Motivating Introductory Computing Students with Pedagogical Datasets

Austin Cory Bart

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

March 22, 2017
Blacksburg, Virginia

Keywords: Introductory Computing, Motivation, Situated Learning

Motivating Introductory Computing Students with Pedagogical Datasets

Austin Cory Bart

(ABSTRACT)

Computing courses struggle to retain introductory students, especially as learner demographics have expanded to include more diverse majors, backgrounds, and career interests. Motivational contexts for these courses must extend beyond short-term interest to empower students and connect to learners' long-term goals, while maintaining a scaffolded experience. To solve ongoing problems such as student retention, methods should be explored that can engage and motivate students.

I propose Data Science as an introductory context that can appeal to a wide range of learners. To test this hypothesis, my work uses two educational theories — the MUSIC Model of Academic Motivation and Situated Learning Theory — to evaluate different components of a student's learning experience for their contribution to the student's motivation. I analyze existing contexts that are used in introductory computing courses, such as game design and media computation, and their limitations in regard to educational theories. I also review how Data Science has been used as a context, and its associated affordances and barriers.

Next, I describe two research projects that make it simple to integrate Data Science into introductory classes. The first project, RealTimeWeb, was a prototypical exploration of how real-time web APIs could be scaffolded into introductory projects and problems. RealTimeWeb evolved into the CORGIS Project, an extensible framework populated by a diverse collection of freely available "Pedagogical Datasets" designed specifically for novices. These datasets are available in easy-to-use libraries for multiple languages, various file formats, and also through accessible web-based tools. While developing these datasets, I identified and systematized a number of design issues, opportunities, and concepts involved in the preparation of Pedagogical Datasets.

With the completed technology, I staged a number of interventions to evaluate Data Science as an introductory context and to better understand the relationship between student motivation and course outcomes. I present findings that show evidence for the potential of a Data Science context to motivate learners. While I found evidence that the course content naturally has a stronger influence on course outcomes, the course context is a valuable component of the course's learning experience.

This material is based on work supported by the NSF under Grants No. DGE-0822220, DUE-1140318, DUE-1444094, and DUE-1624320.

Motivating Introductory Computing Students with Pedagogical Datasets

Austin Cory Bart

(GENERAL AUDIENCE ABSTRACT)

Introductory computing courses struggle to keep students. This has become worse as students with more diverse majors take introductory courses. In prior research, introducing fun and interesting material into courses improved student engagement. This material provides a compelling context for the students, beyond the primary material. But instead of only relying on fun material, courses should also rely on material that is useful. This means connecting to students' long term career goals and empowering learners. Crucial to this is not making the material too difficult for the diverse audience. To keep more students, we need to explore new methods need of teaching computing.

I propose data science as a computing context that can appeal to a wide range of learners. This work tests this hypothesis using theories of academic motivation and learning theory. The components of a learning experience contribute to students' motivation. I analyze how the components of other existing contexts can motivate students. These existing contexts include material like game design or media manipulation. I also analyze how good data science is as a context.

Next, I describe two projects that make it simple to use data science in introductory classes. The first project was RealTimeWeb. This system made it easy to use real-time web APIs in introductory problems. RealTimeWeb evolved into the CORGIS Project. This is a diverse collection of free "Pedagogical Datasets" designed for novices. These datasets are suitable for many kinds of introductory computing courses. While developing this collection, I identified many design issues involved in pedagogical datasets. I also made tools that made it easy to manage and update the data.

I used both projects in real introductory computing courses. First, I evaluated the projects' suitability for students. I also evaluated data science as a learning experience. Finally, I also studied the relationship between student motivation and course outcomes. These outcomes include students interest in learning more computing and their retention rate. I present evidence for the potential of a data science context to motivate learners. But, the primary material has a stronger relationship with course outcomes than the data science context. In other words, students are more interested in continuing computing if they like computing, not if they like data science. Still, the results show that data science is an effective learning experience.

Acknowledgments

I owe tremendous thanks to so many people, without whom this dissertation would never have happened. This entire document is dedicated to them and all of the things they have given me. I apologize to all of my friends, family, colleagues, and inspirations who I did not include by name – you remain in my thoughts and in my heart.

First, thanks go to my advisors and committee members. Dr. Shaffer, for always pushing me to find real questions and to solve real problems. Dr. Tilevich, for your guidance and inspiration in both my research and my career. Dr. Kafura, for letting me into the trenches of course development and trusting my work, even when it was five minutes before class and the server was down. Dr. Jones, for opening my eyes to entirely new ways to look at education, and developing theories that made this dissertation even possible. Dr. Conrad, not only for using my software, but for mentoring and sculpting me into a better researcher.

Next, I must recognize the influence of so many professors from my undergraduate institution, the University of Delaware. Dr. Harvey, for pushing me into research in my second computing class, even though I still had no idea what I was doing. Dr. Atlas, for making me realize that teaching computing was a potential career path. Dr. Pollock, for guiding me through an undergraduate thesis even as I flip-flopped around. Dr. Decker, not only for letting me teach with him, but even using my research software in his classrooms. Dr. Carterette, for taking me on for two summers and introducing me to this strange phenomenon called "Computer Science Research". And finally, to the Honor's Program and Computer Science department, thank you for always pushing me to do better.

A number of student researchers, teaching assistants, peers, and learners deserve my thanks. To Ryan, Omar, and Jason, for all your superb work in helping with the CORGIS project and wrangling those crazy datasets. To Bushra, Javier, and Luke, for all your support in developing and maintaining this course that has consumed our lives. To Kaitlyn, Joey, Alyssa, Ishita, Karina, Maggie, Taylor, Amy, Prab, Lori, Jennifer, Sam, Danielle, and all the other teaching assistants that have grown along with the course, I am continually amazed at what you all have accomplished. To Kevin, Eric, and Mohammed, thank you for making this look easy. To each of the undergraduate students who contributed research data to this dissertation, and especially the ones who had to suffer me as their instructor – I am proud of you, and I value the lessons you have each individually taught me about teaching.

The furry side of my household provided a tremendous amount of personal motivation, and I owe them thanks. To Klaus, my loyal hound, for getting me outside daily to recharge my batteries. To Jiji, my regal cat, for keeping me warm when coding into the dark night and making me so much bread. To Pumpkin, my corpulent kitten, for always keeping me amused and rarely destroying my valuables. To Pele, my ovoid chinchilla, for constantly plotting my downfall but never actually following through... yet. To Wrex, my restive hamster, for tirelessly resisting the threat of the Reapurrs.

My family and friends have had perhaps the biggest influence of all in my life, and I must thank them too. My father, Keith, for his careful and excellent brainwashing me into loving computing at a young age. My mother, Tina, for imparting her zeal for education and belief in the power of educational technology to change the world. Tim and Cindy, for always being a part of my family and supporting me along the way. My brother, Trevor, for always having my back and sometimes letting me win at video games. My newest family – Mark, Michelle, Marty, Emma, and Sam – for welcoming me into your lives so readily. I want to thank my entire family, and all of my friends, for being amazing to me and always being by my side. To my best friend Rebecca Trexler, even though my opinions may be stupid and wrong, they are less stupid and wrong for having you in my life – You are truly the Hobbes to my Calvin.

Finally, and most of all, thanks to my dearest Eleonor, who I love beyond all rhyme and reason. You are the most special thing in my life, and I am not sure what I would do without you. This dissertation would certainly never have happened without you, both on a practical level and a theoretical level. You teach me new lessons every day, and make every moment brighter. I am so excited to continue our journey together, even as things change around us. Ellie, I love you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Low retention rates are an ongoing challenge for Computer Science education [63, 33], even as students of all majors and career paths are increasingly expected to learn about computing. Many institutions (such as Virginia Tech) now offer courses in "Computational Thinking", which cover foundational computing concepts — including algorithm development, data abstraction, and computing ethics — to educate diverse majors from the sciences, arts, humanities, and other fields [92, 60].

These students are often at-risk for a number of motivational hazards. Although the material covered is not always as rigorous or in-depth as traditional introductory computing courses, thinking computationally is often new to these students and they may suffer from low perceptions of their ability. Further, as non-major students often fail to immediately appreciate how computing can benefit their long-term career [38], their resulting motivation can prove insufficient to see them through the more challenging aspects of the curriculum.

This work develops new pedagogical approaches and technical tools to better understand how we can motivate and engage introductory students. Student motivation in this research is modeled with a five-component representation that incorporates students' sense of empowerment, perception of the materials' usefulness, self-efficacy, interest, and their feelings towards other stakeholders in the learning experience [44]. With this richer, multi-dimensional model, I can more accurately capture the nuances of student motivation with regards to the course context. Theoretically, by understanding and then increasing student motivation, we can impact a number of engagement outcomes such as retention and students' intent to continue learning.

The particular approach used in my work is to contextualize introductory computing through pedagogically-driven Data Science. Data Science is a method of answering questions through the use of data and computational methods. Data Science in this dissertation is pedagogically-driven, meaning that the process, scaffolds, and tools are adjusted to best fit the desired learning experience. Ideally, these adjustments positively impact student motivation by retaining the authenticity and spirit of the experience, without negatively impacting student learning or motivation with extraneous complexity.

A major contribution of this work is CORGIS (the Collection of Really Great and Interesting dataSets) a large repository of datasets with associated metadata. A compilation process allows each dataset to be transformed into a wide variety of beginner-friendly libraries for languages such as Java, Python, Racket and BlockPy, as well as file formats like CSV and JSON. The web-based gallery also provides useful and simple tools to explore both the structure and the characteristics of the data, all from within the browser. Using complicated, real-world datasets is a known technique already used in advanced computing courses [29, 82, 85]. However, these kinds of datasets can now be used at the introductory level through the CORGIS Dataset Project.

In this dissertation, I describe the design, development, and evaluation of these tools in introductory computing settings. Further, I conduct novel studies on the nature of motivation, engagement, and learning in introductory computing courses, especially with respect to core course components such as the content and context. The results of my studies are discussed, and their implications for instructors and researchers are given.

1.1 Research Questions

This dissertation attempts to answer the question, "Can a Data Science context motivate introductory computing students, particularly non-Computing majors?" To attend to this question, I focus on a sequence of objectives and questions:

1. How can student motivation in introductory computing be modeled?
2. How does Data Science compare to other introductory contexts?
3. What affordances and barriers does Data Science offer as an introductory context?
4. How can technology support Data Science as a learning context?
5. What is the impact of a Data Science context, as opposed to other course components, on student motivation and course outcomes?

To explore student motivation (Question 1) and the role of Data Science (Questions 2 and 3), I describe multiple educational theories in Chapter 2 to construct a framework that separates the components of introductory computing courses. In Chapter 3, I describe the technology and best practices I have created to mediate the use of Data Science as a learning context (Question 4). Finally, the results of multiple interventions using this technology in introductory computing classes are described in Chapter 4, along with potential future avenues of work. These results are used to answer Question 5.

1.2 Definition of Terms

Many of the following terms are used throughout this dissertation. Some are given further treatment and described in the literature. However, to prepare the reader we present initial, high-level definitions. They are not meant to be universally accepted, but should be sufficient to orient the reader.

Computing: The broad class of knowledge that includes Coding, Computer Science, and Computational Thinking.

Computational Thinking: Solving problems by abstracting information in a manner that can be manipulated using computational agents, including algorithms. Computational Thinking is a general, high-level way of thinking, rather than a specific set of skills. Although coding is a particular way to Think Computationally, there are many ways to approach and model problems.

Coding: Solving problems specifically by writing formal instructions with a syntax and grammar understandable to computers. In particular, this often deals with practical and tactical matters – knowing syntax, understanding how components of an algorithm fit together. There are many strategic elements here too, such as understanding the ecosystem and cultural mileau of how software has evolved.

Computer Science: The scientific/engineering study of how to solve problems at scale better through various computational methods. As a diverse discipline, many kinds of methods are encapsulated in Computer Science. These methods include mathematically rigorous approaches to prove fundamental truths about computing in general and specific algorithms. However, Computer Science also involves methods to solve coding and computational problems using sophisticated strategies and reusable schemas. Although Computer Science overlaps with both coding and Computational Thinking, the field is distinguished by its more formal methodology.

Motivation: The internal force by which students choose to engage in learning. Motivation has a direction and energy.

Engagement: The observable behaviors of motivated learners. Motivation is an internal mental and affective state, while engagement is observable through things like attendance, assignment completion, and voluntary course enrollment.

Chapter 2

Literature Review

In this chapter, I provide background and context for the research. The chapter begins with an overview of several relevant theories in order to develop a conceptual framework for student motivation and course design. This framework is then used in the two subsequent sections to evaluate existing approaches to motivating students in introductory computing. The final section is dedicated to describing the nature of Data Science as a discipline, and its nature with respect to this research.

2.1 Educational Theory

There are many elements of the learning process, and many theories of how people learn, how to teach, and how to engage with students. In this dissertation, I investigate Academic Motivation - the process by which people choose to direct energy towards learning. Motivation is well understood to be a crucial part of education, particularly in Computer Science [16, 71]. I apply motivational models against a learning model inspired by Situated Learning Theory in order to focus on all of the component aspects.

2.1.1 MUSIC Model of Academic Motivation

My work uses the MUSIC® Model of Academic Motivation [44] as its primary motivational framework. Although there are many motivational models available, few strive to be holistic models specifically developed for education. For example, theories like Expectancy-Value and Cognitive Evaluation Theory have a wider scope and have stemmed from other disciplines such as healthcare. The MUSIC Model is derived from an analysis of these other theories, incorporating only the academically relevant components. Further, the MUSIC model is a tool meant for both design and evaluation, allowing it to be used in all phases of this work.

Finally, the model and its associated instrument, the MUSIC Model of Academic Motivation Inventory (MMAMI) , has been extensively validated and used in other educational domains, making it a reliable assessment tool[45].

The MUSIC model identifies five key constructs in motivating students [44].

eMpowerment: The amount of control that a student feels that she has over her learning (e.g., course assignments, lecture topics, etc.). This is the amount of agency that a student perceives.

Usefulness: The expectation of the student that the material she is learning will be valuable to their short- and long- term goals. There is no clear delineation of the time-scale for these goals, but there is nonetheless a distinction between strategic skills that a student needs to be successful in careers and personal interests and the tactical skills she need to complete present-day tasks.

Success: The student's belief in her own ability to complete assignments, projects, and other elements of a course with the investment of a reasonable, fulfilling amount of work. Most students desire experiences that are successful, but still challenging enough to be worth it.

Interest: The student's perception of how the assignment appeals to situational or long-term, individual interests. The former covers the aspects of a course related to attention, while the latter covers topics related to the fully-identified areas of focuses of the student. It can be difficult to parse out the difference between individual interests and long-term career goals – there can be alignment between these two components.

Caring: The student's perceptions of the quality of the relationships between them and others in the learning environment. These stakeholders primarily include their instructor and classmates, but also can be extended to consider other members of their learning experience (e.g., administration, external experts, etc.). It can also be viewed as the extension towards society as a whole.

Students are motivated when one or more of these constructs is sufficiently activated. They are not all required to achieve maximal levels, and in fact that is not always desired – it is possible, for instance, for a student to feel too empowered, and become overwhelmed by possibilities. For some of these constructs, a careful balance is required, and it may not be possible to ever achieve a minimal level; no matter how exciting you make your lecture, you may never convince your students it is interesting, although it is possible that they will still consider it useful and stay motivated. Crucially, students' subjective *perception* of these constructs is a defining requirement and is more important than any objective reality. It does not matter if you believe that your lecture is useful, if you have not convinced your students that it is (although an instructor's beliefs are a powerful tool for convincing their students).

The MUSIC model is often used as an organizational framework and an evaluative tool. As the former, it is a list of factors to consider when building modules, assignments, and content of a course. At all times, instructors can consider whether they are leveraging at least one construct to motivate their students. As the latter, it offers both a quantified instrument (MMAMI) and a structure to anchor a qualitative investigation. The model has also been used directly as an intervention tool: Jones describes a controlled classroom experiment to motivate students by having an experimental group reflect on how a course satisfies the constructs of the MUSIC model. Students were simply prompted to answer the question "How will the material presented here be useful to you?". Quantitative data gathered after the experiment indicated a significant increase in motivation [54]. The MUSIC model is therefore a flexible resource for studying motivation.

The MUSIC model has not been applied to Computer Science education prior to my research, although a number of other motivational frameworks have been leveraged in other studies. Many of these studies often find results that hint at many of the elements of the MUSIC model [71]. For instance, Mitchell describes interest, usefulness, and "intellectual challenges" (success) as three primary elements influencing success in computer science [56] . Although they do not discuss empowerment and social elements, many other studies often suggest these themes. Still, it should be acknowledged that it is an open question whether the MUSIC model can be applied directly, or if the domain has different characteristics from other disciplines.

2.1.2 Situated Learning Theory

A learning experience is a complex sequence of contextualized events, and Situated Learning Theory can help explain the structure of this experience — and in turn, this structure can be cast against the MUSIC model to explain the motivation over the course of a learning experience. Situated Learning Theory, originally proposed by Lave and Wenger, argues that learning normally occurs as a function of the activity, context, and culture in which it is situated [48]. Therefore, tasks in the learning environment should parallel real-world tasks, in order to maximize the *authenticity*. Contextualization is key in these settings, as opposed to decontextualized (or "inert") settings. The key difference is that learning is driven by the problem being solved, rather than the tools available. Therefore, the problem being solved should lead directly to the tool being taught. This argument suggests that this contextualization is a key aspect of motivating students.

Authenticity is a crucial, recurring theme within Situated Learning Theory. All instruction and assessment must be aligned with reality such that success in the former leads to success in the latter. However, there is a nuance here – authenticity is a perceived trait, not an objective one. Students derive value from their learning only if they *perceive* authenticity, regardless of whether the experience is genuinely authentic. Often, authenticity is measured against a perceived *Community of Practice* (CoP), a group of professionals that share a craft or trade.

Situated Learning Theory has been applied to Computer Science before, with mixed results. A paper by Ben-Ari [11] explores its application and limitations. This paper is somewhat hasty in its application of SL Theory by taking a macro-level view – they narrowly look to Open-Source and Industry Software Development communities as the only potential Community of Practice and interpret SL Theory as strictly requiring constant legitimacy, largely ignoring the possibility for gradual development of authenticity within individual courses and modules throughout a curriculum:

> *"What I am claiming is that situated learning as presented in their work cannot be accepted at face value, because it simply ignores the enormous gap between the world of education and the world of high-tech CoPs [Communities of Practice], which demand extensive knowledge of both CS subjects and applications areas. This gap can only be bridged by classical decontextualized teaching in high schools, colleges and universities."* [11]

However, other researchers have found it a useful lens for analyzing curricula. For instance, Guzdial and Tew [38] used the theory to explore and deal with the problem of inauthenticity within their Media Computation project. SL Theory clearly has value, but only as a function of the way that it is applied. In this dissertation, I use SL Theory as a generalized tool for exploring the topic of authenticity throughout introductory Computer Science.

The original work in Situated Learning Theory was not about pedagogy or instructional design. It described how people learn and the importance of context and collaboration, but it did not recommend a particular teaching style or course development process. Subsequent research by Brown [13] and others expanded the theory so that it could be applied to the design of learning experiences. These expansions often naturally dictate the use of active learning techniques, reducing the role of lecture in favor of collaborative, problem-based learning activities. Choi and Hannafin [18] describe a particularly useful, concrete framework for designing situated learning environments and experiences. The Choi and Hannafin framework has four key principles:

Context: Described as "... The problem's physical and conceptual structure as well as the purpose of activity and the social milieu in which it is embedded"[70].

Content: The information intending to be conveyed to the students. In some sense, these can be simple learning objectives. However, the concept also covers the general body of knowledge embedded in a field.

Facilitations: The modifications to the learning experience that support and accelerate learning (commonly done through scaffolds). These can be tools, resources, human supports, and even just general changes to the experience.

Assessment: The methods used to evaluate the learning experience and measure the progress of the student.

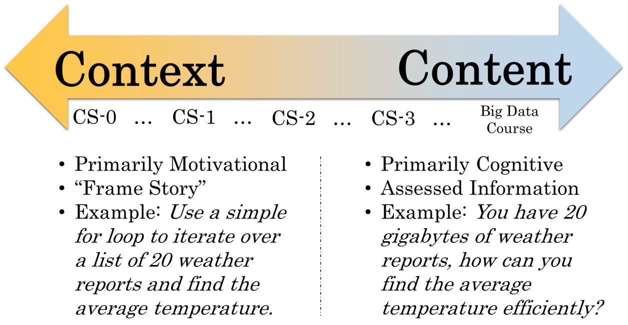
This structure provides a useful framework for analyzing the components of a course. Table 2.1 gives a cross-comparison between these learning experience components and elements of motivation. The columns represent different learning experience components, while the rows represent the elements of motivation. The first row ("Examples") gives examples of the given learning experience components. The other cells of the table are examples of the kinds of questions a learner would ask themselves with regards to the components.

Table 2.1: Components of Motivation across Components of Situated Course Design

	Context	Content	Facilitations	Assessment
Examples	Game Design, Web Development, Data Science, etc.	Functions, For loops, Computer ethics, code tracing, etc.	Algorithm visualizations, teaching assistants, etc.	Exams, Quizzes, performance review, code review, etc.
eMpowerment	Am I restricted by the context to explore what I want?	Do I have control over the depth/breadth/direction of what I am learning?	Do these scaffolds help my learning, or do they artificially prevent me from exploring what I want?	Does this assessment allow me to explore my strengths and weaknesses?
Usefulness	Is this situated in a topic that's worth learning?	Is the content itself worth learning?	Do the scaffolds let me learn enough to be useful?	Do I feel that the assessment is important?
Success	Do I believe I can understand this context?	Do I believe I can understand this material?	Do these scaffolds hinder me or help me?	Can I suceed at this assessment?
Interest	Is this situated in something I find boring/interesting?	Is the material inherently interesting?	Do the scaffolds support my interest?	Am I interested in the assessment experience?
Caring	Does the context give opportunities for the instructor and peers to show they care?	Does the content give opportunities for the instructor and peers to show they care?	Do the scaffolds give opportunities for the instructor and peers to show they care?	Does the assessment give opportunities for the instructor and peers to show they care?

Contexts vs. Content

Figure 2.1: Content vs. Context


Context
Content
CS-0 ... CS-1 ... CS-2 ... CS-3 ... Big Data Course
• Primarily Motivational
• "Frame Story"
• Example: Use a simple for loop to iterate over a list of 20 weather reports and find the average temperature.
• Primarily Cognitive
• Assessed Information
• Example: You have 20 gigabytes of weather reports, how can you find the average temperature efficiently?

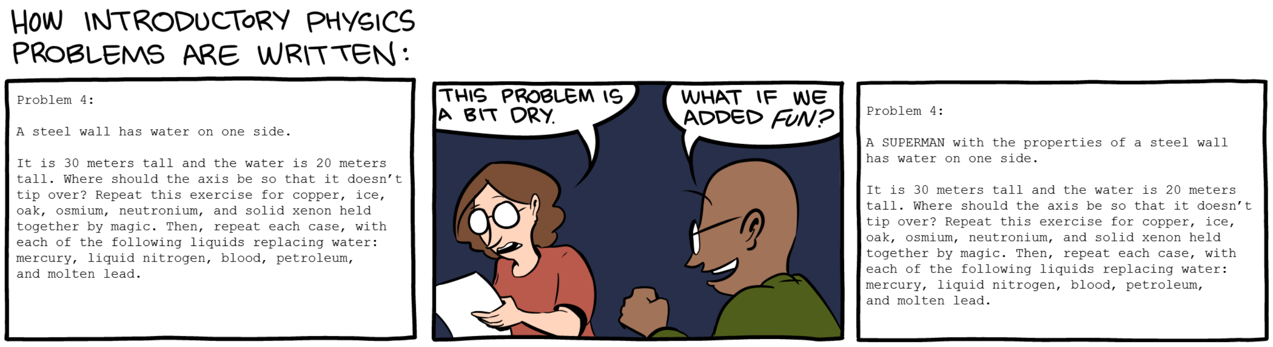
There is a reciprocal relationship between contexts and content. Figure 2.1 demonstrates an example of this relationship for the expected emphasis on big data as a context vs. content throughout an undergraduate curriculum, from a CS-0 (non-majors) course all the way to an upper-level course specifically on big data. Consider teaching Big Data as a topic: while an upper-level course could naturally have big data as both context and content, an introductory course would be unlikely to have it as content, although it could be used as a framing story for assignments. Of course, many topics often bring some associated content with it. When students learn programming in the context of, say, game development, they are almost necessarily learning content related to game development that may not be universal to computer science (e.g., how graphical resources are organized and accessed within the game engine). This content may be seen as a distraction by the instructor, or as useful side knowledge. For example, if a student had to learn how to use a command-line program in order to compile their game, they would be learning an authentic skill that might not be considered part of the core content, but is nonetheless generally useful. When evaluating a context, it is useful to consider what content it represents, and how authentic and useful it is. The authenticity of content that is attached to a context affects the authenticity of the learning environment as a whole. A good context enables a student to find recognizable elements and build on prior understanding, eventually being able to freely transfer their learning to new contexts.

Fascinatingly, the need for a strong context diminishes as learners mature and become domain-identified – the content itself becomes the context. Learners start to see other contexts as nothing more than distractions and unnecessary. This makes sense – you would hope that Computer Science majors in their third semester would be naturally interested in the material, and this is borne out in experimental data. For instance, Yarosh & Guzdial attempted to integrate Media Computation in a CS-2 (Data Structures) course, and found that the learners had "outgrown the desire for a context" [93]. These results are similar to those found in my interventions with a course on Data Structures and Algorithms, where students seemed more irked by the surrounding context than intrigued, as described in Chapter 4.

Of course, it is up to the instructor to choose the depth and breadth of the context's integration. The trade-off between the value and distraction added by the context is a delicate formula. Consider the scenario in Figure 2.2. A steel wall is a relatively relatable concept

Figure 2.2: How to Add Fun to Education

Making the context "Fun" is not necessarily trivial, whether in physics education or computer science education. [87]


HOW INTRODUCTORY PHYSICS PROBLEMS ARE WRITTEN:
THIS PROBLEM IS A BIT DRY.
WHAT IF WE ADDED FUN?

Problem 4:

A steel wall has water on one side.

It is 30 meters tall and the water is 20 meters tall. Where should the axis be so that it doesn't tip over? Repeat this exercise for copper, ice, oak, osmium, neutronium, and solid xenon held together by magic. Then, repeat each case, with each of the following liquids replacing water: mercury, liquid nitrogen, blood, petroleum, and molten lead.

Problem 4:

A SUPERMAN with the properties of a steel wall has water on one side.

It is 30 meters tall and the water is 20 meters tall. Where should the axis be so that it doesn't tip over? Repeat this exercise for copper, ice, oak, osmium, neutronium, and solid xenon held together by magic. Then, repeat each case, with each of the following liquids replacing water: mercury, liquid nitrogen, blood, petroleum, and molten lead.

for most students – they can readily imagine such a large, durable object, and it somewhat reasonable to expect that objects would interfere with it. In this scenario, the instructors consider replacing the wall with a comic book character – something that they anticipate will be more "fun". If they are in tune with their learners, this might be an effective context – perhaps they know that their learners are comic book fans. However, because the integration is only at the surface level, it is possible that the learners will see this as a forced reference, and they will have a more negative reaction. It is also possible that they will not recognize the reference, or feel no positive emotions with it – many contexts do not take into account gender, racial, or socio-economic characteristics of the anticipated learner. The process of motivating students using a context is non-trivial, and in the following section I will explore prior work in contexts for Computer Science Education.

2.2 Introductory Computing Content

Different Computer Science programs have different introductory curricula, depending on whether they focus on Object-Oriented programming, Functional programming, etc. Complicating this discussion is the separation of undergraduate introductory computing into Computational Thinking (sometimes referred to as CS-0) and Computer Science (sometimes referred to as CS-1), which are also distinct from the introductory curricula used in K-12 education (e.g., the new AP CS Principles course [4]). However, it can be argued that any good introductory computing course will cover certain topics, such as abstraction (representing complex phenomena more simply, usually as coded data), algorithms (using formal language to specify computational solutions to problems), some form of decision-making construct (e.g., `if` statements, `cond` statements), and some form of iteration (e.g., `for` loops, recursion) [47, 1, 42]. Of course, each curricula will lay out the material differently. The "How to Design Programs" curriculum emphasizes a functional programming model, using a LISP-descended language named Racket, and the only looping mechanism that students are taught is recursion [31]. The dominance of the Object-Oriented Model in software engineering usually leads to a strong emphasis in introductory courses on abstracting data using objects and classes – such as the "Objects First" curriculum [19]. Beyond just technical knowledge, curricula also include softer skills such as a tolerance for unstructured problems and collaborative attitudes [42, 35].

This dissertation does not take a strong view on what should be included in an introductory computing experience. However, for practical purposes, materials and research work are grounded and tested in actual courses, and have adaptions for that content. Specifically, most of my work is used in a Computational Thinking course, so that is how the content is oriented. "Computational Thinking" was coined by Seymour Papert in 1993 [62] and popularized by Dr. Jeannette Wing's 2006 paper [92]. Unfortunately, there is still limited consensus on *what* exactly CT is, whether it should be universally taught, how it should be taught, and how to identify when it has been taught.

An excellent resource for summarizing the history of Computational Thinking research is the 2013 dissertation by Wienberg [86]. This comprehensive survey analyzed 6906 papers directly or indirectly related to Computational Thinking from 2006-2011, describing research efforts and findings. Over half of the research on CT describes approaches to pedagogy (Curriculum and Program Description), leaving a small amount to modeling (Philosophy and Opinion) and assessment (Research and Evaluation). The lack of assessment research is understandable given the youth of this area of research, but still troubling. Even more troubling, however, is the further analysis of the 57 empirical studies. Only fifteen (26%) studies include or sought an operational definition of computational thinking, and only six go beyond the superficial (solely describing computational thinking as a "way of thinking", a "fundamental skill", or a "way of solving problems"). The failure to identify an operational definition weakens the theoretical strength of the studies. This weakness likely stems from the background of the researchers: only 18% of the articles involved education experts.

Weinberg reflects on the continuing debate about the importance of Computational Thinking:

> Many, like Wing, believe computational thinking to be a revolutionary concept, one as important to a solid educational foundation as are reading, writing, and arithmetic (Bundy, 2007[15]) (Day, 2011[21]). Others believe its potential and significance are overstated (Denning, 2009[22]; Hemmendinger, 2010[41]), and some have voiced concern that by joining forces with other disciplines computer science might be diluting either one or both of the participating disciplines (Cassel, 2011[17]; Jacobs, 2009[43]). Both the praise and the criticism for computational thinking could perhaps be tempered by reflecting on a historical quote by Pfeiffer in 1962: "Computers are too important to overrate or underrate. There is no real point in sensationalizing or exaggerating activities which are striking enough without embellishment. There is no point in belittling either." (Pfeiffer, 1962[66]).

Although it is ambiguous what Computational Thinking is, we will take it as a given that it requires students to learn some amount of non-trivial programming: using iteration constructs (e.g., `while`, `for each`, recursion, etc.), decision constructs (e.g,. `if`, `unless`), have some sense of program state (through mutating variables or passed through composed functions), the basics of input and output, and require the programmer to create instructions in a form the computer can understand. This is not meant to be a strict definition of everything a programmer should learn – simply an acceptable, minimal subset. Chapter 4 describes a concrete Computational Thinking curriculum used for this research project.

2.3 Introductory Computing Contexts

As part of the overarching goal to bring more students into Computer Science, a large number of contexts have been explored in introductory computing. The context of a learning experience grounds the learner in what they already known, in order to teach the new material. Many introductory computing experiences focused on presenting the content as purely as possible, which can come across as abstract and detached [94]. However, starting with Seymour Papert's work with robotics and the LOGO programming environment in the 70s [62], instructors have been interested in motivating students' first experience with richer contexts. Some of these contexts rely on Situational Interest (e.g., Digital Media "Computation" (Manipulation) [32] and Game Design [94]), while others attempt to provide enduring career value (e.g., Big Data Science [3]) or short-term social applicability (e.g,. Problem Solving for Social Good [34]). Although each of these approaches draws on different facets of motivation, they may be compatible with each other. In this section, I will discuss the implications of these different approaches.

2.3.1 Abstract Contexts

Denning describes the early perception of Computer Science by the public as "stodgy and nerdy" [23], since many early computer science classes were driven so strongly by mathematics and logic. A common early introductory programming problem, for instance, is writing a function to compute a Fibonacci number – a relatively simple task if you are familiar with the recurrence, and one that leads quite nicely to discussions on the implementation of algorithms, computational complexity, and a host of other subjects [57]. These contexts are "abstract" because they are already at a similar level of abstraction as the content they are attempting to convey. However, Oliveira [61] suggests that the discussion about "abstract vs. concrete" contexts is a misleading one, because the purity with relation to the content is less important than *prior knowledge.* According to modern constructivist and cognitivist theories, learners build on prior knowledge, and the ability to relate to what they know is crucial. The simple fact is that most students are not good at mathematics, so relying on it as a context is not a useful approach compared to finding subjects that students know and understand readily.

Kay [46] identifies a potential problem of relying strongly on contextualized experiences. What happens once a student has completed the introductory course and is ready to move onto further courses? Most later courses are more decontextualized, and will not use contexts such as game development, robots, etc. Kay goes so far to say that it is unethical to suggest to students that a contextualized introductory course is representative of the curriculum as a whole. Transfer (transferring knowledge and skills between contexts) is a difficult problem in education, so it is unclear whether this argument has merit.

2.3.2 Situationally Interesting Contexts

As it became clear that Computer Science had a serious image problem, work began on making Computer Science "fun" and approachable. A key goal was to increase diversity and to broaden participation. This led to the rise of situationally interesting contexts, emphasizing problems and projects that would be immediately appealing to a wide audience. Guzdial, for instance, was largely responsible for the creation of the Media Computation approach, where students use computational techniques (e.g., iteration and decision) to manipulate sound, images, videos, and other digital artifacts. As an example, students might use a nested, numerically-indexed `for` loop to adjust the red-value of the pixels in an image, treating it as a two-dimensional array of binary tuples, in order to reduce red-eye in a photo. A major advantage of this approach, besides their appealing nature, is that it can make the results of the computation more tangible (e.g., visible pixels as opposed to numbers in memory).

Although wildly deployed, a review of these curricular materials by Guzdial [38] in light of Situated Learning Theory found that students did not find this an authentic context, and intense rhetoric was insufficient to convince them that it was authentic. Few students find it expedient and helpful to remove red-eye from family photos by writing python scripts, and so are unconvinced that the context has long-term value to them (regardless of whether the content does). Guzdial leaves open the question of what contexts can be truly authentic for non-majors, given the relative novelty of teaching introductory computing for non-majors. Ben-Ari echos this question by suggesting a narrow selection of authentic contexts and communities in his paper exploring the application of Situated Learning Theory to Computer Science [11]. Critically, the opposite problem could occur – if an instructor is effective at convincing students that a context is authentic, the students may believe the instructor even if the context is not authentic. There are serious ethical issues involved in mispresenting the utility of a context, leading students to develop an embarrassing misconception of the field. Imagine a young child believing that all of Computer Science is game design, because that is what they started off doing.

There are other disadvantages of an interest-driven approach. The motivation literature describes "seductive details" (interesting but irrelevant adjuncts) [40] as interfering both with short-term problem completion and long-term transfer. In other words, students get hung up on unimportant aspects of the context, so that they ignore the content. Consider a student using the game and animation development environment Scratch, which allows beginners to create sprites from images. A young learner might be so amused by the ability to change the color and shape of their image, that they neglect their assigned work. Although a well-regulated learner would not be distracted, most of the at-risk population that would benefit from these contexts is unable to deal with such distractions. Of course, this could be said of any context, but there is particular danger from an intentionally interesting context.

2.3.3 Empowered Contexts

Empowering a context is the idea of giving students more freedom and agency to control their learning. Compared to interest-driven contexts, this approach is comparatively less researched, although it is not an uncommon practice. Instructors will often allow students to choose from a range of projects or assignments. Stone [75] ran a two-year study where students were allowed to choose their projects from a wide range of domain areas (e.g., Biology, Math, Business, Etymology), and were then surveyed on their engagement. Unfortunately, their experiment suffered strongly from low enrollments and even lower survey responses. Therefore, it is difficult to trust the reliability any of their results (including the idea that women are more likely to prefer biological- and meterological-themed projects). However, their experiences do suggest an interesting challenge: normalizing the difficulty (both in terms of computational knowledge and domain knowledge) across many different projects and assignments is a struggle. Often, instructors will find pieces of the learning experience that students can customize, giving a sense of empowerment without undue burden on the instructional designer.

2.3.4 Contexts that Promote Caring

Most modern educational theories argue that learning is inescapably affected by social factors. There is evidence that the instructor [81] and fellow students [5] are the most important factors in an introductory experience, for instance. Kay [46] discusses this explicitly. It is possible that the most important element in a context is not whether it is fun or useful, but whether the instructor can get excited about it and impart that enthusiasm to the student. And not just enthusiasm, but a thorough understanding of the problem, its usefulness, and the rest of its attributes. By this theory, any context can promote caring if the instructor is suitably enthusiastic, suggesting the value in finding aligned context preference between the instructor and the learners.

2.3.5 Useful-Driven Contexts

The final motivational component to consider in contexts is usefulness, the idea that the context should have immediate or eventual benefit to the learner's needs. The ideal situation for any instructor is to create contexts that specifically suit the long-term needs of their learners [25]. To some extent, it is impossible (or at least prohibitively difficult) to find a one-size-fits-all context that will be useful to all learners (designing learning experiences without learners in mind is an example of preauthentication [65]). However, in practice, some contexts are broadly useful and are likely to engage a diverse crowd of learners. In this section, I suggest two distinct contexts that might fall into this role: real-world problem solving and Data Science.

At a high level, computer science provides tools for solving problems in a wide range of complexity and abstraction [49, 34]. An ITiCSE Working Group collaborated to produce a new framework centered around these ideas – "Social Computing for Good", a collection of approaches and projects for interdisciplinary students to solve using computing [34], which they felt would provide a motivationally powerful context for learners. They raise a number of issues with using socially relevant materials: that students and instructors can become cognitively overloaded by the addition of domain knowledge, that instructors can even be intimidated if they don't have expertise in the domain area, and that games and graphics can appeal to instructors as a "cheap" source of motivation. The working group also created a valuable rubric for developing and evaluating problems, which I map to elements of the MUSIC model below:

1. The degree to which the problem is student-directed (eMpowerment)
2. The amount of scaffolding needed (Success)
3. The amount of external domain knowledge needed (Success)
4. The contribution to the social good (Usefulness, Caring)
5. The "coolness" or "sexiness" (Interest)
6. The amount of explicit student reflection incorporated (Usefulness, since it directs students to reflect on the value of the problem with relation to the students' goals)

Although this framework presents some ideas, there are still unsolved technical and pedagogical problems in how to optimally bring these materials to learners. Their paper ends by raising questions about the effectiveness of this approach compared to existing methods.

A number of other researchers have created course materials along similar lines. Erkan [30] had a sustainability themed curriculum – student surveys suggested some level of effectiveness, although the sample size (N=16) was far too small to generalize the results. In addition to providing two case studies, Buckley [14] suggests an interesting delineation between Social problems (uppercase S, indicating problems general to society) and social problems (lowercase S, indicating problems of personal relevance to the learner).

Barker ran a large survey asking why students persist towards majoring in CS (N=113, only freshmen) and performed a factor analysis [5]. Critically, they found that meaningful/relevant assignments (subsuming both interest and social usefulness) were a major factor in whether students would persist in the major. However, it wasn't one of the primary factors (that is, students suggested other factors were more important for deciding whether they would persist). Interestingly, whether students felt that their workload and pace was appropriate was a much bigger source of importance, suggesting that care and attention should be given to making a context suitably difficult before it is made interesting and useful. This focus on ensuring normalized, appropriate difficulty is echoed by several other authors, all of whom suggest that doing so is not trivial [67, 74].

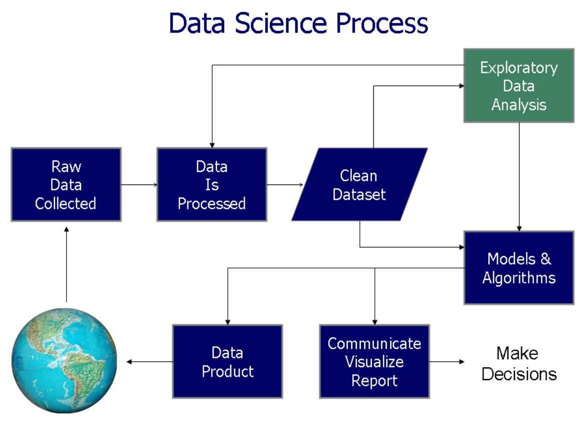
In the past two decades, the field of Data Science has emerged at the intersection of computer science, statistics, mathematics, and a number of other fields. This field is concerned with answering real-world problems through data abstractions, and offers a less socially-conscious path to usefulness. As a context, there are pedagogical penalties for using it, since it introduces a wide variety of new content including visualization, statistics, ethics, and social impacts [2]. However, an instructor can downplay the focus on these topics, or emphasize subject matter's strengths (e.g., a statistics major might find it interesting to use their mathematical background to strengthen their problem-solving investigation). However, there are other difficulties. Bringing in messy data requires programming and data management expertise of the instructor, especially when working with Big Data.

The use of data analysis as a form of contextualization is not novel, but represents a new and actively growing movement where instructors create programming assignments with specific datasets in mind [3, 76, 39, 24]. Upper division courses have employed these situated learning experiences using data of varying size and complexity for several years [29, 82, 85]. However, in all of these research papers, there is typically little evaluation of the advantages and disadvantages of using Data Science in introductory education. Although Sullivan [76] did conduct a study on the difficulty and usefulness of the datasets they provided, they do not go far in identifying lasting lessons for educators creating such datasets. Other researchers conducted the barest of assessment: DePasquale [24] included exactly ONE anecdotal student response in his evaluation.

2.4 Data Science as a Holistic Motivating Context

In this dissertation, I hypothesize that Data Science, involving real-world datasets, can be an effective introductory computing context. My core thesis is that Data Science, used properly, is an excellent context for motivating students across multiple components of motivation: providing opportunities for agency, a sense of usefulness and authenticity, controllable com-

Figure 2.3: A Generalized Model of Data Science [89]



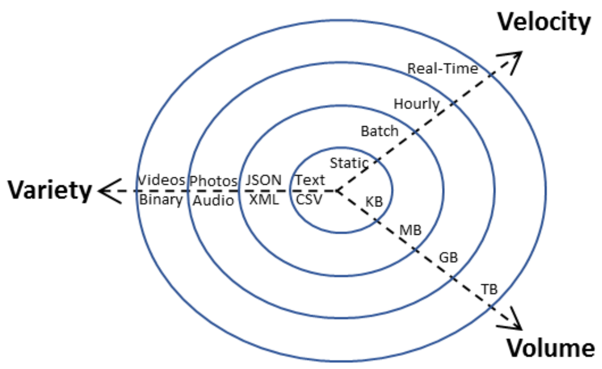
plexity, the potential to be interesting situationally and for individualized interest, and a promotion of caring and ethics. Data Science accomplishes this by being a "meta-context" – it is a general framework for working with a wide array of different data contexts. However, there are many challenges in doing so – managing data is tricky and can be a unstructured problem. For introductory students, in fact, it can be too overwhelming to find and dive into an arbitrary dataset, requiring significant scaffolds.

As established in the previous subsection, this is not a novel approach – upper-level courses have historically used datasets in courses on Machine Learning and Database Design, and recently lower-level courses have started too [3]. However, there is little evaluation of the success of this approach, and numerous roadblocks exist in using this context. For my dissertation, I will investigate what technology can be used to take full advantage of this context within a number of constraints. Before describing what I have done and will do, however, this section will describe this context in more detail.

2.4.1 Data Science

Data Science is the process of answering questions by building, exploring, and processing datasets. There are many theoretical models that define the term more strictly, but in general it is described as an iterative model of collecting, sanitizing, processing, rendering, and interpreting information. Figure 2.3 gives a visual overview of this process. My research will not attempt to narrowly define Data Science; instead, the goal is simply to use elements of the Data Science process to contextualize the experience of learning about computing topics such as abstraction and algorithms.

Figure 2.4: The 3V Model of Big Data



It is crucial to understand that the goal is not to teach students how to become data scientists, anymore than it is the goal of Media Computation to teach students how to be professional computational artists. As a context, using datasets is simply a means to an end – it may involve any components of the generalized Data Science process at any level. Some professors may identify Data Science as a learning objective in and of itself. That is fine, but they are attempting to teach something that is, at some level, distinct from computing proper.

2.4.2 Big Data

One particularly important type of data is known as Big Data. Big Data is much in the news these days, from reports of massive data dredging by the NSA to tens of millions of credit cards stolen by hackers from commercial databases. Big Data has become crucial to scientific advances from understanding the genome to predicting climate change. Even more so than regular Data Science, there are many obstacles to effectively educating students on Big Data. Its representation, manipulation, and expression is, by definition, challenging, and modern curriculum and programming tools are typically inadequate.

Big data has been loosely described as quantities of information that cannot be handled with traditional methods [53]. But "traditional methods" is a vague phrase that has different meanings to different learners. To a Humanities major in their first CS-0 course, the traditional method to sum a list might be to use Excel. In this scenario, "big data" means anything that won't comfortably fit into Excel's working memory or tabular abstraction model. However, to a third-year Computer Science major, the traditional method might be to write an iterative or recursive sequential loop over a file of data; being given a data file that spans multiple terabyte drives might force them to explore parallel models of execution. Clearly, "bigness" is a function of the learner's experience, but that is still not a solid definition. A more precise definition is the "3V Model" [26], which posits that there are three dimensions that distinguish big data from ordinary, run-of-the-mill data:

Volume: The total quantity of the information, usually measured in bytes or number of records. However, this also extends laterally: the number of fields in the structure of the data also impacts the complexity and size. The threshold at which data becomes big is a function of the hardware and software being used. For instance, embedded systems may consider gigabyte-sized files to be big, while modern servers might not struggle until the petabyte level.

Velocity: The rate at which new information is added to the system. High velocity big data implies a distributed architecture, since new data must be arriving from somewhere. The dynamicity of data can vary widely across these architectures, with data updating every year, every day, or even multiple times a second.

Variety: The format or formats of the data. Ideally, data are always distributed in a way that is readily accessible. For instance, simple text-based formats such as CSV and JSON are widely supported, relatively lightweight, and human-readable. More sophisticated data formats for image and audio are also typically well-supported, although still more complicated. However, projects using specialized, compressed binary formats or, more dangerously, multiple formats (e.g., image archives organized with XML files), are more complex.

Silva et al taught an introductory course [72] truly focused on techniques for tackling Big Data: NoSQL, MapReduce, NewSQL. Unfortunately, Silva did not conduct any kind of evaluation of their work across any of the expected dimensions. Learning to work with Big Data can add extra authenticity to the context, but it also raises a large number of new challenges. Once again, it is not the goal of my research to teach students how to work with truly "Big Data" datasets. Instead, I will explore whether the use of Big Data is a useful means to motivate students.

2.4.3 Pedagogical Datasets

Literature on designing and developing datasets specifically for learning experiences (what I will subsequently refer to as "pedagogical datasets") is scarce. There have been some projects that attempt to fill this gap. The UCI Machine Learning Repository is an attempt to collect and organize a plethora of pedagogical datasets suitable for introductory Machine Learning topics [50] – although an impressive collection, the materials are targeted for specific tasks in advanced computing topics. The STARS project sought to make real-world datasets available for introductory statistics courses [12]. Although they did create a small repository of datasets, the project seems to have ended some time ago.

More crucially, neither of these projects sought to research the formal process of developing pedagogical datasets. Radinksy et al. poses some design principles for integrating datasets into inquiry-based learning curriculum, which includes a call for cultivating data but stops short of best practices for organizing and disseminating said data [68]. Verbert describes organizational principles for pedagogical datasets for educational data, specifically for Learning and Knowledge Analytics [83]. But neither of these projects attempted to codify best practices of preparing generally appealing Pedagogical Datasets for introductory computing.

Chapter 3

Technology

In this chapter, I describe the design and evolution of technology developed to facilitate a Data Science context in introductory computing courses. The first section details my first attempt at integrating Data Science into introductory courses: the RealTimeWeb project. The second and larger section is about the CORGIS project, the final iteration of the concept. In addition to the technology of CORGIS, that section enumerates design opportunities and challenges inherent in the process of creating pedagogical datasets.

To begin this section, I will describe design goals consistent across my technology. First and foremost, the technology must be hassle-free. Second, As a scaffold, the presence of the technology should be as unintrusive as possible. The technology here contributes to the context, and therefore should limit the amount of content that it introduces. Third, as a research project, all technology and resources should be open-source and open-access. Finally, the tools must contribute to the authenticity of the learning experience – they should limit their artificiality as much as possible, and follow professional practices if they can.

3.1 RealTimeWeb

The initial research project that led to my dissertation work incorporated real-world systems into introductory computing courses. This led to the development of the RealTimeWeb (RTW) project [8], which provided libraries that simplified the process of connecting to real-world, web-based data APIs for various services. Every library of RTW was available in three major introductory programming languages: Java, Python, and Racket. Data was returned at different levels of complexity. The goal of this system was to allow instructors to peel away layers of scaffolding around the client library to expose successively more inner details of the process. Initially, the library would return fully-cleaned data in proper object-oriented style. Instructors could specify a different function to have the system return uncleaned data using raw data structures (e.g., Hashmaps instead of classes). Finally, at the lowest level, the library would return a raw string of data, requiring the student to parse the data themselves. At all levels, the library would still be responsible for managing calls to the online service and retrieving the data.

Internalized Data Cache

Figure 3.1 demonstrates the flow of data when using a client library. A student using the library calls a function that is meant to retrieve real-time data (such as the current temperature in a city, as an integer). This request is formatted into an appropriate HTTP request, and then sent to either the web service or the local cache. Either way, the resulting data (e.g., JSON-encoded data) is returned asynchronously.

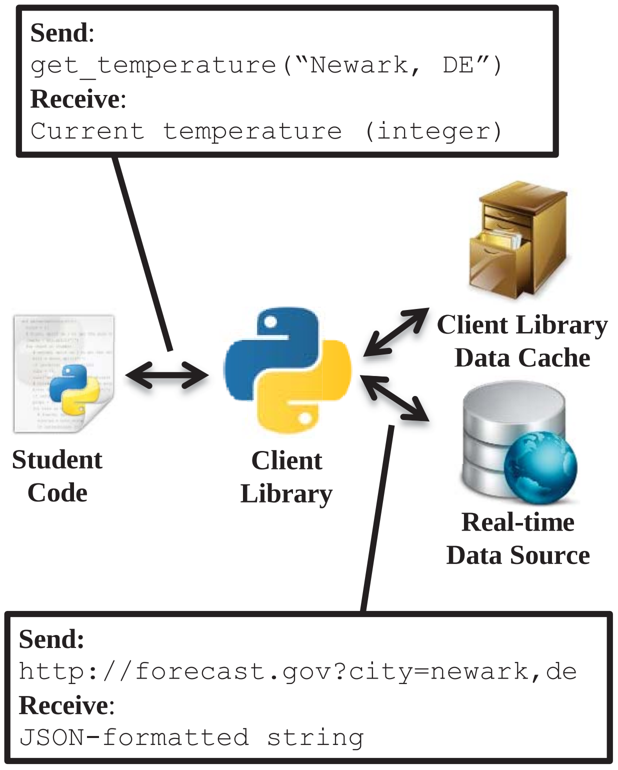

Send:
get_temperature("Newark, DE")
Receive:
Current temperature (integer)
Student Code
Client Library
Client Library Data Cache
Real-time Data Source
Send:
http://forecast.gov?city=newark,de
Receive:
JSON-formatted string

Figure 3.1: Client Library Architecture

Perhaps the single most useful feature of the RTW architecture was the *internalized data cache*. The cache interrupted requests directly to the actual data source, instead accessing a local, static data store (a file). The cache option offers a number of advantages:

Idempotency: By its nature, real-time data is subject to rapid temporal changes. Weather forecasts change on a daily basis, for example, and services like Reddit or Twitter change by the minute. Developing a program that uses such volatile data can be tricky, since it can change between runs of the program. To accommodate introductory students, the libraries make it possible to used cached local data, so consistency of input from run to run, if only for testing purposes, is guaranteed.

Consistency: The content and structure of the web is highly dynamic, as are the web services composing it. Because service APIs commonly evolve at a dizzying pace, the libraries that depend on them must be updated accordingly. Even during a single semester, changes in the data source could be introduced that would result in an out-of-date library. However, this problem can be bypassed by keeping to the internal data cache until an updating fix is released, avoiding any serious delays in student projects.

Connectivity: Although most campuses have gigabit internet connections, this is not universal. Additionally, many students live in off-campus settings with varying internet capabilities. Requiring students to develop an internet-based application with poor internet connections can be an impediment to their learning. However, the cache allows client libraries to be used offline, if necessary.

Efficiency: Even assuming a fast and stable connection, the performance of many web services is limited by the number of connections that the server can handle. Student developers can be polite consumers by developing with client-side data, greatly reducing the number of calls to the online service. This is especially important since many APIs throttle the number of requests (e.g., 100 API calls per day), sometimes penalizing abusers with time-outs and bans. To avoid this, web service clients typically use a polling architecture (with an artificial delay between requests). Running with local-data negates the need for the delay, and makes large-scale testing practical.

Tests: Instructors can use the cached data for testing, ensuring uniform coverage by all students. There is no need to worry about students missing out on edge cases not covered during a particular time interval, since they can be provided with the important cases in test files. Since the cache is stored in a convenient JSON-based data format, it can easily be modified by instructors to return specific results. The ability to ensure consistent tests is particularly useful for automated grading systems, such as Web-CAT [28].

3.1.1 Usability Evaluation

We piloted our Business Service client library in a CS2-style course offered at Virginia Tech (CS 2114) during Spring 2013. This course covers topics typical to a second semester Computer Science course, including object-oriented concepts and the Java programming language. All students in the class are Computer Science majors and minors in their first or second year, with mostly limited prior programming experience. Every week, students use paired

Statement	Strongly Disagree	Disagree	Somewhat Disagree	Somewhat Agree	Agree	Strongly Agree	Total
"I found the content of the... lab to be applicable to my day-to-day computing experiences."	0	0	1	9	5	2	17
"The classes in the API were clear to use based on their names."	0	0	1	4	7	5	17
"The methods of the API were clear to use based on their names."	0	0	0	5	12	0	17
"Overall, the API was easy to use."	0	0	0	2	14	0	16

Table 3.1: Results of the CS 2114 survey on the Business Service assignment

programming to complete a lab assignment. There are also three large projects meant to be completed over the course of the semester.

During one lab session, students were assigned a multi-part problem that required the use of the Business Service library. To personalize their experience with the library, the internal data cache was filled with data from the Blacksburg area. The description of the lab began with an overview of the problems and advantages inherent in using real-time data, and then described how the Business Service library could be used. In the first task, students used the library to create a list of highly rated businesses in the area. In the second, they expanded that list with more detailed information. The primary intent of this lab was to familiarize students with the client library.

Students had a second opportunity to work with the library in the third project of the course, where they built a more complicated "Restaurant Guide" Android application. Students again searched for a list of restaurants in a given region. This time, however, they were required to place the results in their implementation for a circular linked list data structure.

A voluntary survey was administered to the class after students had completed the project, and 17 responses were gathered. Overall, the results were positive. All but one student indicated that they found the API "applicable to day-to-day computing experiences", and all students thought that the API was "easy-to-use". Qualitative feedback from the students highlighted "clear, easy-to-figure-out methods" and that they "provided information that was easy to work with". Full results from the survey are reported in Table 3.1.

The Flaws of RealTimeWeb

The original proposal for the RealTimeWeb system described using WSDL files (Web Services Description Language) as a way to automatically build client libraries from a specification. Explorations of the web-based APIs available revealed that few published a WSDL specification for their service.

Limited APIs: By the end of the RealTimeWeb project, it became apparent that there simply were not many publicly available APIs. The APIs that were available tended to fall into a limited number of categories: stock trading, social media, digital media, etc. Although some of these services had potential, they were typically biased towards high-technology systems and not generally interesting for the average student. There were few real-time APIs related to topics such as civics, history, biological sciences, chemical sciences, or any of a number of disciplines.

Many of these services further imposed restrictions that severely limited their value as high velocity data streams. Twitter, for instance, limits API calls to just 25 tweets at a time. Facebook required individualized app registration for every user. Although the RealTimeWeb project could overcome many of the more technical limitations, it greatly reduced the effectiveness of the resulting system.

Maintenance: Curating the datasets became an enormous task. First, although each dataset was generated from a specification file, most needed to be refined by hand for each library. The architecture was unable to handle the idiosyncrasies of each API. For instance, the stock trading service relied on the Google Finance API, which incorporated dummy characters in results for esoteric client purposes – before the result could be parsed, these extraneous characters had to be removed.

Although the RealTimeWeb project greatly simplified the process of creating web-based libraries by using configuration files to generate libraries, this approach failed to scale. Once a library was generated, it became an independent code base with its own copy of the baseline process needed to access data – duplicating a tremendous amount of code between libraries. Novel features of each library would become mired in boilerplate and library specializations, making it extremely difficult to navigate and update a library over time. For example, consider the differences between the Gutenberg Books library and the Weather library, both of which exposed a single function to connect to an online data source and returns a data structure mixing lists and maps. These two libraries are significantly similar except for their initial method to submit the web request and their final method that processes the retrieved data into the proper form. When an update is made to the web API, the developer must identify these two functions and make modifications, without disturbing the unchanging surrounding boilerplate. Even worse is when improvements were needed to the core architecture. Despite sharing a common general architecture, each library was an independent code base with minute modifications. Therefore, a change to the architecture must be propagated to three dozen other codebases.

A second problem with the original architecture was the disorganization of the documentation and metadata that is associated with each library. Getting to know an API is a difficult process akin to learning to a new language. Supplementary documentation, including tutorials and API references, are necessary. The RealTimeWeb project provided tools for documenting the libraries it generated, but these were limited to creating simple API reference materials that were not adaptable to different levels of learners and did not instruct the learner on its use. Creating tutorials to use the libraries was a manual, cumbersome effort that was redundant across similar libraries. Further, RealTimeWeb had no tooling to generate supporting documentation related to metadata for the library – information such as origin of the data, explanation of terminology used, and terms of its use.

Varying Hardware: Students using RTW software can have computers with different computational power: seniors might have a laptop from their freshman year, or run an older version of Mac OS. Ideally, the students should always be able to run their code quickly and efficiently while developing, without noticeable lag from their programs. However, several libraries suffered from bad caching strategies and poorly-sized datasets, resulting in poor performance that could frustrate beginners. A 100 MB library that runs fine on a developer's new machine can be slow on a students' older laptop. In one incident, a student had to complete their final project with each execution taking up to a minute to run – even when they were only doing an initial data import.

3.2 CORGIS

This section covers my major technological contribution: the Collection of Really Great and Interesting dataSets, or CORGIS. The CORGIS project [10, 6] represented a change in focus from high-velocity, distributed data towards high volume, static data. Philosophically, the project seeks to provide a wide range of motivating datasets of appropriate complexity for introductory, and particularly novice, learners. Developing and maintaining these datasets has required a substantial amount of new technology and processes, both to simplify their creation and to keep maintenance better under control.

This section is organized into five subsections. First, I describe opportunities and affordances that I have observed when preparing datasets for the CORGIS project. Next is an overview of the build chain used to prepare CORGIS datasets for different target outputs. Third, I describe each of the ways that datasets are available (e.g., libraries for popular languages, raw datafiles, and interactive web tools). Fourth, I describe the web-based gallery that disseminates all of the CORGIS material. Finally, I conclude this section with an attempt to characterize the CORGIS datasets.

3.2.1 Designing Pedagogical Datasets

In this subsection, I review design opportunities, affordances, and problems derived while building the CORGIS dataset collection. The CORGIS project aspires to have multiple datasets relevant to each major career path that potential students might seek. To achieve this, we draw on data from wide-ranging, open-access sources including governments, research publications, journalists, non-profits, and industry. At the time of writing, this collection includes more than 44 datasets. Before the data is ready, it must be collected and preprocessed into a condition usable by introductory programmers. For example, removing data fields where most of the data was missing. Most of the human effort for the project comes at this phase, as organizing datasets requires expertise, and each dataset raises different issues. Many of the issues are similar to those encountered by expert data scientists. However, other issues are unique to designing datasets specifically for educational purposes.

I have organized and summarized the most salient issues for future developers into 6 categories.

General Advice: Issues to keep in mind throughout the development of datasets, but particularly when beginning work with a new dataset.

Collecting Data: Issues that come up when data is being collected from external sources.

Restructuring Data: Issues related to changing the structure or format of data, without respect to the values themselves.

Manipulating Data: Issues related to the values within the dataset, as opposed to the structure encapsulating the data.

Working with Data Types: Issues related specifically to the various types of data, and their possible schema.

Knowing Data: Issues related to students' and designers' comprehension of the dataset that they are working with.

1. General Advice

The following issues and ideas should be kept in mind throughout the process. Much of this advice holds true for any software engineering project. However, this does not make it less valuable to review, especially since in some cases conventional wisdom may not hold true.

1.1. Have a plan: Datasets are organized for a number of reasons, such as to facilitate manipulation, to conserve space, to facilitate access across certain indexes or specific paths. Typically, the developer has the intended purpose of the dataset in mind, and this can guide the process. Pedagogical datasets can be organized for specific problems or for open-ended exploration. The former, where an instructor has problems and activities already in mind, can be much easier to develop for, since the instructor will have particular characteristics in mind. The latter, where students will explore the dataset and derive their own research questions, is more difficult to design.

When designing open-ended datasets, the developer needs to provide a range of fields and values that can be used to explore many kinds of questions related to the dataset. I recommend a number of higher order strategies to make the students' experience as motivating as possible. First, try to intentionally lay out multiple obvious possible paths: imagine that you were using the dataset, and what kinds of questions you could conceivably want to answer with the data. Second, try to keep the paths diverse, so that students can explore different directions. Third, allow for serendipity – you may not have an immediate use case for some data that you find, but if the quality is high, consider including it anyway. Finally, accept that not every potential use-case will be covered – students may wish you had collected some other kind of data, but you will never be able to predict all possible student preferences.

For example, consider developing a dataset about crime reports in the United States. A primary path of questioning to design the dataset around is to facilitate research into the trend of crimes over time; however, with the right data collected, the dataset could be used to answer more wide-reaching questions related to public policy, enforcement strategies, and population habits. Incorporating the right statistics to allow students to explore all of these issues might make the dataset too big, but choosing a reasonable subset can make the dataset more widely appealing.

1.2. **Build for your audience:** Typically, datasets are intentionally cleaned and organized so that they are as easy-to-use as possible. However, pedagogical datasets can be designed to provide structured difficulties, representing learning opportunities for novices to encounter specific classes of problems. The designer must consider the learning objectives and prior skills of their audience. In general, the goal is to remove all difficulties that are not explicitly related to the intended learning goals. For example, if a dataset incorporates null values for some observations, students need to know how to use conditionals to avoid plotting the null values – if the original lesson was simply about plotting lists of numbers, the learning objectives have suddenly been expanded (similar to the software engineering phenomenon of feature creep [90]). It is important to choose your learners' battles carefully, so that they struggle with the right aspects. It can be tempting to leave a flaw in the dataset and casually wave it off as "a learning experience", but this must be done with the learners' best interests in mind.

1.3. **Iterate:** As with any design project, you should work in an iterative manner. Expect to refine or discard earlier attempts. Decisions that seem reasonable early on may have surprising consequences when the results are brought to a learner. It is valuable to expose early iterations to expected users to gauge reaction.

1.4. Standardize your process: Never treat code for preprocessing a dataset as single-use; plan early to make it a process that can be run repeatedly. Almost no dataset is final – new data is published, errors are found, or possible improvements are discovered. Structure the preprocessing as a pipeline; ideally include code to automate the retrieval, to clean the data, and to generate the completed dataset. At any time, you should be able to regenerate the completed datasets almost entirely automatically.

As with any coding project, include whatever documentation would be needed to explain the code. Document the source of data, both direct links and the pages that contain them (the former for convenience, the latter for redundancy and future-proofing). Keep track of citations and data documentation wherever possible, since information that seems obvious early on can become more obscure with time. Use assertions to verify expectations about the dataset. Make stylistic decisions and stick to them, so that it is easy to pick-up old build pipelines and navigate them.

1.5. Keep a clean workspace: Verson Control Systems like GitHub can be a great asset in managing preprocessing code. However, many of these systems frown on keeping large data files around. Rather than driving up repository size by including all data files, consider incorporating rules into your repository (e.g., through .gitignore systems) that prevent storing data files (e.g., CSV, JSON, TXT, etc.). Then, add in code to automatically retrieve any data files on demand from a remote system and keep that data cached. This cache, along with intermediate and output files, should be removable as part of the build process. A self-hosted mirror of the original data source can provide safe redundancy, while also maintaining the separation of the processing and the data.

1.6. Manage dataset health: Datasets are never perfect: Users will encounter mistakes and errors; Better structure and organization is realized; New data becomes available; Old data becomes obsolete; Better data is found. You need to plan early to track and manage these changes. Using an issue tracker (such as the one on systems like GitHub) to manage problems with the datasets is an excellent way to stay on top of these changes and to provide a record of the design decisions. It can be difficult to remember the justifications for decisions that were made early-on, but issue trackers can be used to find the original reasoning.

1.7. Beware breaking convention: A recurring danger when scaffolding a dataset is that students may inadvertently learn incorrect behavior. They may believe that the instructor has created the dataset with conventional best practices in mind, and they will (possibly subconsciously) adopt these strategies. Instructors should work with interested students to better understand why a dataset was organized the way it was, and other decisions that could have been made. This can be a teachable moment: consider an assignment where students work to improve a dataset for some purpose. Otherwise, the instructor should follow best practices wherever possible, assuming it does not conflict with managing students' struggles.

1.8. Work in phases: Processing large data files can be time consuming, simply by the nature of data at scale. However, as with any coding process, the development process will involve errors and debugging time. If a later step fails, then resuming all the way from the beginning can greatly hamper development as long delays prevent proper debugging. Structure your build pipeline so that phases can be developed in chunks, and individual phases can be debugged independently. For example, by writing out intermediate results to disk, progress can pick up from where it left off.

1.9. Understand the context: Datasets are typically generated for some purpose; studying more about that purpose can greatly aid in the development. Websites often publish complementary data dictionaries and information about datasets that can explain nuances. Sometimes, publications have been written that can better explain the context. Find and reach out to subject-matter experts in order to resolve questions. Ensure that the dataset you end up preparing aligns with the kind of data that professionals might expect to see, at least in nature if not shape or format. Generally, you should do research on the context before you collect and prepare the data.

2. Collecting data

The first step in preparing a dataset is obtaining the dataset. This section details issues and ideas related to finding, mining, generating, or otherwise obtaining datasets.

2.1. Hunting sources: Because data is pervasive across the internet, the problem of obtaining a data source is largely one of filtering and searching. Search engines are an obvious first step in finding suitable datasets. However, knowing the capabilities and available techniques of your search engine can be invaluable. For instance, Google features a number of powerful ways to improve searches, including boolean logic, site-specific searches, and even a filetype search. This last feature has proven particularly useful, since we are typically interested in data files of certain formats. Knowing how to navigate search results effectively can also be helpful – recognizing what sites are particularly promising can help filter datasets more quickly. For instance, repository sites (e.g., GitHub), government sites, and non-profit organizations are often good places to start.

In the past few years, data enthusiasts have collected varied data sources into meta lists. Some meta lists are more useful than others. In my experience, however, many of these lists have considerable overlap, making it difficult to find new sources. Further, their datasets tend to be limited in size and scope, typically narrowed for a specific purpose. This is particularly true for datasets that are meant to teach specific subjects. The UCI Machine Learning Repository [50] features a tremendous archive of Pedagogical Datasets, but the datasets tend to only contain a few limited features targeted for specific machine learning tasks (which can be too arcane or complex for beginners to tackle). Many datasets are created by and therefore tend to appeal to people who naturally find data interesting and useful – therefore, those tend to be about computational subjects that may have little cross-discipline appeal. Of course, there are still many excellent resources for finding datasets in meta-lists: for instance, the DataIsPlural project publishes novel, high-quality datasets twice a month [73], and a number of its datasets have been incorporated into the CORGIS project.

2.2. **Working with file formats:** Data can come in many exotic formats, which can be troublesome to deal with. When collecting data, it is important to be aware of the various formats you might encounter. It is not possible to make an exhaustive list, but specific and general classes of data formats include the following:

- Spreadsheet data, in a row-column format, is one of the most conventional and stereotypical data formats. CSV (Comma-separated Values) and TSV (Tab-separated values) are common examples of this format. These data formats are usually easy to manipulate and view in both text editors and spreadsheet software such as Microsoft Excel and Google Sheets.
- JSON is a hierarchical format that incorporates lists, objects, and four primitive types: integer, floats, strings, and booleans. I chose JSON as the designated format for completed datasets, since it allowed for considerable flexibility, expressiveness, and simplicity. There are variations on the JSON format, such as the JSON-line format, where every line of the file represents a parseable JSON object (removing the need to encapsulate the top-level structure as a list).
- XML data (and HTML data by extension) are document markup formats, not strictly intended for structuring data. Although popular in many industry settings, for most purposes, XML is a less efficient file format than JSON [59], in terms of space, transmission rate, and complexity. In my experience, processing datasets in XML is cumbersome and tedious compared to other data formats.
- Many popular statistics packages have their own proprietary formats: STATA and SAS, for instance, both have their own file types. These formats require specialized software, and can be inconvenient if the developer is not familiar with them. Typically, I tried to avoid these types if possible.
- PDF is a popular format for transporting documents reliably, and is sometimes used to publish data. Data stored in the PDF format is a tragedy, frozen in time and locked away from convenient access. Although PDF can be a useful format for conveying information to be reviewed by humans, it is highly problematic for computational purposes. Tools exist for parsing PDFs and extracting out their data, but this is extremely difficult, and usually not worth it. Typically, even the best tools introduce garbage data, fail to understand formatting and table structures, and require human effort to correct mistakes.

2.3. Scraping web data: Not every website that collects data makes it easily available. In some cases, *scraping websites* (visiting pages using a script and downloading their contents) can be a great way to make your own dataset. In general, scraping can be considered abusive behavior, and some websites respond aggressively: limiting user access, banning IP addresses, and other defensive actions can greatly hamper your ability to access the site. It is critical to make sure that you are not violating the Terms of Service of the website you are scraping. You should also check carefully that the website does not have their data available in some other, more convenient way. You might even consider requesting the data directly, if contact information is available on the site. If scraping is the only option, then do so politely: use non-peak hours, spread out your scraping to reduce server load, and cache intelligently to reduce redundant server trips.

2.4. Mining real-time data: Mining a real-time data source is an easy way to translate a high-velocity, low-volume dataset into a low-velocity, high-volume dataset. The idea is to take a data source that updates regularly and to retrieve data from it on a consistent schedule, aggregating the data over time. This can be particularly useful for services that aggressively limit users' access rates. Mining requires time, planning, and coordination. It is a good idea to keep tabs on your data collection, in case the API service changes. Logs can help identify problems, especially if notification systems (e.g., email alerts) are used.

2.5. Legality of your data: Some datasets make it obvious whether you are allowed to republish and use data, but sometimes it can be ambiguous. Keep the legality of your data in mind, and investigate the allowed uses before collecting the data. Typically, fair use policies are generous towards using data for educational purposes. When in doubt, reach out and request permission to use the data.

Remember that some datasets might need release forms from every individual student that intends to use it. The ICPSR (Inter-university Consortium for Political and Social Research) [1] is a high-quality collection of datasets from the humanities and social sciences. Unfortunately, the vast majority of its data requires users to individually obtain release forms, since much of the data involves human subjects. In our experience, we have avoided working with ICPSR and other services that have these requirements, since it adds additional complexity disproportionate to our learning objectives.

2.6. Synthesizing datasets: Many datasets can be improved and expanded by incorporating additional data from other datasets. These kind of conjoined datasets are popularly known as "Mash-ups". It can be tricky to find data with enough common indexes to perform the join operation. An example of a mashup would be to take state/year level data (such as for scholastic abilities in US states, or crime statistics) and to integrate information about population, income, or some other kind of metric. This sort of process is most appropriate with script support to recreate the mashup.

[1] `http://www.icpsr.umich.edu/icpsrweb/`

3. Restructuring data

Once data has been collected, it must be structured in a way that aligns with the projects the instructor has in mind. The following entries describe common operations performed when manipulating data. In general, we recommend using a scripting language (e.g,. Python, Perl) that you are comfortable with in order to do any data processing. Many tools and guides already exist to aid in the process of manipulating data, so I consider reviewing low-level language details to be outside of the scope of this dissertation. Instead, I have tried to only include advice at a high level.

3.1. Choose your target structure: For the CORGIS collection, every dataset is eventually represented as a list, where each element is a dictionary of either primitive fields or further dictionaries. This format was chosen since it allowed convenient filtering and sampling of elements while still providing opportunities to practice complex data traversal (both of heterogenous and homogenous data). There are many options available for the structure of data, however. In some cases, you may wish to allow the data to dictate its own structure, based on inherent characteristics. In other cases, you may want to plan a structure that will be easy to access using the students' intended learning environment. If preparing multiple datasets, maintain consistent structure across datasets, so that instructors can switch between datasets more quickly and easily.

3.1. Layering columnar data: Restructuring datasets into list-of-dictionary form sometimes begins with spreadsheet data. The row/column format of spreadsheets is similar to the concept of a list-of-dictionaries, with the exception that dictionaries can contain other dictionaries whereas row/column typically holds singular primitive values. When working with spreadsheets that are particularly wide (i.e., have a lot of columns), chunking columns can be an excellent way to help users navigate the data's structure. These chunks of columns can represent sub-abstractions that group related fields. For example, an address can be grouped with latitude and longitude fields under a "location" field.

How big should a chunk be? A classic text in psychology says that the human memory is best at remembering 7 things in a chunk, plus or minus 2 [55]. However, later research argues that the earlier estimate of 7 was based solely on remembering numeric sequences, and that other kinds of data have different optimal chunk sizes [52]. For textual data (in this case, the names of the chunks which are keys of the dictionary), experts suggest smaller chunk sizes (4 has been suggested [20]), but different experts draw on different factors to suggest a number, including the size of the words, how well-known the words are, and whether schemas can connect the words together. Ultimately, we aimed for chunks 4-5 keys wide, particularly at the top of our dictionary hierarchy, making allowances for the factors established above.

3.2. Converting XML to JSON: As previously discussed, XML can be a cumbersome format, since it is meant for documents instead of data. This may be surprising, since technically XML can represent the same kinds of data as JSON (both are hierarchical data formats). There are many reasons why this is not as simple in practice, however. The syntax for XML requires more characters and rules than for JSON, for instance. Tools exist that can create a direct translation from XML into JSON - however, the results can be surprisingly garbled. One of the major reasons comes from XML's ability to annotate a node with text data, child nodes, and attributes. The comparable structure in JSON (dictionaries) expect to have either a single primitive value, a list, or a dictionary. Mapping between these two structures often requires some level of human oversight.

3.3. **Working with indexes:** Datasets often have *natural keys* – attributes or sets of attributes that uniquely identify each instance in the dataset. Natural keys, unlike surrogate keys, have a natural existence within the dataset, and are not artificially generated for the dataset. The State Crime dataset, for instance, has two attributes that, together, form a natural key: state name and report year. In the CORGIS project, we always attempt to identify the attributes of natural keys as Indexes, as they are often useful for creating problems and mechanisms around the dataset. When sorting the data before release, we use the natural keys as sorting criteria.

Typically, we try to limit our datasets to have 2 indexes when building our natural key. When a dataset with two indexes is filtered on one of the indexes ("fixed), the remaining rows of the dataset are usually immediately useful for some purpose. Consider filtering the crime data by the state of Virginia – since the remaining index is report year, the remaining reports will represent all of the data for Virginia over time, requiring little effort to plot as a line graph.

Besides the advantage of 2-indexes simplifying the filtering of datasets, I have found that introductory students struggle with the implications of datasets that have more than 2 indexes. Earlier versions of our cancer dataset incorporated 5 distinct indexes (gender, race, state, year, and location of cancer on the body), which was often overwhelming for students. Students seem to find it difficult to consider what happens when an index is fixed, that the data that remains will be over the other possible indexes. For the cancer dataset, if the state is fixed to be Virginia, it is still unsuitable to plot the remaining data as a line plot – all of the non-year indexes must be given fixed values.

3.4. **Collapsing fields:** If you find your dataset is too big, you can consider collapsing on a field (also known as pivoting a table). That is, group each instance in the dataset by the different values of the collapsing field, and then use an aggregation function (sum, average, count, min, max, etc.) to flatten the groups into single values. This drops information, but does so in a consistent way and retains some of the original information. For example, the Airlines dataset (which gives information about flights at airports over the past decade) was several standard deviations larger than our other datasets, so we decided to collapse its Carrier field (which indicated which company controlled the flight). This substantially reduced the size of the dataset, at the cost of some fidelity of information.

There are often specialized ways to perform the grouping operation. For instance, increasing the time scale of data (days→months, months→years) or generalizing geographic region (counties→states, cities→countries) can reduce the dataset dramatically. There are few general principles here, but instead careful application of domain knowledge is required.

A field might be promoted into an index by grouping values carefully, in a process known as "binning". Consider the Cars dataset, which has information about many cars over multiple years. Each year, there is usually more than one car present, and the year is not an index for the data. A possible transformation of this dataset would be to bin the cars for each year, presenting averaged data, which would make the year a suitable index for the data. Although for the Car dataset this would make the dataset prohibitively small, there are other cases where this might be preferable.

3.5. **Stacking data:** In the opposite operation of collapsing data, *stacking data* means to extract out a series into its own column. This process removes embedded lists from inside a dataset. The CORGIS project avoids having nested lists (to reduce user complexity), so this technique was used quite often when designing datasets. Every time you stack data, however, you naturally introduce a new index, so consider the ramifications carefully. This can also be used to increase the number of records your dataset has, and by extension its size. Data that is fully stacked is often known as "Tidy Data" [88], representing how this form of structure can be particularly easy to work with (compared to nested lists).

3.6. **Redundant total field:** A common occurrence in datasets is a set of a keys within a single dictionary that perfectly partition a whole. In the Airlines dataset, for example, the time that flights were delayed given in a dictionary as 5 categories, one for each of the possible causes of a delay. Summed, these fields represent the total time that flights were delayed. Although students could be tasked with calculating this number, they would be required to perform either an iteration over a heterogeneous collection (which might not be in the original learning objectives, the way a homogenous collection iteration would) or to write code that individually adds each of the keys. The latter is particularly bad for datasets with tremendously large numbers of keys, such as the Building Construction dataset (which has over 20 categories of construction types, represented as dictionary fields). A common solution to avoid this is to provide an extra field that represents the total of the other fields. However, this total field hampers the student from doing an iteration over the keys, since the result will be twice what it would otherwise be. Caution and careful consideration of learning objectives should guide the decision to include summation, or other aggregation, fields for convenience.

4. Manipulating the data

Beyond working with the structure of the data, the developer must also consider the contents of individual fields. This includes both the value and the key associated with that value. In this section, I detail a number of issues related to the massaging of the data before it is ready for use. Discussion about the Types used in creating data is largely reserved for the following section.

4.1. Standardize fields: Be as consistent as possible across fields names, types, and their values. Ensure that every field name has the same style of capitalization, spelling, use of symbols, and punctuation, and make sure there are no errors in any of the above. Make sure that every instance of a field has the same type and uses the same kinds of units. If any of the data is nullified in some way, ensure that that nullification happens in a reasonable, consistent way. I strongly recommend using a consistent structure across objects, unless you explicitly want to teach students about how to deal with inconsistent structures. Automated tools should be used to supplement the process and verify that these conditions hold. Although using such tools to check the structure of the datasets is obvious, they can also be used to check other conditions, such as correct spelling. Develop a tangible list of standards and conventions that can be referenced during development. This is useful not only for large teams, but for maintaining consistency across multi-dataset projects.

4.2. Names are important: When developing fields for your dataset, meaningful names are critical. The field name is students' first impression of the data, and an important time to convey as much information as you can. Nobody reads documentation [69], and students are not exceptions – in fact, they are probably even bigger offenders of the rule. Just like designing user interfaces, documentation is not a substitute for bad design. When naming things, consider both fields and values: although the field is obvious, sometimes you will want to recode values to be more clear. For example, if a string value was given as an abbreviation, you might use the fully expanded phrase.

Names should be as long as necessary to convey all the needed information, but not longer. Names that are too long will force students to do more typing, make it more likely for typos to occur, and potentially increase frustration unnecessarily. Be careful when using abbreviations, especially if they are not common domain-specific ones – be sure to provide documentation if you must use an abbreviation. If possible, you can also use names to hint at units (e.g., instead of a "Time" field, you could have an "Hours" field).

Be careful with choices in language. It can be easy to introduce offensive or politically correct terminology, especially as a layman. When developing the Immigration dataset, which has information about legal and illegal immigrants entering the United States, the term "Alien" was used in some places to describe illegal immigrants; after a user raised concerns about the vocabulary, we chose to find more accurate and less offensive terminology. Proper research on the source context can help dramatically with navigating these kinds of issues, but in general it helps to simply be open to the possibility and to be responsive to user issues.

The mental errors that students make will not always be be obvious, although students do seem to be predisposed to find certain kinds of patterns in the data. The Publishers dataset features information about the top 10,000 books sold on Amazon, where each row represents a single book at a specific point in time. A number of properties for each book is recorded, including the daily number of books sold on Amazon. This field was named "Daily Books Sold", and encoded as an integer. Almost every student that interacted with the dataset developed the misconception that, because the word "Daily" was in the name, the data was indexed by time. Although the documentation was quite explicit that the dataset was a snapshot of a collection of books at a point in time, students would attempt to plot the Daily Average; since the dataset was sorted by the book's rank in terms of sales, this would invariable lead to a relatively smooth downward curve. The conclusions and extrapolations that students made from this data are irrelevant, because they were fundamentally led astray by the misleading name.

4.3. **Working with bad data:** Datasets are rarely perfect. In fact, more often than not they have errors in their data: typos, dropped fields, encoding errors, or simply non-existent data. As many of these imperfections should be *cleaned* as possible before they are given to students. Many different strategies abound, depending on the nature of the malformed data.

When data are missing, one option is to substitute a null value. Null data can be complex for students to handle: it can require more conditional checks, breaks some standard tools and operations, and needs more critical thinking to reason about. Different formats and languages have different ideas of non-existence. The JSON format provides a "null" type, which is often present in programming languages ("`None`" in Python, "`null`" in Java). Some languages support multiple kinds of non-existence, such as "`undefined`", "`null`", and "`NaN`" (Not a Number) in JavaScript. If these forms of non-existence are not available, you might replace the missing data with 0, 99999, or -1 or some other "Bad data" value. The added value is that the fields will be of the same type – the downside is that it becomes considerably less obvious that the data was incomplete, and learners might not realize what has happened. Learning objectives should be considered carefully to determine whether students should need to confront bad data, or whether it is suitable to ignore the problems for pedagogical purposes.

The process of substituting values for non-existent data is called *imputation*, and is a more sophisticated technique than using nullified values. Imputation can be performed in several different ways, each of which has trade-offs. The "last-observation-carried-forward" rule simply carries over the value from an adjacent instance in the natural ordering. Mean substitution replaces the missing value with an average from the entire dataset, or from a subset of instances that share a similar key. Although sometimes effective, mean substitution can become problematic when the fields being interpolated are independent from the indexes being considered. For example, in the Earthquake dataset, there is no relationship between the “Gap” metric and the datasets’ natural keys; if missing “Gap” data was being interpolated using mean substitution, the values would be worse than guesses, since it would introduce a non-existent relationship. The final form of imputation, and the most sophisticated, is to use regression to fill in missing values. Stochastic variations of regression can be used to incorporate noise into the dataset, increasing its feeling of authenticity. Although typically more accurate than other forms of imputation, regression suffers from the inverse problem that it can introduce new alternative realities: patterns become reinforced, even if they didn’t really exist. The Global Development dataset’s source material (the World Bank) uses linear interpolation to fill in years worth of data, leading to smooth gentle curves that belied the true nature of population data, and in some cases whitewashes tragedies and diseases in foreign countries. It is crucial to document the use of imputation for students, and communicate with the learners about the nature of the data.

Listwise deletion is a simple and direct method for handling bad data – deleting any instances that have bad data. This can have the side benefit of reducing the size and complexity of the dataset, if it is too large already. Consider carefully whether the data being removed are random or represent a specific case of interest. If possible, retain any important statistical relationships with a proper sampling algorithm that gets a random distribution, or intentionally picks values that maintain the relationship. Removing outliers to smooth data can lead to odd results, and needs to be approached with caution. In the CORGIS Immigration dataset, outliers were naively (and briefly) trimmed, leading to a complete halt of Mexico’s immigration to the United States. Deletion has other variations beyond deleting just the instance: *pairwise deletion* removes all instances that share index values, for instance, and there is always the option of just removing a field across the dataset.

Removing or changing the data can be useful for avoiding complex situations. For example, earthquake magnitudes can be negative - but explaining this to students is an extra headache that may be more distraction than helpful as a context. Therefore, the Earthquake data filters out earthquakes smaller than magnitude 1. Although the documentation details this fact, it is likely that students would never realize this – which can be dangerous if they are relying on the dataset to be completely honest!

4.4. **Cleaning up by hand:** Sometimes, automated tools are insufficient for cleaning up a dataset. When developing the Medal of Honor dataset, for instance, award dates and locations were often mangled and incomplete, which became a problem when geocoding was applied to the dataset. The strategy to overcome this limitation was to automate as many broad cases as possible, and to simplify the process of handling exceptions as much as possible. This process needs to be formalized and repeatable; consider that it might need to be run multiple times, that it might need to be interrupted (whether because of a developer error or some other issue), or that it might need to be reviewed. One suitable approach is to have a step in the preprocessing pipeline write the data out to a spreadsheet or other convenient data format, for human annotation. This file can be updated in-place, without deleting existing annotations, and then read back in during a later step to fix the most exceptional cases. Mechanical Turks and undergraduate assistants can be powerful tools for fixing data by hand.

4.5. **Reshaping data:** Visualizing data that is heavily skewed or has an alternative distribution can be difficult. For example, when creating histograms of data that is log-normal, the graph may appear to be completely skewed in one direction; after applying a logarithm function, however, the data becomes normally distributed. Data scientists will often transform data using other common mathematical functions, such as a Square-Root, in order to reshape the data into a more easily visualized distribution. For data that is already normal, it is also common to apply a standardization operation, which uses the mean and standard deviation to recenter the data around zero. Rather than expecting students to apply this operation themselves, a field can be preprocessed using these operations. Doing so requires the developer to explain the field further, both through its name and in the formal documentation; documenting and explaining these kinds of transformations can be challenging, however.

4.6. **Extending a dataset with divined data:** Sometimes, a dataset can be extended without introducing any external data. The method leverages metadata and extrapolate information already contained within the dataset. A relatively simple case is to add a rank field to the dataset, if there is a natural ordering of instances. However, more sophisticated metrics can be added too. The Classics Dataset originally incorporated large texts of classic, public-domain books; however, it featured few numeric metrics on the books. A number of textual analysis techniques, including sentiment analysis, word count, and text difficulty estimations, were applied to expand the dataset.

5. Working with Data Types

Types are an important topic in computing and when creating datasets. The developer must be careful about choosing what types to incorporate into their database. What types do you want your students to be familiar with? Most datasets will incorporate at least some basic numeric types (integer, decimal) and text types. It is also common to have boolean types. Some datasets benefit from having enumerated types. However, enumerated types exist as metadata, not necessarily as an inherent part of the structure. There are also the "Non-existence" types (`null`, `None`, `undefined`, `NaN`, etc.) that require further explanation.

The CORGIS collection limits its data types to integers, decimals, booleans, and string types (in addition to lists and dictionaries). However, even within this scope, there are more complex schemas [36] and nuances to be captured. Specialization types, such as currencies, dates, and addresses, can be represented in many different ways, with trade-offs for each. In this section, I describe concerns and opportunities provided by these types and schemas.

5.1. Numbers: In some ways, numbers are the quintessential type of data. Most formats and environments separate between integer and decimal (floating point) types, but there are other variations: some languages have special types dedicated to massive numbers (`long` in Java and Python), tiny numbers (`char` in Java), and decimal numbers with more or less precision (`double`, `single`, in Java).

The CORGIS project generates datasets ready for multiple languages and environments, which requires a lowest-common denominator approach. Therefore, the only two numeric types are float and integer. We have a mild recommendation to prefer integer numbers over floating point numbers, since decimal numbers can end up sacrificing some level of accuracy in certain environments. Further, it is advisable to keep the size of numbers manageable – it can be mentally difficult to keep track of numbers greater than 2^{64}, and some languages coerce these numbers to a special numeric type (as in Pyton 2.7) or, even worse, simply cut off the unused data (as in the formal JavaScript spec).

Percentages are common in many datasets. These are typically in the range of 0% to 100%, but not always. A small decision is required when encoding percentages: do you multiply it by 100, or leave it as a decimal number? That is, if you were representing a half, would it be 50 (50%) or .5? Whichever option you choose, you should be consistent and make it clear in the documentation. Given that either format can lead to confusion (is that 5000%? .5%?), there is no clear solution. In the CORGIS project, we use whole numbers to represent percentages.

5.2. Textual: Textual data, often referred to as `string` data, comes with a host of headaches. Any data can be encoded as a string. However, students must then struggle with type conversions and string parsing, which may or may not be relevant to the intended learning objectives. Review data carefully to make sure that no numeric fields have been accidentally encoded as strings – while this sounds like a silly mistake, we find it happens in almost every dataset at least once. For example, persistent bug reports with the Cars Dataset led to the discovery that several fields were accidentally encoded as strings for certain observations, because the units were included in certain strings (i.e. `‘‘15kg’’` instead of the integer `15`).

Text data opens up a host of lessons and opportunities to teach students about string parsing, manipulation, and other common operations. However, in more mathematically driven learning contexts (e.g., Data Science) where these skills might be considered distractions, it can be difficult to find a purpose for text data. Analyzing the text itself for metrics can provide numbers, assuming the text is of a sufficient quality and interest (e.g., a note-worthy book, forum posts with associated metadata). These metrics can be applied at multiple levels: individual characters, words, sentences, paragraphs, chapters, equally-sized chunks, or even on complete text blobs. The following is a brief description of computational textual analysis techniques that have been explored or applied in the CORGIS project. This list is biased towards techniques that work particularly well for English-language text. Not all of these metrics lead to particularly useful results.

Sentiment Analysis: The positivity or negativity of a text, calculated by the frequency of certain words that have previously been tagged as positive or negative. Many popular packages exist off-the-shelf to perform this kind of analysis.

Word and Sentence Statistics: Simple measures of the way that the text is written, such as number of characters, number of words, number of sentences, average length of word, average length of sentence, etc.

Word Difficulty: An estimate of the difficulty of all the words in a text. Many formula and packages exist to calculate this, but the two general approaches rely on either the number of syllables in the text or using an index of word difficulty.

Word Frequency: Word frequency analysis can be used to isolate the patterns of use of certain words, assuming a list of suitable words to mine can be created.

Word Etymology: As shown in [78], the etymological source of words (what language they were imported from, such as German, French, Latin, etc.) can be estimated using language source dictionaries.

Grammatical Mood Analysis: The English language expresses uncertainty through certain kinds of grammatical constructions. The frequency of these constructions can be used to classify text as “imperative”, “indicative”, “subjunctive”, or “conditional” [91].

Vocabulary Richness: The diversity of words in a text, also known as Yules I [79], which the original creator argues is a measure of complexity.

Tonal Analysis: Advances in deep learning have led to tools such as IBM Watson, which features an application to analyze a text for emotional content (anger, sadness, joy, disgust, fear), language style (analytical, confident, tentative), and social tendencies (openness, conscientiousness, extraversion, agreeableness, emotional range) [2].

When preparing a text field, check strings carefully to make sure that special characters are correctly escaped and decoded, according to the expected use of the field. Data stored in Unicode encodings presents a particular challenge. Not every language, format, and platform has equivalent support for Unicode. Further, proper handling of Unicode is tricky without proper understanding of the mechanisms involved in bit-level string representation. In some cases, it may be simplest to simply ignore the problem of Unicode data by mapping characters to their nearest ASCII equivalents. Although

[2] `https://tone-analyzer-demo.mybluemix.net/`

this is the approach taken by the CORGIS project, it is most likely unsatisfactory for most serious text datasets. I recommend reviewing more substantive literature for any project interested in working with Unicode data correctly.

5.3. Dates and times: Date and Time data is common in most datasets, since it opens up a wide range of interesting questions and problems. However, encoding and structuring such data is a difficult problem. In some cases it is useful to provide multiple, alternative representations of the value, but this increases file size and the total cognitive load needed to traverse the dataset. Although documentation can supplement the process, we have previously noted that documentation is not well-used by novices or experts.

Including a simple text representation of a a date/time can make it easier to immediately comprehend the value. If this text representation is structured correctly, it can also simplify sorting datasets on time. For instance, writing out a date in the format of "Year/Month/Day", using zeros to pad each number to a fixed width, makes the dataset easily sortable. Unfortunately, these formats may impact the readability of the value, since the zeros contribute noise and the format is not necessarily one most people are familiar with. Whatever format is used, it is important to document the schema as clearly as possible, since users may end up having to parse it.

A more sophisticated approach is to encode each component of the recorded date and/or time as a separate field in its own dictionary: a field for the year, a field for the month, etc. Even this introduces questions: should the month be encoded as a number, its full name ("January"), its abbreviation ("Jan"), or some combination of the above? A simpler approach, with its own caveats, is Epoch time, or the number of seconds after the Unix Epoch (Thursday, 1 January 1970, UTC). Epoch Time will often be unfamiliar to students, requiring an explanation and increasing the complexity of the activity. Epochs are particularly large numbers, which can introduce its own form of complexity – 32-bit signed integers can only represent Epochs from 1901 to 2038, which may be problem for datasets with particularly long prospects. Although Epochs seem like a simple solution to a difficult problem, we tend to avoid using them in the CORGIS datasets for the reasons above, instead favoring a combination of individual fields and a full string representation.

Ultimately, handling all the intricacies of dates and time is impossible, and most programmers do not have any idea of the scope of the difficulties involved [77]. As with any difficult situation, consider what you need your learners to be able to do and what they need to learn – choose a structure and format that is most suitable for this purpose. I recommend just doing your best and hoping that it works out.

5.4. Measurements: Numbers rarely come without context, but programming languages rarely include units in their types. Numbers are used to store currencies, temperatures, distances, volume, and many other measurements. Are the units in dollars or euros? Centigrade, Kelvin, or Fahrenheight? Documentation is usually required to explain what a number means with respect to its field. Clever naming can be used to reduce the complexity – instead of a field named "Time Delayed", for instance, consider the name "Minutes Delayed", which is no more complicated but benefits from immediately explaining the units.

When designing for a specific audience, it can be helpful to use a measurement format they are already comfortable with. Inversely, putting data into an awkward format can provide an opportunity to have students practice manipulating data. In other situations, the context may demand or benefit from a certain format (e.g., using metric for scientific data, or sortable dates). Sometimes, converting data may result in unacceptable loss of precision, tainting the authenticity of the dataset. Floating point representations can be imprecise: doing conversations and then rounding can lead to less accurate results, for example.

5.5. Locations: A common form of data is locations around the world. These locations can be present at varying levels: specific points, addresses, cities, counties, states, countries, continents. Each of the varying levels provides different problems and opportunities.

Many of the CORGIS datasets feature data from states in the USA. To conserve space, it may be tempting to use the two-letter state abbreviation ("VA" instead of "Virginia"). I strongly recommend against this approach, however, because of the ambiguities that can emerge when the location type expands, not to mention the increased cognitive load of converting from a two-letter abbreviation to the full state name. In fact, simply naming the field "State" often becomes a misnomer, tempting as it is. For instance, in many datasets, additional regions are also reported alongside states: the District of Columbia is not a state, but its data is often included. Puerto Rico and other United States territories are not states, but often contribute data to United States-oriented datasets. Their abbreviations will be unexpected to students. For some historical data, this situation becomes even worse: the Kingdom of Alaska and the Alaska Territory only formally joined the United States in 1959 (a scant 58 years before this document was published!).

The United States census breaks the country into 4 major regions with 2-3 divisions each. Other governmental bodies use other partitions, such as the Office of Management and Budget's 10 federal regions. Although these regions are often compelling divisions for data, students may not be familiar with them, requiring them to consult documentation. If the data is included alongside state and national level data, it can exacerbate the previously described redundant summing problem: with three distinct partitions of the data, summing all the instances will result in triple the expected total.

Global data often incorporates information about different countries. Unfortunately, the names, geographic sizes, and existence of countries varies widely over time. Keeping track of changes and aligning data time-wise in a meaningfully manner is exceptionally difficult. Be aware of political and cultural land-mines when choosing names, and what locations to include: some countries may refuse to recognize each other. I strongly recommend referring to experts and the literature to provide arguments for and against including or excluding countries – it is beyond the scope of this dissertation to make any arguments in any direction.

Encoding postal addresses raises similar concerns to representing dates and times. Should an address be broken into individual fields, encoded as a single string, or are multiple representations appropriate? Before answering this question, it is usually also useful to ask: what parts are actually important? Cities, states, countries, and zip codes are typically usable within geographic plotting systems, and are often sufficient on their own. Addresses alone are usually not particularly useful to students, unless they have access to a convenient geocoding system which can give them latitude and longitude coordinates. Geotagging services can be expensive, slow, and rate-limited; they are best done as batch operations. The CORGIS project geocodes during the preprocessing phase, rather than expecting users of the datasets to do so.

5.7. URLs: Similar to postal addresses, some datasets make URL data available. These URLs can connect students to further documentation, data, or other resources associated with an instance of a dataset. One of the more common use cases, for instance, is linking to an image stored on a remote server. This keeps the dataset's file size down while still allowing the user access to richer data. When using URLs, be sure to include the whole URL schema for convenience and clarity's sake (including the "http://" or other relevant protocol).

Distributed data that is not specifically under your control may change at any time, without notification. This change could be the simple disappearance of the data, but it could also be something more malicious (e.g., malicious code, offensive material). Consider mirroring the original linked data so that you can maintain control over the students' experience. Unfortunately, running your own mirror introduces a whole new collection of problems. In fact, many of the technical problems outlined in the RealTimeWeb project (Section 3.1) apply directly to relying on linked web data. The challenges of working with distributed data are so great that you should carefully consider whether students will actually need to take advantage of the linked data, or whether you are simply padding the dataset unnecessarily.

5.8. Enumerated data: Typically, a fields' value falls into a narrow range of options, which can be represented as an Enumerated Type. Many datasets use a numeric code to reduce the space that the data takes up. This requires external documentation to figure out (e.g., using a lookup table or a data dictionary). I strongly recommend against leaving a field as an encoded numeric value – instead, replace the value with a more meaningful textual value (keeping the full data dictionary information in the documentation). Although redundant, the benefit to students' cognitive load is substantial.

6. Knowing the data

Eventually, a created dataset will be encountered by learners. These novices must quickly learn the structure and meaning of the fields, in order to use the dataset for the intended learning experience. This section highlights our experience with the problems and opportunities that occur when students encounter pedagogical datasets.

6.1. Nobody reads the documentation: As previously stated, both students and experts skip the documentation [69]. When they do use it, it tends to be for specific, targeted tasks. However, this does not make the documentation unimportant. Make sure that the documentation is always readily available, easy to access, and able to answer whatever questions they have. The fewer the barriers, and the better the documentation is at answering questions, the more students will learn to use it and build confidence in documentation as a concept. One method for improving documentation is to log what questions students typically ask about a dataset, and using that data to improve the existing documentation.

6.2. Learning the structure: One of the earliest steps in learning to use a dataset is to learn its structure. Students can sometimes struggle to understand the structure of data, especially hierarchical datasets like the ones in the CORGIS project. For instance, we note many students who believe that the top level of our datasets are dictionaries, not lists, despite repeated instruction. This is a symptom of a bigger phenomenon: learners focus on the interesting part of a data and gloss over the finer details. Care and attention must be given to feedback for the students, and provide them opportunities to demonstrate their understanding (or misconceptions) of the nature of the data. This can be structured as an explicit activity where students map out the data's structure as a diagram, such as the one shown in figure 3.11.

6.3. Learning the distribution: As the students learn the structure of the data, they must also gain an understanding of the nature of the data. When they encounter a numeric value in an instance, is it a high value or a low one? Was it unlikely to have that value? What is the greater meaning of the number? These questions of distribution, scale, and characteristics of the data might be questions that students are explicitly assigned to answer, but they can also be distracting side-effects of using data as a context. You might consider providing tools and supplementary information to document the nature of the data. Automatic analysis can reveal suggestions about data, such as median values, the range of data, and to some extent what kind of mathematical distribution is represented. You might consider using these statistical tools yourself to judge the nature of your data, so you can help guide your students to data appropriate to their tasks.

6.4. Disseminating materials: Ultimately, a pedagogical dataset must be delivered to students. The CORGIS project releases datasets and their associated files through a central website, as described in Section 3.2.4. We strongly recommend that the dissemination problem be solved early in the development process. How big is the dataset, and will it fit on your server? If the datasets will be delivered through email, do the systems involved have an issue with file sizes or types? If the datasets will be delivered through a third-party service, do they provide a convenient download interface? Will the dataset fit on students' development machines, and will they be capable of loading the dataset into memory? Mechanical issues hamper the learning process and distract students from more important learning objectives.

We note a particular form of confusion that we have seen students struggle with, no matter how we phrase instructions. When working with the CORGIS datasets, some students will download the database file and possibly the associated code file, attempt to open the database file, and then get confused. These files are typically meant to be processed with a programming language or other specialized software, but students are used to being able to download and open files directly. In the CORGIS project, besides striving for clarity in our instructions, we often include comments in the file to make it clear to students that they should not be opening the data file directly. This at least reduces the incidences of confusion.

6.5. Monitor usage: Pay attention to how students interact, struggle, and comprehend datasets. There is no substitute in design for user interaction. No matter how frustrating students' mistakes can be, we recommend adopting the principle from user interface design: Never blame the user. If it does not contribute to their learning, be prepared to treat a problem with a dataset as a mistake in its design. To facilitate this process, consider adding tracking systems to the dissemination platform, both for the datasets themselves and for the documentation. If possible, provide a specific place for users to report problems and confusion, either on the site or in the learning environment (e.g., a post-activity survey).

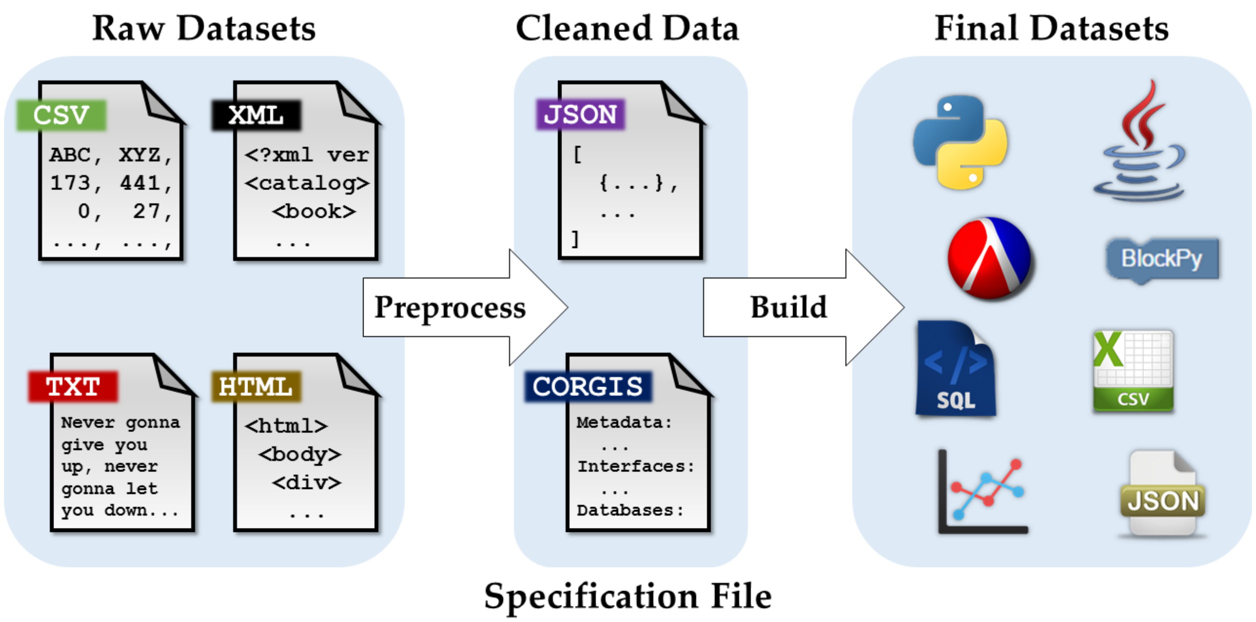
3.2.2 Builder

After data has been preprocessed into a "cleaned" state, it is ready to be processed into various output formats. The CORGIS Builder is a semi-automated tool to process the data. This system parses the specification file (the .CORGIS file) and the dataset file(s) and compiles them into the various output targets.

High-level specification files (with the extension CORGIS) are written in YAML to describe the metadata, fields, and interfaces for a cleaned dataset. Appendix A.1 gives the full format of a specification file. Briefly, however, it is divided into four sections:

Metadata: Information about the dataset, such as its name and version. This also includes a description of the dataset and its provenance.

Figure 3.2: How Datasets Are Added to CORGIS



Locals: A list of the actual data files available for this dataset. Historically, this was a list because multiple data files could be associated with a single dataset. In practice, none of the datasets needed this feature, so this is typically only one data file.

Indexes can be specified for each data file. A dataset is not required to have an index, but in practice we find that most datasets have at least one and more often two. These indexes represent particularly important fields in the dataset, and can be used to conveniently filter and sort the data.

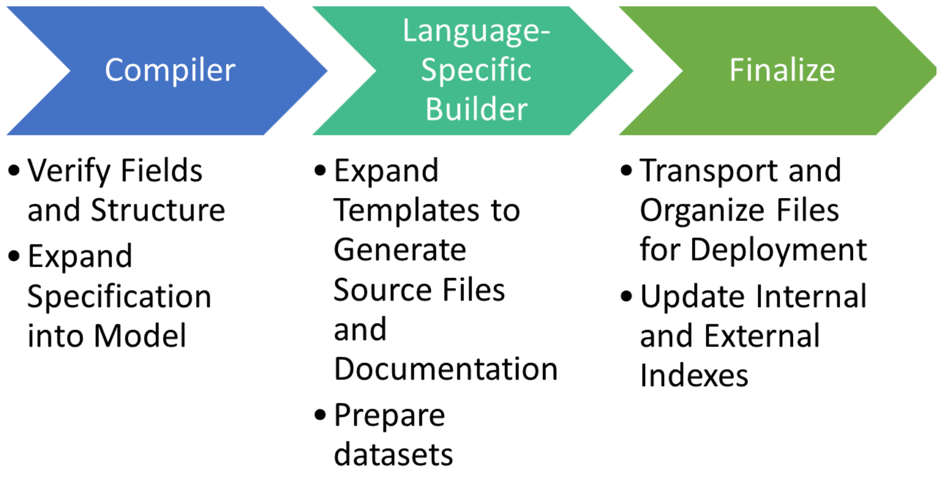
Interfaces: A list of the ways that this dataset can be accessed. Most datasets will provide at least one interface that selects all the data. However, datasets can also provide interfaces that filter and select the data.

Interfaces are specified using SQL query strings. Since the client libraries all natively support SQLite, SQL is a convenient mechanism for accessing the data uniformly across libraries. In theory, interfaces can be accessed using any conventional SQL operations, including both filtering and grouping.

Comments: This optionally specifies comments for the dataset, either through a hierarchical representation of fields in the dataset or through JSON Paths.

Figure 3.3 is a diagram of the CORGIS Build process. First, the CORGIS file is loaded into memory and its schema is verified. The compiler is aware whether each field is required,

Figure 3.3: The CORGIS Build Process


Compiler
Language-Specific Builder
Finalize
• Verify Fields and Structure
• Expand Specification into Model
• Expand Templates to Generate Source Files and Documentation
• Prepare datasets
• Transport and Organize Files for Deployment
• Update Internal and External Indexes

recommended, or optional. Types for each field and the schema's structure is also enforced. Warnings and errors are reported at the individual line level.

After verification is finished, the compiler will generate a complete model of the specification file, filling in defaults and expanding any fields to simplify its schema. During this phase, the data files' structure is traversed recursively. This function verifies the structural integrity of the data file and develops a model of its organization. Comments stored in specification file are mapped to the structure of the data.

The completed model is then ready to be passed along to the language-specific builders. Each output target has its own specific builder that consumes the data file and model in order to generate the required number of files. The CORGIS project uses the Jinja2 templating language, a popular templating system for Python [3], to create source files in the various output languages. Often, the data files will also be manipulated according to some rules. Finally, additional documentation and supporting files (e.g., icons) are generated.

Figure 3.4 shows a Jinja2 template file for a language before it is expanded into a finalized library source code file. Although Jinja2 was originally created to create web files, the system is flexible enough to generate any kind of code. The templating language has support for conditionals, loops, variable assignment, and function declaration and calls. In fact, the language is almost a full implementation of Python.

In the final phase, generated files are transported into the build folder and the indexes are updated. JSON-formatted files are used as indexes to track the existence and build version of each dataset; these files provide programmatic access to the CORGIS collection by external systems and the Builder itself. HTML files are generated as indexes to facilitate public traversal of the CORGIS collection; this is discussed further in Section 3.2.4.

[3]http://jinja.pocoo.org/

Figure 3.4: Example Jinja2 Templating

```
{% for interface in interfaces %}
{% if interface.comment %}
# {{ interface.comment }}
{% endif %}
def {{ interface.name }}({% for arg in interface.args %} {{ arg }} {% endfor %}):
    ...
{% endfor %}
```

3.2.3 Output Targets

The CORGIS project creates a variety of library bindings, data file formats, and web interfaces. Each of these different output targets has a different build process, although many involve significant overlap. In this section, we describe each output target. The first subsection is devoted to the client libraries for Python, Racket, and Java. The following subsection is devoted to the data file formats for JSON, CSV, and SQL. The last two subsections are devoted to the Visualizer and Explorer.

Client Libraries for Python, Racket, and Java

There are currently three languages that CORGIS can generate libraries for: Java, Python, and Racket. These languages were chosen for several reasons: they represent popular teaching languages, they were all relatively easy to develop for, and the developer had extensive experience with all of them.

Each library exposes the same kinds of interfaces, as specified in the CORGIS file. Figure 3.5 gives examples of how a generated library can be used in the target languages. In each example, the same data is accessed using the built-in function interfaces, stored, and then processed.

The client libraries for these languages all use function calls to simplify the process of loading data. Although all three languages have native support for processing data files, doing so is not trivial for introductory students, making these functions preferable for a number of reasons. First, file handling brings a number of hazards: developers need to check for the existence of the file, open the file in the correct manner (e.g., as a text file, as a binary file), need to handle paths correctly, etc. Once the file is opened, it must be loaded into memory and parsed using the appropriate mechanisms (e.g., as a JSON file, as a SQL data file, etc.). Once in memory, the actual data must be converted to appropriate types in the language.

The Python and Racket libraries share similar organization and strategy. Both libraries use a SQLite-based storage strategy and provide a series of module-level functions based on the specifications' interfaces. The Racket and Java libraries convert objects from the data file into native structs and classes, while the Python library leaves them as dictionaries.

The RealTimeWeb architecture used JSON-based cache files, which was initially carried over to the CORGIS architecture. However, the JSON-based approach suffered from a major flaw: the entire dataset had to be read from disk in order to load the data into memory. This was particularly problematic when the datasets were meant to be sampled, since the entire dataset would have to be read in, when only the first thousand entries were desired. The first attempted solution was to embed the test data inside the source code files through self-modification and reflection. Although technically successful, this made the initial file load time unacceptable and made it more difficult to develop the files (since a relatively large base-64 encoded string data blob was stored in each one). Eventually, the JSON files were replaced with SQLite database files. SQLite provides on-disk access, so that sampling can occur efficiently. SQLite support is excellent across most modern languages, making it easy to support without 3rd party libraries.

Internally, the SQLite database stores each row as a JSON encoded string in a "Data" column, which can be decoded at run-time. This "Data" column is indexed according to specific entries established in the CORGIS specification file.

The names of datasets and interfaces are used to generate identifiers in the code, following the appropriate naming convention for the language (Underscore case, snake case, kebab case, etc.). Extra care must be taken when generating the Java classes. The field names in the data file can incorporate characters that are not valid for class names in Java. Symbols and numbers are expanded into their written forms ("%" becomes "percent", for example).

Raw Data Files

Language bindings simplify access to datasets, but some instructors may want students to use more traditional methods. Therefore, we expose datasets in popular raw formats: JSON, SQL, and CSV.

The SQL database represents each child map as a distinct table, linked with a primary key. The CSV table is a flattened representation of the data, using the key hierarchy to disambiguate column names. For example, consider the following chain of keys in a datasets' hierarchy: X->Y->Z. If another field in the dataset also was named Z, then the field would be renamed Y->Z. This process can be repeated as many times as necessary until the fields are disambiguated from each other. The JSON format leaves the datasets unchanged. However, documentation for each field must still be generated.

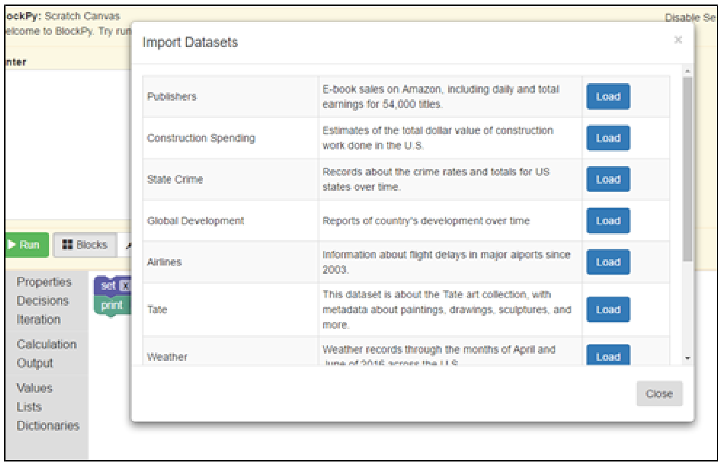
Client Library for BlockPy

BlockPy is a web-based programming environment for Python [9, 7]. At a high level, the environment allows users to edit Python code in either a block view or a text view, switching freely between them. Code is executed locally on the users' browser, with no server required.

Figure 3.5: Example of Using a CORGIS Library

```
### Python ###
# Import data
import crime
crimes = crime.get_all()
# Access data
population_1990 = 0
for crime in crimes:
    if crime["Year"] == 1990:
        population_1990 += crime["Data"]["Population"]
# Report result
print("US Population in 1990: "+population_1990)

;;; Racket ;;;
; Get data
(require crime)
(define reports (crime-get-all))

; Access data
(define (year-is-1990 a-report)
  (= 1990 (crime-year (a-report))))
(define population_1990
  (foldr + 0
    (map crime-population
      (map crime-data
        (filter year-is-1990 reports)))))
; Report result
(println population_1990)

/// Java ///
import corgis.crime.StateCrimeLibrary;
import corgis.crime.domain.Report;
import java.util.*;

public class Main {
  public static void main(String[] args) {
    // Get data
    StateCrimeLibrary stateCrimeLibrary = new StateCrimeLibrary();
    ArrayList<Report> reports =  stateCrimeLibrary.getAllCrimes();

    // Access data
    int population_1990 = 0;
    for (Report r: reports) {
        if (r.getYear() == 1990) {
            population_1990+= r.getData().getPopulation();
        }
    }

    // Report result
    System.out.println("US Population in 1990: "+population_1990);
  }
}
```

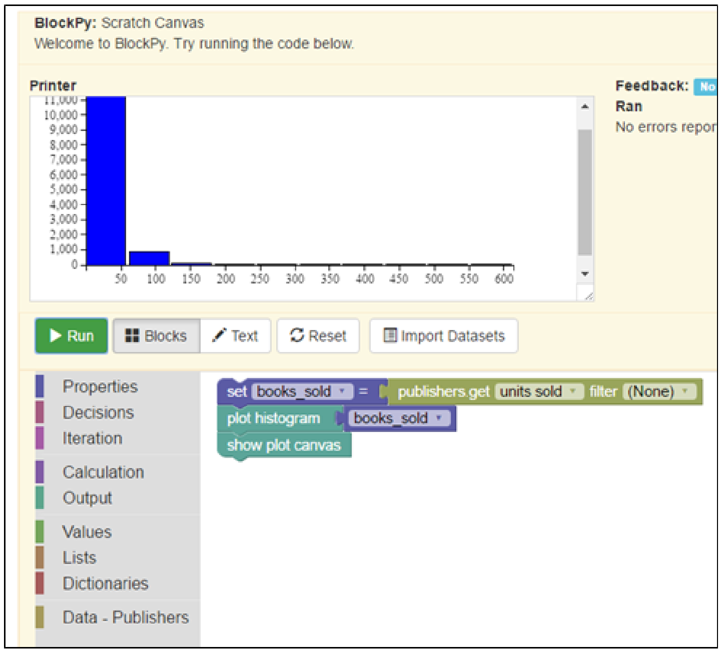
Figure 3.6: CORGIS Integration with BlockPy



Import Datasets

Publishers	E-book sales on Amazon, including daily and total earnings for 54,000 titles.	Load
Construction Spending	Estimates of the total dollar value of construction work done in the U.S.	Load
State Crime	Records about the crime rates and totals for US states over time.	Load
Global Development	Reports of country's development over time	Load
Airlines	Information about flight delays in major aiports since 2003.	Load
Tate	This dataset is about the Tate art collection, with metadata about paintings, drawings, sculptures, and more.	Load
Weather	Weather records through the months of April and	Load

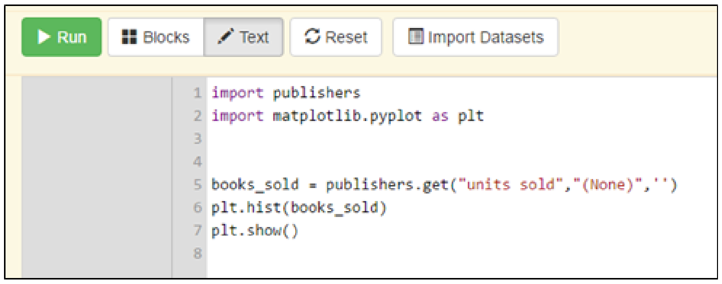
1) Import datasets with the Import Dialog



BlockPy: Scratch Canvas
Welcome to BlockPy. Try running the code below.

Feedback: No
Ran
No errors repor

2) Data Block becomes available



```
import publishers
import matplotlib.pyplot as plt


books_sold = publishers.get("units sold","(None)",'')
plt.hist(books_sold)
plt.show()

```

3) Generates Python code that can be executed in browser

BlockPy supports both CORGIS datasets and a reduced version of the MatPlotLib library for creating visualizations, making it a useful platform for introductory programming with a Data Science context.

Figure 3.6 shows how CORGIS datasets can be imported into BlockPy. First, the user clicks a button to bring up the "Import Datasets" menu. BlockPy queries the CORGIS collection and retrieves the current list of BlockPy datasets from the JSON index. This list is displayed along with a button that loads the dataset into the system. Once loaded, a new menu option appears in the Block interface for the relevant data block.

Each data block contains a dropdown menu to control which field is being accessed from the dataset. The names of these fields are generated in a manner similar to the Visualizer and for CSV files (as described above). However, the names are shortened even more aggressively in order to keep the width of the blocks at a convenient size. Since only one field is available at a time, each block represents a function call that returns a list of numbers or text.

Each datablock can optionally be filtered by a datasets' indexes. In this way, the list of data can be tuned to more specific questions. For instance, since the Airlines dataset has indexes for Airports, users can filter based on a specific airport (e.g., "BWI", or the Baltimore-Washington International airport) and return only fields associated with that airport. This is identical to the functionality exposed by the Visualizer's filter mechanism.

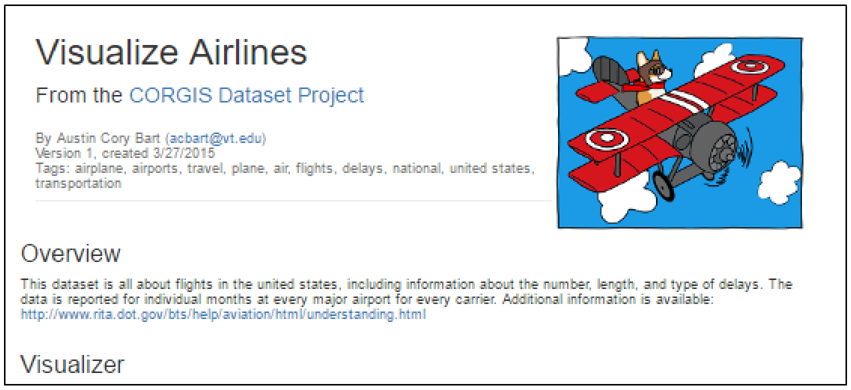
BlockPy is a fully-featured Python execution environment; the blocks are only an interface around Python code generation. At all times, the blocks are generating syntactically valid Python code that is actually run. Figure 3.6 shows an example of the generated code at the bottom of the diagram. The API is somewhat unusual compared to other CORGIS interfaces, since most of the parameters are specified as string values. This is largely done for simplicity, with the understanding that students will not often review how the datasets are loaded.

Since BlockPy has support for line plots, histograms, and scatter plots, the range of visualizations that users can make is also similar to the Visualizer (described in the following subsection). However, the BlockPy interface requires a deeper level of algorithmic and coding knowledge. Still, the simplicity of the data blocks keeps the level of complexity lower than the conventional Python, Racket, and Java client libraries.

Visualizer

The Visualizer allows students to engage with datasets in their browser without programming, as shown in Figure 3.7. The Visualizer supports four kinds of graphs: line plots, histograms, scatter plots, and bar charts. Once a graph is selected, users can choose a numeric field from the dataset to populate the graph. These fields can be filtered using index fields (described below). Users have control over axis labels, title, and even the graph's color scheme.

Figure 3.7: The CORGIS Visualizer



Visualize Airlines

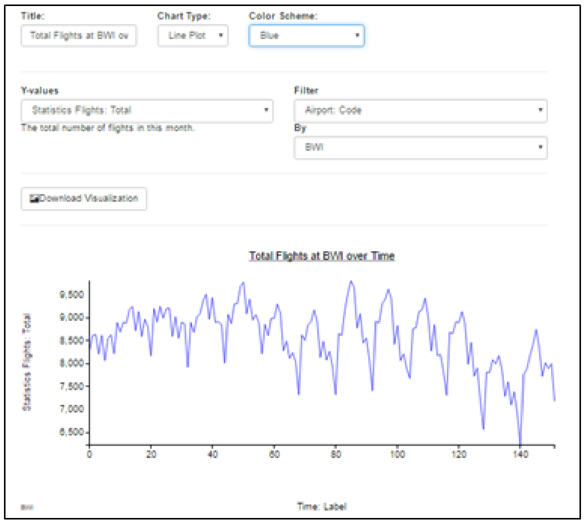
From the CORGIS Dataset Project

By Austin Cory Bart (acbart@vt.edu)
Version 1, created 3/27/2015
Tags: airplane, airports, travel, plane, air, flights, delays, national, united states, transportation

Overview

This dataset is all about flights in the united states, including information about the number, length, and type of delays. The data is reported for individual months at every major airport for every carrier. Additional information is available: http://www.rita.dot.gov/bts/help/aviation/html/understanding.html

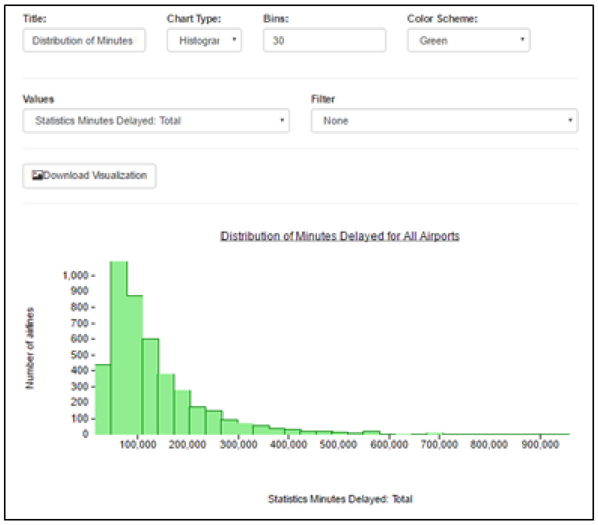
Visualizer



Title: Total Flights at BWI ov
Chart Type: Line Plot
Color Scheme: Blue

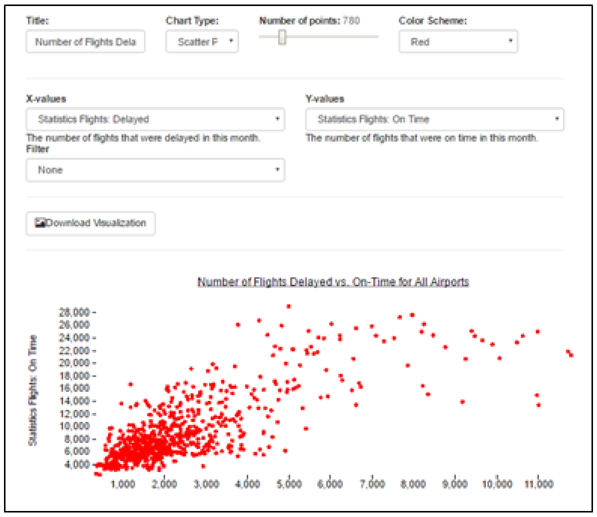
Y-values: Statistics Flights: Total
The total number of flights in this month.

Filter: Airport: Code
By: BWI



Title: Distribution of Minutes
Chart Type: Histogram
Bins: 30
Color Scheme: Green

Values: Statistics Minutes Delayed: Total
Filter: None

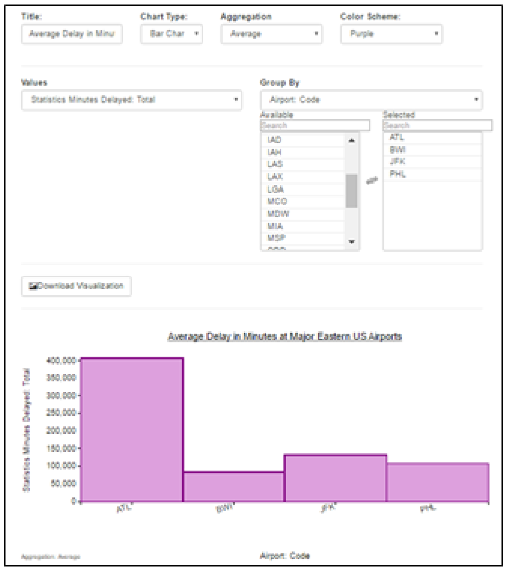


Title: Number of Flights Dela
Chart Type: Scatter P
Number of points: 780
Color Scheme: Red

X-values: Statistics Flights: Delayed
The number of flights that were delayed in this month.

Y-values: Statistics Flights: On Time
The number of flights that were on time in this month.

Filter: None



Title: Average Delay in Minu
Chart Type: Bar Char
Aggregation: Average
Color Scheme: Purple

Values: Statistics Minutes Delayed: Total

Group By: Airport: Code

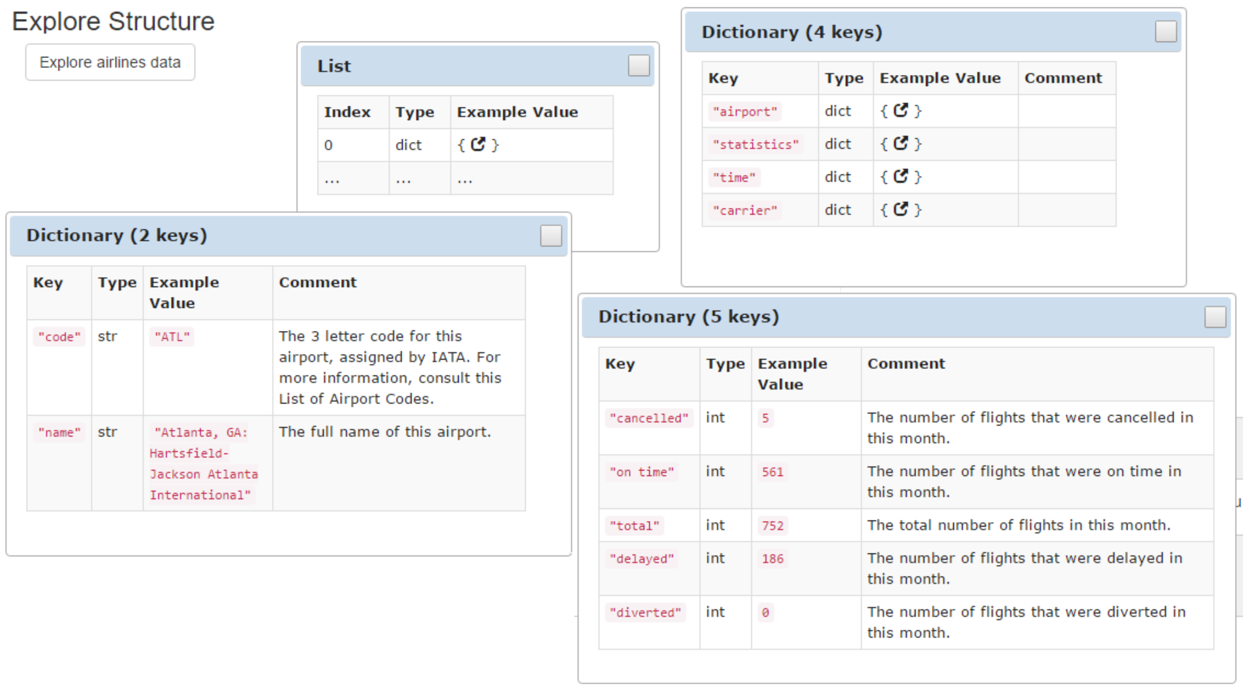
Available: IAD, IAH, LAS, LAX, LGA, MCO, MDW, MIA, MSP
Selected: ATL, BWI, JFK, PHL

Datasets can be optionally marked to have outliers removed when plotted in the visualizer. The outlier detection works by iterating through each field and computing the mean value and standard deviation. Any field that is more than a constant number of deviations (typically 2) away is considered an outlier and removed from the Visualizer's copy of the dataset. An issue occurs if an outlier is removed from one fields' list of values without removing the corresponding values in all other fields', since scatter plots will no longer be consistent when the graph maps two fields' list of values against each other. Therefore, when an outlier is detected, the entire set of values (across all fields) must be removed. The primary advantage of removing outliers is to smooth datasets with considerable noise, since noise can make interpreting certain plots difficult. Consider the earthquake data, which contains earthquakes over the past 2 months. Particularly massive earthquakes may disrupt histograms and overly skew the data. The secondary advantage of removing outliers is to keep large datasets manageable within the browser, since all datasets must be transmitted over the Internet. Removing outliers is only appropriate for incident-level data, rather than report-level data. For example, it may make sense to filter out or smooth unreliable sensor data (e.g., from an earthquake detection system), but it may not make sense to remove the state crime report for Virginia in 1971 simply because they had little-reported crime – the data point might be considered too crucial to ignore.

In the CORGIS specification file, a local dataset can have indexes specified. These indexes are specific fields in the dataset that have particular relevance to the process of filtering and sorting the data. The Visualizer automatically uses these indexes to provide a Filter function. Users select an index field from a drop-down menu (or a "None" field) and a specific value from that index in another drop-down menu. For example, the State Crime dataset is indexed by both States and Year. A user could index on State and choose "Delaware" – the effect of which is to leave a list of crime reports that are associated with Delaware in each of the available years of the dataset. Since these two indexes represent complete coverage of the data, this is sufficient for creating a line plot of the trend in crime rates in Delaware across years. If a specific "Year" had been chosen instead, say "1990", then the remaining data would instead represent each of the states' reported values in that year; in this case, a histogram could be generated of the distribution of crime rates across states in 1990. The filter is a powerful feature for the Visualizer, but can be confusing for novices, since they must think about what data remains after the field's value has been filtered out.

Histograms, Scatter plots, and Line plots are relatively straight-forward graphs for the Visualizer to make, but bar charts are a little more complex. Bar charts are categorized by the indexes from the specification file. So in addition to a field, the user must choose an index and a set of values for that index. Finally, the user must also specify an aggregation function (one of Count, Sum, or Average). For example, in Figure 3.7, the bottom right visualization shows the results of using the Average aggregation to compare four airports (BWI, JFK, ATL, and PHL) total delay time in minutes.

Figure 3.8: The CORGIS Explorer



Explore Structure

Explore airlines data

List

Index	Type	Example Value
0	dict	{ ↗ }
...	...	...

Dictionary (4 keys)

Key	Type	Example Value	Comment
"airport"	dict	{ ↗ }	
"statistics"	dict	{ ↗ }	
"time"	dict	{ ↗ }	
"carrier"	dict	{ ↗ }	

Dictionary (2 keys)

Key	Type	Example Value	Comment
"code"	str	"ATL"	The 3 letter code for this airport, assigned by IATA. For more information, consult this List of Airport Codes.
"name"	str	"Atlanta, GA: Hartsfield-Jackson Atlanta International"	The full name of this airport.

Dictionary (5 keys)

Key	Type	Example Value	Comment
"cancelled"	int	5	The number of flights that were cancelled in this month.
"on time"	int	561	The number of flights that were on time in this month.
"total"	int	752	The total number of flights in this month.
"delayed"	int	186	The number of flights that were delayed in this month.
"diverted"	int	0	The number of flights that were diverted in this month.

Explorer

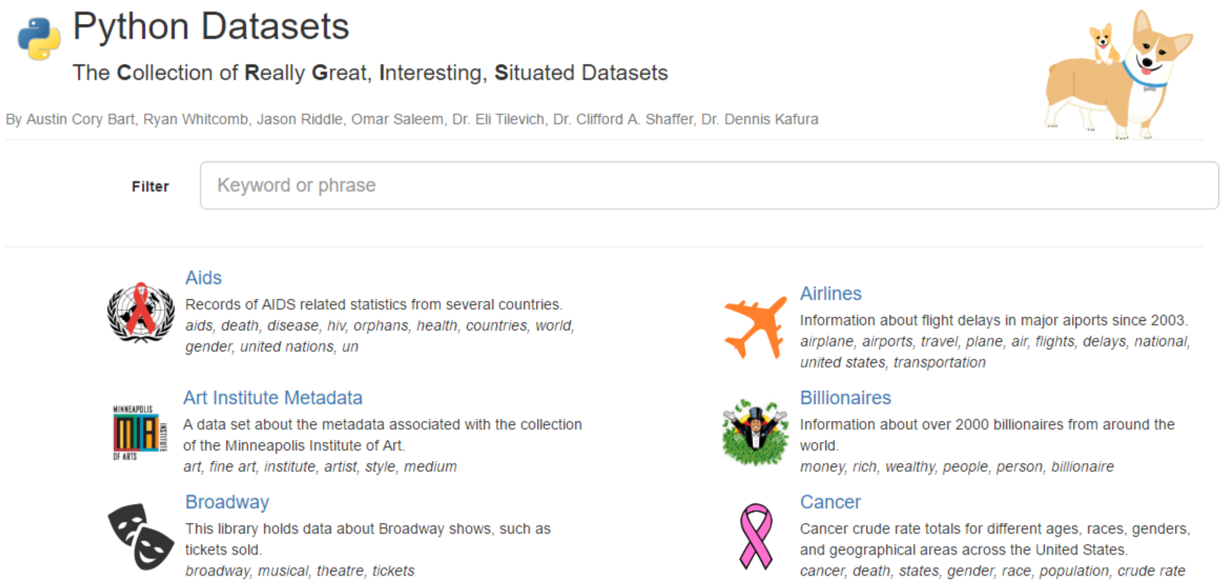
As previously stated, all data files are organized as a list of dictionaries at the top level. Internally, those dictionaries contain nested dictionaries and leaf fields of various basic types. Ultimately, every field must be documented for its type and meaning. Although one option was to document each field in a flat laundry-list of names definitions, the CORGIS project instead provides most documentation for the client libraries through the Explorer.

The explorer is an interactive tool for exploring both the structure and definition of data. The hierarchy of data is explored through chained windows. Clicking a field in a dictionary, for instance, might open a new dictionary's window, which in turn might have atomic fields or links to further dictionaries. This layered effect is intended to portray the hierarchical and nested structure of the data.

3.2.4 Gallery

All CORGIS materials (data files, libraries, and web-based tools) are available through the online CORGIS gallery (https://think.cs.vt.edu/corgis). The gallery is organized by the various output targets, each of which has its own page with the associated datasets. An example page in this gallery for the Python output target is shown in Figure 3.9, and a

Figure 3.9: The CORGIS Gallery



Python Datasets

The **C**ollection of **R**eally **G**reat, **I**nteresting, **S**ituated Datasets

By Austin Cory Bart, Ryan Whitcomb, Jason Riddle, Omar Saleem, Dr. Eli Tilevich, Dr. Clifford A. Shaffer, Dr. Dennis Kafura

Aids
Records of AIDS related statistics from several countries.
aids, death, disease, hiv, orphans, health, countries, world, gender, united nations, un

Airlines
Information about flight delays in major aiports since 2003.
airplane, airports, travel, plane, air, flights, delays, national, united states, transportation

Art Institute Metadata
A data set about the metadata associated with the collection of the Minneapolis Institute of Art.
art, fine art, institute, artist, style, medium

Billionaires
Information about over 2000 billionaires from around the world.
money, rich, wealthy, people, person, billionaire

Broadway
This library holds data about Broadway shows, such as tickets sold.
broadway, musical, theatre, tickets

Cancer
Cancer crude rate totals for different ages, races, genders, and geographical areas across the United States.
cancer, death, states, gender, race, population, crude rate

complete rendering is show in Appendix A.5. Each output targets' gallery can be searched by the datasets' specified tags, description, and text.

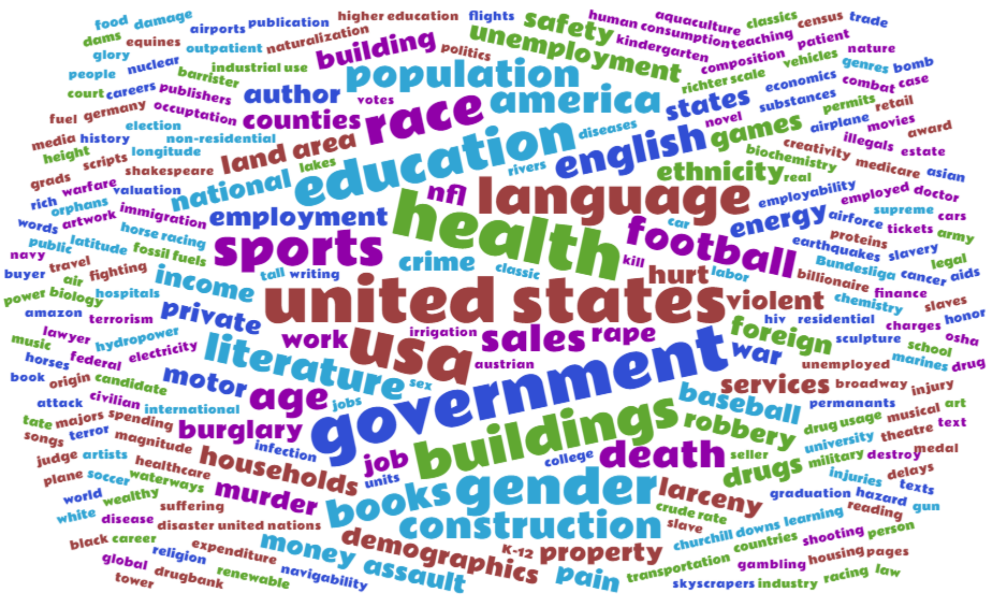
Each individual page for a dataset provides all the necessary files to use the dataset or client library, along with the documentation. This documentation includes information about the interfaces available, the meaning of the fields, and an overview of the dataset. Typically, this overview will include the provenance and history of the dataset.

The CORGIS projects' gallery is automatically generated during the build process. The RealTimeWeb was originally available through a Wiki-based gallery. This portal was updated by hand, which provided flexibility at the cost of convenience. Unfortunately, this quickly proved intractable as the number of datasets increased. This motivated the decision to automate the CORGIS' projects' gallery creation.

3.2.5 Dataset Characteristics

At the time of writing, there are over 40 datasets in the CORGIS gallery, and we are actively working to add more datasets. Figure 3.10 shows a word cloud of all the descriptive tags associated with the CORGIS datasets. This graphic illustrates the range of datasets associated with the project. This also reveals certain biases and trends in the selection of datasets. "United States", for instance, is the single largest word in the cloud. "Education", "language", "health", "government", and "buildings" are the next most popular terms, each of which maps closely to specific datasets.

Figure 3.10: The CORGIS Datasets' Tags as a Word Cloud



What does the typical CORGIS dataset look like in terms of structure and content? Previously, we have noted that every CORGIS dataset is a list of dictionaries, with each dictionary being composed of dictionaries and primitive fields (integer, float, strings, and booleans). The quantity, nesting depth, and width (number of keys) of the dictionaries varies across datasets. Figure 3.11 gives an example of the structure of a dataset (the Classic Books dataset).

The left graph of Figure 3.12 reveals characteristics of the datasets within the corpus. If the data's structure is seen as a tree, the Average Branch Factor (ABF) is the mean number of fields in a child. The height of a dataset is the maximal depth of the tree, and Fields is the number of leaves. Rows is the number of records in the dataset, while size is the amount of disk space used by a dataset. Although we are pleased with the narrow distribution on some of the attributes (e.g., heights), the dispersion of ABF and Fields suggests that some datasets need to have fewer fields organized into more branches.

The right graph of Figure 3.12 shows the distribution of atomic and composite types within datasets. Numeric and string types dominate. Most datasets have few or no boolean types, and few datasets have more than just the top-level list (which is present in every dataset). The x-axis shows percentages of all types within the dataset, with numerics and strings as the most common. The chart does not distinguish between different types of string values, such as unique identifiers, URLs, classification codes, or textual data.

Figure 3.11: Data Map of a CORGIS Dataset

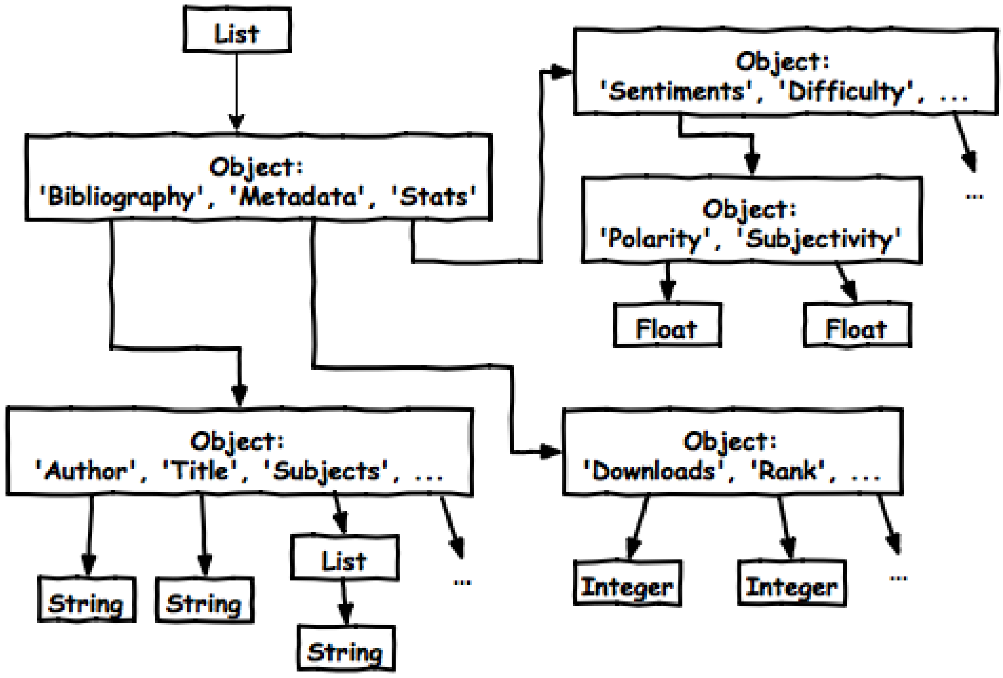
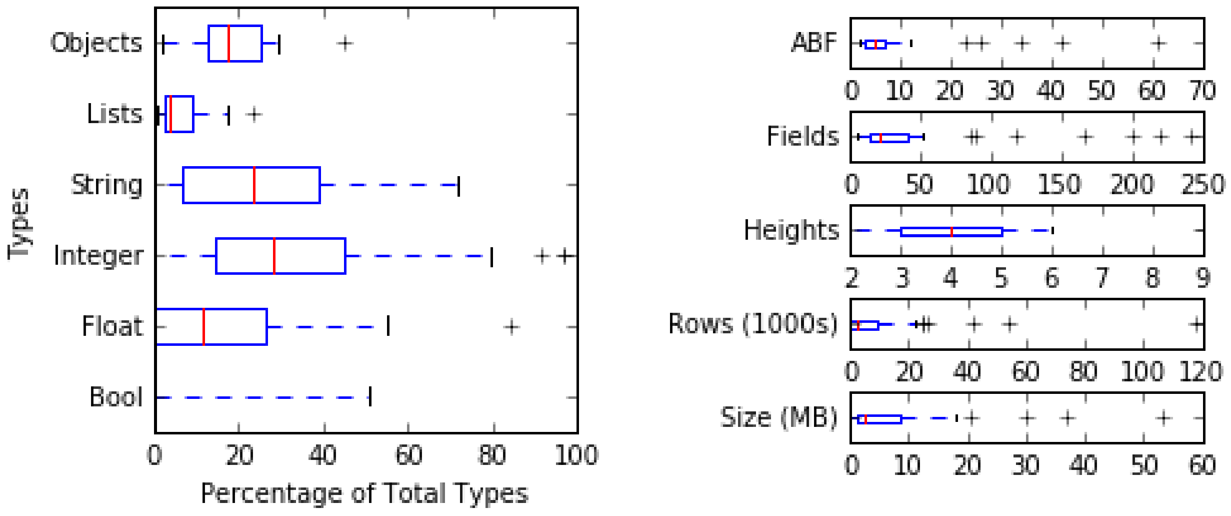

List
Object: 'Bibliography', 'Metadata', 'Stats'
Object: 'Sentiments', 'Difficulty', ...
...
Object: 'Polarity', 'Subjectivity'
Float
Float
Object: 'Author', 'Title', 'Subjects', ...
Object: 'Downloads', 'Rank', ...
String
String
List
String
...
Integer
Integer
...

Figure 3.12: Characteristics of CORGIS Datasets



3.2.6 Building Problems

To better illustrate the pedagogical uses of the CORGIS dataset collection, this section will describe some ways that introductory computing courses can incorporate CORGIS materials.

Exploratory Analysis

The Visualizer facilitates early exploration of datasets, without the need to program. In the beginning of a course, students can generate graphs to answer questions, either from an instructor or of their own choosing. By creating visualizations, students can explore questions about the distribution, trends, and relationships of the data. For example, in the crime dataset, they can identify the downward trend of violent crime rates over time in different states. This is an opportunity to discuss how complex, real-world entities and phenomenon can be represented with computable abstractions (e.g., numbers). Additionally, this gives students practice with selecting and interpreting different kinds of charts. In our experience, many students struggle with aspects of graphs such as distinguishing bar charts and histograms, or knowing when to use line plots.

Practice Problems

Many introductory courses start students by writing short programs (less than 10 lines) that can solve simple, well-defined tasks. Although the scope of these problems is similar to those found in systems like CodingBat [64], the CORGIS collection can make these problems more realistic. The complexity of using complete datasets is avoided by using the simpler interfaces exposed for libraries. For example, students might be assigned the task of writing a program to print whether an umbrella is necessary depending on the weather in their current city (requiring only a function call to the Weather library, an if-statement, and the print statement).

A wide range of programming topics can be contextualized with the CORGIS libraries, including collection-based iteration, decisions, printing, and functions. This is especially true when having students perform common operations on lists of numerical data. Descriptive statistics, such as the Mean, Sum, and Count can give students practice with accumulation-style iteration. Minimum and Maximum require a loop with a decision based on the accumulated variable. Retrieving the first, last, or median value from a list can let students practice list indexing functions. More sophisticated problems can teach students functional list-handling operations: filter, reduce, and map have many applications on CORGIS datasets. Finally, the CORGIS datasets can also be a platform for more complex mathematics, such as inferential statistics operations: calculating variance, standard deviation, correlation coefficients, or even conducting ANOVAs and matched-pairs tests.

Chapter 4

Student Impact

In this chapter, I discuss a number of studies conducted in various introductory computing courses in order to answer fundamental questions about student motivation in those courses. I begin by describing a series of initial studies that provided guiding data for my research, based on small interventions in existing computing courses using the RealTimeWeb software. These interventions are presented in Section 4.1. As the RealTimeWeb project evolved into the CORGIS project, the scope and depth of the interventions shifted dramatically into full-blown course creation. This course, described in detail in Section 4.2, provided a test bed for the major research questions of my dissertation. Section 4.3 describes these research questions, while Section 4.4 describes the mechanisms for collecting data to answer the questions. Finally, Section 4.5 reviews the results collected from 5 iterations of the course, and discusses the implications with regards to my major research questions.

4.1 Exploratory Interventions with RealTimeWeb

During the development of the RealTimeWeb project, a number of studies were conducted to guide initial development of both the technology and my motivational research questions. Table 4.1 summarizes these studies and surveys. The first and last surveys were associated with interventions, while the middle survey was an exploratory study to gather baseline results. All of these related only to the RealTimeWeb phase of my research.

4.1.1 Spring 2013 - Usability Study

I piloted the Business Service client library in a second-semester (CS2) course offered at Virginia Tech (CS 2114) during Spring 2013. The goal was to collect usability data that could refine the interface of the libraries and the assignments incorporating them. This

Table 4.1: Overview of RealTimeWeb Surveys

Panel A: Pilot Intervention Study

Semester	Course	Institution	# of Students	Library	Language
Spring 2013	CS2	Virginia Tech	17	Business	Java

Panel B: Relevancy Survey

Semester	Course	Institution	# of Students
Fall 2013	CS 1	Delaware	77
Fall 2013	CS 2, DSA	Virginia Tech	376

Panel C: Major Interventions Study

Semester	Course	Institution	# of Students	Library	Language
Fall 2013	CS1	Delaware	70	Weather, Reddit	Java
Fall 2013	CS2	Virginia Tech	176	Business	Java
Fall 2013	DSA	Virginia Tech	22	Earthquakes	Java

course covers topics typical to a second semester Computer Science course, including object-oriented concepts and the Java programming language. Students in the class are Computer Science majors and minors in their first or second year. Every week, students use paired programming to complete a lab assignment. There are also three large projects meant to be completed over the course of the semester.

During one lab session, students were assigned a multipart problem that required the use of the Business Services client library. This library reports information from the Yelp API about businesses. To personalize their experience with the library, the internal data cache was filled with data from the local Blacksburg area. The description of the lab began with an overview of the problems and advantages inherent in using real-time data, and then described how the Business Service library could be used. In the first task, students used the library to create a list of highly rated businesses in the area. In the second, they expanded that list with more detailed information about the businesses. The primary intent of this lab was to familiarize students with the client library. Students had a second opportunity to work further with the library in the third project of the course, where they built a more complicated "Restaurant Guide" Android application. Students again searched for a list of restaurants in a given region and compiled the results into a data structure.

A voluntary survey was administered to the class after students had completed the project, and 17 responses were gathered. Overall, the results were positive. All but one student indicated that they found the API "applicable to day-to-day computing experiences", and all students thought that the API was "easy-to-use". Qualitative feedback from the students highlighted “clear, easy-to-figure-out methods” and that they “provided information that was easy to work with”. The encouraging results meant that few, if any changes, were needed when expanding the project.

4.1.2 Fall 2013 - Relevancy Survey

In Fall 2013, a survey was conducted of 370 undergraduates at Virginia Tech and the University of Delaware. This survey was administered at the start of the semester, without any interactions involving the research software. The population of 452 students spanned three courses: CS 1 at the University of Delaware, CS 2 at Virginia Tech, and a Data Structures and Algorithms (DSA) course also at Virginia Tech. 65 (14%) of the respondents identified as female, and 300 (66%) of the respondents identified as Computer Science majors. Most students were in their Junior year (145, or 32%) or Sophomore year (133, or 29%), with relatively fewer Freshmen (94, or 21%) and Seniors (70, or 15%); the remaining 11 (2%) did not identify as Freshmen through Senior.

Beyond basic demographic information, the survey asked three multipart questions:

1. “Class projects for CS courses often involve some form of computing technology. For each of the following, please indicate how interested you would be in taking a class where the programming projects used this technology.”
2. “For each of these, how much do you think that having experience in programming with that technology would benefit your career prospects?”
3. “If you were to do a project using real-time data streams, how interested would you be in using the following data sources?”

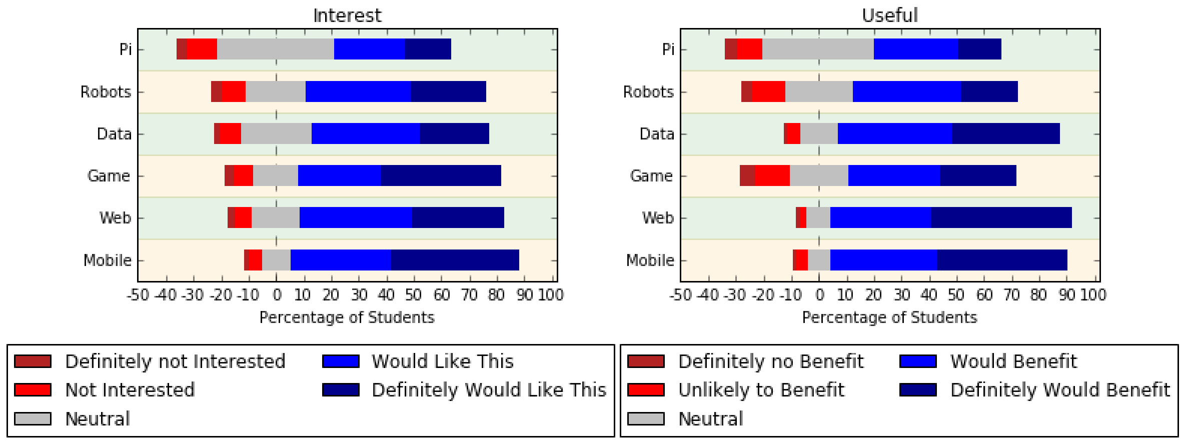
The first two questions presented the same list of introductory contexts:

- “Programming Robots”
- “Using Real-time data (e.g. weather forecasts, twitter posts, stock data)”
- “Mobile app development”
- “Website and web application development”
- “Game development”
- “Raspberry Pi development”

These questions used similar 5-point Likert scales, with the options for “Definitely not interested”, “Not interested”, “Neutral”, “Would like this”, and “Definitely would like this” (the second question replaces “interest/like” with “benefit”). The third question lists 8 datasets that were available or planned for the RealTimeWeb collection.

Figure 4.1 shows the distribution of student responses to the first two survey questions. The figure on the left is their interest in the context, and the figure on the right is their sense of the usefulness of each context. A Mann-Whitney U test was conducted between each contexts’ interest and usefulness distribution. Only Robots, Games, Web, and Real-time Data had significant differences. Students expressed similar levels of disinterest and uselessness for working with Raspberry Pi devices and similar levels of interest and usefulness for working with Mobile development. Working with Robots and Games featured relatively higher interest than usefulness, and Real-time Data and Web Development featured higher usefulness than interest.

Figure 4.1: Students’ Interest and Sense of Usefulness in Introductory Contexts



Working with Real-time Data, Web Development, and Mobile Development were considered the most useful contexts. Mobile Development was considered one of the most interesting, followed by Web and Game development. Working with Real-time Data scored reasonably in both categories, with a similar median to most others (only Raspberry Pi scored lower in both categories, and only Web Development scored higher in Usefulness). Interestingly, when the data is broken down by demographics, non-majors were significantly (but not dramatically) less interested and saw less benefit from Web Development and Game Development than majors did. My conclusion from this analysis is that Real-time Data, while perhaps not the best context for all students, is certainly competitive with other options used in introductory courses; this is especially true when emphasizing the usefulness of the context to non-majors.

Figure 4.2: Students' Interest in Data Streams

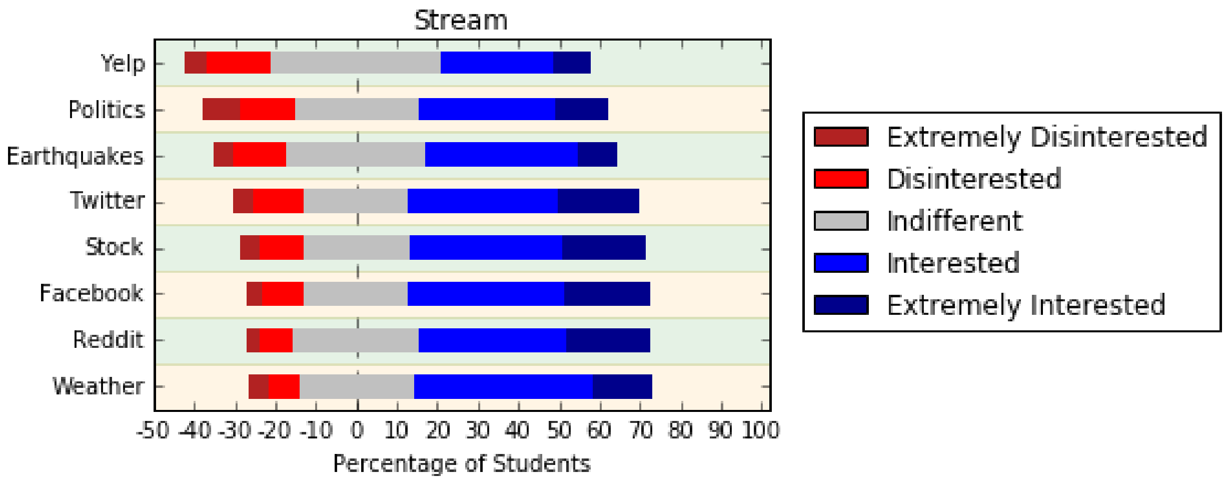


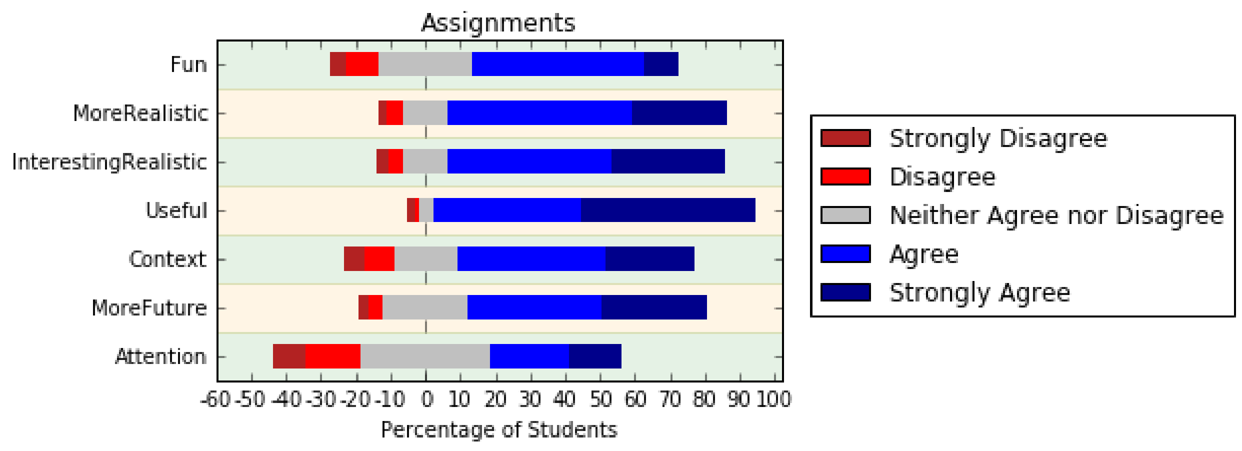
Figure 4.2 shows students' interest in 8 different datasets. Only a few datasets elicited particularly negative results: the Yelp (business reviews), Politics (Congressional voting information), and Earthquakes datasets. Conducting an all-pairs Mann-Whitney U test reveals significant differences between these datasets and the others; in fact, the median of each of these datasets was a full level lower (i.e. every student who said "strongly interested" about one dataset would only be "interested" in those three). The remaining datasets have no significant differences, suggesting that they may be more suitable when assigning datasets to the entire class.

4.1.3 Fall 2013 - Post Survey

During the Fall 2013 semester, 3 interventions were conducted using the RealTimeWeb libraries. These interventions were staged in the same classes where the Relevancy Survey were administered. Students were given at least one assignment that used a RealTimeWeb library; at the end of the semester, they were given a survey that asked questions related to the library's usage and its motivational impact. Unfortunately, the intervention in the DSA course suffered a number of serious technical issues related to the automated grading system. This had a serious impact on the response rate and the nature of the responses. I have chosen to leave this data out of the analysis, so that the results are not affected by the technical issues.

Figure 4.3 describes students' responses across a variety of statements, from "Strongly Disagree" to "Strongly Agree".

Figure 4.3: Students' Response to Survey Questions about RealTimeWeb



Fun: Were the assignments fun? Students generally seemed to think so.

MoreRealistic: Did the Real-time data make the project more realistic? Students seemed to agree.

InterestingRealistic: In general, are projects more interesting if they are realistic? Again, students seemed to agree.

Useful: Is it useful to know how to work with Real-time data? This was the strongest positive response, showing the value of the context in terms of usefulness, once again.

Context: Did the real-time data context make the data structures' purpose more clear? Students generally seemed to think so.

MoreFuture: Do students want more datasets in the future? Students seemed to want more.

Attention: Did students pay more attention to the assignment because it incorporated Real-time data? This was easily the most negative response, suggesting that Real-time Data did not have an impact on engagement.

4.1.4 Summarizing RealTimeWeb

Data collected during the RealTimeWeb project was promising, but the conclusions from analyzing this data were not particularly revolutionary. As a context, Real-Time Data did not seem particularly better than many existing contexts. The real-time nature of the data added a number of difficulties that, while interesting technical problems to solve, ultimately did not seem worth the added struggle. In some cases, such as the DSA course from the third study, the unexpected difficulties had a somewhat negative impact on the course. Although Real-Time Data has potential to be both interesting and useful to students, the lukewarm response led to a shift in both target audience and nature of the context being researched. Specifically, the CORGIS project would focus on non-majors (who seemed to react more positively to Real-time Data as a context) and would remove the Real-time element of Data Science as a context.

4.2 Designing a Computational Thinking Course with CORGIS

The initial versions of the RealTimeWeb project were used in traditional introductory computing courses, and some initial data was collected there. However, the vast majority of the research data used in this dissertation was collected in a Computational Thinking course designed using CORGIS from the ground up. I was a primary developer of the course material, along with Dr. Dennis Kafura and Bushra Chowdhury. Although there are a number of existing Computational Thinking curricula, I defined a particular set of educational objectives and course topics that I believed were valuable for our students. Large portions of our course were created from scratch to achieve our pedagogical goals. In this section, I describe the courses' audience, content, context, structure, technology, and assessments.

4.2.1 Audience

The Computational Thinking curriculum was designed to fulfill a new general education requirement at Virginia Tech, to provide "Computational Thinking" to all undergraduates. Therefore, students come from the arts, social sciences, humanities, agriculture, and more. Although some science and engineering majors enroll, it is expected that most students will not be STEM majors. The students are not expected to have prior programming experience, and the course itself has no prerequisites. Students could be at any level, from Freshman to Senior.

4.2.2 Course Content

The three major topics in the course are:

- **Abstraction:** The idea that real-world entities (e.g, people, objects, events, actions, processes, etc.) can be simplified and concretely represented to suit the needs of some stakeholder.

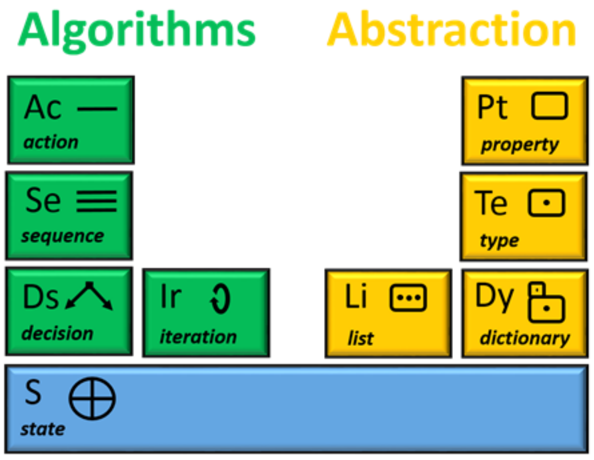


Figure 4.4: Fundamental Elements of Algorithms and Abstraction

- **Algorithms:** The idea that computational abstractions can be manipulated using formal language.
- **Social Impacts:** The idea that computing provides new powers and knowledge that can influence society.

Our core learning objective is for students to be able to create an algorithm that manipulates data to answer real-world questions. Students must express their algorithms using a practical programming language – in this case, Python. Further, they must be able to navigate and describe data in contexts related to their own careers and across other fields. Finally, they should be able to discuss the social impacts, relevant stakeholders, and ethics of the questions and answers they find.

Figure 4.4 illustrates the hierarchical and connected nature of two of the courses' topics, Algorithms and Abstractions. Algorithms, fundamentally, describe *actions* in a specific *sequence*. Depending on the formality of the algorithmic agent, these actions can vary in complexity and atomicity. Higher-order constructs empower algorithms with the ability to branch (make *decisions*) and repeat (*iterate*). Abstractions are concretely represented as variables or *properties*, which formally have *types* and quantitative characteristics. We refer to variables as properties to avoid confusing students with the algebraic term. Higher-order organizational structures such as homogeneous *lists* and heterogeneous *dictionaries* allow for construction of complex data. Connecting these two ideas is *state*, the collection of properties whose structure is defined by Abstraction and whose values are used or changed dynamically during the performance of an Algorithm. Taken together, these concepts describe our ideal "notational machine" for students, unifying their understanding of control structures with the data structures being manipulated.

The topics of algorithms, abstraction, and social impacts were chosen for a combination of philosophical and pragmatic reasons. Philosophically, the algorithmic constructs in Figure 4.4 are embodied in the Turing machine model of computation and are elements of imperative programming languages. In addition to giving robust meaning to "computation", the algorithmic elements are sufficient for our learners to develop cognitive skills in process-oriented thinking, which we view as one component of the "thinking" in computational thinking. Pragmatically, time constraints prevent us from including more advanced programming concepts (e.g., objects, functions). Philosophically, the abstraction elements in Figure 4.4 carry through on our commitment to presenting computation as being a tool for answering questions about the real world. These abstraction elements are sufficient to define instances of complex entities using structured collections with information-bearing properties. In addition to helping students see the world through the lens of information, the abstraction elements stimulate cognitive skills in the management of complexity through "layers of abstraction", which we view as the second component of the "thinking" in computational thinking. Pragmatically, time constraints limit the number of structures that we can introduce, and the ones we did include were influenced by our choice of Python as the programming language. Philosophically, the social impacts topic encourages our learners to think about the ways in which computation shapes decisions made in the real world and influences the character of that world. This course component is in line with calls for computer science education to include the social and ethical dimensions of computing. Pragmatically, we were influenced by the requirement of our university's general education guidelines to include a components on ethics.

In the original version of the course, we also taught a unit on creating functions to encapsulate reusable chunks of code. However, time restrictions forced us to remove those lessons. Instead, modules and functions are introduced as mechanisms for reusing existing code written by other developers, without any class time devoted to creating functions. In many cases, we attempt to provide this material on functions in individual lessons with students.

4.2.3 Course Context

Orthogonal to the course content, the course context is centered around Data Science, supported directly by the CORGIS project. In the course, students must extract meaning from data using computational analyses. Our argument to students is that most disciplines benefit from Data Science, since the world is becoming increasingly data-driven. Students use datasets provided through the CORGIS project in order to create visualizations (e.g,. histograms, line plots) and perform simple statistical calculations (e.g., average, sums, counts). These kinds of analyses, while not statistically rigorous or computationally difficult, provide a range of appropriate introductory-level problems and projects. Students interpret these visualizations in a social context, further supporting the core learning objectives. The course context provides the hook for deploying and evaluating the CORGIS project.

Figure 4.5: Overview of Course

Module	Topics	Time Period
1	Course Overview	Days 1-4
2	Computational Modeling and Abstraction	Days 5-9
3	Algorithms and Programming	Days 10-14
4	Python and Big Data	Days 15-19
5	Mini-project	Days 20-22
6	Final project	Days 23-29

4.2.4 Course Structure

As shown in Figure 4.5, there are several distinct phases to the course: introduction, hands-on conceptual activities, BlockPy, Mini-project, and Final Project.

Module 1 is a brief overview of the course, presenting the major topics and the cohort system (which uses peer learning). After their introduction to the course, students originally used NetLogo[1] and then the Blockly Maze game[2]. In later iterations of the course (starting with the Fall 2016 semester), NetLogo was replaced with the CORGIS Visualizer, and the Maze activity was moved to Module 2. These applications introduce the overarching ideas of Abstraction and Algorithms, respectively.

In Module 2, students begin learning fundamentals of computational abstraction and basic control structures, working mostly with paper-and-pencil and kinesthetic activities. In one lesson, groups of 5 (referred to here as a "cohort") work together to write instructions for a "card-sorting robot", executed by the course staff on playing cards in order to demonstrate the basics of algorithm design and the ambiguities of natural language. The course staff also performs a "play" that models the state of the program, the program counter, and console for a simple algorithm to process a list of data. During execution of the program, the instructor regularly stops the lesson and has students predict the next state. There are also a series of smaller pencil-and-paper activities for topics relating to decision, iteration, and abstraction.

During Module 3, students must complete a series of programming assignments in BlockPy, a block-based programming environment for Python [7]. These assignments incorporate real-world datasets from the CORGIS project, and students create charts and compute simple statistical information (averages, counts, thresholds, etc.). By the end of Module 3, students are working in the "text mode" of BlockPy, to prepare them for the transition to Module 4. This module builds on their Python experience by having them complete similar programming assignments, except this time in a professional desktop IDE named Spyder[3].

[1]https://ccl.northwestern.edu/netlogo/

[2]https://blockly-games.appspot.com/maze

[3]https://pythonhosted.org/spyder/

They continue to use Spyder in Modules 5 and 6, where they complete a mini-project and then their final project. During the mini-project, students work in their cohort to answer questions about the CORGIS State Crime dataset. As a group, they create visualizations and a presentation, which they are individually responsible for presenting as a video. This is largely practice for their final project, which is a month long, individually-paced activity to create a 5-minute video presentation answering questions related to a dataset of their choosing using Python-based visualizations.

Throughout the course, most of the modules end with a day on Social Impacts. In the first module, students review final projects from previous semesters that are relevant to their career interests, and must discuss and critique the social impacts of the video. In the second module, students watch a video about an ethical scenario where a computer potentially caused an accident, and they are asked to debate within their cohorts the ethical ramifications. The third module ends with a brief lesson about various ethical frameworks (e.g., Utilitarianism, Duty, Common Good, etc.) and students apply these frameworks to the debate from the previous module. In both the mini-project and the final project, students are expected to discuss the social impacts of the questions and answers that they determine.

Students are assigned daily readings accompanied by reading quizzes. The readings, written specifically for the curriculum by the course staff[4], introduce the material for the day with a few pages of prose and images. These quizzes are short, multiple-choice assignments that are expected to take only a few minutes. They are administered and automatically graded by Canvas. Students are allowed 3 attempts before the quiz closes

The class meets twice a week for 75 minutes. A typical day of the course begins with a short introductory lecture on the material. Students then work in their cohorts or individually to complete the interactive component (e.g., paper-and-pencil activity, collaborative discussion, programming assignment, project work). Finally, the class comes back together to discuss the lessons of the day and any challenges that occurred. Most lessons include a required homework assignment meant to reinforce students' learning and prepare them for the next class.

Cohorts

On the third day of the course, students are grouped into collaborative cohorts of 5-6 students. Each undergraduate teaching assistant (UTA) is responsible for two cohorts. A graduate teaching assistant is responsible for managing the UTAs, and in turn is overseen by the course instructors.

Most early classwork activities are explicitly collaborative, and students are expected to generate a single answer from their cohort. During the programming section of the course, most assignments require individual submissions, but students are allowed to work together

[4]http://think.cs.vt.edu/book/

and get help. Students are forbidden from sharing their solutions, but are encouraged to share their thought process and provide support. Near the end of the course, students complete a group "mini-project" to prepare for their individual final project. This mini-project allows a cohort to create questions and visualizations collaboratively, although they are required to submit individual presentations.

4.3 Research Questions

Creating a new course for non-Computer Science majors provided a unique opportunity to test and grow the CORGIS project. At a high level, this course allows us to study the question: What is the impact of a Data Science context, as opposed to other course components, on student motivation and course outcomes? In order to answer this major research question, I break down the problem into multiple parts.

4.3.1 What is the Impact of a Data Science Context on Motivation?

In Chapter 2, I argued that a Data Science context would be able to serve multiple aspects of motivation. As previously described, the MUSIC Model states that motivation comes through five different avenues: sense of empowerment (agency), sense of usefulness, sense of success (self-efficacy), interest, and a sense of being cared for. In the Computational Thinking course, I seek to understand how a students' motivation to engage in the course changes over the semester, particularly in relation to the content, context, and scaffolds of the course.

1. What course components are particularly effective or not effective at providing motivation?
2. How does students' motivation change over the course of the semester?
3. How does motivation compare between genders?
4. How does Data Science appeal to students compared to other potential introductory course contexts?

4.3.2 What is the Impact of a Data Science Context on Course Outcomes?

Although motivation is an important end goal, course designers consider a range of course outcomes when evaluating the success of a course. When designing our new course, we were interested in achieving a number of objectives beyond motivation. These objectives establish the criteria for the success of a course oriented around Data Science. In Chapter 2, I reviewed literature that argues that increased motivation will lead to improved course outcomes. Therefore, I am also interested in whether students' motivation has any connection to course outcomes, whether in terms of correlation or prediction. Ultimately, I look at course outcomes to answer two questions:

1. Is a course designed around a Data Science context viable with respect to key course outcomes?
2. What aspects of motivation correlate with course outcomes at the beginning and end of the semester?

In order to do this analysis, I will collect several different course outcomes:

Course Performance Measured as their final grade in the course as a percentage out of 100 in categories of the assignment types: attendance, classwork, homework, projects, and cumulative course grade.

Project Performance A number based on the students' performance on the final Data Science project, assessed via a rubric.

Procrastination Measured as a number indicating how many times the student handed in an assignment late.

Intent to Continue Students' self-reported interest in continuing engaging with computing.

4.4 Data Collection

Data has been collected over five semesters. During that time, the type and quality of the data has changed.

Figure 4.6 gives an overview of the data collection timeline. Different kinds of data are collected over the duration of the course. At the start of the semester, student demographics are collected to support our learner analysis. A set of motivational surveys are administered at the beginning, middle, and end of the course. In the second module, and then later at the end of the course, a number of assessment instruments are used to measure learner outcomes. During the BlockPy module, we collect a large quantity of user interaction log data as students write solutions to simple programming problems. Attendance is also logged throughout the semester. Finally, a number of participation-based classwork, homework, and

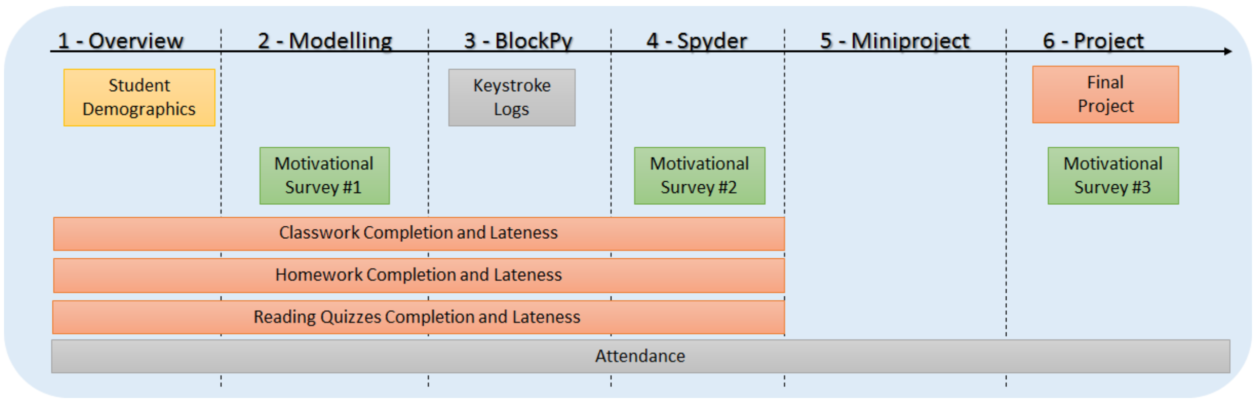


Figure 4.6: Data Collection Timeline

other course indicators are collected during most phases of the course. Classwork, homework, and reading quizzes do not occur during Modules 5 and 6 because students work on projects.

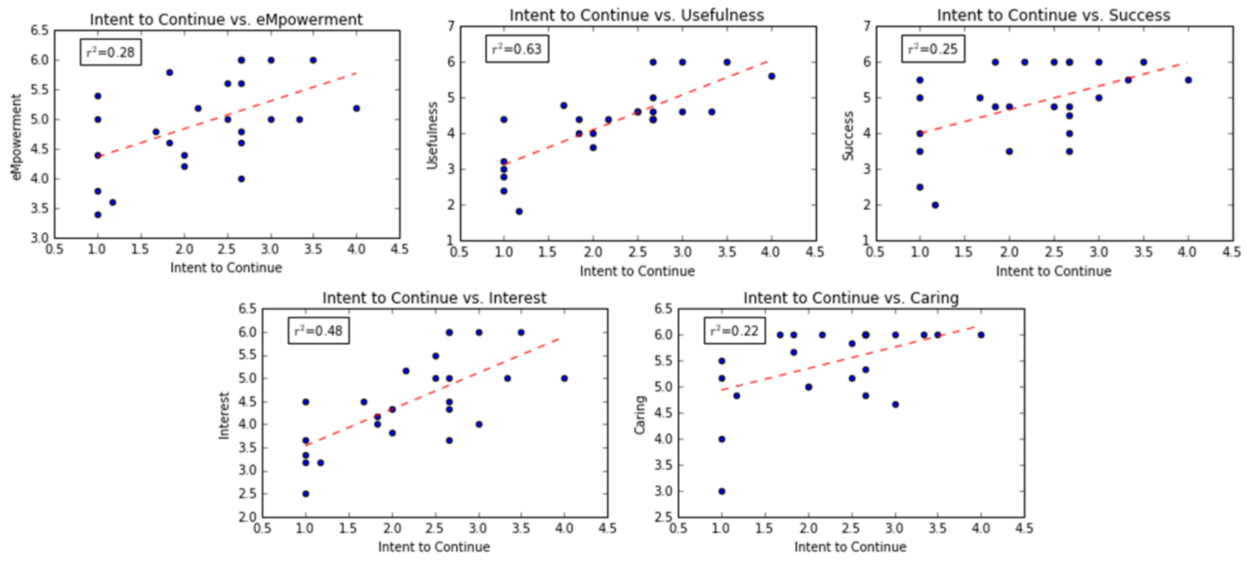
All data was collected with full compliance from our Institutional Review Board. Students were informed that data would be collected from the various computational tools. Consent was gathered using paper release forms. When students did not give consent to the use of their data, their data was removed from the analysis process. After collection, all data was anonymized, and are only reported in aggregate.

4.4.1 Surveys

Surveys were used to determine students' motivation, representing the core data of this research project. Although self-reported data has threats to validity, it is a convenient and practical way to gather quantitative data about students' attitudes towards the course. Surveys were administered every semester of the course, integrated directly into the design of the course. In Chapter 2, I described the 5-component MUSIC model, which was the motivational framework for developing all of the surveys. The MUSIC model has an associated instrument, named the Music Model of Academic Motivation Instrument (MMAMI) or the Music Inventory, that can be used to quantitatively measure these attributes. Early on, the survey mirrored the Music Inventory [44] much more closely, sometimes incorporating exact items from the instrument. However, over time, this survey evolved to better answer specific research questions. This evolution is parallel to the evolution and growth of the course.

Surveys were administered in-class as a non-optional activity. In the early semesters, the survey was given at three points in the course (near the beginning, near the middle, and at the end). However, during the third and fourth offerings (Fall 2015 and Spring 2016), the survey was only administered at the beginning and the end. This change was reversed in the latest offering (Fall 2016) so that finer-grained data could be gathered. Students received classwork credit for participating in the survey (regardless of what answers they

Figure 4.7: Students Intent to Continue Computing vs. Their Self-reported Motivation



gave). Although students were required to complete the survey, the first question of the survey asked if they consented for their responses to be used for research purposes. In order to match up survey responses with students' course outcomes, students were required to include their email address – however, this information was stripped before analysis so that students would be anonymized. To further protect students, survey data was not analyzed until after final grades were assigned.

All of the data before the Spring 2016 semester should be considered preliminary results, and was largely used to guide development of the course and the survey. For example, data (shown in Figure 4.7) gathered in Spring 2015 suggests that students' sense of the usefulness of the material is more closely correlated with their intent to continue in computing than their interest in the material (N=35). The data on which that figure is based comes from a survey more strongly based off MMAMI, which means that it only asks about motivation at the course level (referring generically to "coursework"), not at the course component level (as in, referring to specific course components). The r^2 between each MUSIC component and the students' reported intent to continue was calculated using linear regression, and are shown in the figure in the top-left corner of each graph. Usefulness had the highest coefficient of determination at .63, followed by Interest at 0.48; both of these were much higher than eMpowerment, Caring, and Success, which were all below .3.

The current version of the survey was created in Fall 2015, and has been used since then. This survey uses more in-depth questions to identify students' intent to continue, uses a more sophisticated series of questions to isolate student motivation with respect to course components, and uses a Likert scale with more points. The complete text of the current survey is included in Appendix A. However, to summarize the survey, there are three main parts.

Motivation × Course Components Questions

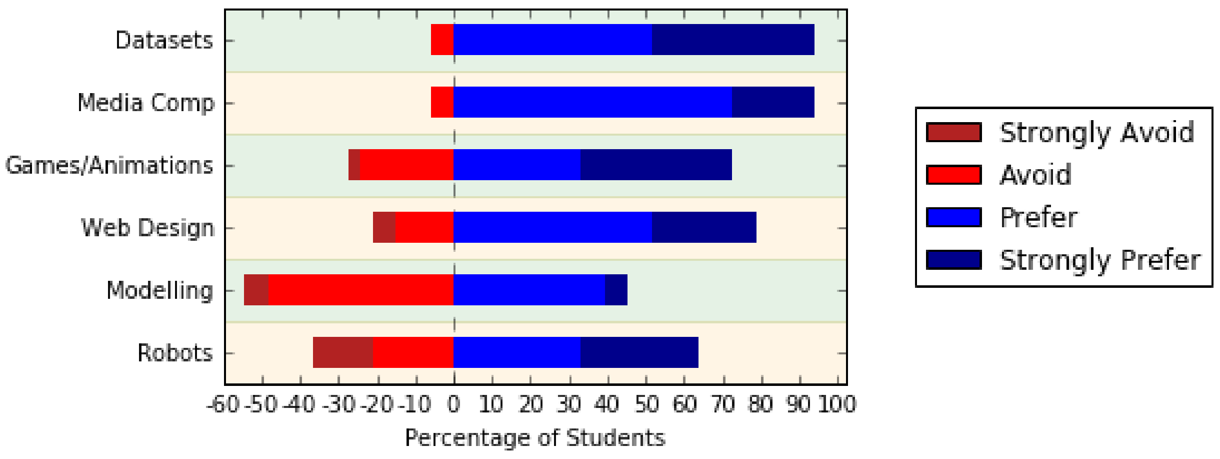
The first part of the survey is five series of five 7-point Likert statements (Strongly Disagree to Strongly Agree). Each series relates to a different part of the MUSIC model ("I believe it will be interesting to..."), and each Likert statement relates to a different part of the course:

- "... learn to write computer programs" - course content related to algorithms.
- "... learn to work with abstraction" - course content related to abstraction.
- "... learn about the social impacts of computing" - course content related to social ethics.
- "... work with real-world data related to my major" - course context related to Data Science.
- "... work with my cohort" - course scaffold related to the collaborative nature of the course.

So, for example, a student would be asked to rate their agreement with a statement such as, "During the course, I felt that it was useful to my career to learn how to program" or "During the course, I felt it was interesting to learn about social ethics of computing". These five elements were chosen as some of the most clear and important elements of the course that would be visible and understandable to a student at both the beginning and ending of the course. Although there are other course elements that would be desirable to gather data on (e.g., the block-based environment), there is a limited number of questions that we can ask students. These questions get at the three main course content objectives, the course context, and one of the most visible scaffolds available to students. As the cross-product of the five motivational components and the five course components, this results in a total of 25 questions.

The 7-point Likert scale was chosen to increase the continuity of the data. Technically, data collected through Likert scales must be treated as ordinal or nominal data, rather than continuous. However, by using a large number of options, along with a sufficiently large sample size, it becomes more appropriate to use parametric statistical tests [58]. Many of the analyses favor nonparametric tests, but in some cases parametric tests and continuous measures (e.g., the mean instead of the median) are used instead.

Figure 4.8: Student Preference for Various Possible Introductory Contexts (Spring 2015)



Course Outcomes Questions

The second part of the survey instrument asks three sets of questions related to student course outcomes. These questions are also 7-point Likert scales. Unfortunately, the last two sets of questions (preference for course contexts and overall rating) were not included in the Spring 2016 survey, and so does not present any baseline. In the Spring 2016 version of the instrument, students were asked a single question: whether they intended to continue to learn more computing (either formally or informally), recommend the class to others, or to apply their new knowledge in their career. In the Fall 2016 version, this first question was split into three separate questions, to better refine student responses. The 7-point Likert scale offers students a choice from "Strongly Agree" to "Strongly Disagree".

The next quantitative question introduced in the Fall 2016 version asks students about their preference ("Strongly Prefer" to "Strongly Avoid") for various introductory course contexts. These contexts were chosen based on alternative ways that the course could have been taught: working with data sets related to their major (Data Science), working with media (Media Comp), making games and animations (Game Dev), making websites (Web Dev), making scientific models (Modelling), controlling robots (Robots), and making phone apps (Mobile Dev). Figure 4.8 shows the preliminary results from an almost identical survey question in the Spring 2015 version of the instrument, albeit on a 4-point Likert scale instead of a 7-point scales. The small sample size makes this data unsuitable for more summative analysis, but was useful at the time for verifying that the course's trajectory was on a good track.

The last set of quantitative questions asked students to give an overall rating for the course and their instructor. Although I was not particularly interested in students' assessment of the instructor, student satisfaction with the course is a relevant course outcome. These two questions are rated on a 7-point Likert scale from “Terrible” to “Excellent”.

Free Response Questions

The third part of the survey instrument has five open-ended qualitative questions relating to the components of the MUSIC model, phrased to ask for particularly extreme examples of motivating and demotivating aspects. For example, the first question is “So far, what parts of the course seem particularly interesting or boring to you?” While the first half of the survey provided structured data about the components of the course, this section provided data about the “stand-out” parts. The data collected was analyzed using a grounded theory qualitative coding method to establish recurring themes in the course components – for instance, a large percentage of students reported that the block-based environment made them feel particularly successful.

4.4.2 Assessment and Other Course Data

A number of course outcomes are based on assessments and activities completed in the course. In this subsection, the general grading schema of the course is described, along with the individual final assessments.

Course Performance

The final course grade was calculated based on a combination of students’ completion of classwork, homework, and reading quizzes, their attendance in the course, and their project scores. Classwork and homework, representing the largest time span and quantity of work, make up the largest proportion of the final grade, at 60%. Attendance and reading quizzes make up comparatively less, at 10% each. This leaves 20% total for all of the projects.

Classwork and homework that were not gradeable by automatic means were graded manually by the teaching assistants. Graded assignments were given individualized feedback, and students were strongly encouraged to resubmit corrected work without penalty. This represents a mastery-based learning approach, which was hoped would encourage students to learn the material more closely than they would otherwise. Reading quizzes were structured as automatically verifiable multiple choice questions so that they did not need manual grading.

Daily attendance was recorded using a Canvas plugin by the Undergraduate Teaching Assistants on a daily basis. The attendance score was calculated as a ratio of the number of days the student attended class and the number of days in the semester (typically 28 days). If a student had an excused absence (e.g., because of illness), the day was subtracted from the number of days in the semester for their final attendance score.

The course used a two-tier system for managing late assignments: due dates and lock dates. Reading quizzes are due before a class starts, classwork is due at the end of class, and homework was due before the start of the next class (projects have variable due dates depending on the project). However, submissions for a module become unavailable at the lock date, one week after the module ends. Students are not penalized for submissions past the due date, but are strongly urged and frequently reminded to follow the due dates as closely as possible. When a student fails to submit work by the lock date, they receive a zero on the assignment. Late submissions are recorded automatically within Canvas and used to calculate the frequency of students' lateness.

Final Project Performance

As previously described, the final project for the course is a month-long assignment where students use a dataset typically related to their career interests to conduct a computational analysis and answer questions. Once they have developed answers to their questions, students create a 5 minute video presentation (e.g., by using PowerPoint). This presentation is assessed using an 8-item rubric, which students are encouraged to reference while they are developing the presentation. The components of the rubric are as follows:

Abstraction What is the relationship between a real-world entity and the data?

Data Structure How is the data for the project organized? What fields in the data are particularly relevant to the stakeholder?

Questions What questions were explored?

Visualizations How did students graphically present the results of their analysis?

Answers What were the answers to the questions? How does the student interpret them?

Limitations What are limitations of the data provided, and the kinds of analyses that could be performed?

Social Impacts Who is interested in the results of the analysis, and who is affected? What potential concerns or problems can arise with regards to the analyses' impact?

Communication Were the slides well constructed? Did the student convey them clearly?

The complete text of the project rubric is available in Appendix A.3. Appendix A.2 gives an example project created by the instructor, shown to students.

4.5 Data Analysis

In Section 4.3, I listed a set of seven sub-questions, divided into two categories, that would be used to answer the last overarching research question of my dissertation: What is the impact of a Data Science context, as opposed to other course components, on student motivation and course outcomes? Each subsection in this section presents data and associated analyses that can answer one of these sub-questions. As previously described, the data for this research project was collected over five semesters, but primarily only the latest and largest course offering will be analyzed to answer these questions – the constantly evolving nature of the course makes it tenuous to compare most results across semesters. Table 4.2 gives an overview of the demographic data for the five iterations of the course captured so far. In an ideal experimental setting, one would find balanced representation across majors, years, and genders, and the sub-groups would be both large and equally sized. Although the demographics are not quite ideal (the course makeup is dominated by women and freshmen), the large sample size still makes this the most compelling population to review results for.

4.5.1 Effective Course Components for Motivation

What course components are particularly effective or not effective at providing motivation?

The five major components of the course provide varying opportunities throughout the semester to foster motivation in the five different aspects of the MUSIC model. Figures 4.9, 4.10, 4.11, 4.12, and 4.13 show (respectively) students' interest, perception of usefulness, success expectancies, perception of instructors' caring, and sense of empowerment across course components. The five groups of bars correspond to a major course component: learning to Program, learning to work with Abstraction, learning about the social impacts of computing (Ethics), working with real-world Data related to their major, and working with their Cohort. Each group has three rows, one for each time the survey question was administered. To answer this subquestion, consider the last bar in each group, representing their self-reported motivation at the end of the semester.

To supplement these quantitative results, the free response questions were qualitatively coded based on positive and negative versions for each of the MUSIC components. Table 4.3 shows the positive codes and Table 4.4 shows the negative codes. The three columns divide the results by the three times the survey was administered over the semester. The codes are clustered by the positive MUSIC elements (Easy, Empowering, Useful, Interesting, and Cared For) and the negative MUSIC elements (Hard, Limiting, Useless, Boring, and Uncared For). Each individual code is given along with the frequency of that code in parentheses. Only codes that appeared five or more times were included. Note that there were no codes that appeared five or more times for the "Uncared For" tag, across the entire semester. Again, for this question, we only consider the results of the last column (the last survey administered).

Table 4.2: Summarizations of Demographic Data in the Course over Time

(a) Gender of Students in the Computational Thinking Course

	Fall 2014	Spring 2015	Fall 2015	Spring 2016	Fall 2016	Total
Female (column %)	7 (27%)	21 (54%)	11 (35%)	24 (48%)	62 (61%)	125 (50%)
Male (column %)	19 (73%)	18 (46%)	20 (65%)	26 (52%)	40 (39%)	123 (50%)
Total (row %)	26 (10%)	39 (16%)	31 (12%)	50 (20%)	102 (41%)	248 (100%)

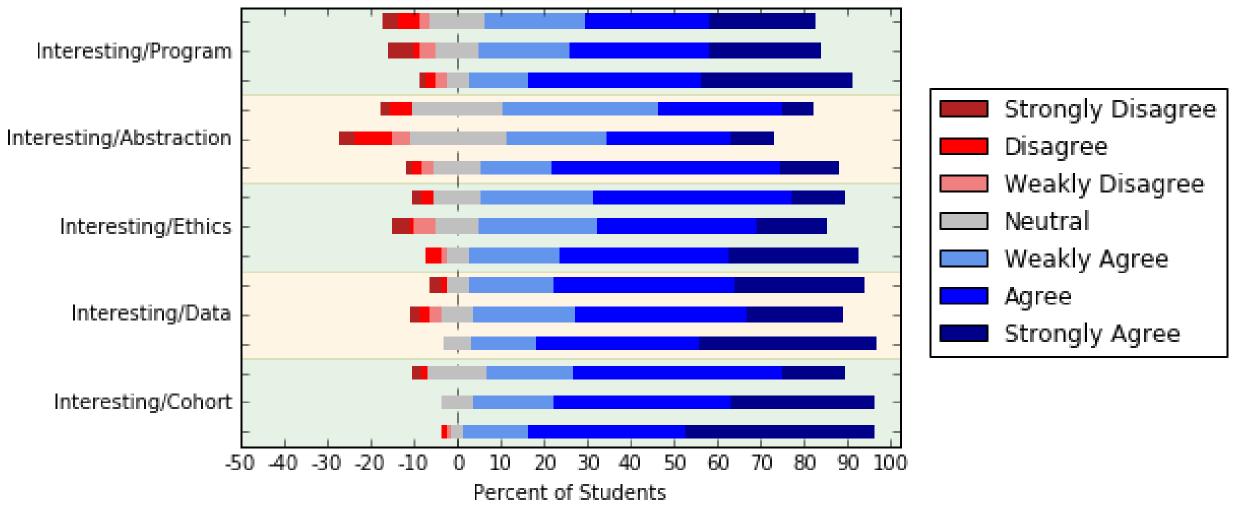
(b) Year of Students in the Computational Thinking Course

	Fall 2014	Spring 2015	Fall 2015	Spring 2016	Fall 2016	Total
Freshmen (col%)	3 (12%)	5 (13%)	4 (13%)	9 (18%)	50 (49%)	71 (29%)
Sophomore (col%)	7 (27%)	12 (31%)	5 (13%)	18 (36%)	19 (19%)	61 (25%)
Junior (col%)	8 (31%)	11 (28%)	11 (35%)	16 (32%)	18 (18%)	63 (25%)
Senior+ (col%)	8 (31%)	10 (26%)	11 (35%)	7 (14%)	15 (15%)	51 (21%)
Total (row%)	26 (10%)	39 (16%)	31 (13%)	50 (20%)	102 (41%)	248 (100%)

(c) College of Students in the Computational Thinking Course

	Fall 2014	Spring 2015	Fall 2015	Spring 2016	Fall 2016	Total
Architecture (column %)	0 (0%)	7 (18%)	5 (16%)	21 (42%)	15 (15%)	48 (19%)
Liberal Arts (column %)	9 (35%)	24 (62%)	15 (48%)	24 (48%)	69 (68%)	141 (57%)
University Studies (column %)	1 (4%)	0 (0%)	2 (6%)	2 (4%)	2 (2%)	7 (3%)
Agriculture (column %)	0 (0%)	1 (3%)	0 (0%)	1 (2%)	1 (1%)	3 (1%)
Science (column %)	8 (31%)	5 (13%)	3 (10%)	1 (2%)	11 (11%)	28 (11%)
Business (column %)	1 (4%)	0 (0%)	3 (10%)	1 (2%)	1 (1%)	6 (2%)
Engineering (column %)	7 (27%)	2 (5%)	3 (10%)	0 (0%)	3 (3%)	15 (6%)
Total (row %)	26 (10%)	39 (16%)	31 (13%)	50 (20%)	102 (41%)	248 (100%)

Figure 4.9: Students' Interest with Regards to Course Components over 3 Surveys



Interest

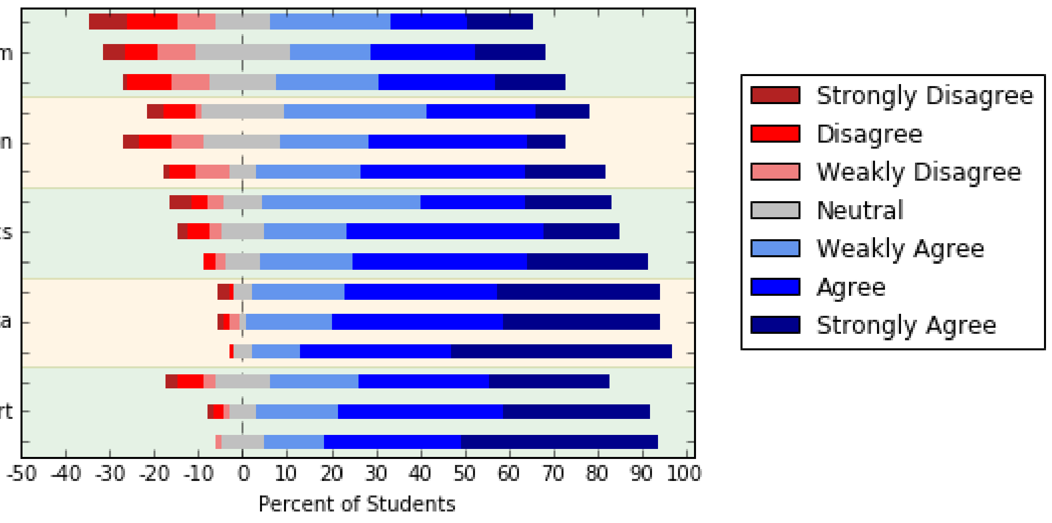
Figure 4.9 indicates that students were generally interested in all five course components. The Data Science context, however, was particularly interesting to students, with no negative results. Working within a cohort was also viewed positively. Interest in learning the course content (programming, abstraction, and ethics) had more negative results, but not a substantial number (less than 10%).

Many students thought that programming was interesting, with no corresponding vocal minority saying that it was boring. However, abstraction was identified as boring. BlockPy was considered boring by a small group of students, but more than twice as many considered it interesting to work with. The prevalence of the BlockPy tag across the qualitative results make it a candidate for inclusion in future versions of the major course components – students clearly identified BlockPy as a major course component.

Usefulness

Figure 4.10 shows a similar result for Usefulness as compared to Interest. The Data Science context and working within the cohort were once again almost completely positive. The course content showed more stratification. Students reported relatively high Usefulness for learning about Ethics in computing; in fact, the number of positive responses was actually comparable to the Usefulness of working within a cohort. However, the number of positive responses for the Usefulness of the Abstraction material was lower (by almost 10%), and Programming material was even lower still (by more than 20%). This validates the core value of the Data Science context: even if students are not particularly motivated by the

Figure 4.10: Students' Perception of Usefulness with Regards to Course Components over 3 Surveys



course content, the course context can provide a sense of Usefulness.

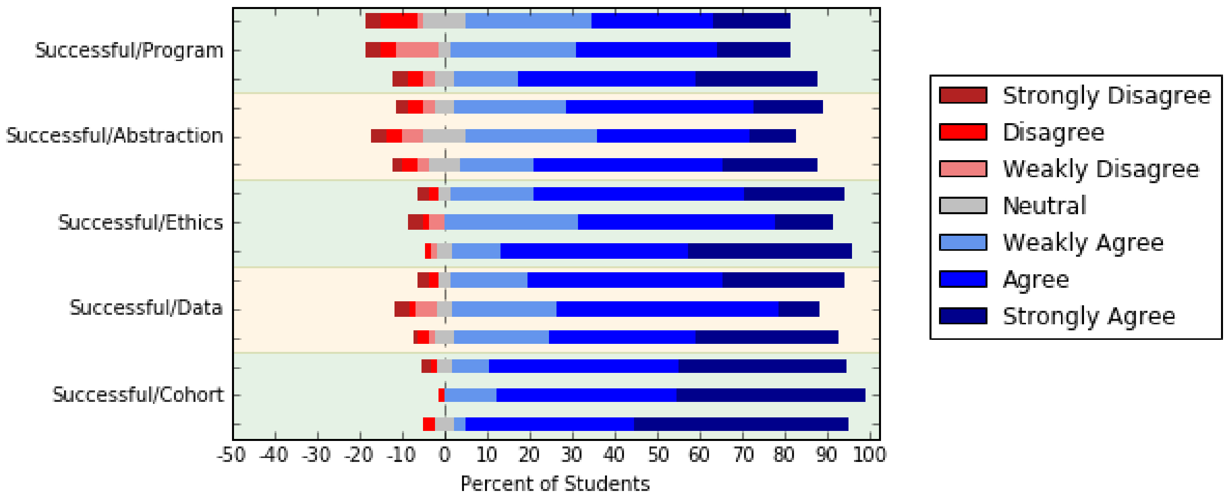
Although many students suggested the Usefulness of learning to program, there was a vocal minority that felt it was particularly useless to their long-term career goals. The rest of the qualitative results for the Usefulness category reinforce the value of the Social Impacts material, the Data Science context, and working within cohorts. Even Abstraction makes an appearance in the list of popular codes. The high frequency of positive Usefulness tags gives further credence to the argument that the course is designed as a Useful experience for learners.

Success

Figure 4.11 shows similar levels of self-efficacy across all the course components. Students seemed to find themselves more successful with regards to the Ethics material and working within a cohort, but relatively less successful with Programming, Abstractions, and working with Real-world Data. This matches my own understanding of the difficulty of the course components, so the results are unsurprising. The encouraging takeaway from this figure is that most results are positive, indicating that most students did not consider the material too challenging for themselves.

Unlike the other negative qualitative tags, the Hard group had considerably more items, both compared to other negative tags and to the positive Success tag. However, it is worth considering that "Hard" and "Easy" are not necessarily negative and positive in the way that tags like "boring" and "interesting" are. Ideally, a course should be in a zone of "just

Figure 4.11: Students' Perception of Their Success with Regards to Course Components over 3 Surveys

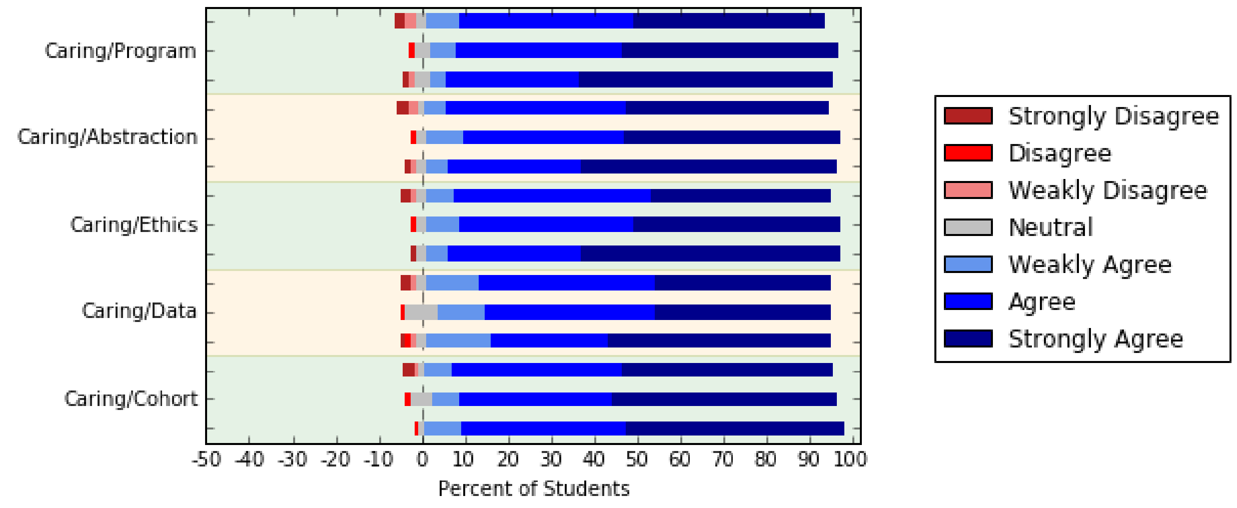


hard enough", rather than being particularly hard or easy. Still, it is useful to see what students are vocal about, although the results are somewhat polarizing. Roughly 16.6% of the students suggested that programming was hard, and about half as many as that suggested that programming is easy. Meanwhile, 6.8% students suggested that the social impacts were difficult while 4.9% suggested that the social impacts were easy. Considering the relative difficulty of Programming course content versus Social Impacts, the skew is not particularly surprising. Curiously, despite receiving the highest percentage of negative quantitative results, Abstraction did not appear frequently in the free response section. Also missing was working with data, which suggests that the technological scaffolding for the Data Science context was appropriate.

Caring

Figure 4.12 presents an interesting result. Almost universally across course components, students felt that instructors cared that they learned that component. Similarly, there were no high frequency codes for the "Uncared For" tag. One interpertation of this is that the instructors did an excellent job at expressing their caring for the students' learning. Of course, as a question that rates the instructor, students may be influenced to give biased answers. The major advantage of including this question in the analysis is to attempt to control for students' appreciation or dislike of the instructor. The idea being, if the student had particular feelings about the instructor, they would express it in this question rather than in one of the other four MUSIC components. Ultimately, the results shown have little to tell about students' motivation with regards to Data Science as a context, but at least suggest that the instructors themselves were not a weak element of the course.

Figure 4.12: Students' Perception of Instructors' Care of their Learning with Regards to Course Components over 3 Surveys



Empowerment

Figure 4.13 continues the pattern established in the Interest and Usefulness questions. The Data Science context provided the most sense of empowerment, followed by the Ethics material and working within a cohort. Programming provided less empowerment, but more so than Abstraction. It is important to recognize that the differences between the Abstraction and Data components are only about 10% in total. In general, this shows that the course empowers the students, even if there are slight differences between the components.

Students point to the variety of datasets available as one of the most empowering features of the course. However, a vocal minority of students wanted more datasets, or felt that the datasets they had were insufficient; unfortunately, students did not make recommendations for new data sources or identify gaps in the existing dataset selection. There is a continuing struggle to provide a sufficient number of datasets, and it is difficult to determine when a saturation point has been reached. Regardless, many students responded quite favorably to their freedom in selecting questions and visualizations when completing their projects. Although more work is needed to expand the CORGIS collection, these results are encouraging.

4.5.2 Trends in Student Motivation

How does students' motivation change over the course of the semester?

The previous subsection analyzed the quantitative and qualitative results from the last survey, in order to highlight the importance of students' motivation at the end of the course.

Table 4.3: Frequency of Positive Codes across MUSIC Components and Surveys

Category	First	Middle	Last
Easy	everything (11) abstraction (6) homework (5)	blockpy (11) programming (10)	blockpy (17) programming (8) social implications (5)
Empowering	data (18) nano (12) multiple solutions (10) everything (9) maze (8) deadlines (7)	data (17) everything (5) projects (5)	data (20) projects (14) multiple solutions (5)
Useful	data (17) cohort (10) abstraction (9) computers (9) new thinking (9) problem solving (7) visualization (6) programming (6) social implications (6) algorithms (5)	programming (23) data (12) social implications (12) python (7) new thinking (7) abstraction (6) problem solving (6) cohort (6)	programming (28) social implications (15) data (13) new thinking (9) cohort (8) abstraction (7) everything (7)
Interesting	maze (27) abstraction (12) programming (12) real world (9) algorithms (7) social implications (6) new thinking (6) visualization (5)	programming (23) python (16) blockpy (10) data (8) real world (6) social implications (5)	programming (25) blockpy (12) data (9) spyder (8) social implications (8) real world (6) python (6), projects (6) majors (5), cohort (5)
Cared for	learning (20) TA (17) answers (16) enthusiasm (10) feedback (5) attentive (5)	helpful (33) learning (23) TA (14) answers (10) office hours (9) instructors (8) communication (8) walking (7) questions (7) enthusiasm (6)	helpful (29) instructors (24) TA (23) answers (15) learning (14) office hours (14) communication (7) enthusiasm (6) walking (5)

Figure 4.13: Students' Perception of their Empowerment with Regards to Course Components over 3 Surveys

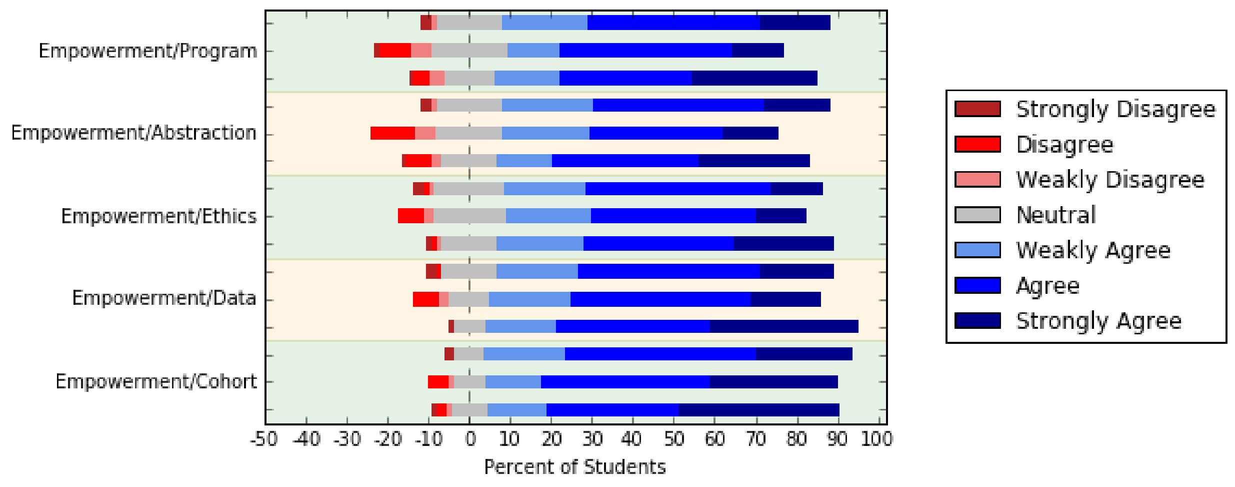


Table 4.4: Frequency of Negative Codes across MUSIC Components and Surveys

Category	First	Middle	Last
Hard	algorithms (9) nothing (7) maze 10 (7) maze (7) programming (6) precision (5)	programming (17) spyder (11) python (6) transition (5) dictionaries (5) visualization (5)	programming (17) spyder (13) social implications (7) python (7)
Limiting	data (5)		data (6)
Useless	nothing (10) programming (8)	programming (17) nothing (5)	programming (16)
Boring	lecture (5)	abstraction (6) social implications (5) visualization (5)	abstraction (6) blockpy (5)
Uncared For			

However, student motivation is not a fixed trait, but one that develops over the course of the semester in response to the learning experience. In this subsection, I note trends and patterns in the data related to students' motivation. It is important to keep in mind the placement of the three surveys. The first occurs shortly after the start of Module 2, on Day 7 (right before they begin working with BlockPy). The second is administered at the end of the class before the Mini-project begins (right after they have finished working in Spyder on the practice problems). The third is administered on the final day of the course (after final projects, and all other course work, have been submitted). Most of the motivation/course components pairs followed one of two patterns: either they constantly increased, or they followed a $\vee$-shaped curve (indicating a decrease in the middle of the semester, and a jump at the end). The frequency of the latter pattern may suggest a more general pattern of fatigue during the middle of the semester. Figures 4.9, 4.10, 4.11, 4.12, and 4.13 show these trends in each pairing of course components and motivational components.

Interest in learning about abstraction, learning about ethics, and working with real-world data all follow the aforementioned $\vee$ shape: they increase from the start of the semester to the end, but with a sharp decrease in the middle. Interest in programming follows a more monotonic pattern, although there is little increase from the start to the middle; although throughout the qualitative data, there is a recurring presence of students strongly interested in programming. It is difficult to infer the trend from the start to the middle: one possibility is that students are generally less motivated during the middle phase of the course, where they are completing daily programming problems, and forget the other course content and context. It is possible that the Data Science context is even a detrimental force, as students become bored with the repeated problems dealing with the same few data sources (weather, earthquakes, classic books, and theater data). The qualitative data seems to reinforce this conclusion, with a number of tags related to this repetition. It is curious that none of this fatigue extends to their interest in programming, but it is worth noting that their interest was already relatively lower than most of the other course components. Although the first pair of surveys are difficult to explain, the second pair show a more positive result. Given the timing of the surveys, the sharp increases suggest a powerful effect from the open-ended final project, as students regain interest in the context and content.

Students' sense of Usefulness increased for each component consistently across the semester, with the exception of learning about Abstraction. Much like Interest in learning about Abstraction, that element followed the $\vee$ shape, suggesting once again that the middle section of the course caused student motivation towards Abstraction to drop off. The qualitative data shows an even more interesting trend: at the beginning of the course, students more vocally ascribed Usefulness to working with data and learning to work in a cohort. However, by the middle of the course (and repeated at the end), students became more vocal about the Usefulness of learning programming and Social Implications. Although working with Data stayed frequent, working in a cohort slipped down dramatically. Although students perceived that the context has tremendous value in terms of Usefulness, it seems that many students also percieve an innate usefulness to programming.

Students' self-efficacy follows a pattern similar to students' Interest. Abstraction, Social Impacts, and Data follow a $\vee$ shape, while Programming stays neutral before increasing. Working within a cohort follows a $\wedge$ shape, decreasing overall during the course. The quantitative data seems to be somewhat at odds with the qualitative data, which suggests that programming was among the hardest elements of the course, even to the end. To explain this, it is helpful to consider the nature of the survey questions: the quantitative data asks if students felt they were successful, while the free response question asks what parts of the course were particularly hard or easy. One interpretation of these results, therefore, is that even though the programming may have been difficult, students' felt they were up to the challenge. It is interesting to note the positive impact of the projects on student self-efficacy across the contexts and content components.

Empowerment follows the most distinct $\vee$ shaped trend during the semester, in every component. The trough of this trend is most likely because the middle survey comes when students have just spent the past few weeks completing many small programming problems, fairly constrained assignments compared to the freedom of the final project. This design for the course was intentional, to give students a consistent and finely-tuned base of knowledge to prepare them for the open-ended, self-directed final project. Still, the context of working with real-world data, particularly in the projects, came across in both the middle of the semester and the end within the free response data. It is telling that students only suggested "multiple solutions" (referring to how many programming problems can be solved in different ways) during the beginning and end of the course; this reflects the fairly narrow bounds of the middle section of the curriculum. Somewhat discouraging is that the overall trend of empowerment, with respect to learning to program and abstraction, is negative; this is somewhat alleviated by the overall positive trend of the data context.

Students' sense of caring had high values consistently across the entire semester. Almost from the beginning, students pointed out a number of ways that instructors showed they were interested in their learning, were helpful in guiding them, or provided steps to the answers. Although this result is pleasant, it does not say much about the value of the Data Science context.

4.5.3 Motivation across Genders

How does motivation compare between genders?

Gender remains an important issue in introductory computing, particularly because of the low percentage of women entering and staying in computing [27]. Therefore, it is both useful and interesting to identify differences in motivational reactions between genders. Although other demographic indicators were collected, the relatively small population (roughly 100 students) makes it difficult to compare other subgroups such as major, college, or even year. Even dividing the population in half affects assumptions needed to use parametric tests (that the sample size is sufficient to invoke the central-limit theorem). Therefore, I use the non-parametric Kruskal-Wallis test for each motivation/course component pair, for the first and last surveys, between each gender. A significance level of $\alpha < .05$ was chosen to determine if results were significant. Table 4.5 shows the survey items where there was a significant difference between male and female responses. Each row of the table indicates the survey (first or last), motivational component, and course component that was different; the last column indicates the absolute difference in means between the two genders, and the penultimate column indicates which gender was higher. The absolute mean difference can be roughly taken as the differences between individual levels of agreement on the Likert scale: a difference of 1, for example, represents the change from "Agree" to "Strongly Agree".

Table 4.5: Significant Results of Kruskal-Wallis Test for Motivational and Course Components across Multiple Surveys between Genders

Survey	Motivational Component	Course Component	Higher	Difference in Means
First	Caring	Abstraction	Females	0.47
First	Caring	Ethics	Females	0.50
First	Caring	Cohort	Females	0.61
First	Caring	Program	Females	0.62
First	Caring	Data	Females	0.63
First	Successful	Cohort	Females	0.74
First	Usefulness	Data	Females	0.95
Last	Successful	Program	Males	0.87

At the beginning of the semester, women scored significantly higher in a number of categories. Most obviously, the entire group of items related to the Caring motivational component had women scoring roughly half a level of agreement higher. In general, this suggests several possibilities. One possibility is that women were more attentive and open to the instructors' attempts to make students feel cared for. Another is that men were actually less cared for than women and they accurately perceived this (though neither instructor reported anything like this). A third possibility is that women overestimated the amount of caring by their instructors. Women were also more than a half level higher in terms of their self-efficacy to work in a group, suggesting a similar set of possibilities: perhaps that the female students were more attuned to the group work, that the male students were less cared for, or that the women overestimated. Finally, female students felt that it would be more useful to work with real-world data, by almost a full level of agreement. This is consistent with results that suggest female students may be more driven by introductory material that can be connected to solving real-world problems [27]. If so, this can be seen as a major benefit of the Data Science context.

However, over the course of the semester most differences in gender disappeared, leaving only a single item with a significant gender difference (which, coincidentally, was also the only result where the males scored higher). By the end, males were almost a full level of agreement higher than females in terms of their self-efficacy at learning to program. It is impossible to tell if males are over-estimating their skills or women are under-estimating, but it should be noted that there was no significant differences between genders in actual performance of most course activities. It is also worth noting that women scored significantly higher on the final project, according to a One-way ANOVA, by almost a full letter grade (9.36 point difference in means). Although it is spurious to connect students' ability to program too closely to these outcomes, it is still telling that students seem to have gender-based differences in ability to self-evaluate.

4.5.4 Data Science vs. Introductory Contexts

How does Data Science appeal to students compared to other potential introductory course contexts?

Figure 4.14 shows the results of a survey question about students' preference for various potential course contexts. Each of the potential contexts was briefly described to students in a short sentence (e.g., Media Computation was described as "Working with pictures, sounds, movies"), and students were asked how much they would prefer or avoid a course taught in that context. Scientific computing was easily the most poorly rated context, followed by controlling robots and then Mobile App development; these three contexts were significantly ($\alpha < .05$) lower than both Media Computation and Data Science. Although Media Computation was consistently a popular context, the Data Science context was the most preferred choice by the end of the course (although there was no significant difference between the two in a matched-pairs T-test).

This survey question revealed a number of interesting significant differences between genders (non-parametric Kruskal-Wallis test, with $\alpha < .05$). Men rated roughly a full level of preference higher than women for the contexts of Game Design, Controlling Robots, Scientific Modelling, Mobile App Development, and Web Development. These results seem to strongly be in favor Data Science and Media Comptuation as preferred contexts for designing gender-agnostic courses.

There are three major threats to validity from the methodology for answering this question. First, students were given little explanation for the contexts. Although the contexts are described in a sentence, there is hardly time to give concrete examples or go into any serious depth. It is difficult to know what students imagine these contexts to be like, especially given their inexperience with computing in general. Second, students know that the instructors are biased in their own preference, since they are enrolled in an introductory computing course constructed around a Data Science context – the students may simply be telling the instructors what students think the instructors want to hear. Third, it is possible that some students may be biased themselves in favor of a Data Science context, since they may be taking the course specifically for this pre-existing reason.

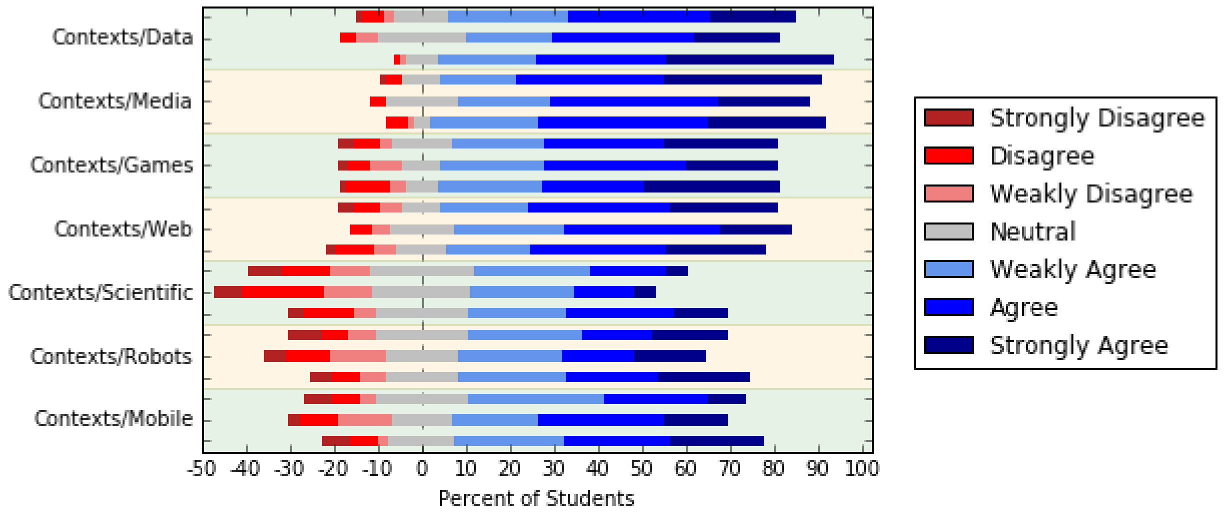


Figure 4.14: Students' Preference for Potential Course Contexts over 3 Surveys

4.5.5 Viability of a Data Science Context

Is a course designed around a Data Science context viable?

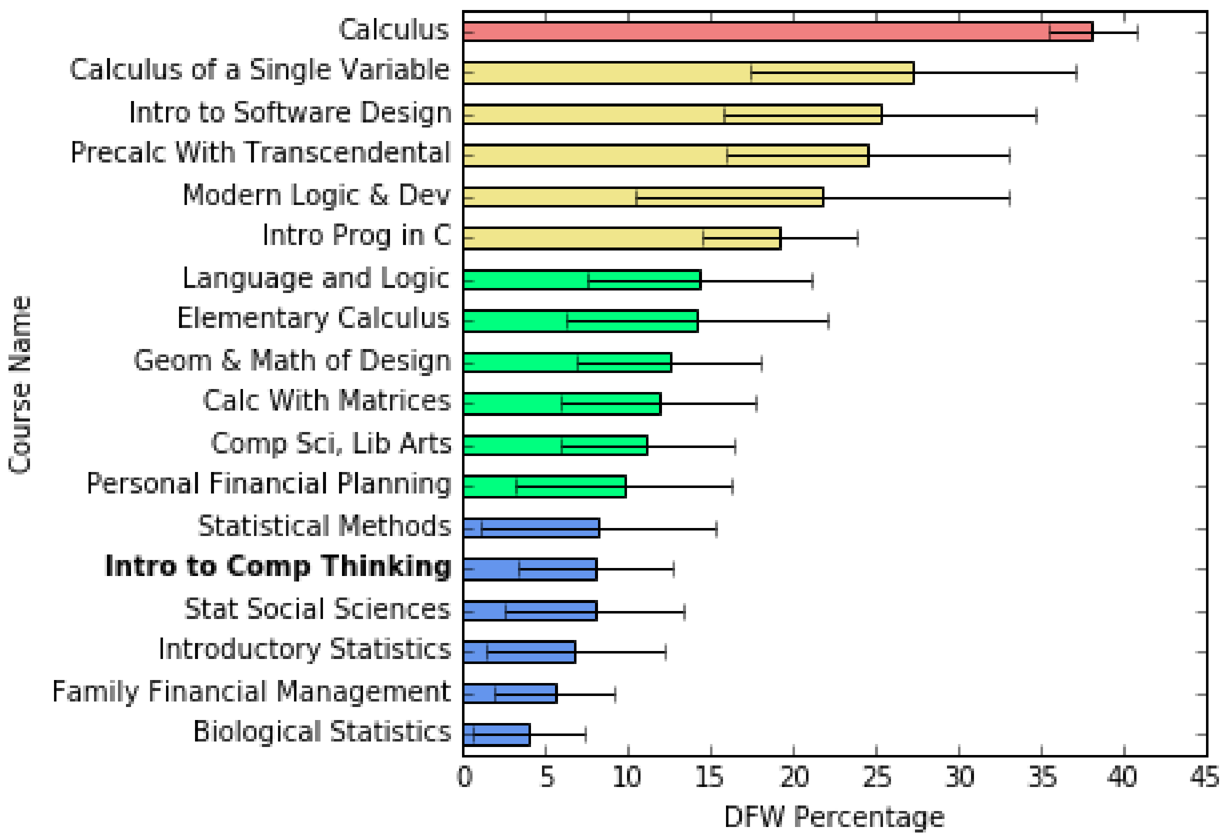
Although academic motivation within a course is important, there is a fundamental question of whether a course structured around a Data Science context can work. In order to answer this question, I loosely follow criteria established by Guzdial [37] when he evaluated his Media Computation curriculum. Specifically, I take his Retention, Gender, Learning and More-Computing hypotheses as different ways of critically analyzing a course. The Plagarism Hypothesis (that there would be fewer reported incidences of cheating) was not evaluated for our course, since we had no expectations or problems related to plagarism.

The Media Computation project collected data over a 10 year period in order to evaluate their hypotheses, so it is still early to compare our Data Science course, which has been around for a quarter of that time. These hypotheses are also being evaluated post hoc, which means that different criteria may have been imagined when designing the Data Science course, compared to the Media Computation course; it is possible that other criteria may be more appropriate. Still, the results should prove to be a barometer of the progress of the course so far.

Retention Hypothesis

The first hypothesis evaluated was the Retention Hypothesis. This hypothesis uses a measure called "DFW" rate - the number of "D" and "F" grades, combined with the number of students who withdrew, divided by the total number of students initially enrolled [37]. Ideally, a course should minimize the DFW rate (without artifically decreasing the courses' difficulty).

Figure 4.15: Percentage of DFW (Ds, Fs, and Withdrawls) across All Area 5 Courses since Fall 2014



Guzdial targeted a failure rate of no more than 15%, and achieved an actual rate of around 12%. These numbers were established to compare the Media Computation course with existing introductory computing courses (which were not contextualized for non-majors).

Virginia Tech collects and publishes data on grade distribution and drop-out rates across every course [84]. This makes it easy to compare the Introduction to Computational Thinking course with courses that fulfill similar requirements. Figure 4.15 shows the grade distributions of courses across the university that also fulfill the Quantitative and Symbolic Reasoning credit (also known as "Area 5" courses). Also included is the traditional introductory computing class for Computer Science majors ("Intro to Software Design"). Our own course is rendered in bold ("Intro to Comp Thinking"). The colors of the bars represent clustering performed via a K-Means algorithm to group courses with similar DFW percentages.

As can be seen, the Computational Thinking course is in the lowest DFW% group, at roughly 8%. The black error bars (indicating the standard deviation) show that the vast majority of the offerings were below the 15% threshold successfully. This compares well with the Media Computation course, and favorably to the traditional introductory computing

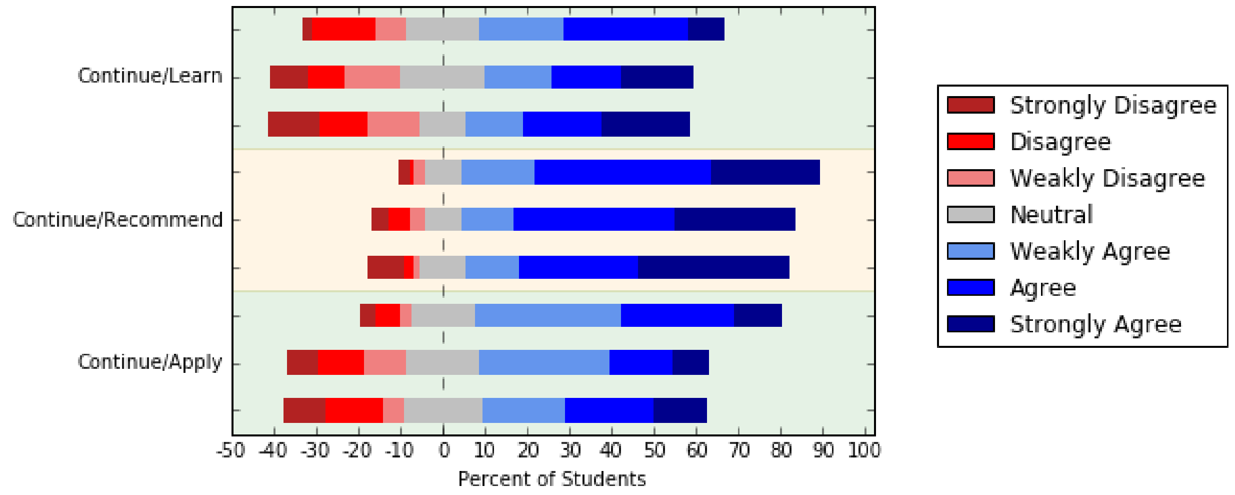


Figure 4.16: Student Response to Intent to Continue Questions over 3x Surveys

course. Interesting to also note is the "Comp Sci, Lib Arts" course, another attempt at an introductory computing course for non-majors, which shares a DFW% more similar to the Media Computation course than our own.

More-Computing Hypothesis

The second hypothesis tested in [37] was the More-Computing Hypothesis, which tested whether students would continue to take computing courses after the Media Comp course. Unfortunately, our Data Science course has no direct successor. Further, the relatively young nature of the course makes it difficult to gather meaningful longitudinal data that speaks to students' continuing to another CS course. However, included in the survey as an engagement outcome is a set of three questions meant to determine whether students intend to continue with computing. Figure 4.16 shows students' agreement with the following three statements:

Continue/Apply "I will directly apply what I have learned in my career."

Continue/Learn "I will try to learn more about computing, either through a course or on my own."

Continue/Recommend "I will recommend this class to others."

Only half the respondents indicated an intent in continuing to learn about computation and apply computation to their long-term careers. However, closer to three-quarters of the respondents were willing to recommend the class to their friends. It is difficult to understand what concrete action this will translate into – will students enroll in more computationally-oriented courses? Will they take advantage of free online courses? Further research is necessary to determine whether students actually follow through on their intent. However, we believe that it is useful to be able to set goal posts for such future research.

The trend in student responses over the semester is consistent across the three separate questions. The number of students intending to continue or recommend decreased from the first survey point to the second. The number of students intending to continue or recommend stayed constant from the middle to the last surveys, but students on both sides became more strongly polarized. That is, students who previously indicated a “weak agreement” with the statement now indicated “agreement”, students that indicated a “weak disagreement” now indicated a “disagreement”, and so on. The project helped solidify students’ intent to continue, while the programming assignments generally decreased their intention.

The different form of our data makes it unsuitable to compare our own course to Media Computation for this hypothesis. It is worth pointing out that Guzdial considered their course to not meet the expectations for the More Computing Hypothesis. Of course, neither of our courses consider the more computing hypothesis a particularly high-value objective [38]. Still, measuring and improving the number of students interested in continuing in computing is an ongoing research direction worth pursuing.

Gender Hypothesis

Gender imbalance in Computer Science continues to be a serious problem, impacting the culture and productivity of the discipline. As a course explicitly for non-majors, the course designers hoped to draw in genders equally. Reviewing the historic data in Table 4.2, the instructors were pleased to find that roughly 50% of the students across the five semesters were women. In fact, the most recent iteration of the course was 61% women. This level of gender parity is close to the ideal, although there is hope in the future to more closely approach an even split. Reviewing the results of the Media Comp course, it can be seen that the Media Computation course achieved similar demographics.

Learning Hypothesis

The last measure, the Learning Hypothesis, suggests that students would achieve some level of proficiency at the end of the semester. Guzdial notes the many difficulties in meaningfully comparing knowledge across introductory courses [80]. Efforts to create an accurate instrument have struggled, and there is still disagreement about what exactly should be measured and whether the quantity measured is appropriate [51].

It is beyond the scope of this dissertation to introduce a conclusive instrument for measuring student performance across introductory courses. Instead, I present the formal assessment of the course to attempt to measure student performance at the end of the semester. These results are reported here in order to at least justify that some level of learning occurred

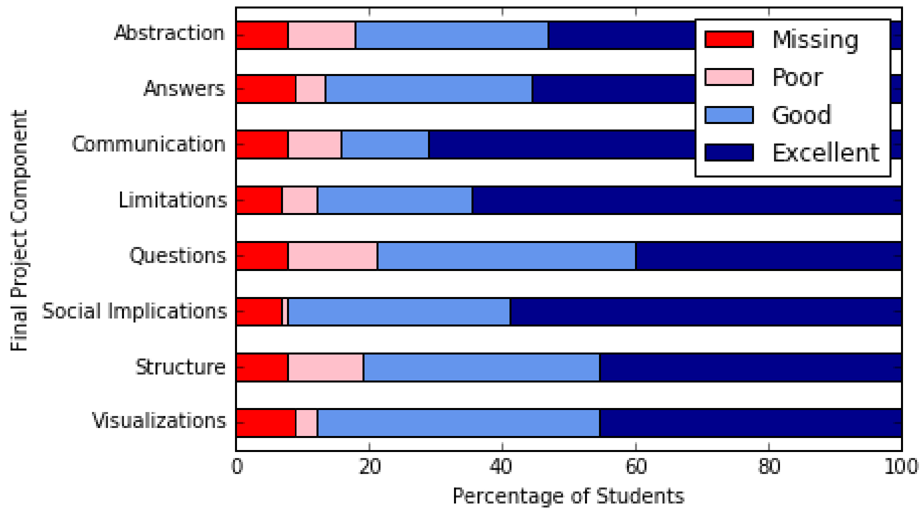


Figure 4.17: Final Project Grade Distribution (F16)

within the course. Earlier, we established three major course learning objectives:

1. Students should be able to create algorithms that manipulate data using a practical programming language (Algorithms).
2. Students should be able to navigate and describe data (Abstractions).
3. Students should be able to discuss social impacts of their computations (Social Impacts).

The major form of assessment for these objectives was the Final Project, which was graded using an 8-item rubric. Figure 4.17 shows the results for each rubric element, which are quite positive. The majority of students (85%) received an Excellent or Good rating on each rubric element.

The Algorithms learning objective relates to the Questions, Answers, and Visualizations components of the rubric. Students performed worst on the Questions component out of all of the components, where they are tasked with creating questions that can be solved through computational analysis of their data. Although the vast majority of students had no problems creating questions, many students failed to properly justify their questions and argue for their importance using evidence. Fortunately, most students were able to competently create visualizations and use them to answer the questions they generated. Most students lost points because they used an insufficient breadth of visualizations, insufficiently labeled their diagrams, or interpreted the results of their visualization incorrectly.

The Abstractions learning objectives relates to the Abstraction, Limitations, and Structure components of the rubric. Although the results are also positive here, some students did struggle with explaining the finer details of their data and justifying their relevance to their project. Although they could often list some relevant fields of their data that they used in their projects, they sometimes had difficulty connecting those abstracted fields to real-world entities and ideas. In terms of the structure of their data, students were easily able to identify the types of individual fields, but often gave confusing explanations for the lists and dictionaries those fields were composed in. Students' limitations were sometimes too generic rather than specific to their data – for example, students would simply report that their project was limited by the quantity or detail of the data, rather than a unique characteristic.

The Social Impacts objective related directly to the Social Implications component of the rubric. Students were capable of identifying and justifying stakeholders who would find their results relevant, but many students failed to explain conflicts between those stakeholders using the ethical frameworks they had previously learned.

Without externally validated instruments for measuring computational knowledge, it is difficult to conclude that learning did or did not occur within our course. Certainly, students performed at some arbitrary level of proficiency. The instructors were satisfied by this level, although there is hope to continue improving these outcomes. Of course, given that many of these students were non-STEM majors with no prior computing background, the fact that many of these students completed the project successfully is compelling, albeit informal, evidence of learning. Ultimately, I leave it to the reader to determine for themselves if they are satisfied with these measures.

4.5.6 Correlating Motivation and Course Outcomes

What aspects of motivation at the beginning and end of the semester correlate with course outcomes?

In Sections 4.4.1 and 4.4.2, I established a number of course outcomes that I hoped to connect to the motivational data. Each of these outcomes was compared to each motivation/course component pair to find their Pearson correlation, which should indicate the association between the two measures. A significance level of $\alpha < .05$ was chosen, and only significant results are included in this section.

Table 4.6 shows the correlation for data collected during the Spring 2016 semester in a pair of tables. The first table represents the data for one course outcome: students' intent to continue (which in that semester, was represented by only one survey item). Each column represents a Motivational element of the MUSIC model (eMpowerment, Usefulness, etc.) and each paired row represents a course component. Each paired row consists of the result from either the first survey or the last survey. The intersection of a row and column represents the correlation of that course/motivation component with the Intent to Continue course outcome for one of the two surveys. This format was chosen due to the high number of significant correlations related to this particular course outcome, and to highlight a particular trend in the types of motivation/course components that have a high correlation with this outcome. In particular, it can be seen that the course content (Programs, Abstraction, and Ethics) are the only course components with any correlation: the Data Science context does not have any statistically significant association with students' intent to continue. The highest correlations are from the students' sense of Usefulness of the material, followed by their Interest and Self-efficacy (which is predicted by Guzdial in reviewing Eccles work on educational decision-making [37]). This suggests that, when attempting to influence students' intent to continue with computing, the context is less important than the content. This is contrary to original hypotheses that the course context would have significant connections to students' intent to continue learning.

Table 4.6: Correlation between Course Outcomes vs. MUSIC/Course Components in the Spring 2016 Semester

(a) Pearson's Correlation between Students' Intent to Continue vs. MUSIC/Course Components in the Spring 2016 Semester across Two Surveys (First and Last)

Component	Survey	M	U	S	I	C
Abstraction	Last	**0.455***	**0.690***	**0.706***	**0.504***	0.176
	First	0.25	**0.537***	**0.582***	**0.463***	-0.209
Cohort	Last	0.059	0.087	0.002	0.044	0.057
	First	-0.157	0.236	0.08	0.228	0
Data	Last	0.105	0.293	0.276	0.276	0.202
	First	-0.011	0.023	0.031	-0.055	-0.307
Ethics	Last	0.284	**0.465***	**0.465***	0.298	0.165
	First	0.05	0.21	0.162	0.015	-0.155
Programs	Last	**0.441***	**0.828***	**0.701***	**0.648***	0.052
	First	0.224	**0.497***	**0.466***	**0.534***	0.009

(b) Pearson's Correlation between Various Course Outcomes vs. MUSIC/Course Components in the Spring 2016 Semester across Two Surveys (First and Last)

Course Outcome	Motivation	Course Component	Survey	Correlation
Final Project	Caring	Data	First	-.383
Quizzes Grade	Successful	Data	First	.452
Final Grade	Successful	Programs	Last	.401
Final Project	Successful	Programs	Last	.458
Final Project	Useful	Programs	Last	.419
Final Project	Successful	Abstraction	Last	.345
Attendance	Successful	Abstraction	Last	.335
Attendance	Useful	Ethics	Last	-.343

The second table holds the remaining course outcomes. Most of these correlations are low, and so are not particularly compelling. In particular, most of them make little sense – why does a students' perception of the instructors' caring whether they learn to work with data negatively correspond to their Final Project grade? Why would students' sense of the usefulness of learning ethics negatively correspond to their attendance? Although explanations could be hypothesized, I consider it more likely an artifact of applying statistical tests across too many features, which can incorrectly identify significance when there is none. For the remaining outcomes, there is some credence to the idea that students sense of their own Successfulness in the course content would be correlated with course outcomes such as the Final Project and the Final Grade. However, none of the results seem to have much importance.

Table 4.7 shows the significant correlations for the major course outcomes in Fall 2016. First, it is necessary to point out the relatively low correlations in the table, much like the previous semester, suggesting limited trust-worthiness of the data. In fact, none of the outcomes are particularly well-correlated with motivation/course component items, either at the beginning of the semester or at the end. That said, it is telling that the most highly correlated items are those related to learning about programming – particularly with regards to Interest and Usefulness. The most well-correlated outcome is again the Intent to Continue questions (applying computation and learning computation). The only two cognitive outcomes with relatively high correlations are students' Final Grades and their Classwork grades, which correlate with students' self-efficacy at learning to program – which suggests only that students have some ability to self-assess. Many of the previously established cognitive outcomes fail to even appear in the table, representing the limited connection found between motivation and cognitive outcomes.

Table 4.7: Correlation of Outcomes vs. MUSIC/Course Components in Fall 2016 Semester

Course Outcome	Motivation	Course Component	Survey	Correlation
Attendance	Caring	Program	First	.316
Recommend Course	Interesting	Cohort	First	.304
Continue Applying	Interesting	Program	First	.368
Continue Applying	Useful	Program	Last	.390
Continue Applying	Caring	Program	Last	.375
Continue Applying	Caring	Abstraction	Last	.377
Final Grade	Successful	Program	Last	.396
Quizzes	Successful	Abstraction	Last	.304
Classwork	Successful	Program	Last	.354
Continue Learning	Interesting	Program	First	.405
Continue Learning	Interesting	Program	Last	.341
Continue Learning	Useful	Program	Last	.406
Continue Learning	Successful	Program	Last	.354

Overall, the major result that I glean from this data is that the course content, particularly with regards to students' interest and sense of usefulness, are the most important components connected to students intent to continue. These two semesters worth of results seem to suggest that few connections can be drawn from motivational data and cognitive course outcomes. However, the engagement course outcomes (i.e. intent to continue) can still be seen as valuable targets, if nothing else.

Chapter 5

Conclusions

At the beginning of this dissertation, I posed five research questions to be answered. Here, I restate those questions and summarize the results and their associated conclusions.

5.1 How Can Student Motivation in Introductory Computing Be Modeled?

Academic motivation is a complex, internal process where students choose to engage with a learning experience. To answer the research question, we need to not only model student motivation, but also model the course itself. With regards to motivation, introductory students become motivated by a multitude of factors at variable levels, in reaction to many different elements of a course. In the MUSIC Model, these factors can be described with a five-component model named MUSIC: students' sense of eMpowerment, students' sense of the Usefulness of the material, their sense of Successfulness with the material, their Interest in the experience, and their perception of the instructors' Care that the students learn the materials. When analyzing a course, it is useful to view the learning experience as the sum of four distinct elements: the Content to be learned, the Context it is learned within, the Scaffolds that support the learning experience, and the Assessments that measure and direct the learning. By combining these two models, one of academic motivation and the other of course analysis, into a coherent framework, I can describe and analyze student motivation within a specific course with greater precision. During a learning experience, these different course elements affect these motivational components in different ways.

5.2 How Does Data Science Compare to Other Introductory Contexts?

A wide range of introductory computing contexts have been introduced over the years to better motivate students. These contexts can be categorized based on how they attempt to motivate students. For instance, contexts such as Media Computation and Game Design rely on appealing to students' situational interest, rather than their sense of usefulness. Data Science instead focuses more heavily on students' sense of long-term and short-term usefulness rather than their interest. In addition, Data Science still maintains many opportunities to engage students' interest, potentially making it superior to many other kinds of contexts. Although true for most contexts, Data Science can also naturally allow opportunities for students to be empowered, and proper scaffolding ensures that learners will feel a healthy level of success. Although Data Science provides opportunities for the instructor to express their sense of caring, this can be true of almost any context. Ultimately, Data Science seems to offer strong advantages over many other popular contexts across the motivational components.

5.3 What Affordances and Barriers Does Data Science Offer as an Introductory Context?

Data Science can be a challenging context to integrate into an introductory setting. Finding and obtaining suitable pedagogical datasets is difficult. Once these datasets have been collected, they must be shaped and molded into a shape suitable for introductory students. This work requires expertise in data manipulation, statistics, and the subject matter domain. Then, after the datasets have been delivered to students, there is an entire process where these learners must integrate the data into their algorithms and work. Depending on the level of the learner, that integration can be through simple interactive visualization tools, through scaffolded client libraries, or even professional programmatic interfaces. Finally, students must make sense of the data and any derived results, navigating documentation, the subject matter, and the data's structure itself.

However, the Data Science approach also offers many pedagogically useful opportunities for introductory computing content and multiple avenues of motivational support. Realistic datasets provide implicit justification for computational methods and programming constructs – iteration is needed because of the data set's inherent structure; IF statements are needed to handle variable types; and abstraction is needed to handle the scope and size of the data. The wide breadth of datasets mean that students can be given many kinds of diverse assignments, possibly of their own choosing, empowering the student and potentially offering many ways for them to be interested. The insights gained from analyzing datasets can have serious implications on public policy and knowledge, and reflects genuine practice from professionals in their field – this gives a sense of authenticity and usefulness to activities involving Data Science. Finally, data can be provided at varying levels of complexity, as the learning experience demands, which means that students can be challenged at a level appropriate for each student.

5.4 How Can Technology Support Data Science as a Learning Context?

In Chapter 3 of this dissertation, I reviewed two projects, RealTimeWeb and the CORGIS Collection, that provided technology to support Data Science as a learning context. The RealTimeWeb project was a prototypical approach based on integrating real-time APIs, and incorporated a number of technologically interesting scaffolds. These scaffolds include a client-side cache to prevent networking outages, simplified data retrieval interfaces, and layered response types to support different levels of complexity in returned data. The CORGIS project represented an evolution of the idea, as the Collection of Really Great and Interesting dataSets. Not only does CORGIS make a large repository of datasets available across many introductory environments and through easy-to-use tools, but the project also has recorded a large collection of strategies, tips, opportunities, and design issues involved in Pedagogical Dataset development. CORGIS client libraries make it easy to immediately access datasets without knowledge of file handling mechanics, esoteric file formats, or the hazards of testing large data files. As free and open-source tools, these materials and technologies can immediately benefit educators and researchers interested in introductory computing.

5.5 What Is the Impact of a Data Science Context, as Opposed to Other Course Components, on Student Motivation and Course Outcomes?

In Chapter 4 of this dissertation, I reviewed the results of several pilot studies involving RealTimeWeb and the CORGIS project. The former was piloted in existing introductory computing courses for computing majors at various levels, while the latter was used in a new course specifically for non-computing majors. This new course provided a test-bed for a number of important sub-research questions that describe the impact of a Data Science context on student motivation and course outcomes. At a high level, data collected during multiple iterations of the course shows that not only can a Data Science context provide more motivation (across multiple facets of motivation) than the course content, but it is competitive with the best of the common contexts used in introductory computing. Even better still, as a context it is fairly gender neutral, making it suitable for a diverse audience. I provided evidence that the course designed around the CORGIS project was also successful, especially in comparison to existing popular contexts such as Media Computation. However, analysis of the association between student motivation and course outcomes revealed few connections. In fact, the strongest connection showed that the course content is significantly more correlated with students' intent to continue in computing than the context. These results can guide course designers who must consider course outcomes related to both motivation and engagement, especially when evaluating how they can incorporate a Data Science context. If the goal is to motivate students, then focusing on the course context may be the best approach; however, if the goal is to convince students to consider future computing experiences, then focusing on motivating the content itself is important.

5.6 Suggestions for Future Research

A number of future directions exist for the CORGIS project and this research effort. Here I describe a number of ways that this research could evolve further.

5.6.1 Expanding the Size of the CORGIS Project

The CORGIS collection currently contains over 40 libraries for 3 languages, 3 formats, and with multiple online tools. Data collected in the Computational Thinking course suggests that there are students who feel that there are not enough choices among the existing datasets. Taken one way, this means that the collection needs to expand and cover more domains: for instance, there are few medically-oriented datasets, few datasets about fashion and merchandising, and few datasets relevant to the natural sciences. A full needs assessment could be used to determine what career paths and disciplines are not covered by the current collection. Although this assessment might reveal the need for new kinds of datasets, it could also reveal a need for more redundancy among the datasets that already exist: given the large percentage of building construction majors each semester, for example, it would be beneficial to have more than two libraries suitable for these students.

Although the CORGIS project covers a number of popular introductory languages and formats, not every course uses Racket, Python, or Java. Many courses still teach in languages such as C++, C, and MatLab, for instance, which could benefit from simplistic libraries. Other data formats exist too, such as the popular XML format, or formats specifically for tools such as SPSS, STATA, and JMP.

In general, expanding the CORGIS Project should also involve expanding the CORGIS community. Bringing in more contributors, both in terms of finding, developing, and managing datasets, would allow the project to scale considerably. This would require substantial effort to formalize procedures within the CORGIS project and document the methodologies, which would be of additional benefit to the long-term health of the project.

5.6.2 Establishing the Connection between Motivation and Course Outcomes

In this dissertation, only tenuous connections were found between most major course outcomes and motivational/course components. Although some motivational/course components were able to connect more strongly with engagement outcomes, there were few convincing associations with cognitive outcomes. Although one possibility is that motivation is not particularly well-linked to course outcomes, other possibilities exist too. For example, the survey instruments may have flaws that make the data being collected insufficient to find a meaningful relationship. Alternatively, the assessment instruments used to measure course outcomes could be faulty or inadequate to the task. Further studies, with larger populations and more balanced demographics, might also shed light on stronger correlations. These studies might also investigate other populations, such as computing majors and similar computing-oriented disciplines. Ideally, motivation data could be used to predict course outcomes from the start of the course.

5.6.3 Extending Pedagogical Datasets to other Domains

Introductory Computing is a natural field to integrate real-world Data Science. However, other fields could potentially benefit from the introduction of pedagogical datasets. Obvious candidates include statistics, math, and computational modeling and data analysis. However, similar arguments made for Computational Thinking could also be made for Data Science – that every workforce can benefit from such skills. These pedagogical datasets could become more specialized elements of learning experiences in other disciplines to empower students with better computational assets and tools.

5.6.4 Better Tools to Support Pedagogical Dataset Development

The process of designing and developing pedagogical datasets is ad-hoc, requiring a significant amount of technical expertise. Few methods are generalizable across datasets, and each dataset introduces its own barriers and hurdles. Although there are many sophisticated tools in many languages for preparing datasets, more specialized tools could be developed to prepare pedagogical datasets. These tools could be used for verification of constraints or expectations of datasets, to mine and prepare data collected from online sources, or to manipulate and process data in a standardized way.

These tools could also be used to review and improve existing datasets. While reviewing datasets for this project, several were found with systemic issues, such as having all string data (indicating a failure to parse data), having incorrectly coded data (e.g., a year value replacing a precipitation value), and in some cases violating structural constraints of the project (e.g., having an object at the top level of the dataset, instead of a list). No large system is without bugs, and it can be difficult to track down these bugs when they are stored across tens of megabytes of data.

Artificially-generated pedagogical datasets could be used to supplement datasets where there is little source material available. These datasets could be created by subject-matter experts to match the kind of datasets that would be found by professionals in various fields. Further research is required to determine if these datasets have the same motivational value that other datasets do, and whether the perception of authenticity can be imparted.

5.7 Conclusion

In this dissertation, I have explored whether Data Science can serve as a motivating context for introductory computing courses. I began answering this question in Chapter 2 with a literature review to devise a framework for describing students' motivation with respect to various course components. Using that framework, I explored literature on existing course contexts and how they compared to Data Science. In Chapter 3, I described two research projects (RealTimeWeb and CORGIS) that scaffolded a Data Science context within a course. This description included not only the interfaces and tools that students could use, but described processes, challenges, and opportunities for pedagogical dataset developers when preparing appropriate data sources. Finally, in Chapter 4, I reviewed a number of research studies that involved the use of RealTimeWeb and CORGIS. These studies evaluated the systems for usability and explored deeper research questions related to the effectiveness of a Data Science context in terms of motivation, engagement, and other course outcomes. Although Data Science was shown to be a compelling context, possibly superior to many other common introductory contexts, it was also shown that there was a stronger link between the course content (rather than the course context) and student's intent to pursue additional study in computing. Further research is required to better understand the connection between student motivation and course outcomes, and the effectiveness of Data Science as a course context.

Bibliography

[1] ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer science curricula 2013. Technical report, ACM Press and IEEE Computer Society Press, December 2013.

[2] R. E. Anderson, M. D. Ernst, R. Ordóñez, P. Pham, and B. Tribelhorn. A data programming CS1 course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 150–155, 2015.

[3] R. E. Anderson, M. D. Ernst, R. Ordóñez, P. Pham, and S. A. Wolfman. Introductory programming meets the real world: Using real problems and data in CS1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 465–466, 2014.

[4] O. Astrachan, T. Barnes, D. D. Garcia, J. Paul, B. Simon, and L. Snyder. CS principles: piloting a new course at national scale, 2011.

[5] L. J. Barker, C. McDowell, and K. Kalahar. Exploring factors that influence computer science introductory course students to persist in the major. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 153–157, 2009.

[6] A. C. Bart. Situating computational thinking with big data: Pedagogy and technology (abstract only). In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 719–719, 2015.

[7] A. C. Bart, J. Tibau, E. Tilevich, C. A. Shaffer, and D. Kafura. Implementing an open-access, data science programming environment for learners. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, volume 1, pages 728–737, 2016.

[8] A. C. Bart, E. Tilevich, S. Hall, T. Allevato, and C. A. Shaffer. Transforming introductory computer science projects via real-time web data. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 289–294, 2014.

[9] A. C. Bart, E. Tilevich, C. A. Shaffer, and D. Kafura. Position paper: From interest to usefulness with BlockPy, a block-based, educational environment. In *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE*, pages 87–89, 2015.

[10] A. C. Bart, R. Whitcomb, E. Tilevich, C. A. Shaffer, and D. Kafura. Computing with CORGIS: Diverse, Real-world Datasets for Introductory Computing. In *Proceedings of the 48th ACM Technical Symposium on Computer Science Education*, SIGCSE '17, 2017.

[11] M. Ben-Ari. Situated learning in computer science education. *Computer Science Education*, 14(2):85–100, 2004.

[12] P. Bidgood. Creating statistical resources from real datasets–the STARS project. In *Seventh International Conference on Teaching Statistics (ICOTS7), Salvador, Bahia, Brazil*, 2006.

[13] J. S. Brown, A. Collins, and P. Duguid. Situated cognition and the culture of learning. *Educational researcher*, 18(1):32–42, 1989.

[14] M. Buckley, J. Nordlinger, and D. Subramanian. Socially relevant computing. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, pages 347–351, 2008.

[15] A. Bundy. Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, 1(2):67–69, 2007.

[16] J. Carter, D. Bouvier, R. Cardell-Oliver, M. Hamilton, S. Kurkovsky, S. Markham, O. W. McClung, R. McDermott, C. Riedesel, J. Shi, and S. White. Motivating all our students? In *Proceedings of the 16th Annual Conference Reports on Innovation and Technology in Computer Science Education - Working Group Reports*, ITiCSE-WGR '11, pages 1–18, 2011.

[17] L. N. Cassel. Interdisciplinary computing is the answer: Now, what was the question? *ACM Inroads*, 2(1):4–6, 2011.

[18] J.-I. Choi and M. Hannafin. Situated cognition and learning environments: Roles, structures, and implications for design. *Educational Technology Research and Development*, 43(2):53–69, 1995.

[19] S. Cooper, W. Dann, and R. Pausch. Teaching objects-first in introductory computer science. In *ACM SIGCSE Bulletin*, volume 35, pages 191–195, 2003.

[20] N. Cowan. The magical mystery four: How is working memory capacity limited, and why? *Current directions in psychological science*, 19(1):51–57, 2010.

[21] C. Day. Computational thinking is becoming one of the three R's. *Computing in Science and Engineering*, 13(1):88–88, 2011.

[22] P. J. Denning. The profession of IT: Beyond computational thinking. *Communications of the ACM*, 52(6):28–30, 2009.

[23] P. J. Denning and A. McGettrick. Recentering computer science. *Communications of the ACM*, 48(11):15–19, Nov 2005.

[24] P. DePasquale. Exploiting on-line data sources as the basis of programming projects. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '06, pages 283–287, 2006.

[25] B. DiSalvo and A. Bruckman. From interests to values. *Communications of the ACM*, 54(8):27–29, Aug. 2011.

[26] L. Douglas. The importance of "Big Data": A definition. *Gartner (June 2012)*, 2012.

[27] W. M. DuBow, B. A. Quinn, G. C. Townsend, R. Robinson, and V. Barr. Efforts to make computer science more inclusive of women. *ACM Inroads*, 7(4):74–80, 2016.

[28] S. Edwards. Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. In *In Proceedings of the International Conference on Education and Information Systems: Technologies and Applications*, EISTA, 2003.

[29] A. E. Egger. Engaging students in earthquakes via real-time data and decisions. *Science*, 336(6089):1654–1655, 2012.

[30] A. Erkan, T. Pfaff, J. Hamilton, and M. Rogers. Sustainability themed problem solving in data structures and algorithms. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 9–14, 2012.

[31] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. How to design programs, 2001.

[32] A. Forte and M. Guzdial. Computers for communication, not calculation: Media as a motivation and context for learning. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 10–pp, 2004.

[33] M. N. Giannakos, I. O. Pappas, L. Jaccheri, and D. G. Sampson. Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness. *Education and Information Technologies*, pages 1–18, 2016.

[34] M. Goldweber, J. Barr, T. Clear, R. Davoli, S. Mann, E. Patitsas, and S. Portnoff. A framework for enhancing the social good in computing education: a values approach. *ACM Inroads*, 4(1):58–79, 2013.

[35] Google Inc. Exploring computational thinking. `http://www.google.com/edu/computational-thinking/`, 2011. Accessed: 01-30-2017.

[36] R. Guha. Introducing schema.org: Search engines come together for a richer web. *Google Official Blog*, 2011.

[37] M. Guzdial. Exploring hypotheses about media computation. In *Proceedings of the ninth annual international ACM conference on International computing education research*, pages 19–26, 2013.

[38] M. Guzdial and A. E. Tew. Imagineering inauthentic legitimate peripheral participation: an instructional design approach for motivating computing education. In *Proceedings of the second international workshop on Computing education research*, pages 51–58, 2006.

[39] O. A. Hall-Holt and K. R. Sanft. Statistics-infused introduction to computer science. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 138–143, 2015.

[40] S. F. Harp and R. E. Mayer. How seductive details do their damage: A theory of cognitive interest in science learning. *Journal of Educational Psychology*, 90(3):414, 1998.

[41] D. Hemmendinger. A plea for modesty. *ACM Inroads*, 1(2):4–7, 2010.

[42] International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA). Operational definition of computational thinking, 2011.

[43] J. A. Jacobs. Interdisciplinary hype, November 2009.

[44] B. D. Jones. Motivating students to engage in learning: The MUSIC model of academic motivation. *International Journal of Teaching and Learning in Higher Education*, 21(2):272–285, 2009.

[45] B. D. Jones and G. Skaggs. *Validation of the MUSIC Model of Academic Motivation Inventory: A measure of students' motivation in college courses*. 2012.

[46] J. S. Kay. Contextualized approaches to introductory computer science: The key to making computer science relevant or simply bait and switch? In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 177–182, 2011.

[47] J. Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, Apr. 2007.

[48] J. Lave and E. Wenger. *Situated learning: Legitimate peripheral participation*. 1991.

[49] L. Layman, L. Williams, and K. Slaten. Note to self: Make assignments meaningful. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, pages 459–463, 2007.

[50] M. Lichman. UCI machine learning repository, 2013.

[51] A. Luxton-Reilly. Learning to program is easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 284–289, 2016.

[52] W. J. Ma, M. Husain, and P. M. Bays. Changing concepts of working memory. *Nature neuroscience*, 17(3):347–356, 2014.

[53] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*, 2011.

[54] J. J. McGinley and B. D. Jones. A brief instructional intervention to increase students' motivation on the first day of class. *Teaching of Psychology*, 41(2):158–162, 2014.

[55] G. A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[56] M. Mitchell, J. Sheard, and S. Markham. Student motivation and positive impressions of computing subjects. In *Proceedings of the Australasian Conference on Computing Education*, ACSE '00, pages 189–194, 2000.

[57] S. Mneimneh. Fibonacci in the curriculum: Not just a bad recurrence. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 253–258, 2015.

[58] G. Norman. Likert scales, levels of measurement and the "laws" of statistics. *Advances in health sciences education*, 15(5):625–632, 2010.

[59] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta. Comparison of JSON and XML data interchange formats: A case study. *Caine*, 2009:157–162, 2009.

[60] O. of the Senior Vice President and Provost. Academic implementation strategy for a plan for a new horizon: Envisioning virginia tech 2013-2018. Technical report, 2013.

[61] O. L. Oliveira, A. M. Monteiro, and N. T. Roman. From concrete to abstract?: Problem domain in the learning of introductory programming. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 173–177, 2011.

[62] S. Papert. An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning*, 1(1):95–123, 1996.

[63] I. O. Pappas, M. N. Giannakos, and L. Jaccheri. Investigating factors influencing students' intention to dropout computer science studies. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, pages 198–203, 2016.

[64] N. Parlante. Codingbat, 2015. Online. Retrieved 1/08/2011 from http://codingbat.com, 2011.

[65] J. Petraglia. The real world on a short leash: The (mis) application of constructivism to the design of educational technology. *Educational Technology Research and Development*, 46(3):53–65, 1998.

[66] J. Pfeiffer. *The thinking machine.* 1962.

[67] C. Rader, D. Hakkarinen, B. M. Moskal, and K. Hellman. Exploring the appeal of socially relevant computing: Are students interested in socially relevant problems? In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 423–428, 2011.

[68] J. Radinsky, B. Loh, J. Mundt, S. Marshall, L. M. Gomez, B. J. Reiser, and D. C. Edelson. Problematizing complex datasets for students: Design principles for inquiry curriculum. 1999.

[69] M. Rettig. Nobody reads documentation. *Communications of the ACM*, 34(7):19–24, July 1991.

[70] B. E. Rogoff and J. E. Lave. *Everyday cognition: Its development in social context.* 1984.

[71] D. F. Shell, L.-K. Soh, A. E. Flanigan, and M. S. Peteranetz. Students' initial course motivation and their achievement and retention in college CS1 courses. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 639–644, 2016.

[72] Y. N. Silva, S. W. Dietrich, J. M. Reed, and L. M. Tsosie. Integrating big data into the computing curricula. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 139–144, 2014.

[73] J. Singer-Vine. Data is plural. `http://tinyletter.com/data-is-plural/archive`, 2015.

[74] D. E. Stevenson and P. J. Wagner. Developing real-world programming assignments for CS1. *SIGCSE Bulletin*, 38(3):158–162, June 2006.

[75] J. A. Stone and E. M. Madigan. The impact of providing project choices in CS1. *SIGCSE Bulletin*, 40(2):65–68, June 2008.

[76] D. G. Sullivan. A data-centric introduction to computer science for non-majors. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 71–76, 2013.

[77] N. Sussman. More falsehoods programmers believe about time; "wisdom of the crowd" edition, 2017.

[78] M. Taylor. The Layers of English, 2017. Online; accessed 26-January-2017.

[79] S. Teller. Measuring vocabulary richness with python - a geek with a hat, 2017.

[80] A. E. Tew. *Assessing fundamental introductory computing concept knowledge in a language independent manner*. PhD thesis, Georgia Institute of Technology, 2010.

[81] T. Thompson and R. Barnes. The engine of successful education reform: Effective teachers and principals. *Commission on No Child Left Behind. The Aspen Institute. Denver,(Oct. 2009)*, 2009.

[82] L. Torgo. *Data Mining with R, learning with case studies*. 2010.

[83] K. Verbert, N. Manouselis, H. Drachsler, and E. Duval. Dataset-driven research to support learning and knowledge analytics. *Educational Technology & Society*, 15(3):133–148, 2012.

[84] Virginia Tech Office of Institutional Research & Effectiveness. Distribution of Grades, Average QCA by Course and Instructor, 2016. Available from: `http://www.ir.vt.edu/`.

[85] M. Waldman. Keeping it real: utilizing NYC open data in an introduction to database systems course. *Journal of Computing Sciences in Colleges*, 28(6):156–161, 2013.

[86] A. Weinberg. Computational thinking: An investigation of the existing scholarship and research. In *School of Education*. 2013.

[87] Z. Weinersmith. How introductory physics problems are written. `http://www.smbc-comics.com/comics/20131202.png`, Dec 2013.

[88] H. Wickham. Tidy data. *The Journal of Statistical Software*, 59, 2014.

[89] Wikimedia Commons. Data Science Process. `https://commons.wikimedia.org/wiki/File%3aData_visualization_process_v1.png`. Accessed: 06-17-15.

[90] "Wikipedia". Feature creep — wikipedia, the free encyclopedia, 2016. Online; accessed 16-February-2017.

[91] Wikipedia. Grammatical mood — wikipedia, the free encyclopedia, 2017. Online; accessed 26-January-2017.

[92] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[93] S. Yarosh and M. Guzdial. Narrating data structures: The role of context in CS2. *Journal on Educational Resources in Computing (JERIC)*, 7(4):6, 2008.

[94] Z. Zografski. Innovating introductory computer science courses: Approaches and comparisons. In *Proceedings of the 45th Annual Southeast Regional Conference*, ACM-SE 45, pages 478–483, 2007.

Appendix A

Supplementary Materials

A.1 CORGIS Specification File Format

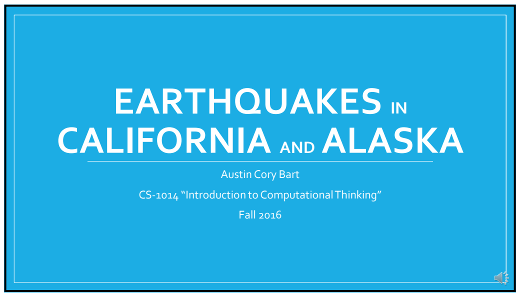
new 7 *Friday, January 20, 2017 11:35 PM*

```
metadata: (object) Information about the dataset, such as its name and version are present here.
    name: (string) The full name of the dataset, including spaces and capitilization. Target
    Builders are responsible for sluggifying the dataset name as needed.
    author: (string) A list of names of the authors who have been involved in creating this
    dataset, separated by a comma.
    version: (int) A number indicating the current version of this dataset, meant to be
    incremented each time there are changes.
    datetime: (string) A date time indicating when this dataset was last updated, meant to be
    updated by hand.
    hardware: (int) A number indicating the default limit to restrict the length of datasets
    when they are sampled (e.g., only the first 1000 elements).
    description: (object) A set of high-level descriptions of the dataset, meant for humans.
        overview: (string) The main description of the dataset, should probably be a few
        paragraphs. This gives any necessary background and context for any users of the dataset.
        short: (string) A short, 1-2 sentence description of the dataset, meant to give a quick
        idea of what it is about.
    appendix: (list of string) A list of filenames in the languages' directory that should be
    included.
    icon: (string) The filename of the icon file for this dataset.
    tags: (list of string) A list of keywords that can be used to search for this dataset.
locals: (list of objects) A list of the actual data files available for this dataset.
Historically, this was a list because multiple data files could be associated with a single
dataset. In practice, that was a nightmare and we don't do that. So this is typically only one
data file.
    name: (string) The name of this local dataset, often identical to the name of the dataset
    itself. It would really only be different if there needed to be more than one data file.
    file: (string) The full path to the data file.
    type: (string) The type of the data file. In theory, other formats were to be availabel
    (e.g., CSV), but in practice JSON is perfect.
    row: (string) A description of what a single row of the dataset looks like, for humans.
    This is used by some Target Builders (e.g., the Visualizer) to explain the dataset better.
    It's also used to name the top-level class in the Java libraries (and other statically
    typed libraries)
    order: (string) A description of how the dataset is sorted, for humans. This is used by
    some Target Builders (e.g., the Visualizer) to explain the dataset better.
    index: (list of objects) A dataset can optionally have an index to more efficiently access
    certain slices of the data. This is particularly useful for the Visualizer.
        name: (string) The name that this index will be given.
        location: (string) A representative JSON Path into the dataset where the data should be
        indexed against (e.g., "school_scores.[0].State.Name"). Must not contain sublist
        elements.
interfaces: (list of objects) A list of the ways that this dataset can be accessed.
Historically, this would allow multiple methods for accessing data, including HTTP APIs.
    name: (string) The name of the interface for accessing the data. This will often be
    translated into a function for the end-user.
    returns: (string) The file type that will be returned for this dataset. Any special objects
    should be consistent with the Local Row attribute from above. For example, if the row was
    "record", then this interface might return "list[record]".
    description: (string) A description of what this interface does, meant for the end-user.
    production: (object) The specific information needed to access data within the locals for
    this interface.
        sql: (string) A SQL query that will be applied by the interface to actually retrieve
        data from the database. It can reference tables that are named via the ``local'' field
        above. Most of the time, this will be very straightforward, to just return the entire
        dataset.
        http: (string) Historically, it was going to be possible to access online data sources
        through these interfaces. That ended up not being particularly valuable, and the
        feature was never completed.
        post: (string) A bit of custom Bark script that can be used to post-process the result
        of the SQL query. Bark was meant to be a mini-language to apply simple filters. There
        isn't really much it can do, but the most common one is to convert the result from a
        raw string to JSON (e.g., "json()") or to convert it to JSON and then access a specific
        field (e.g., "json()|jsonpath(school_scores.[0].State.Name)").
        pre: (string) A bit of custom Bark script that can be used preprocess the dataset
        returned by the dataset.
    test: (object) Identical to the `production` field, except it will only be used with test
```

-1-

```
    data.
    args: (list of objects) The arguments that will be available for this interface.
        name: (string) The name of the argument, ready to be sluggified for the language.
        type: (string) What data type the argument should be (typically a primitive value such
        as int).
        default: (any) Optionaly field to use as the default value for the call in the
        documentation, as an example.
        matches: (string) This field is a SQL string that will allow validation for the given
        argument. Typically, this is used to make a suggestion about a mispelling or something.
        description: (string) A human-ready description of this argument.
    cache: (list) I'll be honest, I don't know what this was meant for.
structures: (list of object) This is where comments and explanations of individual dataset
fields comes in. There are two types of structures.
    dictionaries: (list of objects) Any dictionary nested anywhere inside the dataset gets an
    entry here. We require that dictionaries have unique names, unless they have identical
    structures!
        name: The name of the dictionary, ready to be sluggified as needed.
        fields: (list of objects) The fields in this dictionary.
            type: (string) The type of the value of this field, which might be a primitive
            (e.g., int), another dictionary, or even a list.
            key: (string) The name of this field, ready to be sluggified.
            example: (string) The value taken from the first row, used as an example in
            documentation, typically.
            path: (string) The full JSON path to this key.
            comment: (string) Any associated metadata that's available for this particular field.
    lists: (list of objects) Any lists that were found are described here.
```

A.2 Example Final Project

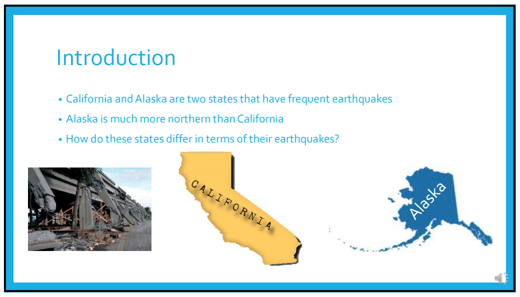


EARTHQUAKES IN CALIFORNIA AND ALASKA

Austin Cory Bart

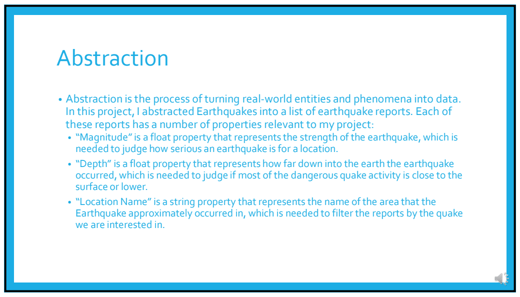
CS-1014 "Introduction to Computational Thinking"

Fall 2016



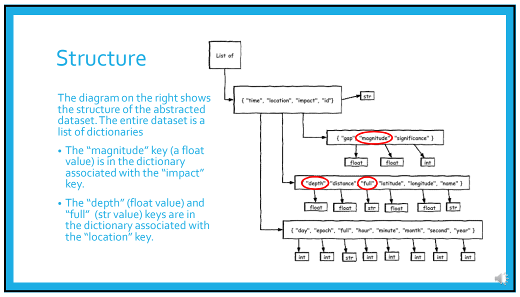
Introduction

- California and Alaska are two states that have frequent earthquakes
- Alaska is much more northern than California
- How do these states differ in terms of their earthquakes?



Abstraction

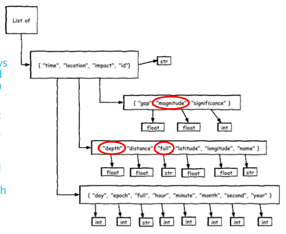
- Abstraction is the process of turning real-world entities and phenomena into data. In this project, I abstracted Earthquakes into a list of earthquake reports. Each of these reports has a number of properties relevant to my project:
  - "Magnitude" is a float property that represents the strength of the earthquake, which is needed to judge how serious an earthquake is for a location.
  - "Depth" is a float property that represents how far down into the earth the earthquake occurred, which is needed to judge if most of the dangerous quake activity is close to the surface or lower.
  - "Location Name" is a string property that represents the name of the area that the Earthquake approximately occurred in, which is needed to filter the reports by the quake we are interested in.

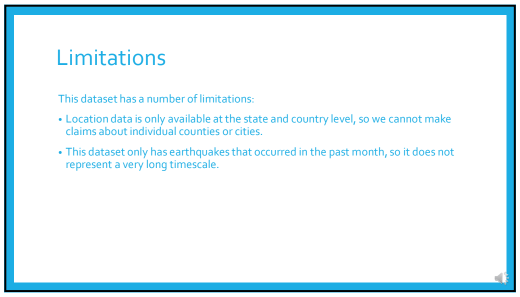


Structure

The diagram on the right shows the structure of the abstracted dataset. The entire dataset is a list of dictionaries

- The "magnitude" key (a float value) is in the dictionary associated with the "impact" key.
- The "depth" (float value) and "full" (str value) keys are in the dictionary associated with the "location" key.

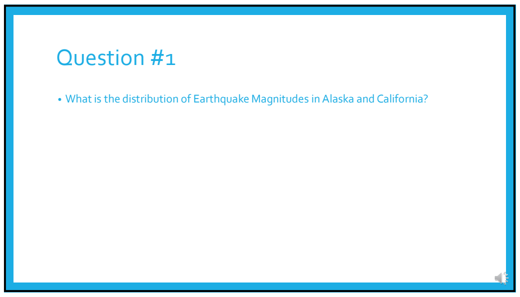

List of
{ "time", "location", "impact", "id"}
str
{ "gap", "magnitude", "significance" }
float
float
int
{ "depth", "distance", "full", "latitude", "longitude", "name" }
float
float
str
float
float
str
{ "day", "epoch", "full", "hour", "minute", "month", "second", "year" }
int
int
str
int
int
int
int
int



Limitations

This dataset has a number of limitations:

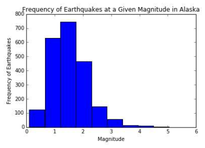
- Location data is only available at the state and country level, so we cannot make claims about individual counties or cities.
- This dataset only has earthquakes that occurred in the past month, so it does not represent a very long timescale.

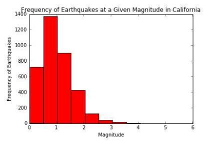


Question #1

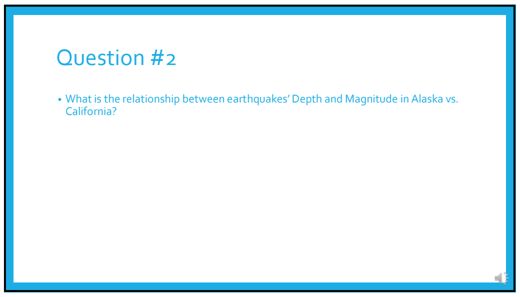
- What is the distribution of Earthquake Magnitudes in Alaska and California?

Visualization/Answer #1





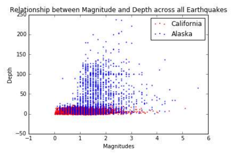
These two visualizations show the distribution of magnitudes in Alaska and California. Both locations tend to have many relatively low magnitude earthquakes, and tend to not have too many high magnitudes. Although the scale of the magnitudes is similar, the quantity of earthquakes seems to be different.



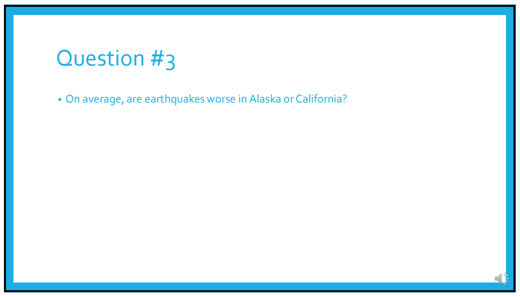
Question #2

- What is the relationship between earthquakes' Depth and Magnitude in Alaska vs. California?

Visualization/Answer #2



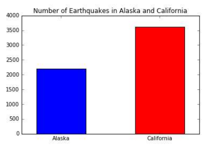
This scatter plot shows the relationship between magnitudes and depths; the red dots are California and the blue dots are Alaska. Although both locations seem to have the same range of magnitudes, Alaska seems to have a much wider range of depths. There doesn't seem to be a relationship between depth and magnitude.

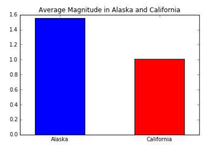


Question #3

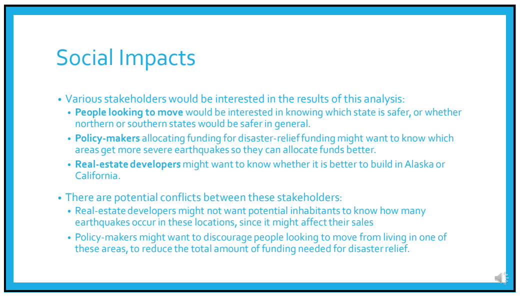
- On average, are earthquakes worse in Alaska or California?

Visualization/Answer #3





These bar charts show the total number of earthquakes and the average magnitudes between the states. It shows that although there are more earthquakes in California, the average earthquake in Alaska is more severe.



Social Impacts

- Various stakeholders would be interested in the results of this analysis:
  - **People looking to move** would be interested in knowing which state is safer, or whether northern or southern states would be safer in general.
  - **Policy-makers** allocating funding for disaster-relief funding might want to know which areas get more severe earthquakes so they can allocate funds better.
  - **Real-estate developers** might want to know whether it is better to build in Alaska or California.
- There are potential conflicts between these stakeholders:
  - Real-estate developers might not want potential inhabitants to know how many earthquakes occur in these locations, since it might affect their sales
  - Policy-makers might want to discourage people looking to move from living in one of these areas, to reduce the total amount of funding needed for disaster relief.

A.3 Final Project Grading Rubric

Book - 7.2) Project - Overview

Outline

There are seven major steps in completing the project. These steps are divided into activities surrounding the data set and those activities related to the computational analysis.

The activities related to the data set are:

- **Abstraction**: Understanding the abstraction of the real-world entities represented by the data set you selected; knowing the organization, structure, and content of the data set, including constructing a data map.
- **Questions**: Deciding on the questions to be answered by the computational analysis of the data set; understanding why the answers to these questions are important and to what audience(s).
- **Limitations**: Listing factors that limit the generality of the possible conclusions that can be drawn from the computational analysis due to restrictions imposed by the data or the method of analysis.
- **Social Impacts**: Identifying possible stakeholders and the benefits or harms that might affect these stakeholders from the results the project; considering conflicts among the stakeholders.

The activities related to the computational analysis are:

- **Programming**: Designing the algorithms and writing the code to perform the computational analysis of the data set.
- **Visualizations**: Designing and computationally generating informative displays of the data set to answer the exploratory questions.
- **Conclusions**: Using the visualizations to answers the exploratory questions; exploring the implications of these answers; developing a

Each of these seven components will be illustrated in the following sections of this chapter.

The outcomes from the project's activities result in:

- a **project presentation**: a video of approximately 5-6 minutes describing the project. This video is created by adding narration to a set of PowerPoint slides, exporting it in a video format, and uploading it to YouTube.
- the **project's program**: the Python code developed to manipulate the data set and generate the visualizations.

The project presentation contains a description of six of the seven activities described above - all those except the programming itself. A rubric for evaluating the project presentation is given below. The programming done for the project will be evaluated by assessing the quality of explanations given about selected parts of the Python code. The mechanics for this part of the project evaluation will be described in class.

Questions

A key element of the project is posing interesting and meaningful questions that are to be answered by analyzing the data set you have selected. Two general categories of questions are analysis questions and existence questions.

Analysis questions explore the quantitative characteristics of the real-world phenomenon described by the data and the possible relationships between or among these characteristics. Forms of questions of these types are as follows:

- **Characterization questions**
  - What is the distribution of ... ?
  - What categories of values are there for ...?
  - What is the range of ...?
  - How much variation is there in ...?
  - What is the likelihood of ...?
  - The proportion falling into a given category is ... ?
- **Relationship questions**
  - How is factor x related to factor y?
  - How are factors x, y, and z related?
  - How does factor x change over time?
  - Is there a trend in the data?
  - How is factor x distributed over space?

Various forms of visualizations help to answer analysis questions. For example, line graphs and histograms are useful ways of showing the variation and distribution of the data; scatter plots are good ways to explore the relationship between two characteristics, revealing trends or clusters of data that suggest interactions between the two characteristics.

Existence questions explore a unique or critical aspect of the phenomenon. These questions focus on identifying data points that have particularly interesting combination of properties. Forms of questions of this type are as follows:

- **Selection Questions**
  - What is the best match to ... ?
  - Are there multiple occurrences of ... ?
  - Are there distinctive categories of ...?
- **Capability questions**
  - Is it possible for...?
  - Under what circumstances will ...?

Different types of visualizations help to explore the answers to these kinds of questions. Line graphs or scatter plots help to identify distinctive points in the data (sometimes called outliers) or identify data within certain ranges of interest. Bar charts and histograms help to explore and compare occurrences of different attributes.

Limitations

Our ability to model the real world using information may be imperfect, especially if the real world entity is highly complex or not well understood. Thus, the answers to question that are obtained by analyzing this information model are limited by the imperfections in the data or our analysis of the data. All information models have limitations. Thus, it is important to identify and acknowledge the specific limitations of the model.

Some possible sources of limitations are the following.

Abstraction: A frequent limitation is due to properties that are excluded from the abstraction. The dilemma in forming an abstraction is to anticipate what information properties should be included and which should be excluded. Including too many properties makes the abstraction unfocused and unwieldy. However, a penalty for excluding some properties is that we cannot ask questions or draw conclusion that rely on the excluded data. For example, if we exclude the weight of a patient from a patient abstraction then we cannot draw conclusions about the role of weight in the patient's health.

Completeness: Another common occurrence is that the values might not be available for some properties in some instances. Gaps in the data can arise for a variety of reasons. If the data is self-reported the responder may know the desired value, may not wish to report the value, or simply overlooked this part off the report. If the data is generated by sensors (e.g., temperatures, humidities, etc.) or other automated means the data may be missing because of "mechanical" problems with the sensor (e.g., loss of power to the sensor, loss of network connectivity for the sensor to report, etc.). Obviously, the more missing data there is the more the conclusions are limited.

Precision: In some problems the accuracy of the data might be insufficient to make conclusions at a very precise level of detail. The lack of precision may be a reflection of the underlying ability to gather the data. This is particularly true when the data is projected or estimated. For example, a number indicated the gross domestic product of the U.S. (in the multiple trillions of dollars) cannot be estimated down to the last penny (or perhaps even the last hundreds of millions of dollars).

In addition to these general limitations, the data that you are working with may be subject to other limitations.

Some factors are, however, not inherent limitations of the data even though they may be challenges for the person using the data. Some examples that are not inherent limitations of the data are the following:

Complexity. The data set may be very complex including more detail than is needed to answer the specific questions of the project. While this extra size may be annoying it is not a source of limitations on the ability to use the data in answering questions.

Structure. The data might be organized in such a way that extracting the data to answer the project's questions is tedious or awkward. While this may be a frustration for the data's user, it does not inherently limit the questions that can be answer.

Evaluation

As noted above, the project has two major outcomes: the project's presentation and the project's Python code. Each of these outcomes will be evaluated as describe further in this section.

Presentation Evaluation

The project's video presentation is evaluated by a rubric that rates as missing, poor, good, or excellent each of the following 8 statements.

1. The presenter explained the questions to be answered and their importance.
2. The presenter explained the role of abstraction in defining the information properties relevant to the project’s questions.
3. The presenter explained the limitations of the available data in answering the questions.
4. The presenter explained the structure of the data.
5. The presenter explained the meaning of the visualizations used to answer the questions.
6. The presenter explained the answers to the questions based on the visualizations.
7. The presenter explained the social implications of the project.
8. The presenter communicated effectively.

General Guidelines

In all cases a rating of *Missing* is given if the required element is not recognizably present. It is the responsibility of the presenter to present each element clearly.

An individual required element may be presented over multiple slides and a single slide may relate to multiple required elements.

It is not important in what order the required elements are addressed in the presentation. However, the presentation should have a logical sequencing that makes the presentation understandable and easily relatable to the rubric elements.

Here are detailed descriptions of these elements.

Specific Explanations

1. **The presenter explained the questions to be answered and their importance.**

   Element: The presenter demonstrated problem-solving skills by describing the questions the project is intended to answer and identifying the significance of these answers to some group.

   Rating:

   - *Poor*: states the questions and claims that answering these questions is important.
   - *Good*: states the questions and gives an argument for the importance of answering the questions.
   - *Excellent*: states the questions and an argument supported by evidence (expert opinion, examples, economic factors, etc.) for the importance of answering the questions.

2. **The presenter explained the role of abstraction in defining the information properties relevant to the project’s questions.**

   Element: The presenter demonstrated an understanding of the concept of abstraction by explaining the information properties used to model the real-world entities relevant to the project’s questions.

   Rating:

   - *Poor:* gives a definition of abstraction and names the real-world entities.
   - *Good:* defines abstraction and describes characteristics of the real-world entities.
   - *Excellent:* defines abstraction, identifies specific information properties of the real-world entities, and explains how these properties are relevant to the project’s questions.

3. **The presenter explained the limitations of the available data in answering the questions.**

   Element: The presenter demonstrated quantitative reasoning by identifying factors that restrict the scope (comprehensiveness, completeness) or accuracy (precision) of the answers that can be obtained from the available data.

   Rating:

   - *Poor:* General factors related to the data are described without clear indication of their impact on the answers.
   - *Good:* Specific factors are identified but without clear indication of their impact on the answers.
   - *Excellent:* Specific factors are identified and the impact of these factors on the answers is clearly explained.

4. **The presenter explained the structure of the data.**

   Element: The presenter demonstrated an understanding of data structures by explaining the technical organization of the data.

Rating:

- *Poor:* A diagram or other visual representation of the data's organization is presented and only partially or confusingly described.
- *Good:* A diagram or other visual representation of the data's organization is presented and well described using general language.
- *Excellent:* A diagram or other visual representation of the data's organization is presented and described using correct technical language.

5. **The presenter explained the meaning of the visualizations used to answer the questions.**

   Element: The presenter demonstrated the ability to analyze complex data by explaining the form, content, and significance of the generated visualizations that answer the project's questions.

   Rating:

   - *Poor:* Inappropriate, incomplete or ambiguous visualizations are presented whose relationships to the questions is not clear.
   - *Good:* One form of visualization appropriate to the data and the questions is used. Visualizations are not always appropriate titled or labelled. The relationship of the visualizations to the questions is reasonably clear.
   - *Excellent:* Multiple forms of visualizations appropriate to the data and the questions are used. Visualizations are appropriately titled and labelled. The relationship of the visualizations to the questions is clearly identified.

6. **The presenter explained the answers to the questions based on the visualizations.**

   Element: The presenter demonstrated a quantitative reasoning ability by explaining how conclusions based on the visualizations answer the project's questions.

   Rating:

   - *Poor:* The implications of the visualizations are not explained or are misinterpreted.
   - *Good:* The implications of the visualizations are clearly explained but not directly related to the answer for each question.
   - *Excellent:* The implications of the visualizations are clearly explained and directly related to the answer for each question.

7. **The presenter explained the social implications of the project.**

   Element: The presenter demonstrated an awareness of the social impact of computing by explaining how the results of the project could have positive or negative effects on individuals, groups, or society.

   Rating:

   - *Poor:* states generally that there are social impacts
   - *Good:* identifies appropriate stakeholders and specific impacts that may be experienced by these stakeholders.
   - *Excellent:* identifies stakeholders, impacts, and explains possible conflicts among the stakeholders.

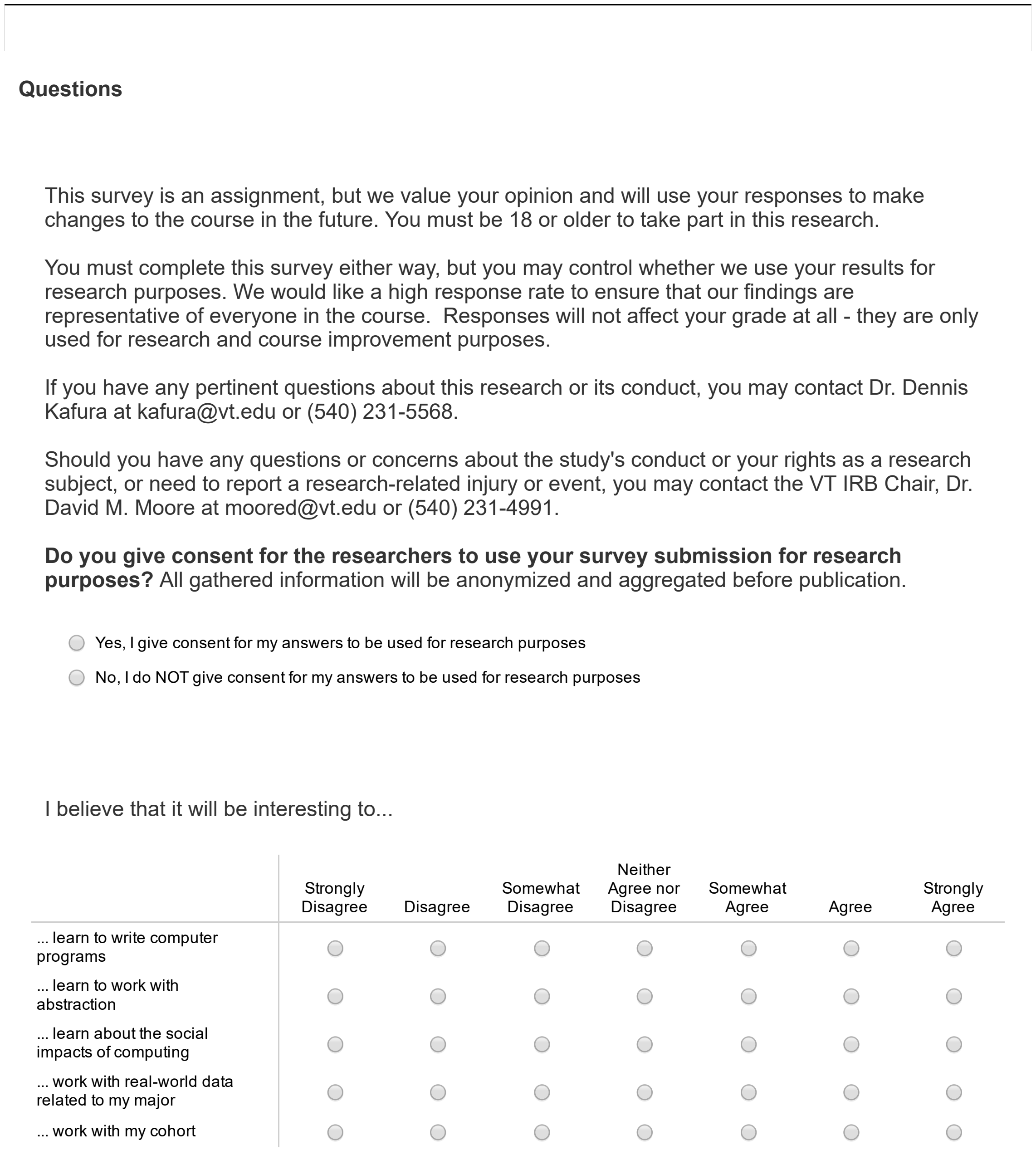
8. **The presenter communicated effectively.**

   Element: The presenter demonstrated oral and visual communication skills by presenting the project in an effective manner.

   Rating:

   - *Poor:* The presentation was of inappropriate length for the content or the presenter was not clearly understandable.
   - *Good:* The presentation was of appropriate length, the presenter was clearly understandable, but the slides lacked some degree of organization or clarity.
   - *Excellent:* The presentation was of an appropriate length; the presenter was clearly understandable, and the slides conveyed information in an organized and clear manner.

A.4 Motivation Survey



Questions

This survey is an assignment, but we value your opinion and will use your responses to make changes to the course in the future. You must be 18 or older to take part in this research.

You must complete this survey either way, but you may control whether we use your results for research purposes. We would like a high response rate to ensure that our findings are representative of everyone in the course. Responses will not affect your grade at all - they are only used for research and course improvement purposes.

If you have any pertinent questions about this research or its conduct, you may contact Dr. Dennis Kafura at kafura@vt.edu or (540) 231-5568.

Should you have any questions or concerns about the study's conduct or your rights as a research subject, or need to report a research-related injury or event, you may contact the VT IRB Chair, Dr. David M. Moore at moored@vt.edu or (540) 231-4991.

Do you give consent for the researchers to use your survey submission for research purposes? All gathered information will be anonymized and aggregated before publication.

○ Yes, I give consent for my answers to be used for research purposes

○ No, I do NOT give consent for my answers to be used for research purposes

I believe that it will be interesting to...

	Strongly Disagree	Disagree	Somewhat Disagree	Neither Agree nor Disagree	Somewhat Agree	Agree	Strongly Agree
... learn to write computer programs	○	○	○	○	○	○	○
... learn to work with abstraction	○	○	○	○	○	○	○
... learn about the social impacts of computing	○	○	○	○	○	○	○
... work with real-world data related to my major	○	○	○	○	○	○	○
... work with my cohort	○	○	○	○	○	○	○

I believe that it will be useful to my long-term career goals to...

	Strongly Disagree	Disagree	Somewhat Disagree	Neither Agree nor Disagree	Somewhat Agree	Agree	Strongly Agree
... learn to write computer programs	○	○	○	○	○	○	○
... learn to work with abstractions	○	○	○	○	○	○	○
... learn about the social impacts of computing	○	○	○	○	○	○	○
... work with real-world data related to my major	○	○	○	○	○	○	○
... work with my cohort	○	○	○	○	○	○	○

I believe that I can be successful in...

	Strongly Disagree	Disagree	Somewhat Disagree	Neither Agree nor Disagree	Somewhat Agree	Agree	Strongly Agree
... learning to write computer programs	○	○	○	○	○	○	○
... learning to work with abstractions	○	○	○	○	○	○	○
... learning about the social impacts of computing	○	○	○	○	○	○	○
... working with real-world data related to my major	○	○	○	○	○	○	○
... working with my cohort	○	○	○	○	○	○	○

I believe that I will have choices as I...

	Strongly Disagree	Disagree	Somewhat Disagree	Neither Agree nor Disagree	Somewhat Agree	Agree	Strongly Agree
... learn to write computer programs	○	○	○	○	○	○	○
... learn to work with abstractions	○	○	○	○	○	○	○
... learn about the social impacts of computing	○	○	○	○	○	○	○
... work with real-world data related to my major	○	○	○	○	○	○	○
... work with my cohort	○	○	○	○	○	○	○

I believe that my instructors in this course care if I...

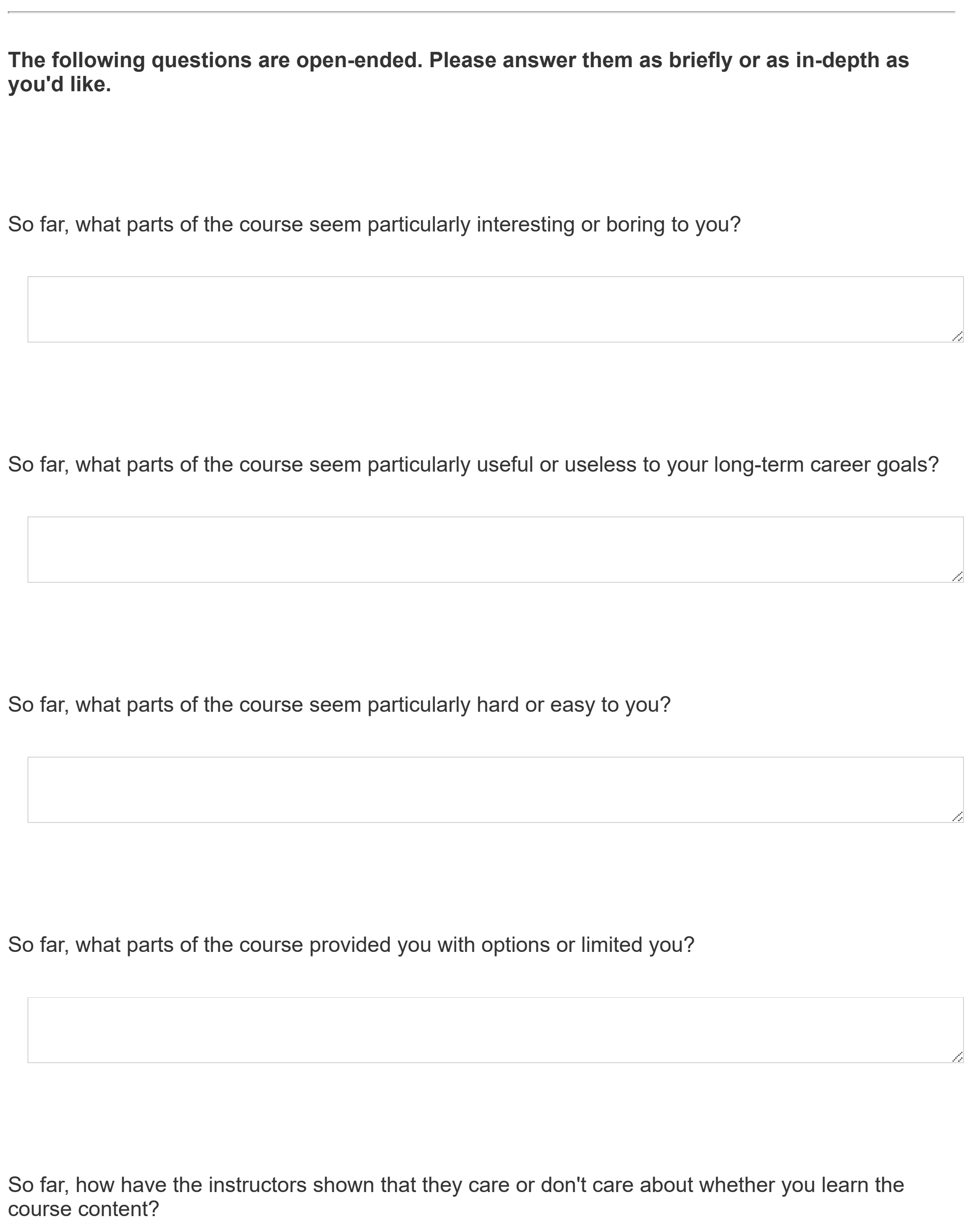
	Strongly Disagree	Disagree	Somewhat Disagree	Neither Agree nor Disagree	Somewhat Agree	Agree	Strongly Agree
... learn to write computer programs	○	○	○	○	○	○	○
... learn to work with abstractions	○	○	○	○	○	○	○
... learn about the social impacts of computing	○	○	○	○	○	○	○
... work with real-world data related to my major	○	○	○	○	○	○	○
... work with my cohort	○	○	○	○	○	○	○

After this course is over, I believe that...

	Strongly agree	Agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Disagree	Strongly disagree
I will try to learn more about computing, either through a course or on my own.	○	○	○	○	○	○	○
I will recommend this class to others.	○	○	○	○	○	○	○
I will directly apply what I have learned in my career.	○	○	○	○	○	○	○

There are many ways to teach a Computational Thinking course. Consider these different topics as a focus for the course. Please indicate how much you would prefer or avoid them.

	Strongly Avoid	Avoid	Somewhat Avoid	Neither Prefer nor Avoid	Somewhat Prefer	Prefer	Strongly Prefer
Working with data sets related to your major	○	○	○	○	○	○	○
Working with pictures, sounds, movies	○	○	○	○	○	○	○
Making games and animations	○	○	○	○	○	○	○
Making websites	○	○	○	○	○	○	○
Making scientific models of real-world phenomenon	○	○	○	○	○	○	○
Controlling robots or drones	○	○	○	○	○	○	○
Making phone apps	○	○	○	○	○	○	○



The following questions are open-ended. Please answer them as briefly or as in-depth as you'd like.

So far, what parts of the course seem particularly interesting or boring to you?

So far, what parts of the course seem particularly useful or useless to your long-term career goals?

So far, what parts of the course seem particularly hard or easy to you?

So far, what parts of the course provided you with options or limited you?

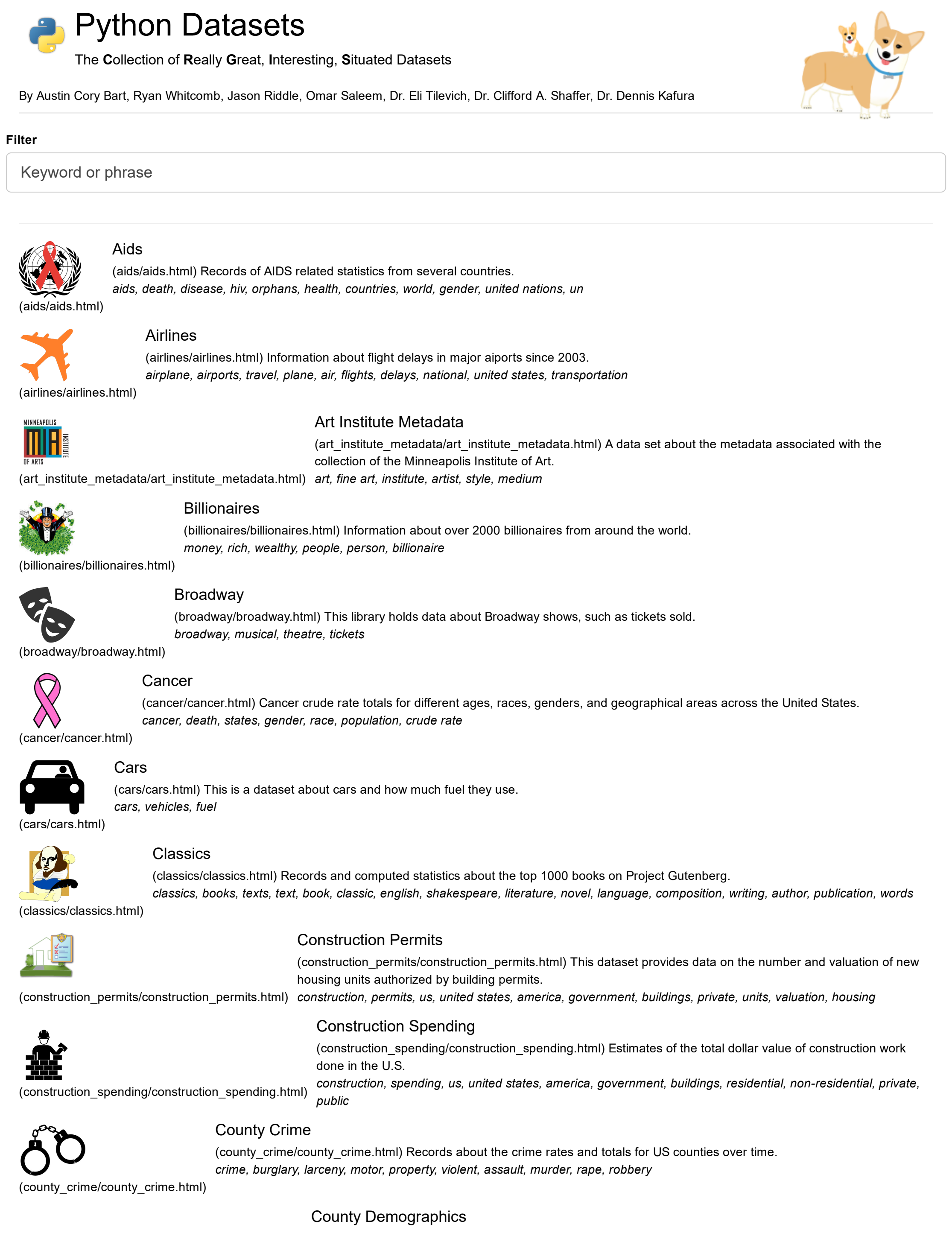
So far, how have the instructors shown that they care or don't care about whether you learn the course content?

Browser Meta Info

This question will not be displayed to the recipient.
Browser: **Chrome**
Version: **56.0.2924.76**
Operating System: **Windows NT 10.0**
Screen Resolution: **1366x768**
Flash Version: **24.0.0**
Java Support: **0**
User Agent: **Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.76 Safari/537.36**

What is your university email address (e.g., @vt.edu)?

A.5 CORGIS Python Collection



Python Datasets

The **C**ollection **o**f **R**eally **G**reat, **I**nteresting, **S**ituated Datasets

By Austin Cory Bart, Ryan Whitcomb, Jason Riddle, Omar Saleem, Dr. Eli Tilevich, Dr. Clifford A. Shaffer, Dr. Dennis Kafura

Filter

Keyword or phrase

(aids/aids.html)
Aids
(aids/aids.html) Records of AIDS related statistics from several countries.
aids, death, disease, hiv, orphans, health, countries, world, gender, united nations, un

(airlines/airlines.html)
Airlines
(airlines/airlines.html) Information about flight delays in major aiports since 2003.
airplane, airports, travel, plane, air, flights, delays, national, united states, transportation

(art_institute_metadata/art_institute_metadata.html)
Art Institute Metadata
(art_institute_metadata/art_institute_metadata.html) A data set about the metadata associated with the collection of the Minneapolis Institute of Art.
art, fine art, institute, artist, style, medium

(billionaires/billionaires.html)
Billionaires
(billionaires/billionaires.html) Information about over 2000 billionaires from around the world.
money, rich, wealthy, people, person, billionaire

(broadway/broadway.html)
Broadway
(broadway/broadway.html) This library holds data about Broadway shows, such as tickets sold.
broadway, musical, theatre, tickets

(cancer/cancer.html)
Cancer
(cancer/cancer.html) Cancer crude rate totals for different ages, races, genders, and geographical areas across the United States.
cancer, death, states, gender, race, population, crude rate

(cars/cars.html)
Cars
(cars/cars.html) This is a dataset about cars and how much fuel they use.
cars, vehicles, fuel

(classics/classics.html)
Classics
(classics/classics.html) Records and computed statistics about the top 1000 books on Project Gutenberg.
classics, books, texts, text, book, classic, english, shakespeare, literature, novel, language, composition, writing, author, publication, words

(construction_permits/construction_permits.html)
Construction Permits
(construction_permits/construction_permits.html) This dataset provides data on the number and valuation of new housing units authorized by building permits.
construction, permits, us, united states, america, government, buildings, private, units, valuation, housing

(construction_spending/construction_spending.html)
Construction Spending
(construction_spending/construction_spending.html) Estimates of the total dollar value of construction work done in the U.S.
construction, spending, us, united states, america, government, buildings, residential, non-residential, private, public

(county_crime/county_crime.html)
County Crime
(county_crime/county_crime.html) Records about the crime rates and totals for US counties over time.
crime, burglary, larceny, motor, property, violent, assault, murder, rape, robbery

County Demographics

(county_demographics/county_demographics.html) Demographic information for counties in the United States.
demographics, population, age, ethnicity, race, language, education, foreign, households, income, employment, unemployment, sales, land area, counties

(drug_bank/drug_bank.html)

Drug Bank
(drug_bank/drug_bank.html) The DrugBank database is a cheminformatics resource that combines detailed drug data with comprehensive drug target information.
drugs, drug, drugbank, chemistry, biology, biochemistry, proteins

(drugs/drugs.html)

Drugs
(drugs/drugs.html) This dataset is about substance abuse (cigarettes, marijuana, cocaine, alcohol) among different age groups and states.
drugs, substances, drug usage

(earthquakes/earthquakes.html)

Earthquakes
(earthquakes/earthquakes.html) Records from different earthquake occurences across the world.
earthquakes, nature, disaster, magnitude, richter scale, latitude, longitude

(education/education.html)

Education
(education/education.html) Information about K-12 education for different states, collected from the US government.
school, education, K-12, kindergarten, learning, teaching

(election/election.html)

Election
(election/election.html) A breakdown of how each county voted in the 2016 Presidential primaries.
politics, election, votes, counties, candidate

(energy/energy.html)

Energy
(energy/energy.html) United States Government reports on consumption, production, import, and export of various fuel sources.
energy, renewable, united states, us, nuclear, fossil fuels, power

(finance/finance.html)

Finance
(finance/finance.html) The Annual Survey of State Government Finances provides a comprehensive summary of the annual survey findings for state governments, as well as data for individual states.
finance, government, expenditure, charges, money, united states, us

(food/food.html)

Food
(food/food.html) Statistics for various food items
food, vitamins, minerals, health, nutrition

(food_access/food_access.html)

Food Access
(food_access/food_access.html) Data about counties ability to access supermarkets, supercenters, grocery stores, or other sources of healthy and affordable food.
counties, states, food, access, availability, supermarket, rural, urban, population, vehicles

THE WORLD BANK
(global_development/global_development.html)

Global Development
(global_development/global_development.html) Reports of country's development over time
world, countries, development, health, population, urban, rural, agriculture, infrastructure

(graduates/graduates.html)

Graduates
(graduates/graduates.html) This library holds data about employment of new graduates by major.
graduation, grads, majors, education, college, university, higher education, jobs, careers

(health/health.html)

Health
(health/health.html) This library holds data about reports of diseases in America.
health, diseases, infection

(hospitals/hospitals.html)

Hospitals
(hospitals/hospitals.html) Records about the outpatient volume and other information for hospitals across the U.S.
hospitals, healthcare, medicare, outpatient, patient, doctor, health

(hydropower/hydropower.html)

Hydropower
(hydropower/hydropower.html) Records about dams in the United States such as location, dimensions, and project information
hydropower, dams, lakes, rivers, waterways, irrigation, human consumption, industrial use, aquaculture, navigability, energy, electricity

(immigration/immigration.html)

Immigration

(immigration/immigration.html) A dataset about the immigration of foreign nationals to the United States.
immigration, naturalization, permanants, united states, us

(injuries/injuries.html)

Injuries

(injuries/injuries.html) A sampling of work-related injury and illness data from the US.
osha, united states, us, usa, government, health, safety, injury, injuries, hurt, pain, job, work, industry, career, occuptation, safety, hazard, health

(labor/labor.html)

Labor

(labor/labor.html) Employment status of the civilian population by race, sex, and age.
labor, race, age, sex, gender, america, usa, census, employed, unemployed, employability, job, work, civilian, black, white, asian, government

(medal_of_honor/medal_of_honor.html)

Medal of Honor

(medal_of_honor/medal_of_honor.html) This dataset has records for the awarding of the United States Medal of Honor, one of the military's highest honors.
us, usa, united states, military, combat, government, army, airforce, marines, navy, fighting, war, award, medal, honor, glory, death

(music/music.html)

Music

(music/music.html) The Music library is a compilation of over one million contemporary songs and information about their audio features and metadata.
music, songs, artists, creativity, media

(publishers/publishers.html)

Publishers

(publishers/publishers.html) E-book sales on Amazon, including daily and total earnings for 54,000 titles.
publishers, amazon, books, sales, genres, literature, english

(real_estate/real_estate.html)

Real Estate

(real_estate/real_estate.html) Real estate information in the United States, including inventory, building, and customer data.
real, estate, buildings, government, building

(retail_services/retail_services.html)

Retail Services

(retail_services/retail_services.html) Monthly sales, inventories, and inventory/sales ratios for retail trade and food services.
retail, services, government, united states, usa, us, trade, food, services

(school_scores/school_scores.html)

School Scores

(school_scores/school_scores.html) SAT scores for students across the United States, organized by year and state.
school, education, K-12, kindergarten, learning, teaching, sat

(skyscrapers/skyscrapers.html)

Skyscrapers

(skyscrapers/skyscrapers.html) Information about tall buildings constructed all over the world.
buildings, construction, building, skyscrapers, international, tall, height, tower

(slavery/slavery.html)

Slavery

(slavery/slavery.html) This dataset has financial records of New Orleans slave sales, 1856-1861. There are almost 16,000 sales recorded in this dataset.
slavery, slave, slaves, buyer, seller, origin, history, economics

(state_crime/state_crime.html)

State Crime

(state_crime/state_crime.html) Records about the crime rates and totals for US states over time.
crime, burglary, larceny, motor, property, violent, assault, murder, rape, robbery

(state_demographics/state_demographics.html)

State Demographics

(state_demographics/state_demographics.html) Demographic information for the states in the United States.
demographics, population, age, ethnicity, race, language, education, foreign, households, income, employment, unemployment, sales, land area, states

(state_fragility/state_fragility.html)

State Fragility

(state_fragility/state_fragility.html) The State Fragility Index scores for countries around the world from 1995 to 2014
world, countries, security, politics, economy, society, effectiveness, legitimacy