A Novel Course in Data Systems with Minimal Prerequisites

Thomas C. Bressoud
Denison University
Granville, Ohio
bressoud@denison.edu

Gavin Thomas
Denison University
Granville, Ohio
thomas_g8@denison.edu

ABSTRACT

The need for understanding data systems, from the structure and constraints of data models to the client-server mechanisms for acquiring and curating data, includes not just computer science students, but extends to data science students and students in a wide range of interdisciplinary programs. In this paper we describe the design and implementation of a second course in computer science, whose data-centric focus emphasizes structural models of data and the skills involved in acquiring and transforming data into forms amenable for analysis. We argue that, from a user-not-designer perspective, these topics need not wait for an upper-level database course, but to achieve sufficient depth, an introductory computer science course is appropriate and sufficient as a prerequisite.

CCS CONCEPTS

• **Information systems** → *Relational database model*; *Hierarchical data models*; *Semi-structured data*; *RESTful web services*; • **Networks** → *Network design principles*; *Application layer protocols*; • **Computer systems organization** → *Client-server architectures*;

KEYWORDS

Computer science education; CS2; data science; curriculum; data models; client-server

ACM Reference Format:
Thomas C. Bressoud and Gavin Thomas. 2019. A Novel Course in Data Systems with Minimal Prerequisites. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), February 27-March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287425

1 INTRODUCTION

The sources and forms of data, along with the demand for acquiring, processing, and analyzing that data are increasing at a prodigious rate[1, 3, 6]. From a curricular perspective, majors, programs, and courses are being designed to enable interested students to understand these topics and prepare themselves for the practice of these associated data skills[2, 5, 8, 14]. As the sources and forms of data increase, with a commensurate increase in the data providers and interfaces for accessing data, so too must our interpretation of what constitutes a data system[4].

The student audience for such curricular efforts is broadening as well. We identify three student constituencies:

Computer Science: students recognizing the value of acquiring and transforming data, and the synergy of this activity with the systems and algorithmic problem-solving aspects of the computer science discipline.

Data Science/Data Analytics: students pursuing these new and emerging undergraduate majors, who desire to build algorithmic and practical skills with an eye towards enabling data exploration, visualization, statistical analysis, and application of machine-learning techniques for use in coursework as well as in practicums and internships.

Multidisciplinary/domain-centric: students whose focus is on practical aspects of acquiring and transforming data for use in their own disciplines. These include students in social sciences as well as natural sciences, and often these students want to learn this material as a prelude to a "methods" course specific to their discipline.

Others[8] have called the union of these audiences the "data-centric" audience, and we extend this idea to define the set of data-related topics and the desired level of understanding the *data-aptitude* for these students. We see two primary dimensions to data-aptitude, corresponding to the *sources* and *forms* of data. Data sources range from local files to relational databases to network-based data providers. Data forms, provided by data sources, range from comma-separated-value flat files to the tables within a relational database to XML and JSON to unstructured data and HTML. Data sources may be further distinguished by the protection and privacy associated with the data they provide. Some data is explicitly open and freely available, while other data may be limited to use by particular applications, and still other data may be comprised of protected resources of one or more authorized resource owners.

Many of the topics associated with data-aptitude have homes spread through the computer science curriculum. In the ACM curricular guidelines [9], the Information Management (IM) knowledge area includes relational databases and data models; the Network Computing (NC) and Systems Foundations (SF) include aspects of client-server computing and the protocols upon which the APIs for accessing provider data are built.

In curricular guidelines [13], and in many actual majors [9], parts of this material can be found in a *Database Systems* course. Such courses are strong on the relational data model, emphasize normalization and design of schema to enforce desirable properties and constraints, and often include mathematical underpinnings of the model and systems design aspects allowing for efficient realizations of database systems.

Database systems courses typically require prerequisites of a CS1 class, an Intro to Systems/Computer Organization class, a Data

Structures class, and a class and/or material in discrete mathematics supporting sets and relations, and so the database systems class is most often not taken until a student is in the late stages of their educational career. In smaller institutions, based on rotations of class offerings, a student may not be able to take a database systems class at all. Being able to actually use the database-relevant material in projects or other classes is not possible. And because of its focus on the relational model and the systems design aspects of efficient database system realizations, it is too narrow to satisfy the needs defined for data-aptitude.

At the other end of the spectrum, and recognizing the needs of data-aptitude and the growing audience of data-centric students, recent efforts have focused on incorporating data-aptitude as part of the CS1 experience. At St. Olaf, Hall-Holt and Sanft have incorporated aspects of data acquisition and analysis along with statistics in an introductory "statistics-infused" CS offering [8]. Anderson et al. use Data Programming in a CS1 class [2]. More broadly, Kaplan has written about teaching computation to undergraduates across the natural and social sciences [10] with a data focus and has a book that uses R and Data Computation for introductory computer science/data science[11]. Earlier work by Sullivan also uses a data centric introduction to computer science and targets the broader non-majors audience [14].

There are some clear advantages to such CS1+data approaches. Students are introduced early to data-aptitude topics alongside algorithmic problem solving. By learning some of this material in a first course, students can bring this newfound knowledge to bear in downstream courses and projects. And clearly it helps to address the need of the growing population of the data-centric student audience. Additionally, having introductory computer science and data-aptitude in a single course has significant appeal on already-crowded majors, including new data science majors as well as students with a data focus in majors in the other natural and social sciences.

Unfortunately, such an all-in-one solution can come at a price: too much content for a single course. Students can feel overwhelmed when data-aptitude content and goals are added to the existing content and goals of an introductory computer science course. Students have difficulty in differentiating the competing aspects of what they are learning, from algorithmic problem solving, programming language syntax, separation of internal/memory resident data structures from external files and formats, statistics, and data acquisition through various means, from files to URL s to more complex networking-based communication exchange. There is also the significant danger that, for the data topics, we try to simplify by "teaching to an interface", focusing on the calls and arguments for a supporting package, and de-emphasizing the underlying systems principles and implementations that are beneath the surface.

To simultaneously solve many of the disadvantages of a late-curriculum and narrow database systems course as well as the content overload and potential lack of data-aptitude depth of a CS1+data approach, we advocate for an early data systems course with a prerequisite of a CS1 class. We define data systems broadly, including relational database systems as well as those systems providing data across the Internet in a client-server manner in the form of web service APIs, like REST (REpresentational State Transfer)[7]. This course differentiates and focuses on the understanding and *using* of these data systems, rather than the *design* of such systems, allowing us to shed much of the material that would be contained in a database systems course. This course would not replace the database systems course; rather, with only a 20% content overlap, it would provide students a foundational understanding of the topics at an earlier stage in their education.

This course employs *data models* as a framework for learning about the various forms of data, from a simple tabular model with appropriate constraints to the relational data model to a hierarchical data model as needed by our definition of data-aptitude. In the data sources dimension, the course follows a progression of data sources, starting the students with local files, moving on to a MySQL database server as representative of relational databases, and then covering more advanced client-server interaction over HTTP using provider APIs, and also extracting data from HTML and addressing protected resources. In this way, we feel we can cover the topics of data-aptitude in sufficient depth beyond a single CS1+data course, while keeping material broad and applicable to the wide range of data-centric students.

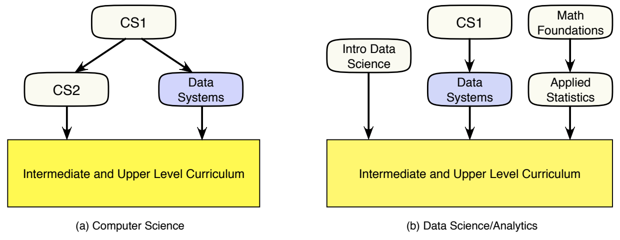

Figure 1: Curricular Context for Data Systems Course

In Figure 1, we show in (a) our proposed course in the context of the early part of a computer science major. Note that we are not replacing a program's CS2 course, but rather suggest a second course that could be taken concurrently with a traditional CS2. In summer of 2016, the Park City Math Institute held a three week program for Undergraduate Faculty on developing guidelines for a Data Science curriculum[5]; in (b), we show a permutation of that curriculum and show our proposed course within it. This fits well with the guidelines' desire for a second course in computer science centered around data acquisition and curation. One could also envision the two course sequence of CS1 followed by our proposed course as an excellent fit for the multi-disciplinary/non-major student constituency.

Our institution and department has designed and, in the 2017-2018 academic year, implemented such a course, supporting a broader systems experience for our computer science students and integral to our institution's emerging data analysis (DA) major. This new course is now required for both majors.

2 COURSE CONTENT

The Data Systems Architecture, as illustrated in Figure 2, enables us to see the elements of the course content. As introduced in Section 1, the two primary dimensions of course content involve the *sources* of data from data systems and the *forms* of that data. On the right side of the figure, we depict the primary data sources, that of local files, database systems, web servers, and providers exporting APIs by which their data may be accessed. Each of the

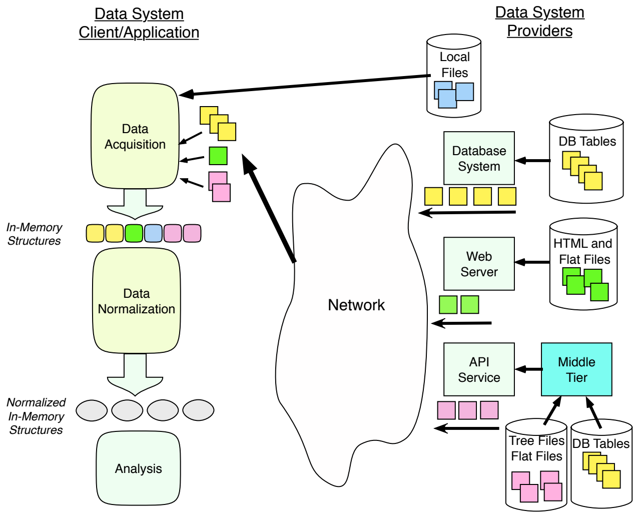

Figure 2: Data Systems Architecture

sources may deliver their data in one or more of the data forms covered in the course. The left side of the figure depicts the main phases of the data pipeline necessary for client applications that are on the consumer side of data systems. The process starts with data acquisition, which may come from local files, or over the network, and proceeds to data normalization, where the data is transformed to satisfy constraints and to put it into a form amenable for analysis.

2.1 Data Models

We use *data models* as the framework by which we cover the various forms and associated formats of data. A data model consists of:

- *structure*: typically one or more two-dimensional structures, or a tree structure,
- *operations*: given the structure, how do we extract, modify, and transform data in that structure, and
- *constraints*: within the structure, what are the legal data types, legal values for elements, and legal relationships between some elements and one or more other elements.

This course investigates three data models, starting with a simple tabular data model, progressing through the relational data model, and then covering the hierarchical data model.

Tabular Data Model. The tabular data model focuses on primarily single table datasets with row and column dimensions. In this model, we are interested in in-memory representations of tabular data with sufficient power in their operations to allow extracting and updating subsets of rows and columns, working with columns as vectors, and transforming from one tabular representation to another. In Python, we use the pandas module to provide such facility.

To enable effective analysis, which includes partitioning data, computing on partitions, and combining results[15], the data must conform to particular constraints, with variables representing columns, and rows defining appropriate functional dependencies between independent variables and dependent variables[1]. Data can be represented in tables in many different ways, but most do not conform to these constraints, and, in the tabular model, constraints are not enforced, but are merely a convention. So the course must teach recognition of violations of the model constraints, and how to transform into its normal form.

The course starts from CS1 topics of two-dimensional data structures using native Python data structures, like lists of lists and dictionaries of lists. We begin with basic line-by-line processing of flat delimited files as a way of building into the topic. This corresponds to the upper right part of Figure 2 with data acquisition from local files. Coverage of the model progresses through various techniques of transforming and normalizing the data[2], thus allowing for exploration/visualization and downstream analysis. Transformation and other complex operations are discussed algorithmically, to allow generalization to other languages and packages. At the end of this section, students are given a project with real-world datasets that they must understand and normalize to allow visualization and communication/interpretation of the underlying data.

Relational Data Model. One of the most important forms of data, and its associated data model, is the relational database system and model. Grounded in nearly 50 years of development and an important element of many larger systems, we cover this model from a primarily client/user perspective. This corresponds in Figure 2 to the second source on the right of the data system providers and, on the left, by using SQL as part of data acquisition and then transforming results into appropriate in-memory structures.

By using the tabular data model as a foundation, we start with projecting columns and selecting rows from single relations/tables, drawing parallels with the prior material, and seeing some of the differences of a declarative versus procedural style of operation. We then extend to multiple tables using join operations to see how the relational model uses constraints on foreign and primary keys to build one-many and many-many relationships.

Utilizing existing well-designed databases as examples, we cover, at an intuitive level, various aspects of sound database design. We use elements of the convention-based constraints of the tabular model to arrive at an intuitive understanding of third normal form. By using database schema, the students see the benefit of design preceding data population, and how a database system can enforce many of the desired constraints in this model, to avoid errors before-the-fact. Coverage of this section concludes with simple, but sound, database design, table creation, and row insertion for some existing datasets derived from the earlier section on the tabular data model.

The relational data model is reinforced and content synthesized in the student's second project, in which they have access to a few significantly large relational databases and conceive of and build queries of significant complexity, and interpret results.

Hierarchical Data Model. When we interact with data systems on the Internet, many providers structure and represent provided data in a flexible form based on a tree data model. Dominant specific examples include XML and JSON formats, but the tree inherent in an HTML document is another significant example. These correspond, in Figure 2, to the "Tree Files" and "HTML Files" in the lower two providers types on the right side of the figure. Data acquisition for tree formatted files can also be provided by local files.[3]

[1]Called *tidy data*[16] by the data science community.

[2]Variously called cleaning, wrangling, and munging by the data science community.

[3]We use the lxml module to support this data model.

Given a tree or subtree, the operations of this model include procedural tree traversal and the extraction of data (tags, text, and attributes) from the nodes of the tree. This model also gives us the opportunity, on the XML and HTML side, to look at declarative operations using XPath. We are also able to see how trees can represent both the notion of a single table, or a set of tables, all within a single file. Constraints in this model are either by convention, or can be explicit, using mechanisms such as Document Type Definition (DTD), or an XML Schema Definition (XSD).

Unstructured Data. Much of the time, data may not be structured in one of the covered forms, and a client, after data acquisition, must perform advanced pattern matching and extract fields of data out of one or more unstructured collections of data. To accommodate this reality, the course includes a unit on pattern matching and regular expressions. The focus is on recognizing the patterns that exist in unstructured data and understanding the tools by which the data can be extracted, regex capture groups, for instance. Using these tools, students build tabular data structures from unstructured data.

2.2 Data Sources

As described above in both the tabular data model and the hierarchical data model, we begin with local files to build our understanding of the model before we move into more complicated means of data acquisition. We progress from there, during coverage of the relational model, to an external database management system as the source of the data. We use a MySQL server available on the local network and shared by all the students in the class. Each student has their own login, within which they can CREATE and INSERT in tables, and each has SELECT access to numerous databases and tables used in the class. Students learn the various parts of the URL employed as a *connection string* to establish communication with the database and are introduced to basic networking ideas like a machine and its address, and a protocol by which a client makes requests that are processed and result in a response by the server. They also see, by example, the notion of data resources having an ownership (the user) and protection for these owned resources.

Our next source of data are web servers, corresponding to the middle provider type on the right side of Figure 2. Students learn the basics of the HyperText Transfer Protocol (HTTP) as a network client that must specify the machine and protocol for a connection over the network. They then must construct and issue requests (GET and POST) and be able to retrieve and understand the constituent parts of the meta-data and data of the response. This type of data source can provide delimited flat files, HTML files, or even XML/JSON files. Given this data source, and combining with the knowledge from the hierarchical data model, students combine these to understand web scraping as an application of these ideas.

With the foundation of HTTP, we then turn to API-based data providers[4], as represented by the bottom right of Figure 2. We extend our understanding of the resource path part of a URL in an HTTP GET or POST to give an *endpoint* of a provider, and the use of URL parameters and HTTP header key/value pairs to parameterize requests made of an endpoint. We can then extend our ideas about authentication and authorization to allow for client applications as an entity that can be authenticated and resources from a provider as being "owned" by a principal, who may not be the same as the writer of the application. We use OAuth with real-world providers to facilitate this material.

[4]We focus on RESTful APIs as appropriate for students in this class

2.3 Classroom Approach

Because of the importance of visualization in exploring acquired data, asking interesting initial questions about a dataset, and communicating such pre-analysis to an audience. We employ the use of Tableau, software available both online and in desktop version, as a powerful tool for generating such visualizations. Once data is in a normalized form from our tabular data model, it is relatively straightforward to build quality visualizations, and the students in this class employ Tabeau in all of their projects.

We use real-world datasets for all projects and most of the examples and practicums in the course. There is an abundance of data, and one of the goals of the course is to understand how to take "messy" real-world data and, in a clear and systematic way, normalize it into a usable form.

In addition to the projects associated with the tabular and relational data models, we use a synthesis final project. Students must employ most of the aspects learned in the class, from using multiple providers, and at least one of those providers must hold protected resources, and the students must engage OAuth to gain authorized access to these resources. Students must build a simple, sound database with data derived from their data acquisition and data normalization. Finally, students must visualize and ask interesting questions of their data.

The topics covered by week in the semester are give in Table 1:

Week	Topic Coverage
1	Python Foundations and Review
2	Tabular Model Structure and Operations using pandas Module
3	Tabular Constraints/Tidy Data Recognition and Transformation
4	Relational Database Access and Single Table SQL
5	Relational Database Multiple Table Joins, Grouping, and Aggregation
6	Programming in sqlalchemy and Sound Database Design
7	Regular Expressions; Networking Fundamentals
8	HTTP Definition and use with the requests module
9	Hierarchical Model - Structure, Operations, JSON Intro
10	XML and XPath using lxml
11	HTML and Web Scraping
12	Authentication, Authorization and OAuth2
13	Basic REST API Services
14	Authorized REST API Services

Table 1: Course Topics by Week

3 INAUGURAL YEAR STUDENT AUDIENCE

In the 2017-2018 academic year, we offered a total of four sections of data systems, each capped at 24 students – three sections in Fall 2017, and one in Spring 2018. All sections filled, and, after initial-semester attrition, we ended with a population of 64 students in Fall 2017 and 26 students in Spring 2018. From an initial class survey, we describe the basic demographics of the student audience in this section, along with their self-reported confidence/proficiency in some of the topics assumed from the CS1 prerequisite.

Over the four sections, 52% of the students were sophomores, 41% were juniors, and 7% were seniors. The number of juniors and seniors is expected to reduce as we move forward, with fewer upperclass students needing to satisfy this new requirement. The female/male ratio in this initial year was 20%/80%, but our data analytics offerings have been showing more diverse appeal, and so we hope our ratio of females taking this course will increase.

Because of the DA major introduction in the prior academic year, and the addition of this course as a required in the computer science major, there was a backlog of demand, including from students further along in their program. We expect, in subsequent offerings, for the class year distribution to shift toward predominantly sophomores, as opposed to the 52% sophomore and 41% juniors seen in this first year.

Majors, as a percentage of the class population, differed significantly between the two semesters, as can be seen in Figure 3. In the fall, nearly 60% of the students were computer science majors, and only 32% were data analytics majors. In the spring, these ratios swapped. In Section 1, we identified three student constituencies, the CS students, the data analytics students, and the multidisciplinary students. Our data-analytics program emphasizes interdisciplinary integration, and so the data analytics majors shown here are really a mix between the data analytics and multidisciplinary constituencies. We do hope that demand from students in other majors ("Other" in the graph) will increase as we move forward.

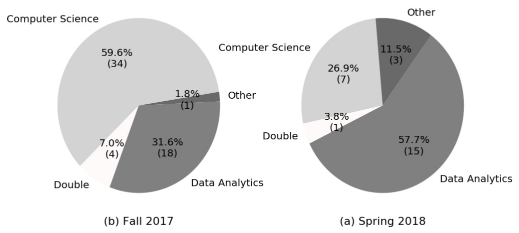

Figure 3: F17 and S18 Distribution by Major

We decided to use Python in a single language approach to this data systems class, because it is taught in our CS1 courses. This allows students to focus on the topics we present, rather than needing to also learn a new language. In the initial survey, designed to understand sufficiency in CS1 topics and to aid in some initial review, we asked students their self-assessed level of understanding in a number of the Python programming topics. In particular we asked students to assess their proficiency in the usage and methods of strings, lists, and dictionaries, the usage and construction of two-dimensional data structures, and creation of basic graphs through plotting packages. Figure 4 shows the results of these survey questions, where students ranked their proficiency on a 1 to 7 scale (bigger is better).

On the positive side, for all six topic areas, the significant majority of the distribution is at ratings of 5-7. Students clearly felt most proficient in lists and strings, followed by dictionaries and objects. The distribution of proficiency on graphics has greater spread, as does two-dimensional representations. Given coverage late in our CS1 offerings, this is not unexpected.

4 EVALUATION AND STUDENT FEEDBACK

We employed multiple surveys to gather student feedback on interest and efficacy of this course. Students were given an initial survey, to gather demographics and solicit feedback on proficiency

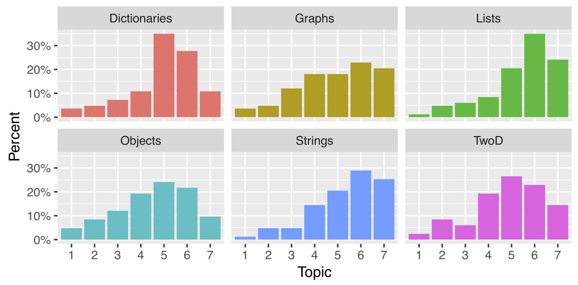

Figure 4: CS1 Topic Proficiency

in CS1 topics and opinions on the expected usefulness of the data systems topics. Students were also given surveys mid-semester and end-semester, asking about learning outcomes in each of the covered topics. Finally, we conducted a follow-up survey aligned near the end of students' summer experiences to help understand applications of the course materials to courses, internships, and projects after the class was over.[5] Out of 90 total students, 85 responded to the initial survey, 72 to the mid-semester survey, and 64 to the end-semester survey. The post-survey only went out to students who agreed to be contacted for follow-up, and we received 29 responses from 60 such students.[6]

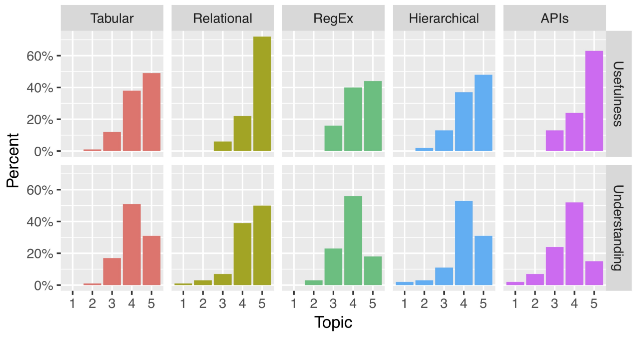

Figure 5: Usefulness and Understanding by Topic

Figure 5 shows percentages of respondents rating, on a 1-5 scale, for each of five central topic areas. In the top row of the figure, students rate their expectation on the usefulness of the topic, prior to coverage. In the bottom row, students self-assess their understanding of the topic material after coverage. The first, second, and fourth column correspond to the data models covered in the course. The column headed RegEx corresponds to coverage of regular expressions, addressed in the course for handling unstructured data. The final APIs column encompasses both simple web server access, as well as more complex authenticated and authorized REST APIs. Even before coverage, students' expectations were high for the usefulness of all of these topics, with Relational and APIs as the perceived "most useful". Further, their self-assessed level of understanding is quite good across all five topics. Using total percentage of 4 plus 5 ratings as a metric, Relational is at the top with 89% 4s and 5s, Hierarchical, Tabular, and RegEx are in the middle at 84%, 82%, and 74%, respectively, and understanding of APIs at 67% 4s

[5]All surveys were anonymous, were with informed consent, were completed without the instructor in the room, and students were reminded that they always had the choice to not take the survey.
[6]For a solicitation to students during their summer, we felt a nearly 50% response rate quite good.

and 5s. These are very positive outcomes of the course, and support our thesis that this approach of a broad-based introduction to data systems is sufficiently accessible given the foundation of a CS1 course.

The post-survey was undertaken after receiving unsolicited feedback from students describing how they had employed data systems course learning in internships, personal projects, and research experiences after the course was completed. To avoid anecdotal and biased reporting, we designed and sent the post survey out to the broader audience of students from all sections asking, in particular, whether students had employed what they had learned in the course, if so, in what kinds of post-course activity, and then, to rate the importance of the data systems class topics in the context of their post-course work. Of the 29 respondents, 27 (93%) indicated use of these topics in some post-course project or activity, and most applied their learning in multiple activities. Of these, 44% used their skills in projects for other courses, 60% in internship or REU experiences, and an amazing 78% in personal or outside projects.

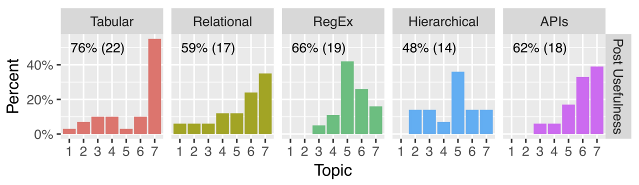

Figure 6: Post Project Importance by Topic

In Figure 6, we depict both the usage of each of the course topic areas and their importance in the post-course project. The usage is shown in each topic column as an annotated percentage and count (out of 29) that employed that topic area in their project. Given that a student used the topic area, we asked how important the topic was to their work. We can see that 76% of the students used Tabular (structure and cleaning/normalization) in their projects and that it was considered very important in the work. Regular expressions were used by a significant number of students, but their importance would appear to have been secondary. APIs, when used, were quite important. And the hierarchical model was used least at 48%, with more of a spread in importance. These numbers clearly support both the selection of content in our course design and, by the fact that so many students have integrated their learning so quickly into work following the class, both within and outside their academic endeavors, clearly argues for such a course being early in their curriculum.

Category	1	2	3	4	5		1	2	3	4	5
Effort	0	2%	4%	45%	49%		0	0	5%	25%	70%
Knowledge	0	2%	9%	32%	57%		0	0	0	20%	80%
Challenge	0	0	13%	40%	47%		0	0	10%	30%	60%
Overall	2%	4%	28%	42%	25%		0	0	5%	35%	60%

Table 2: Course Evaluation Results

Table 2 presents the results of the standard university course evaluations for the two offered semesters. Ratings are on a 1 to 5 scale and, for comparison, we use percentages of counts for each category. The number of evaluation respondents was 52 for the fall (on the left) and 20 for the spring (on the right). In both semesters, students were challenged by the material of the class, and significant effort was required. The knowledge and overall ratings differ between the two semesters, with both having greater spread, and a few students giving the fall instance of the course as low as a 1 or 2. The overall ratings and knowledge were remarkably good for the spring instance of the class. From the instructor's point of view, the course improved between the two semesters. Based on the fall feedback, some material was reduced and refined, and there were some changes to the ordering of topics as well.

In the post-survey, we asked students, with the benefit of hindsight after the course was complete, how they would compare their learning in this course to other courses at the university, and how they would rate the course overall. The course comparison question received 14 (48%) ratings of 6 and 12 (35%) ratings of 7, on a 7 point scale. The hindsight rating of the course overall received 13 (45%) ratings of 4 and 14 (48%) ratings of 5, on a 5 point scale.

When asked in the post-survey to describe their projects and how they had used what they learned, we received a rich set of answers. Some of these internship experiences include students working in physics, actuarial science, data science, banking, software engineering, academic research, and even for the government. Personal projects include stock market analysis, Chicago speed camera analysis, creating Twitter bots, personalizing Spotify, IMDb movies analysis, ESPN baseball analysis, and many more. The interesting and diverse responses, and the excitement conveyed by the students has been perhaps the most rewarding aspect of this undertaking.

5 CONCLUSIONS

The course, in design and in result, has been very successful, but it also has had its challenges. First and foremost involves the resources for students to use in the class. There is no single textbook that covers our desired topics. Books on database systems are too narrow and contain much material that we do not cover. Books on Python modules teach skills focused on a particular interface and lack the framework of the data models or the generality to see how a new interface or different language might accomplish the same goals. Instead, we relied on many online resources in combination with a significant number of Jupyter notebooks [12] written to both engage the students and to convey material. Some students, with good note-taking and skills at managing this disparate set of resources, adapted well, but for others it was a significant challenge. The second biggest challenge was trying to "do too much" in the fall version of the class. We reduced student workload somewhat, dropped a few secondary topics, and reordered slightly, to allow some of the more interesting *data sources* dimension of the class to occur earlier in the semester. These adjustments made a significant difference for the spring semester.

In summary, we have designed and developed a new data systems course that addresses the needs of our curriculum, introducing systems topics earlier and allowing downstream use of the knowledge gained. The course addresses the needs of the growing data-centric audience with a foundation of data-aptitude that can be implemented in institutions with data science programs or traditional computer science programs. We cover a significant range of topics in both the forms and the sources of data. Finally, we avoid content-overload and can attain greater depth by relying on a CS1 prerequisite.

REFERENCES

[1] 2017. *White paper: Top Ten Big Data Trends*. Technical Report 849188. Tableau Software.

[2] Ruth E. Anderson, Michael D. Ernst, Robert Ordóñez, Paul Pham, and Ben Tribelhorn. 2015. A Data Programming CS1 Course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. ACM, New York, NY, USA, 150–155. https://doi.org/10.1145/2676723.2677309

[3] Randy Bean. 2015. Making the Case for the 'Long Tail' of Big Data. *Wall Street Journal* (Aug 2015).

[4] Peter Cornillon. 2005. What Is a Data System, Anyway? *EDUCAUSE Review* 40, 2 (March/April 2005), 10–11.

[5] Richard D. De Veaux, Mahesh Agarwal, Maia Averett, Benjamin S. Baumer, Andrew Bray, Thomas C. Bressoud, Lance Bryant, Lei Z. Cheng, Amanda Francis, Robert Gould, Albert Y. Kim, Matt Kretchmar, Qin Lu, Ann Moskol, Deborah Nolan, Roberto Pelayo, Sean Raleigh, Ricky J. Sethi, Mutiara Sondjaja, Neelesh Tiruviluamala, Paul X. Uhlig, Talitha M. Washington, Curtis L. Wesley, David White, and Ping Ye. 2017. Curriculum Guidelines for Undergraduate Programs in Data Science. *Annual Review of Statistics and Its Application* 4, 1 (2017), 15–30. https://doi.org/10.1146/annurev-statistics-060116-053930 arXiv:https://doi.org/10.1146/annurev-statistics-060116-053930

[6] Peter ffoulkes. 2017. *White paper: InsideBIGDATA Guide to The Intelligent Use of Big Data on an Industrial Scale*. Technical Report. insideBigData. https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/

[7] Roy T. Fielding and Richard N. Taylor. 2000. Principled Design of the Modern Web Architecture. In *Proceedings of the 22nd International Conference on Software Engineering*. IEEE, 407–416.

[8] Olaf A. Hall-Holt and Kevin R. Sanft. 2015. Statistics-infused Introduction to Computer Science. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. ACM, New York, NY, USA, 138–143. https://doi.org/10.1145/2676723.2677218

[9] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, New York, NY, USA. 999133.

[10] Daniel T. Kaplan. 2004. Teaching Computation to Undergraduate Scientists. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '04)*. ACM, New York, NY, USA, 358–362. https://doi.org/10.1145/971300.971424

[11] Daniel T. Kaplan. 2015. *Data Computing: An Introduction to Wrangling and Visualization with R*. Project MOSAIC.

[12] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt (Eds.). IOS Press, 87 – 90.

[13] Task Group on Information Technology Curricula. 2017. *Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology*. ACM, New York, NY, USA.

[14] David G. Sullivan. 2013. A Data-centric Introduction to Computer Science for Non-majors. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. ACM, New York, NY, USA, 71–76. https://doi.org/10.1145/2445196.2445222

[15] Hadley Wickham. 2011. The Split-Apply-Combine Strategy for Data Analysis. *Journal of Statistical Software, Articles* 40, 1 (2011). https://www.jstatsoft.org/v040/i01

[16] Hadley Wickham. 2014. Tidy Data. *Journal of Statistical Software, Articles* 59, 10 (2014), 1–23. https://doi.org/10.18637/jss.v059.i10