DATA MINING STUDENT ACTIVITY PATTERNS IN AN INTERACTIVE ACTIVITY-BASED STEM LEARNING ENVIRONMENT

A Thesis

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science

---

by

SUSAN E. BROWNAWELL

Dr. Grant Scott, Thesis Supervisor

May 2020

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

DATA MINING STUDENT ACTIVITY PATTERNS
IN AN INTERACTIVE ACTIVITY-BASED
STEM LEARNING ENVIRONMENT

presented by Susan E Brownawell,

a candidate for the degree of master of science,

and hereby certify that, in their opinion, it is worthy of acceptance.

Professor Grant Scott

Professor Curt Davis

Professor Tim Matisziw

ACKNOWLEDGEMENTS

First, I would like to express my appreciation to Dr. Grant Scott, my thesis advisor, for his support and encouragement during my master's studies, for suggesting this topic of research, and for his constant availability and feedback as I completed this work. I also would like to thank the other members of my thesis committee, Dr. Curt Davis and Dr. Tim Matisziw, for their support and insightful suggestions and comments.

I would also like to recognize and thank the technical team supporting the PSDS program who helped me get oriented and provided technical tools and access used in the completion of this thesis. Andrew Craighead developed the grapi library and introduced me to the Graylog system. Alex Lasley was always ready and willing to solve my permissions and other issues and to answer my questions about the systems installation and configuration. Will Starms wrote the patch for the T R A K custom log messages and introduced me to the ZeroMQ library for messaging in Jupyter, a major puzzle piece for understanding the most efficient way to develop learning analytics for Jupyter.

TABLE OF CONTENTS

ILLUSTRATIONS

Figures

Tables

TABLE OF TERMS

Category	Term	Definition
PSDS Activity Type	*Lab*	A Jupyter notebook which leads the students interactively through concepts and skills.
	Practice	A Jupyter notebook where students must supply code themselves, but which provides instructor guidance.
	Exercise	A Jupyter notebook in which students apply what they have learned previously by writing independent code.
	Final Project	A Jupyter notebook in which students synthesize and apply what they have learned in previous modules of a course.
Message Features	**user**	The person executing the **notebook**. Student usernames follow the pattern aaa.####.
	notebook_path	The full path of a notebook, which can be decomposed into **class**, **user**, **notebook_type**, and **notebook**. Example: “PSDS2200OP3-2_aaa.1234/Day4/labs/Chi-Squared.ipynb”
	class	An instance of a course, specified by the course name, the contract period, and the number of the instance for that contract period. For example: PSDS 3300 OP3-2 written as PSDS3300OP3-2. Found in the **notebook_path**.
	day	The PSDS instructional module a **notebook** is in, extracted from the **notebook_path**.
	notebook_type	The activity type of the notebook found in the **notebook_path** at the next level up from the **notebook**. Example: in “PSDS2200OP3-2_aaa.1234/Day4/labs/Chi-Squared.ipynb”, the **notebook_type** is “labs”.
	notebook	The name of the Jupyter notebook with the extension .ipynb, which can be extracted from the **notebook_path**.
	timestamp	The date and time of a log message
	code	The code within a code cell that is executed in an EXECUTE_INPUT **event_type** but only logged in a TRAK message, and not extracted as a feature at this time.
	kernel_id	The unique identifier for a kernel, found in a path after "kernels/”,
	log_type	A higher-level type assigned to a message based on labels or keywords found in the message.
	event_type	A sub-type of message based on keywords found in the message or an interpretation of the message.

Category	Term	Definition
Jupyter Log **log_type**	KERNELMANAGER	A tag appearing in some log messages, marking a class in the Jupyter client that manages a single kernel. Log entries tagged with KERNELMANAGER include "execute", "status", and "error" **event_types**.
	TRAK	A label for PSDS custom logging for EXECUTE_INPUT log events. As currently configured, the log message will contain the **user** (who), the **code** (what), the **notebook_path** (where), and the **timestamp** (when).
	PROXY WEB	Some log messages tagged "ConfigProxy" contain the words "PROXY WEB". **event_types** associated with these messages include "load", "kernels", "checkpoint", "terminals", "save", and "error".
	PROXY WS	Some log messages tagged "ConfigProxy" contain the words "PROXY WS". **event_types** associated with these messages include "kernels" and "terminals".
Jupyter Log **event_type**	SAVE	A PROXY WEB event recorded when the contents of a notebook are saved. These messages contain a **notebook_path** followed by "?content=0&_=".
	LOAD	A PROXY WEB event recorded when a notebook is opened in the browser. These messages contain a **notebook_path** followed by "to http".
	EXECUTE_INPUT	Some log messages of **log_type** KERNELMANAGER contain "execute_input". These are included in the EXECUTE **event_type**.
	EXECUTE_RESULT	Some log messages of **log_type** KERNELMANAGER contain "execute_result". These are included in the EXECUTE **event_type**.
	EXECUTE	An **event_type** to include messages containing either “execute_input" or "execute_result"
Graylog Extractors	execute_user	Extracts the **user** from certain Jupyter messages.
	execute_course	Extracts the **class** from certain Jupyter messages.
	execute_notebook	Extracts the **notebook** from certain Jupyter messages.
Graylog API Request Arguments	*query*	The query sent to Graylog, expressed in Lucene syntax.
	fields	The fields to be returned from Graylog.
	from	The starting time of the time window to search Graylog.
	to	The ending time of the time window to search Graylog.
	limit	The maximum number of results to return from Graylog.

Category	Term	Definition
Graylog API Response Keys	query	The query that was sent to Graylog in the request.
	built_query	The Elasticsearch query that Graylog built from the request query.
	used_indices	The indices used that contain the results for the query.
	messages	The collection of messages returned by Graylog. The number of messages returned is capped by the *limit*.
	fields	The fields returned for the query results.
	time	The date and time of the request.
	total_results	The total number of log messages found for the time frame of the query (not the number returned.)
	from	The starting time of the time window for the Graylog search.
	to	The ending time of the time window for the Graylog search.
	decoration_stats	Decorators allow you to alter message fields during search time automatically. If decorators are configured for a stream, this key will return stats for those used in the search.

ABSTRACT

Jupyter Notebook is gaining in popularity for STEM instruction and activity-based learning. This platform for sharing interactive documents via a web interface allows instructors to combine a variety of media together with interactive and editable code, providing rich opportunities for an active learning pedagogy. Other online learning environments, such as Canvas and Moodle, provide or integrate learning analytics for the use of administrators, educators, and students to improve learning outcomes; however, these platforms lack the rich learning environment of Jupyter Notebook. Also, with increasing interest in online learning, research communities have arisen for Learning Analytics and Educational Data Mining. Unfortunately, these research communities have not yet begun to address the Jupyter Notebook learning environment.

The University of Missouri College of Engineering offers a Program of Study in Data Science (PSDS) under contract with the National Geospatial Intelligence Agency (NGA.) This program is delivered online, making heavy use of Jupyter notebooks served by JupyterHub for active engagement with course content. The PSDS infrastructure uses the Graylog log management program to collect Jupyter logs, which are stored in an integrated Elasticsearch document store for a period of months. The PSDS program provides an excellent case study for a proof-of-concept in applying learning analytics to the Jupyter learning environment.

This thesis consists of two major parts. (1) Mining the Graylog system to extract useful log messages, transformation of those messages into student-activities features, and loading the data into a PostgreSQL database for long-term storage. (2) Developing a variety of visualizations of student activity for administrators, instructors and students. The pedological structure of PSDS courses allows unique insights into student engagement with the course material. Finally, recommendations are made for the development of a more comprehensive logging system and additional analyses that could be performed.

Chapter 1: Introduction

Professors and instructors carefully prepare educational materials with a view toward building student understanding and skill proficiency. But how do students make use of those materials? Do they engage thoughtfully with the assignments, or do they skip over ungraded tasks and take shortcuts with evaluated work? Does engagement with preparatory assignments enable success on subsequent exercises, or is the material too hard or too easy? Does skipping preparatory assignments lead to students beating their heads against subsequent exercises with little progress for the time invested? Are students focusing most of their time on the most important learning objectives, or are they sidelined by less important objectives or technological impediments?

These questions apply for any course delivery system, but activity-based, technology-mediated learning allows new ways of measuring student engagement with the material, providing insights previously hidden to instructors.

1.1. Active Learning with Jupyter Notebook

Active learning is a process where students are not passive receivers of knowledge, but active participants discovering and constructing knowledge by engaging in thinking and questioning, problem solving, and higher order tasks such as analysis, synthesis, and evaluation [1], [2]. A large meta-analysis of active learning in higher education STEM courses showed that active learning methods increase student performance on assessments, especially on concept inventories, and result in lower course failure rates [2].

In an online environment, active learning can be accomplished via interactive media that intersperse expository material with student activities designed to discover knowledge, forming a dialog between instructor and student. Jupyter Notebook is such an environment, allowing the sharing of interactive documents via a web interface. These notebooks are composed of cells that can contain code, code output, explanatory text, links to further (and deeper) information, visualizations, static images, and equations [3].

Educators find the ability to combine code and text in an interactive environment invaluable in STEM education. For instance, engineering and architecture students experiment with provided code, seeing for themselves the effect changing the value of different parameters has on a model [4]. The professors of a computer science high performance computing course appreciated the ability of students to try out examples and develop a deep understanding of their code, as well as the ability for both the instructor and student to integrate different materials into one document for flexibility in learning and review [4]. A mechanical engineering professor finds that students can tackle open-ended design problems using active learning in Jupyter notebooks and enter the Analyze, Synthesize, and Evaluate stages of Bloom's taxonomy, moving beyond the Remember, Comprehend, and Apply stages [5].

Jupyter Notebook is particularly popular in data science and data science education [6]. Lorena Barba, a mechanical and aeronautical engineer at George Washington University, praises the ability to document the interactive nature of data exploration as a type of conversation. She states "IPython notebooks (a precursor to Jupyter notebooks) are really a killer app for teaching computing in science and engineering" [6]. Jupyter Notebook can be served to multiple concurrent users via JupyterHub, making it an ideal activity-based learning solution.

1.2. Learning Analytics

With increasing interest in data analytics in all fields, it is no surprise that research communities devoted to applying analytics to education and learning have emerged. The International Educational Data Mining Society defines Educational Data Mining (EDM) as: "... an emerging discipline, concerned with developing methods for exploring the unique types of data that come from educational settings, and using those methods to better understand students, and the settings which they learn in." The Society for Learning Analytics Research defines learning analytics (LA) to be "… the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs." These two communities have significant overlap, but the LA community perhaps has more of a focus on informing instructors and learners [7].

Use cases for learning analytics include improving the quality of teaching, increasing student retention, and informing students of their progress to enable them to adapt and improve. Other possible use cases include identifying differential outcomes among sub-groups of students in order to improve student success, as well as enabling adaptive, personalized learning [8].

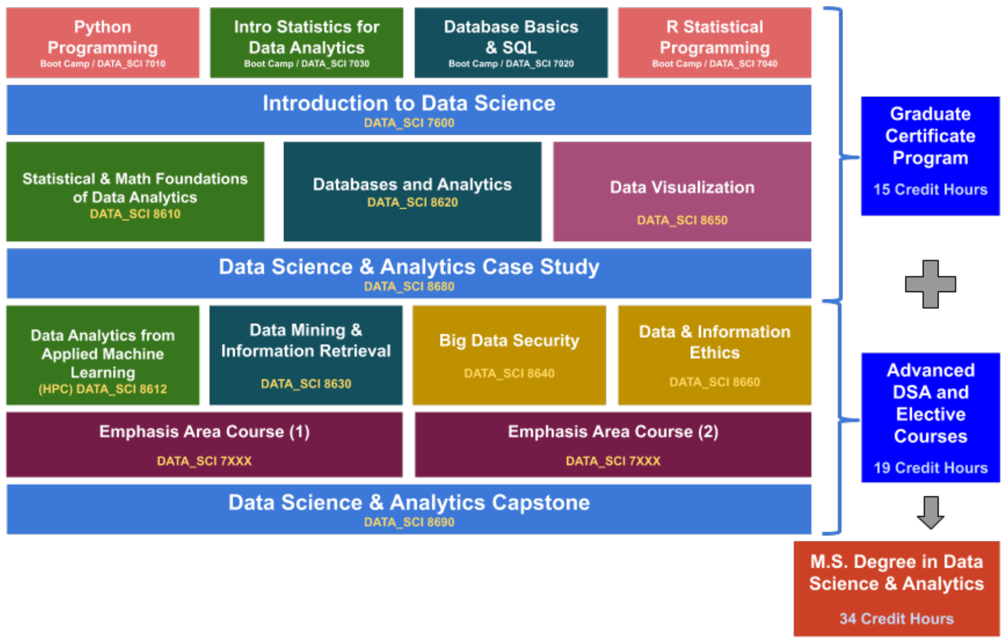
Online learning naturally lends itself to data collection through logging of activity. Third-party or integrated tools for data mining and analytics presentation are available for many, if not most, learning management systems (LMS) and virtual learning environments (VLE). Canvas, for example, offers an analytics dashboard for viewing account or course activity. For courses, plots are available for aggregated activity by date, the percentage of on-time, late or missing submissions for each assignment, and the grade distribution for all assignments. Plots are also available for an individual student's page views, participation, assignments, and current score [9]. This is typical of the high-level view usually offered, providing little insight into the learning that is occurring, the understanding attained on particular concepts, or where difficulties lie [10].

There is research in the literature that strives to illuminate online behaviors in LMS and VLE that are associated with learning outcomes by analyzing log data. For instance, in Jo et al., certain time management conceptual constructs, well-established as factors in student performance, are mapped to proxy variables (total login time, login frequency, and regularity of login interval) to predict student performance in a commercial e-learning course [11]. Nguyen et al. analyze log data in Moodle to investigate how student timing of engagement aligns with the instructors' learning design and how levels of performance and learning design relate to different study patterns for one, largely online module in a distance-learning course [12]. Yu and Jo use multiple linear regression analysis to develop a model for predicting students' academic achievement for an undergraduate, face-to-face course utilizing Moodle. Their six-predictor model is based on total logged time, number of logins, regularity of learning interval in the LMS (i.e., the standard deviation of the average login time), number of posts responding to the instructor, number of posts responding to peers, and downloads of content [13].

However, a literature search for "Jupyter learning analytics" and similar search terms in Google scholar and other journal search providers failed to uncover learning analytics research applied to the Jupyter environment. Results that seemed relevant turned out to use Jupyter notebooks to perform analysis on LMS systems such as Moodle. This absence in the literature provides a great opportunity, because unlike students using LMS systems, students using Jupyter notebooks are engaging in active learning. In an LMS, students read or download documents, videos, and slideshows, engage in on-line discussions, read and send private messages, and upload work completed off-line, as well as view administrative information such as deadlines, grades and syllabi. The nature of analytics with an LMS is necessarily high-level, because these activities do not lend themselves to deeper insights. A student participated in an activity or not; they made a certain number of posts, they spent a certain amount of time online, etc. Logs from Jupyter notebooks can provide deeper insights into student activity patterns. Did they run instructor code or just read the content? Did they experiment with the material themselves? Did they spend little time in a notebook, hurrying through, or did they engage more thoughtfully? Jupyter notebooks are rising in importance and popularity in STEM education due to the interactive, active learning environment they provide. LA and EDM applied to Jupyter logs have the potential to increase the learning advantage of Jupyter even more, by providing insights on the effectiveness of instructors' lesson and course design and providing feedback to both instructors and students on the effectiveness of students' learning approaches.

1.3. Program of Study in Data Science (PSDS) Program

In the summer of 2016, the University of Missouri College of Engineering was awarded a five-year contract to deliver a data science education program for the National Geospatial-Intelligence Agency (NGA). The Program of Study in Data Science (PSDS) is a collaboration between the Data Science and Analytics online master's degree program (DSA) from the MU Institute for Data Science and Informatics, and the Center for Geospatial Intelligence (CGI) [14]. The PSDS program is an adaptation of the DSA program. The PSDS program consists of 16 courses (see Figure 1), ranging from introductory boot camps for basic skills in programming, database access, and statistics; through intermediate courses in applied statistics, and computational methods; to advanced courses in visualization, machine learning, data mining, cloud computing and more.



Python Programming
Boot Camp / DATA_SCI 7010
Intro Statistics for Data Analytics
Boot Camp / DATA_SCI 7030
Database Basics & SQL
Boot Camp / DATA_SCI 7020
R Statistical Programming
Boot Camp / DATA_SCI 7040
Introduction to Data Science
DATA_SCI 7600
Statistical & Math Foundations of Data Analytics
DATA_SCI 8610
Databases and Analytics
DATA_SCI 8620
Data Visualization
DATA_SCI 8650
Data Science & Analytics Case Study
DATA_SCI 8680
Data Analytics from Applied Machine Learning
(HPC) DATA_SCI 8612
Data Mining & Information Retrieval
DATA_SCI 8630
Big Data Security
DATA_SCI 8640
Data & Information Ethics
DATA_SCI 8660
Emphasis Area Course (1)
DATA_SCI 7XXX
Emphasis Area Course (2)
DATA_SCI 7XXX
Data Science & Analytics Capstone
DATA_SCI 8690
Graduate Certificate Program
15 Credit Hours
Advanced DSA and Elective Courses
19 Credit Hours
M.S. Degree in Data Science & Analytics
34 Credit Hours

Figure 1. The PSDS course roadmap for both the Graduate Certificate and the MS degree. Figure provided by the PSDS program.

In both the DSA master's program and the PSDS professional development program, data science is presented as a collaborative, iterative process. This process is depicted in Figure 2 where diverse data repositories feed into a collaborative data science workflow that feeds back into the data repositories, as well as forward into analytical and predictive modeling; all towards building decision making products. Both programs employ best practices in adult-learning pedagogies and repeatable science [14]. Jupyter notebooks are particularly well-suited to this philosophy and approach and are employed for most courses.

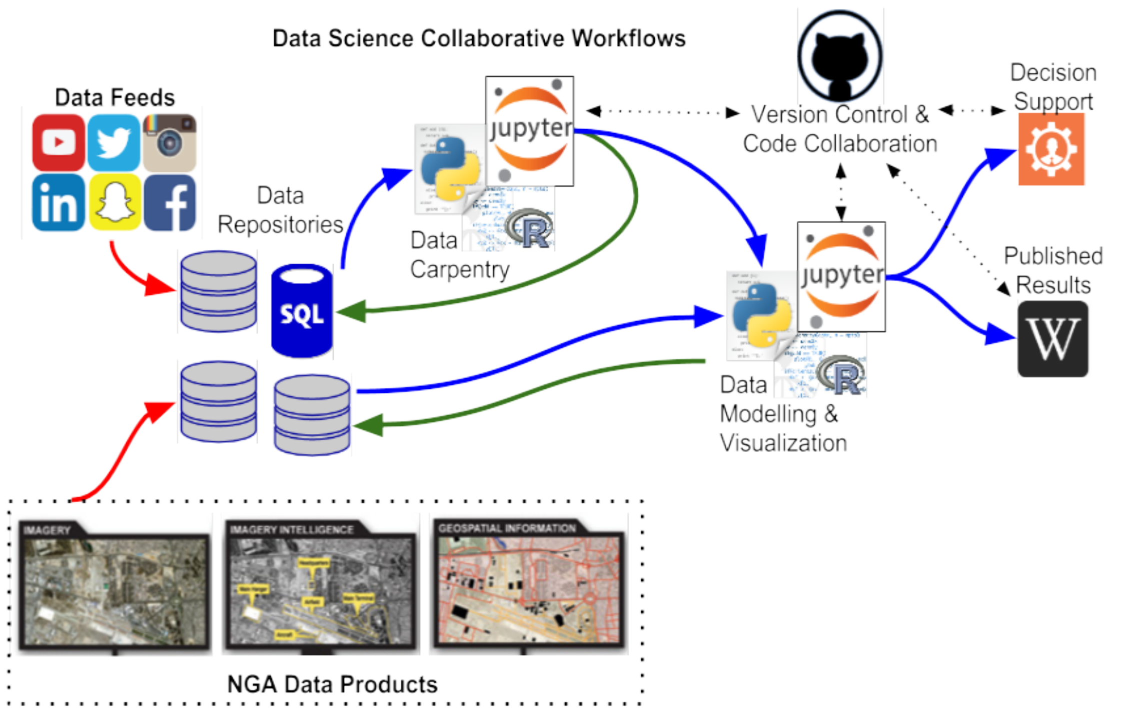


Figure 2. Data science collaborative workflow from PSDS course material. Figure provided by the PSDS program.

A PSDS course is an adaptation of a DSA course. A typical DSA course is divided into eight single-week modules, progressing through three sets of notebooks in each of the first seven modules. A module begins with readings or video lectures and *LAB* notebooks which lead the students interactively through the concepts and skills of the module. This is followed by *PRACTICE* notebooks, in which students must supply code themselves but which provide instructor guidance. The module ends with *EXERCISE* notebooks in which the students apply what they have learned in the previous *LAB* and *PRACTICE* notebooks (as well as prior modules and courses) by writing independent code. The course is capped in module 8 by a *FINAL PROJECT* where students synthesize and apply what they have learned in the previous modules. Introductory boot camp courses are a little different; they typically exclude the *EXERCISE* notebooks, having only readings, videos, *LAB* and *PRACTICE* notebooks, and a *FINAL PROJECT* presented as a *PRACTICE*.

In contrast to an 8-week DSA course, a PSDS course is typically delivered over five instruction days of concentrated coursework and a *FINAL PROJECT*, spread out across two and a half to four weeks' time. Most courses are taught entirely remotely, but some courses are blended, with a few days led by an instructor at an NGA site. Each **day** corresponds generally to two modules of a DSA course, with the *FINAL PROJECT* assigned on the fifth **day**. PSDS students are expected to work on the course during their professional workday, when faculty and staff are contracted to be available for questions and support. Students have the option to apply for university credit for PSDS courses they enrolled in. The accelerated nature of the program presents logistical issues for accurate and timely university enrollment, relying on students performing enrollment steps in a timely manner after the course has started. In the case of PSDS, the enrollment sequence is usually *"NGA enroll" → attend → assess → "MU enroll" → pay.*

The PSDS Program provides an excellent case study for applying learning analytics to the Jupyter learning environment. Not only can we monitor the time spent in active learning in the notebooks, we can use the pedagogical structure provided by the *LAB*, *PRACTICE*, *EXERCISE*, and *FINAL PROJECT* notebooks to gain insights into students' study habits and the effectiveness of the learning materials.

Chapter 2: Log Collection with Jupyter and Graylog

2.1. Logging and Messaging in Jupyter

Log entries are created for web proxy events, user-server events, and JupyterHub events such as user authentication and stopping and starting user servers [15]. Understanding messaging between the Jupyter front end and kernel opens the possibility of custom logging of user actions.

A Jupyter front end communicates with the kernel using the ZeroMQ library for messaging, through modern WebSockets (instead of traditional HTTP/HTTPS). As depicted in Figure 3, the kernel contains separate sockets (or channels) for different functions: the Shell socket handles most requests from the front ends; stdin allows the kernel to request input from the front end; the Control socket handles requests for shutdown and restart as well as debugging messages, and messages on the Heartbeat socket ensure the front end and kernel are still connected. The kernel also has a broadcast channel that publishes side effects (stdout, stderr, and debugging events) and requests from the Shell, stdin, and Control sockets to allow coordination between all clients [16].

A ZeroMQ message is a dictionary of five dictionaries: the message header, the parent header (for when a message is the result of another message), metadata, content, and buffers. The message header contains fields for the message id, the session, the username, the date, the message type, and the message protocol version. A session id may refer to a client session or a kernel session, depending on where the message originates. The client session id will be unique for each client connected to a kernel. If a client reconnects, it will use the same client session id. If the client restarts, it will generate a new session id. Similarly, a new kernel session id will be generated if the kernel is restarted.

A variety of message types are described for each channel. Additionally, custom message types can be specified for extending Jupyter functionality. Any of these messages can be tapped for logging by modifying class ZMQChannelsHandler in the Jupyter Notebook open source code. For instance, the code patch to method _on_zmq_reply() shown in Figure 4 logs EXECUTE_INPUT messages in the PSDS Jupyter Notebook installation.

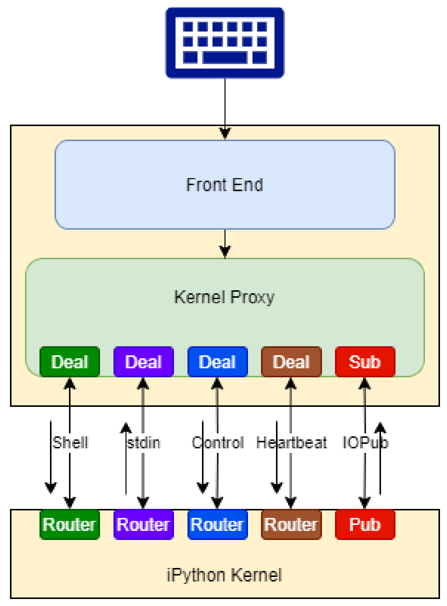
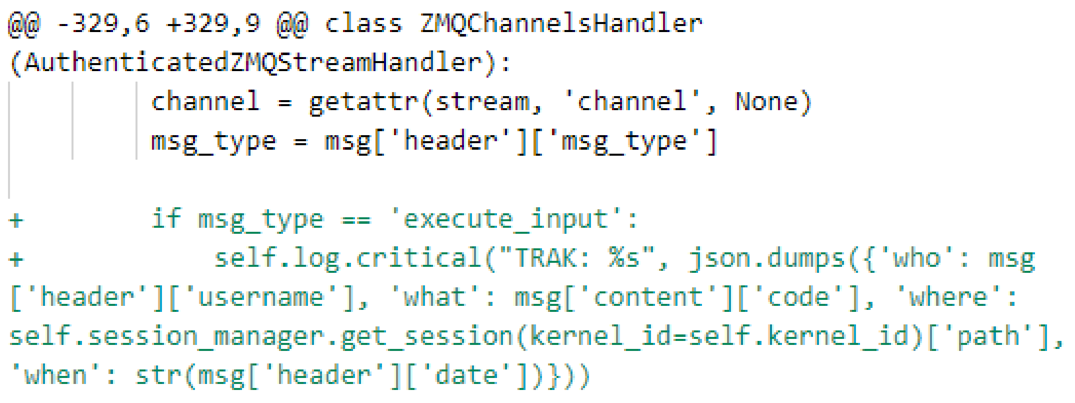


Figure 3. Jupyter messaging between a front end and the kernel.
Unconnected arrows show direction of request. Diagram adapted from [16].



```
@@ -329,6 +329,9 @@ class ZMQChannelsHandler
(AuthenticatedZMQStreamHandler):
         channel = getattr(stream, 'channel', None)
         msg_type = msg['header']['msg_type']

+        if msg_type == 'execute_input':
+            self.log.critical("TRAK: %s", json.dumps({'who': msg
['header']['username'], 'what': msg['content']['code'], 'where':
self.session_manager.get_session(kernel_id=self.kernel_id)['path'],
'when': str(msg['header']['date'])}))
```

Figure 4. Jupyter custom logging patch.
This patch is installed on method _on_zmq_reply() in class ZMQChannelsHandler in the file notebook/notebook/services/kernels/handlers.py. This patch is used by the PSDS system.

2.2. Graylog and Elasticsearch

Graylog is an open-source and open-standard log management platform for collecting, enhancing, storing, and analyzing log messages from sources throughout the enterprise [17]. A variety of message transport standards are supported for collecting log messages. As a message is received, it is parsed into defined fields by Graylog. Rules within Graylog determine what is done with a given message, allow custom extraction and enrichment of the message data, and the removal of unneeded data. After this processing, the data is stored in an Elasticsearch document store [18]. Elasticsearch is a distributed document store, where the documents are serialized JSON and text fields are indexed with an inverted index. This allows fast full-text searches [19]. Using the Graylog REST API to access Elasticsearch, this stored data can be searched by fields, alerts can be activated, and the log data visualized in dashboards [18]. Configuration data for Graylog is stored in MongoDB [20]. The architecture for a minimal setup with collocated services is shown in Figure 5.

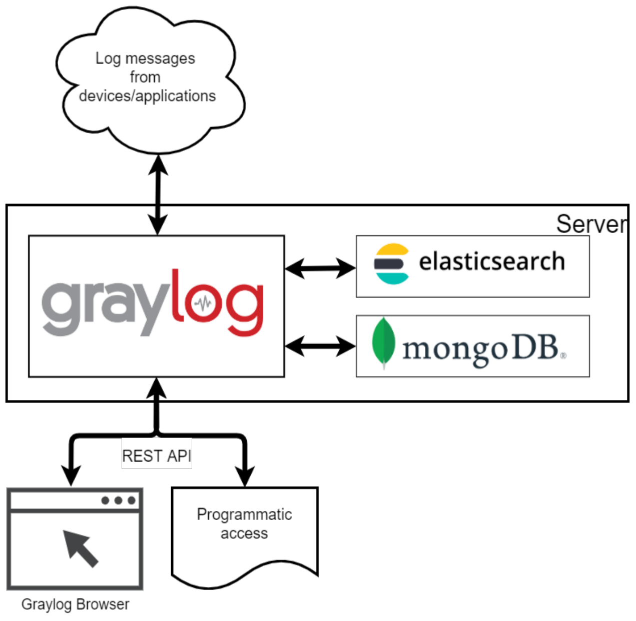


Figure 5. Graylog architecture for a minimal setup.
Services are collocated on the same server. Diagram adapted from [20].

Custom extractors are provided by Graylog to overcome a challenge. Although the specification for the syslog standard should make it easy to parse syslog messages, many devices that claim to follow syslog do not follow the RFCs. Other devices intentionally follow their own format. To accommodate the non-standard nature of all these devices, Graylog allows the user to specify how to extract important fields from any text in the received message. Extracting these custom fields allows Graylog to index them, proving efficient search capabilities [20].

An important limitation of Graylog is that once a message is stored in Elasticsearch, alterations to the schema are not available. Unlike adding a field to a RDBMS table, new fields cannot be extracted from persisted messages and added to the document. However, the pipeline for processing messages can be changed. If such changes are made, newly stored documents may have a different schema than older documents, possibly affecting search results.

Chapter 3: PSDS IT System Design

The data science cyberinfrastructure used by both the MU DSA program and PSDS is depicted in Figure 6. The Jupyter environment is central to the interaction with a variety of ancillary systems that enable data science learning with diverse types of data and technologies. While some courses use Jupyter to launch other systems such as deploying code to Shiny Server or cloud instances, for many courses most or all work is done in the Jupyter notebook itself, allowing detailed monitoring of student interaction.

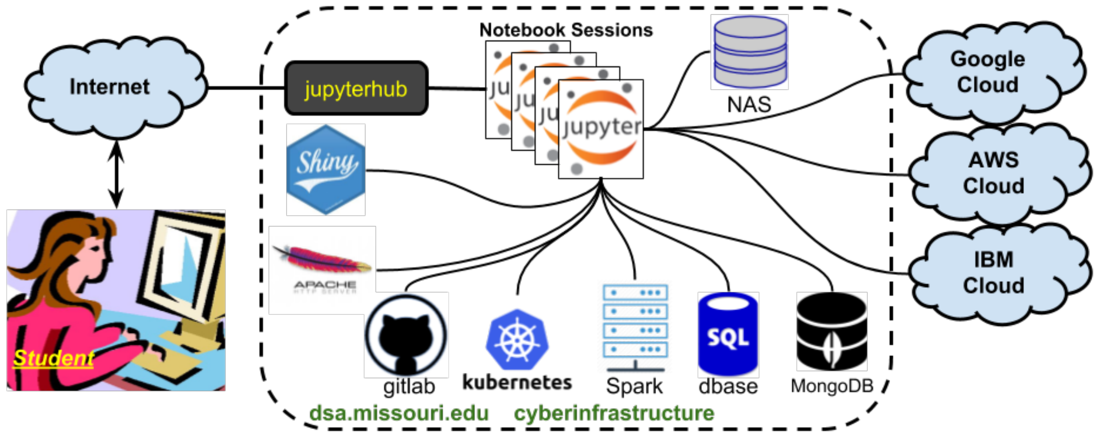


Figure 6. PSDS & DSA cyberinfrastructure.
Figure provided by the PSDS program.

The PSDS logging system is depicted in Figure 7. The log messages collected from JupyterHub are sent to the Graylog server. The Graylog configuration is stored on MongoDB, while the log messages are stored in Elasticsearch (as detailed in Section 2.2). A Jupyter notebook is used to query the Graylog REST API for direct reporting. Since older records in the Elasticsearch document store are pruned periodically for space considerations, the messages are also ingested into a PostgreSQL database for more persistent storage and historical reporting. During this study, Graylog 3.1.4 and Elasticsearch 6.8.6 were used.

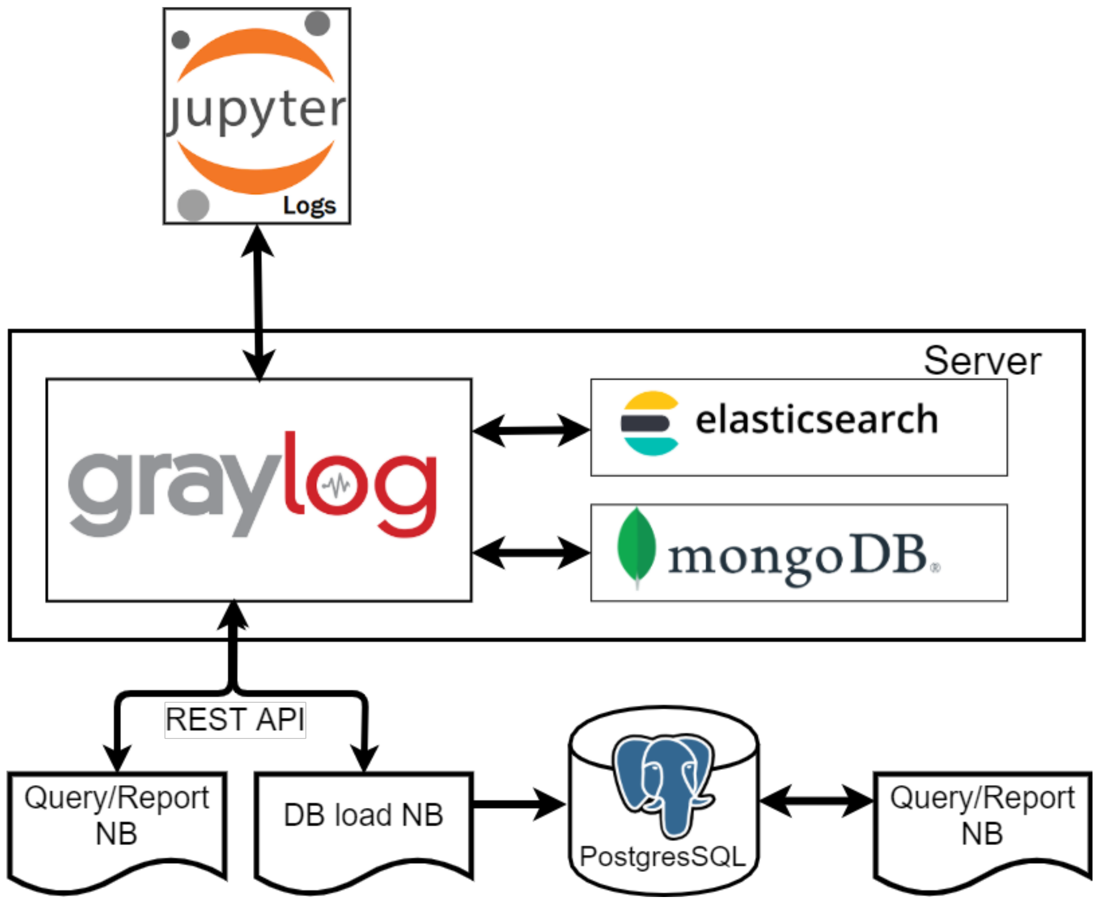


Figure 7. PSDS Jupyter log storage and reporting using Graylog.
Jupyter notebooks are used to query the Graylog REST API for reporting and loading a PostgreSQL relational database for permanent storage and historical reporting. Diagram adapted from [20].

Chapter 4: Data Collection & Ingestion Methods

Jupyter notebooks written in Python are used to query Graylog, extract features, persist the data in PostgreSQL, and visualize and report on student activity patterns.

4.1. Data Acquisition Challenges

With any complex system of systems such as the PSDS and DSA infrastructure, there are many potential challenges to overcome when developing a new, integrated capability such as logging analytics. A necessary condition of developing analytics is that robust and reliable access to the log messages is in place. However, despite the promise and potential of the Graylog system, a few hurdles had to be overcome to have reliable and repeatable techniques for querying Graylog and understanding the log messages. The challenges that were encountered and overcome included log purging, system outages, extractor issues, and logging patch adaptations; each of which is detailed in the subsections below.

4.1.1. Log Purges

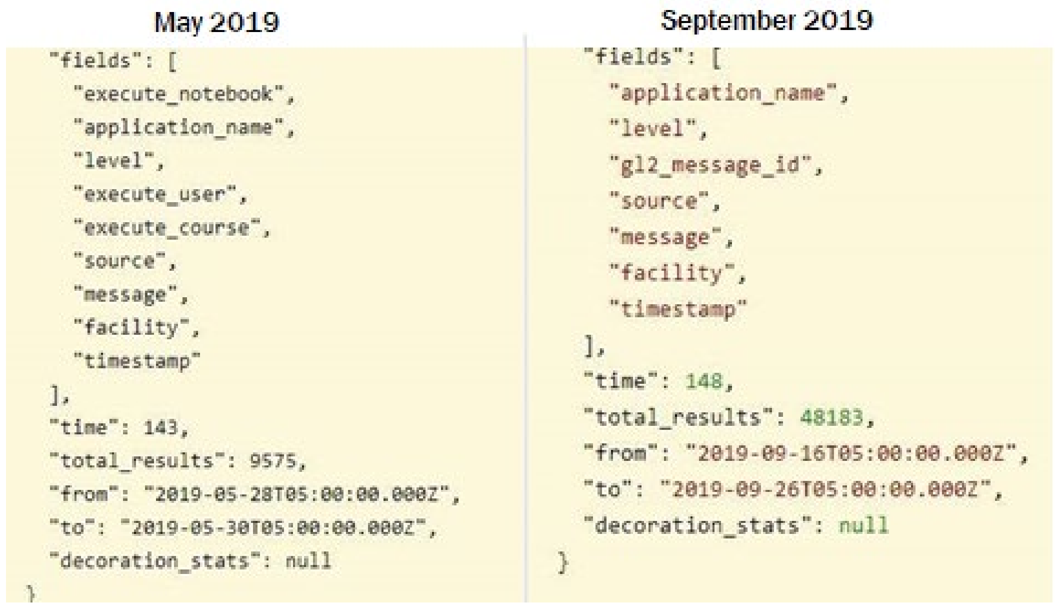
During investigation of the data availability, it became apparent that data were disappearing from the Graylog system. The system administrator confirmed this was a space management design setting. At the time of this study, Graylog was configured to keep 150 indices of 20 million messages. With the rate of usage for all systems monitored by our Graylog instance, the logs consumed about 1 TB and were retained about five months. To have data available for historical reporting, it was determined that Graylog messages need to be loaded into a more persistent data storage platform. For this research, PostgreSQL database was chosen for permanent storage of the data from the Jupyter system, beginning with the relevant log messages with **timestamps** beginning May 23, 2019 at 18:40. The details of the persistent Jupyter log message storage in PostgreSQL is detailed in Section 4.2.

4.1.2. Graylog Outages

An additional consideration regarding the completeness of log acquisition was the possibility of Graylog outages, where the Graylog system being offline causes missing data in the logging streams. The system administrator clarified that outages are typically limited to an hour or less for server patches, which occur about once a month. Additionally, there was a one to two-day outage in September 2019. If an outage occurred during scheduled instruction time or when students were putting in extra work, less student engagement would be reported for students working at that time. For finer-grained analytics that look at student engagement per notebook this missing data could be significant. The longer September outage will skew student engagement results for any course that was running during that time.

4.1.3. Custom Graylog Extractors

As explained in Section 2.2, custom Graylog extractors are useful to extract and index domain relevant information from the torrent of incoming Graylog messages. Jupyter specific extractors were previously configured for extracting the **user**, the **notebook**, and the **class** from certain messages, although there was a period of a few weeks in September 2019 that they were not operational. This was discovered when queries returning all fields showed different schemas in May and September (see Figure 8.) As the data were explored, these extractors were suspected to not be pulling the relevant values from all messages of interest. Because of both the time the extractors were not available, and the desire for confirmable reliability, regex patterns were used to extract these values, as well as others, in Jupyter notebook python code. This will be explained more fully in Section 4.2.2. For future work, rewriting the extractors and confirming they work as intended should be considered.


May 2019

```
"fields": [
  "execute_notebook",
  "application_name",
  "level",
  "execute_user",
  "execute_course",
  "source",
  "message",
  "facility",
  "timestamp"
],
"time": 143,
"total_results": 9575,
"from": "2019-05-28T05:00:00.000Z",
"to": "2019-05-30T05:00:00.000Z",
"decoration_stats": null
}
```

September 2019

```
"fields": [
  "application_name",
  "level",
  "gl2_message_id",
  "source",
  "message",
  "facility",
  "timestamp"
],
"time": 148,
"total_results": 48183,
"from": "2019-09-16T05:00:00.000Z",
"to": "2019-09-26T05:00:00.000Z",
"decoration_stats": null
}
```

Figure 8. Graylog schema change apparent in September 2019.
The extractors execute_notebook, execute_user, and execute_course are present in May 2019 but missing in September 2019.This was rectified September 30, 2019.

4.1.4. TRAK Messages (Custom Jupyter Logging)

A custom logging patch for Jupyter EXECUTE_INPUT messages (labeled as TRAK messages) was installed on the PSDS JupyterHub server from the earliest log record obtained, but the data logged was minimal, including only the **user** and the **timestamp**. On October 5, 2019 at approximately 15:30 the patch in Figure 4 was installed, providing a richer TRAK log message including the **user**, the **notebook**, the **timestamp**, and the **code** executed. These enhancements to Jupyter logging improved TRAK messages greatly for usability. However, in order to have a larger amount of data to analyze, KERNELMANAGER messages for EXECUTE_INPUT and EXECUTE_RESULT were used as a substitute. This was a critical learning event in the progress of this project.

Once the Jupyter messages are appropriately tracked and processed, a measure of student engagement can be approximated by the cumulative time between notebook SAVE events and/or cell EXECUTE events, which will be discussed in more detail in Section 5.3.

Two courses were explored by generating preliminary reports using SAVE messages and either TRAK or KERNELMANAGER EXECUTE messages to test if using the different message types was comparable. For the most part, the times were close (within a rounded minute over a user-day), but there were two user-day instances where the accumulated time was off significantly, and a few instances where the two methods differed by a few minutes. These larger discrepancies are shown in Table 1. If the number is negative there is more time in the KERNELMANAGER EXECUTE report, if they are positive, there is more time in the TRAK report.

Table 1. Time discrepancies in using TRAK vs kernelmanager execute messages. Negative times indicate more time is in the KERNELMANAGER EXECUTE report; positive times indicate more time is in the TRAK report. Student engagement times for all other student-days in these two courses were within a rounded minute over a day.

PSDS3300OP3-2_Final			
User	Date	Hours	Rounded Minutes
user1	10/13/2019	-1.084	-65
user2	10/15/2019	-0.137	-8
user3	10/20/2019	0.0126	1
user4	10/13/2019	-0.032	-2
user4	10/14/2019	-0.096	-6
user5	10/14/2019	-0.03	-2
user6	10/14/2019	-0.017	-1

PSDS2100OP3-3_Final			
User	Date	Hours	Rounded Minutes
user7	10/22/2019	0.0118	1
user8	10/22/2019	-1.4531	-87
user9	10/30/2019	0.0155	1
user10	10/15/2019	-0.0097	-1
user10	10/16/2019	-0.0094	-1
user10	10/18/2019	0.0107	1
user10	10/20/2019	0.0539	3
user10	10/26/2019	-0.0192	-1
user11	10/22/2019	0.0101	1
user12	10/15/2019	-0.1803	-11
user12	10/16/2019	-0.1113	-7

Interestingly, there was not a one-to-one correspondence between TRAK messages and KERNELMANAGER EXECUTE messages. However, this is not because EXECUTE messages include both EXECUTE_INPUT and EXECUTE_RESULT - sometimes there are multiple EXECUTE_INPUT messages for one TRAK message. There are also cases where there are TRAK messages and no KERNELMANAGER messages.

These discrepancies warrant further investigation, but for the purposes of this study, using KERNELMANAGER EXECUTE messages for all reporting was deemed acceptable for providing a proof of concept.

Unlike the richer TRAK messages, KERNELMANAGER EXECUTE messages contain a **kernel_id**, but do not have the **notebook_path**. Because of this, **kernel_ids** are matched with **notebook_paths** using an algorithm described in Section 4.2.3.

4.2. Data Ingestion and Database Loading

Graylog caps the number of results returned with no way to paginate or automatically window the results. Therefore, querying Graylog is done in a loop with the time window for the query determined dynamically. For each batch of results, features are extracted, and the records are loaded into the database. The loop continues to query Graylog, extract features, and load records until all messages up to the current time are loaded.

Figure 9 depicts the query, parse, and load cycle. A step_size of 60 minutes is used for the first time frame. The number of results found for the $query$ determines if the step is adjusted up or down. If more than 10,000 results are found (the size of the $limit$), the step size is reduced proportionally to the number of results divided by the $limit$ and the $query$ is tried again with the new, smaller step_size (a smaller time frame.).

If the $query$ finds fewer results than the $limit$, the messages are parsed from the results, features are extracted, and the records are loaded into the database. If there are fewer results than half the $limit$, the step size is increased by 5 minutes for the next query, so we have a larger time frame. The new start time is the previous end time.

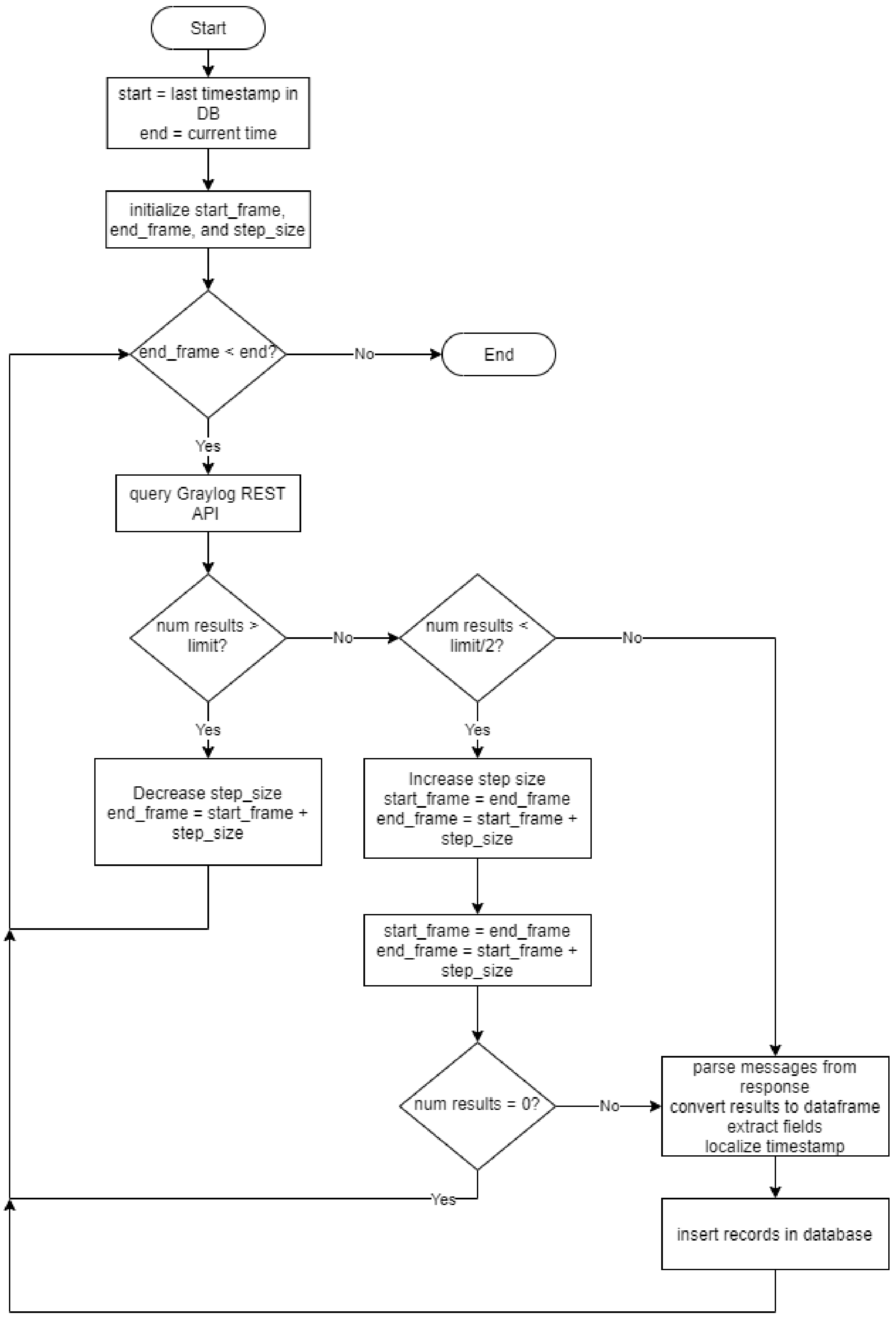

Start
start = last timestamp in DB
end = current time
initialize start_frame, end_frame, and step_size
end_frame < end?
No
End
Yes
query Graylog REST API
num results > limit?
No
num results < limit/2?
No
Yes
Decrease step_size
end_frame = start_frame + step_size
Yes
Increase step size
start_frame = end_frame
end_frame = start_frame + step_size
start_frame = end_frame
end_frame = start_frame + step_size
num results = 0?
No
parse messages from response
convert results to dataframe
extract fields
localize timestamp
Yes
insert records in database

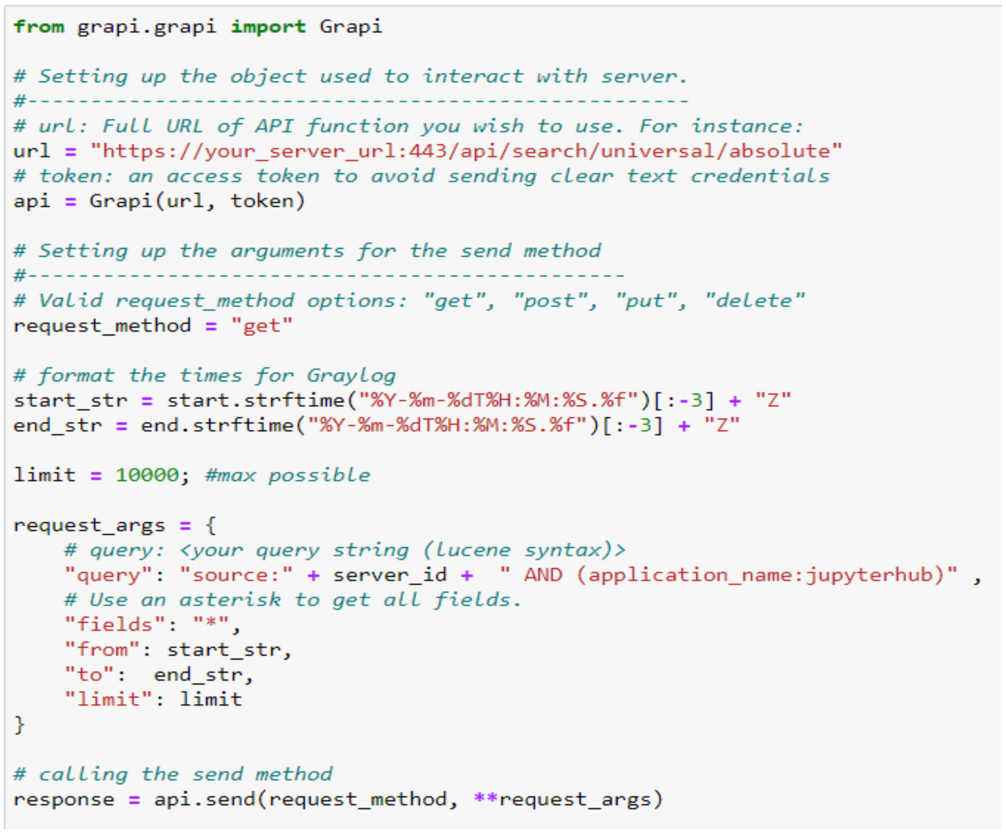
Figure 9. High level view of the query, parse, and load cycle.
The time window size for the query is determined dynamically.

4.2.1. Querying Graylog

Grapi Request

The grapi library is used to query Graylog from Python code [21] as displayed in Figure 10. The grapi object is instantiated with the full URL of the Graylog REST API function and a Graylog access token. Many functions are available in the REST API, which are viewable in the REST API browser [22]. We are using absolute search (search with an absolute time range.)

The grapi send method makes the request, by specifying the request method (such as GET) and the request arguments. Required fields for the request arguments are the $query$ (a string using Lucene syntax), $fields$ (the fields you wish Graylog to return), $from$ (the starting time), and to (the ending time). The number of results to return is specified by $limit$, which is optional.



```
from grapi.grapi import Grapi

# Setting up the object used to interact with server.
#---------------------------------------------------
# url: Full URL of API function you wish to use. For instance:
url = "https://your_server_url:443/api/search/universal/absolute"
# token: an access token to avoid sending clear text credentials
api = Grapi(url, token)

# Setting up the arguments for the send method
#----------------------------------------------
# Valid request_method options: "get", "post", "put", "delete"
request_method = "get"

# format the times for Graylog
start_str = start.strftime("%Y-%m-%dT%H:%M:%S.%f")[:-3] + "Z"
end_str = end.strftime("%Y-%m-%dT%H:%M:%S.%f")[:-3] + "Z"

limit = 10000; #max possible

request_args = {
    # query: <your query string (lucene syntax)>
    "query": "source:" + server_id +  " AND (application_name:jupyterhub)" ,
    # Use an asterisk to get all fields.
    "fields": "*",
    "from": start_str,
    "to":  end_str,
    "limit": limit
}

# calling the send method
response = api.send(request_method, **request_args)
```

Figure 10. Using the grapi library to query Graylog from Python.

A simple *query* is used in order to return all log messages for possible future reporting. Only the server name and application_name (jupyterhub) are specified in the *query* string. The asterisk for *fields* means all fields are returned from Graylog. The *from* and *to* timestamps are determined dynamically as discussed in the previous slide and queries are performed in a loop until all messages are processed.

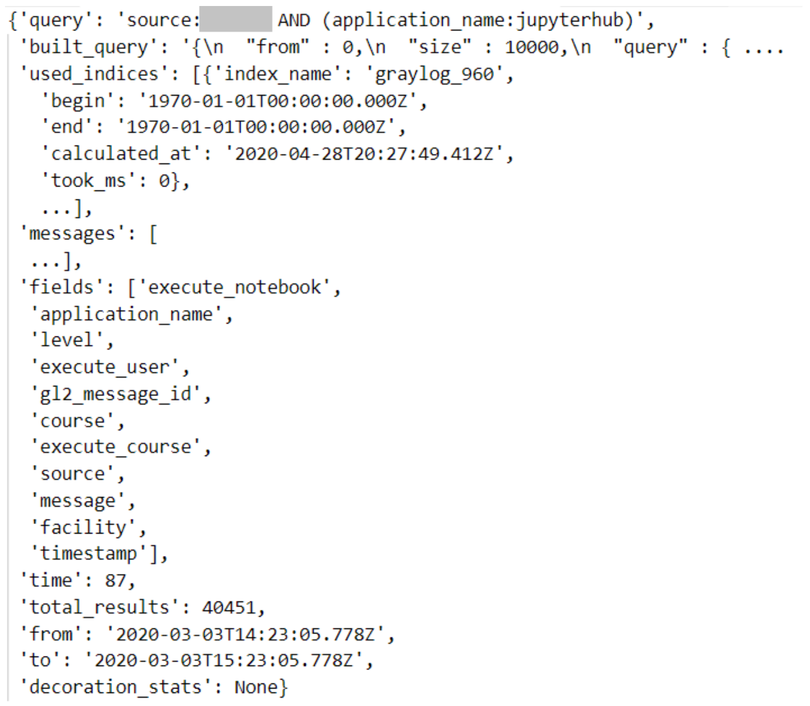
The default number of messages returned by a Graylog query is 150; for more results you must specify a *limit*. The maximum possible *limit* is 10,000, so that is what is used. No matter the number of results returned, the query results also return the number of actual records found for that time frame.

The start time is determined by querying the maximum message **timestamp** already loaded in the database. The **timestamp** returned, when assigned to a variable, is a pandas Timestamp with a local offset; this must be converted to Zulu time without an offset, and then converted to a string correctly formatted for querying Graylog. (For example: '2020-03-03T14:23:05.778Z'.)

Parsing the Response

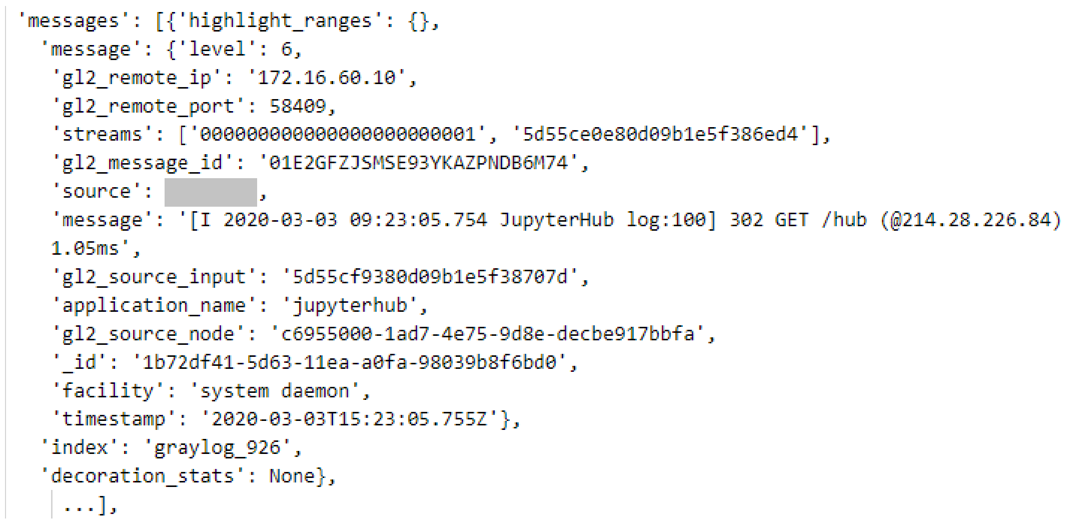
The contents of the response are accessed by response.json() or response.text. The response contains the query as specified in the request; the built_query - the Elasticsearch query built by Graylog; the collection of used_indices; the collection of messages; the fields returned; the amount of time taken by the request; the total_results; the from and to times; and decoration_stats (see Figure 11.)

Figure 12 shows an example of the contents of messages, including the first message. The messages can be retrieved from the response by normalizing the JSON, converting messages to a dictionary where messages[0] is the key and the collection of messages is the value, and appending each message in messages[0] to a list. The list can then be converted to a Pandas DataFrame (see Figure 13.)



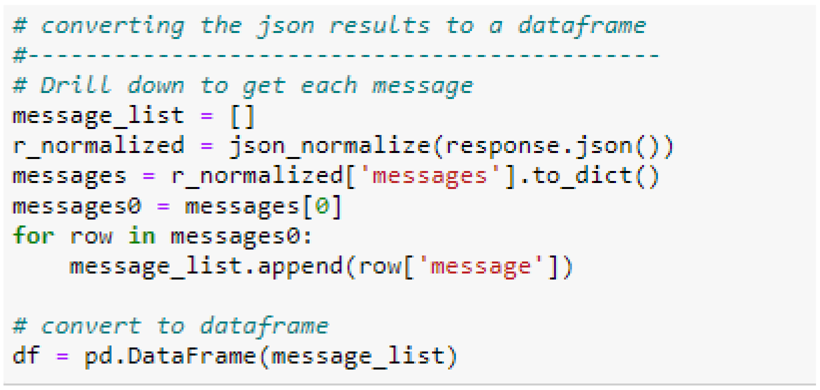
```
{'query': 'source:[redacted] AND (application_name:jupyterhub)',
 'built_query': '{\n  "from" : 0,\n  "size" : 10000,\n  "query" : { ....
 'used_indices': [{'index_name': 'graylog_960',
   'begin': '1970-01-01T00:00:00.000Z',
   'end': '1970-01-01T00:00:00.000Z',
   'calculated_at': '2020-04-28T20:27:49.412Z',
   'took_ms': 0},
   ...],
 'messages': [
  ...],
 'fields': ['execute_notebook',
  'application_name',
  'level',
  'execute_user',
  'gl2_message_id',
  'course',
  'execute_course',
  'source',
  'message',
  'facility',
  'timestamp'],
 'time': 87,
 'total_results': 40451,
 'from': '2020-03-03T14:23:05.778Z',
 'to': '2020-03-03T15:23:05.778Z',
 'decoration_stats': None}
```

Figure 11. The JSON content of the API response.



```
'messages': [{'highlight_ranges': {},
  'message': {'level': 6,
   'gl2_remote_ip': '172.16.60.10',
   'gl2_remote_port': 58409,
   'streams': ['000000000000000000000001', '5d55ce0e80d09b1e5f386ed4'],
   'gl2_message_id': '01E2GFZJSMSE93YKAZPNDB6M74',
   'source': [redacted],
   'message': '[I 2020-03-03 09:23:05.754 JupyterHub log:100] 302 GET /hub (@214.28.226.84)
   1.05ms',
   'gl2_source_input': '5d55cf9380d09b1e5f38707d',
   'application_name': 'jupyterhub',
   'gl2_source_node': 'c6955000-1ad7-4e75-9d8e-decbe917bbfa',
   '_id': '1b72df41-5d63-11ea-a0fa-98039b8f6bd0',
   'facility': 'system daemon',
   'timestamp': '2020-03-03T15:23:05.755Z'},
  'index': 'graylog_926',
  'decoration_stats': None},
   ...],
```

Figure 12. The JSON format of the messages portion of the Graylog API response. This message is missing the extractors execute_user, execute_course, and execute_notebook because they are not applicable to this message. Messages also contain metadata in addition to the fields in the schema.



```
# converting the json results to a dataframe
#---------------------------------------------
# Drill down to get each message
message_list = []
r_normalized = json_normalize(response.json())
messages = r_normalized['messages'].to_dict()
messages0 = messages[0]
for row in messages0:
    message_list.append(row['message'])

# convert to dataframe
df = pd.DataFrame(message_list)
```

Figure 13. Parsing the messages from the API response.

4.2.2. Feature Extraction

Various features are extracted from the message using regex pattern matching. Because we are returning the widest variety of messages (all JupyterHub messages from the specified server), not all features are present for all messages. An overview of these features is presented in Table 2.

Table 2. Features extracted from the log message.

Feature	Description
user	The student username extracted from the message.
class	The specific instance of a course extracted from the message.
day	The PSDS instructional module a **notebook** is in, extracted from the message.
notebook	The notebook name extracted from the message.
kernel_id	The unique identifier for the kernel as extracted from a path
log_type	Values: "kernelmanager", "proxy web", "proxy ws", "trak", nan. Extracted from the message.
event_type	Values: "checkpoint", "error", "execute", "kernels", "load", "save", "status", "terminals", nan. Extracted or determined from the message.
notebook_type	Values: "answers", "challenges", "exercises", "finalproject", "labs", "practices", "readings", "resources", "solutions and "other." Extracted or determined from the message.

Student user ids follow the pattern of three characters, a dot, and four digits, i.e., abc.####. TA or instructor user ids will not follow this pattern. When a TA is running a student notebook, a web proxy message will contain both user ids. Two regex patterns are extracted: one for the specific student pattern and a second for the more general pattern. If they don't match, we know a TA or instructor is running the notebook and the **user** will be replaced with nan. Otherwise, the student user id is the **user** if present.

PSDS course content is organized in a manner that facilitates decomposition of a **notebook_path** into information about the **class**, the **day** the **notebook_type** (*LAB*, *PRACTICE*, or *EXERCISE*, etc.), and the specific **notebook**. Take the following **notebook_path** as an example:

```
“PSDS2200OP3-2_aaa.1234/Day4/labs/Chi-Squared.ipynb”
```

The **class** is “PSDS2200OP3-2”, which is a specific instance of the course PSDS2200. The course **day** is “4”, the **notebook_type** is “labs”, and the **notebook** is “Chi-Squared.ipynb".

Possible values for **notebook_type** are "answers", "challenges", "exercises", "finalproject", "labs", "practices", "readings", "resources", and "solutions." The type is assigned as “other” if there is a notebook that doesn’t match these expected types.

The **kernel_id** is extracted for matching to a **notebook_path** later if it occurs in a path after “kernels/”, such as shown in red in the following message:

```
"21:33:40.783 [ConfigProxy] #033[34mdebug#033[39m: PROXY
WEB /user/aaa.1234/api/kernels/53fe6569-ae4d-47d7-bf68-
96df0dd5b6ee/restart to http://127.0.0.1:32783."
```

Two additional fields are extracted or labeled to help sort through different message types. This was useful for data exploration, and most of these values are not used. A **log_type** may have the values of "kernelmanager", "proxy web", "proxy ws", "trak", or none of these. These are Jupyter components with the addition of our custom T R A K log message. These values are extracted from the message.

An **event_type** may have the value of "checkpoint", "error", "execute", "kernels", "load", "save", "status", "terminals", or none of these. Most of these values are extracted from the message, but some may be not as intended. For instance, since a T R A K message includes the **code** that was executed, if there is a comment including the word "error", this will be extracted. Therefore, using an **event_type** (other than “save” or “load”) in a WHERE clause should always include a **log_type**.

Table 3: Data dictionary for the table psds.log_entry in the analytics database.

Column Name	Nullable	Data Type	Max Length	Notes
_id	NO	character varying		PK; Graylog message id. Returned by Graylog.
gl2_message_id	YES	character varying		Returned by Graylog for some messages.
message	NO	character varying		Returned by Graylog.
application_name	NO	character varying	20	Example: jupyterhub. Returned by Graylog.
facility	NO	character varying	50	Example: system daemon. Returned by Graylog.
timestamp	NO	timestamp with time zone		Timestamp of message. Returned by Graylog as UTC, converted to local time.
execute_course	YES	character varying	20	Graylog custom extractor for course. Returned by Graylog for some messages.
execute_notebook	YES	character varying		Graylog custom extractor for notebook. Returned by Graylog for some messages.
execute_user	YES	character varying	20	Graylog custom extractor for username. Returned by Graylog for some messages.
psds_user	YES	character varying	20	Student username. Extracted from the message if present.
class_code	YES	character varying		Specific instance of a course. Extracted from the message if present.
day	YES	smallint		The PDSD instructional module a notebook is in. Extracted from the message if present.
notebook	YES	character varying		The notebook name. Extracted from the message if present.
kernel_id	YES	character varying		Unique identifier for the kernel, extracted from a path in the message containing /kernels/ if present.
log_type	YES	character varying		Values: "kernelmanager", "proxy web", "proxy ws", "trak", null. Extracted from the message.
event_type	YES	character varying		Values: "checkpoint", "error", "execute", "kernels", "load", "save", "status", "terminals", null. Extracted or determined from the message.
notebook_type	YES	text		Values: "answers", "challenges", "exercises", "finalproject", "labs", "practices", "readings", "resources", "solutions and "other." Extracted or determined from the message.
new_log_type	YES	character varying		For experimenting with additional types.
new_event_type	YES	character varying		For experimenting with additional types.
load_time	YES	timestamp without time zone		The time when this record batch was loaded in the database.

The **event_types** LOAD and SAVE are not directly extracted. If the message includes a **notebook_path** followed by "to http" this is interpreted as loading the notebook in the browser and labeled as LOAD. If the message includes a **notebook_path** followed by "?content=0&_=", this is labeled as a SAVE event.

The message **timestamp** is converted to pandas datetime, localized to UTC, and converted to our local time zone.

Finally, nans are replaced by None for database compatibility and only messages with either a **user** or a **kernel_id** are loaded into the database as relevant to mining student activity patterns. The data dictionary for the table psds.log_entry in the analytics database is shown in Table 3.

For future work, Jupyter custom logging is recommended for all messages of interest for clear identification and unambiguous values.

4.2.3. Kernel to Notebook Mapping

When visualizing student or class engagement patterns related to individual **notebooks** or **notebook_types**, we are hampered by the limited timeframe for which the richer version of TRAK messages has been available. The KERNELMANAGER EXECUTE messages used in place of TRAK only have kernel_ids. Therefore, a method of mapping notebooks information to **kernel_ids** was developed using a combination of database scripts and a Jupyter notebook.

The kernel matching process is depicted in Figure 14. Two tables were created in the analytics database to assist with this mapping - psds.kernel and psds.nb_load. In Step 1, a SQL script load table psds.kernels with the records with the earliest **timestamp** for each **kernel_id**, ordered by **user** and **timestamp**. In Step 2, table psds.nb_load is loaded with records with **event_type** LOAD where **user** is not null, also ordered by **user** and **timestamp**. In Steps 3 and 4, a Jupyter notebook loads these two tables into Data Frames df_kernel and df_load, with df_kernels in descending order by **timestamp**. In Step 5, for each student in df_kernels, we loop through the **kernel_ids** for that student, selecting the records from df_loads for the same **user** that have not yet been matched to a **kernel_id**, and that have a **timestamp** less than or equal to the **timestamp** of the kernel event. These records are loaded into DataFrame df_cadidate_matches. In Step 6 we find the max index in df_candidate_matches. This is the latest unmatched notebook LOAD event that occurred before or at the same **timestamp** as the kernel event. The notebook information (**class, day, notebook, notebook_type**) is copied to df_load. The record in df_load that was matched is set to matched, and the loop continues until all **kernel_ids** for every student are matched to a **notebook**. Finally, in Steps 7 and 8, the database tables are updated and in Step 9 a SQL script is used to update the psds.log_entry table with the notebook data for records that match the **kernel_ids** in psds.kernel.

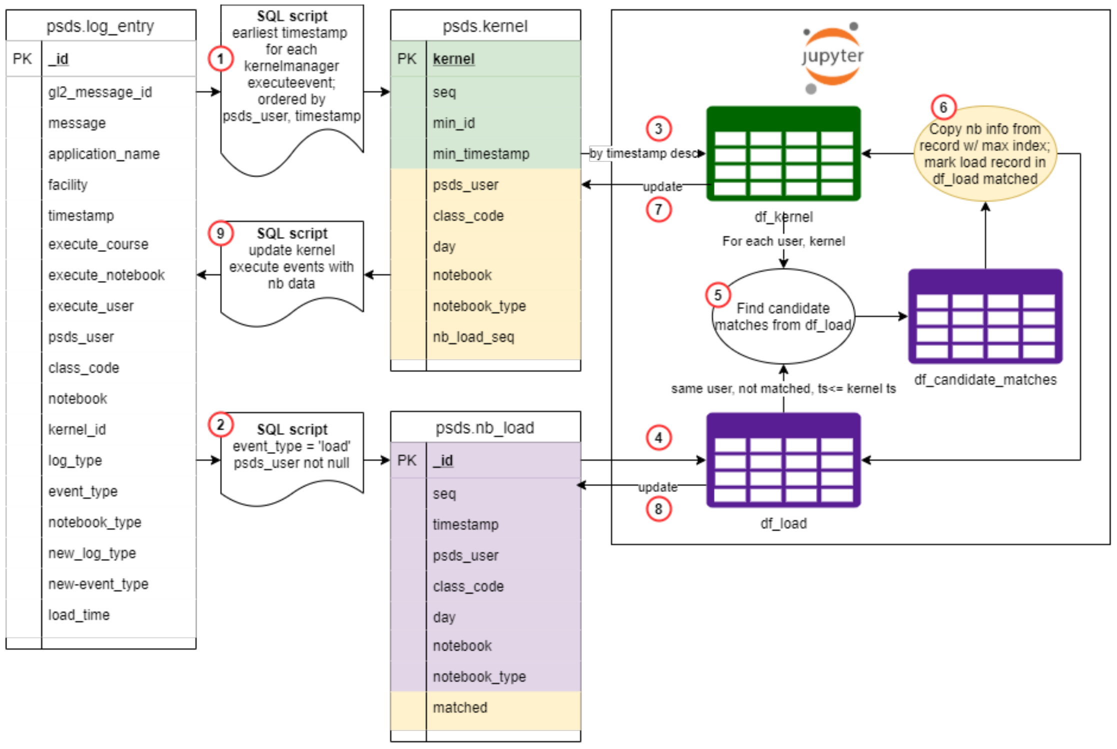
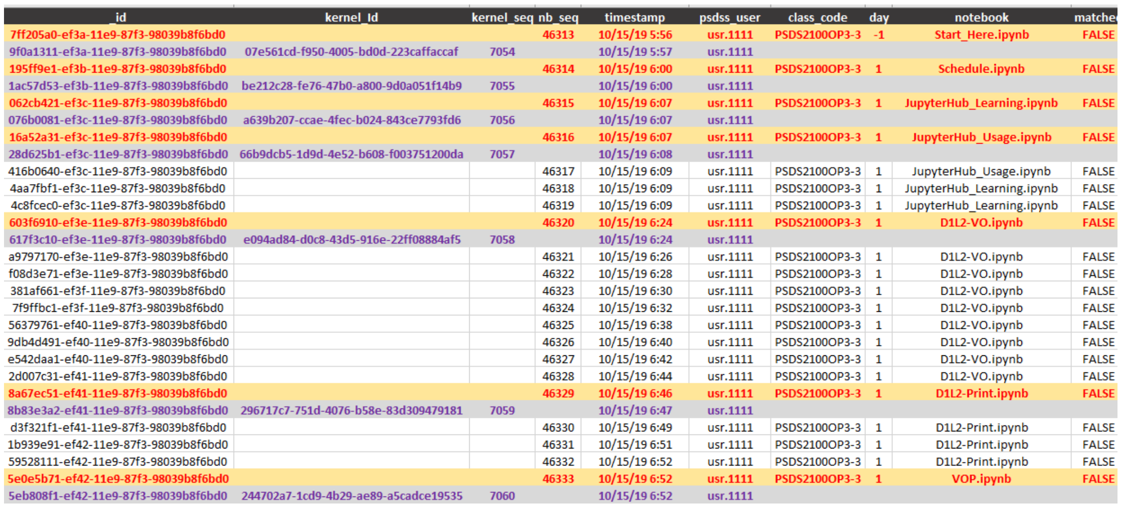


Figure 14. The kernel to notebook matching process.
SQL scripts are used to load the two worktables psds.kernel and psds.nb_load from psds.log_entry. A Jupyter notebook is used for the matching process and the worktables are updated. Yellow fields in the worktables are filled during matching. Finally, the **user, class, day, notebook**, and **notebook_type** are copied to records matching the **kernel_id** in psds.log_entry from psds.kernel.

The **kernel_ids** are ordered in descending **timestamp** order because the kernel event occurs after the LOAD event. Figure 15 shows a spreadsheet with kernel events and LOAD events in ascending **timestamp** order. As we process the kernel events from the bottom up, the last LOAD event from the top down contains the **notebook** to match.

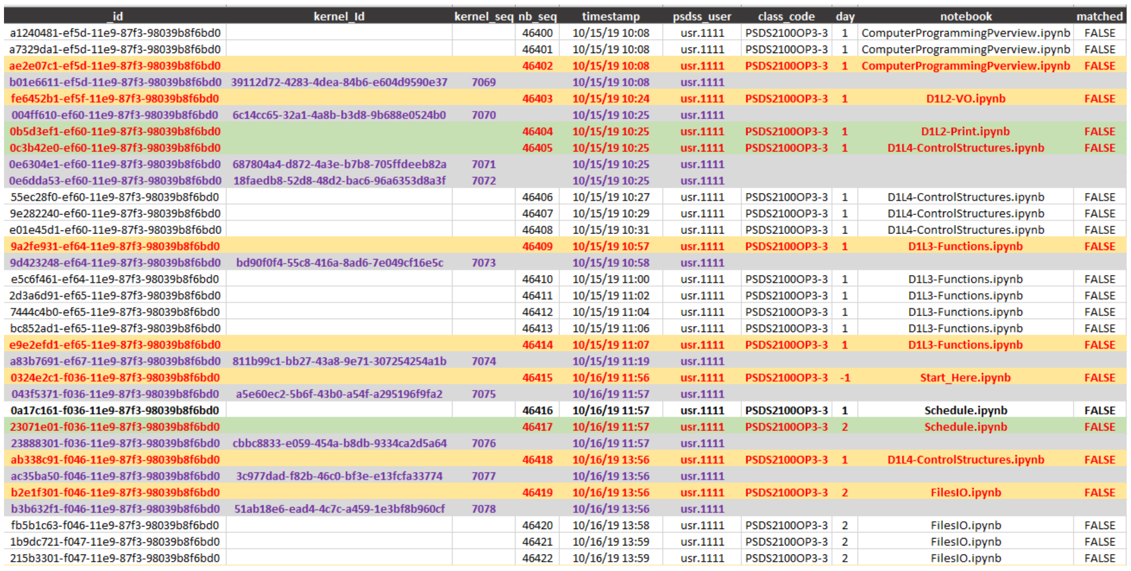


_id	kernel_Id	kernel_seq	nb_seq	timestamp	psdss_user	class_code	day	notebook	matche
7ff205a0-ef3a-11e9-87f3-98039b8f6bd0			46313	10/15/19 5:56	usr.1111	PSDS2100OP3-3	-1	Start_Here.ipynb	FALSE
9f0a1311-ef3a-11e9-87f3-98039b8f6bd0	07e561cd-f950-4005-bd0d-223caffaccaf	7054		10/15/19 5:57	usr.1111				
195ff9e1-ef3b-11e9-87f3-98039b8f6bd0			46314	10/15/19 6:00	usr.1111	PSDS2100OP3-3	1	Schedule.ipynb	FALSE
1ac57d53-ef3b-11e9-87f3-98039b8f6bd0	be212c28-fe76-47b0-a800-9d0a051f14b9	7055		10/15/19 6:00	usr.1111				
062cb421-ef3c-11e9-87f3-98039b8f6bd0			46315	10/15/19 6:07	usr.1111	PSDS2100OP3-3	1	JupyterHub_Learning.ipynb	FALSE
076b0081-ef3c-11e9-87f3-98039b8f6bd0	a639b207-ccae-4fec-b024-843ce7793fd6	7056		10/15/19 6:07	usr.1111				
16a52a31-ef3c-11e9-87f3-98039b8f6bd0			46316	10/15/19 6:07	usr.1111	PSDS2100OP3-3	1	JupyterHub_Usage.ipynb	FALSE
28d625b1-ef3c-11e9-87f3-98039b8f6bd0	66b9dcb5-1d9d-4e52-b608-f003751200da	7057		10/15/19 6:08	usr.1111				
416b0640-ef3c-11e9-87f3-98039b8f6bd0			46317	10/15/19 6:09	usr.1111	PSDS2100OP3-3	1	JupyterHub_Usage.ipynb	FALSE
4aa7fbf1-ef3c-11e9-87f3-98039b8f6bd0			46318	10/15/19 6:09	usr.1111	PSDS2100OP3-3	1	JupyterHub_Learning.ipynb	FALSE
4c8fcec0-ef3c-11e9-87f3-98039b8f6bd0			46319	10/15/19 6:09	usr.1111	PSDS2100OP3-3	1	JupyterHub_Learning.ipynb	FALSE
603f6910-ef3e-11e9-87f3-98039b8f6bd0			46320	10/15/19 6:24	usr.1111	PSDS2100OP3-3	1	D1L2-VO.ipynb	FALSE
617f3c10-ef3e-11e9-87f3-98039b8f6bd0	e094ad84-d0c8-43d5-916e-22ff08884af5	7058		10/15/19 6:24	usr.1111				
a9797170-ef3e-11e9-87f3-98039b8f6bd0			46321	10/15/19 6:26	usr.1111	PSDS2100OP3-3	1	D1L2-VO.ipynb	FALSE
f08d3e71-ef3e-11e9-87f3-98039b8f6bd0			46322	10/15/19 6:28	usr.1111	PSDS2100OP3-3	1	D1L2-VO.ipynb	FALSE
381af661-ef3f-11e9-87f3-98039b8f6bd0			46323	10/15/19 6:30	usr.1111	PSDS2100OP3-3	1	D1L2-VO.ipynb	FALSE
7f9ffbc1-ef3f-11e9-87f3-98039b8f6bd0			46324	10/15/19 6:32	usr.1111	PSDS2100OP3-3	1	D1L2-VO.ipynb	FALSE
56379761-ef40-11e9-87f3-98039b8f6bd0			46325	10/15/19 6:38	usr.1111	PSDS2100OP3-3	1	D1L2-VO.ipynb	FALSE
9db4d491-ef40-11e9-87f3-98039b8f6bd0			46326	10/15/19 6:40	usr.1111	PSDS2100OP3-3	1	D1L2-VO.ipynb	FALSE
e542daa1-ef40-11e9-87f3-98039b8f6bd0			46327	10/15/19 6:42	usr.1111	PSDS2100OP3-3	1	D1L2-VO.ipynb	FALSE
2d007c31-ef41-11e9-87f3-98039b8f6bd0			46328	10/15/19 6:44	usr.1111	PSDS2100OP3-3	1	D1L2-VO.ipynb	FALSE
8a67ec51-ef41-11e9-87f3-98039b8f6bd0			46329	10/15/19 6:46	usr.1111	PSDS2100OP3-3	1	D1L2-Print.ipynb	FALSE
8b83e3a2-ef41-11e9-87f3-98039b8f6bd0	296717c7-751d-4076-b58e-83d309479181	7059		10/15/19 6:47	usr.1111				
d3f321f1-ef41-11e9-87f3-98039b8f6bd0			46330	10/15/19 6:49	usr.1111	PSDS2100OP3-3	1	D1L2-Print.ipynb	FALSE
1b939e91-ef42-11e9-87f3-98039b8f6bd0			46331	10/15/19 6:51	usr.1111	PSDS2100OP3-3	1	D1L2-Print.ipynb	FALSE
59528111-ef42-11e9-87f3-98039b8f6bd0			46332	10/15/19 6:52	usr.1111	PSDS2100OP3-3	1	D1L2-Print.ipynb	FALSE
5e0e5b71-ef42-11e9-87f3-98039b8f6bd0			46333	10/15/19 6:52	usr.1111	PSDS2100OP3-3	1	VOP.ipynb	FALSE
5eb808f1-ef42-11e9-87f3-98039b8f6bd0	244702a7-1cd9-4b29-ae89-a5cadce19535	7060		10/15/19 6:52	usr.1111				

Figure 15. Matching earliest notebook load to earliest timestamp for kernel_id. Yellow rows and red text show earliest notebook LOAD events. Gray rows and purple text show the earliest **timestamp** for each **kernel_id.** In this spreadsheet, records are in ascending **timestamp** order.

The matching process is not perfect. In Figure 16, the green rows are ambiguous matches, with the same **timestamp.** The row in with bold black text is an orphan **notebook** that has no matching **kernel_id.**

As mentioned in Section 4.1.4, comparing reports generated using TRAK messages and KERNELMANAGER EXECUTE messages showed that the times were within a rounded minute over a user-day in most cases. This made it feasible to use KERNELMANAGER EXECUTE log records to analyze courses from before October 5.



_id	kernel_Id	kernel_seq	nb_seq	timestamp	psdss_user	class_code	day	notebook	matched
a1240481-ef5d-11e9-87f3-98039b8f6bd0			46400	10/15/19 10:08	usr.1111	PSDS2100OP3-3	1	ComputerProgrammingPverview.ipynb	FALSE
a7329da1-ef5d-11e9-87f3-98039b8f6bd0			46401	10/15/19 10:08	usr.1111	PSDS2100OP3-3	1	ComputerProgrammingPverview.ipynb	FALSE
ae2e07c1-ef5d-11e9-87f3-98039b8f6bd0			46402	10/15/19 10:08	usr.1111	PSDS2100OP3-3	1	ComputerProgrammingPverview.ipynb	FALSE
b01e6611-ef5d-11e9-87f3-98039b8f6bd0	39112d72-4283-4dea-84b6-e604d9590e37	7069		10/15/19 10:08	usr.1111				
fe6452b1-ef5f-11e9-87f3-98039b8f6bd0			46403	10/15/19 10:24	usr.1111	PSDS2100OP3-3	1	D1L2-VO.ipynb	FALSE
004ff610-ef60-11e9-87f3-98039b8f6bd0	6c14cc65-32a1-4a8b-b3d8-9b688e0524b0	7070		10/15/19 10:25	usr.1111				
0b5d3ef1-ef60-11e9-87f3-98039b8f6bd0			46404	10/15/19 10:25	usr.1111	PSDS2100OP3-3	1	D1L2-Print.ipynb	FALSE
0c3b42e0-ef60-11e9-87f3-98039b8f6bd0			46405	10/15/19 10:25	usr.1111	PSDS2100OP3-3	1	D1L4-ControlStructures.ipynb	FALSE
0e6304e1-ef60-11e9-87f3-98039b8f6bd0	687804a4-d872-4a3e-b7b8-705ffdeeb82a	7071		10/15/19 10:25	usr.1111				
0e6dda53-ef60-11e9-87f3-98039b8f6bd0	18faedb8-52d8-48d2-bac6-96a6353d8a3f	7072		10/15/19 10:25	usr.1111				
55ec28f0-ef60-11e9-87f3-98039b8f6bd0			46406	10/15/19 10:27	usr.1111	PSDS2100OP3-3	1	D1L4-ControlStructures.ipynb	FALSE
9e282240-ef60-11e9-87f3-98039b8f6bd0			46407	10/15/19 10:29	usr.1111	PSDS2100OP3-3	1	D1L4-ControlStructures.ipynb	FALSE
e01e45d1-ef60-11e9-87f3-98039b8f6bd0			46408	10/15/19 10:31	usr.1111	PSDS2100OP3-3	1	D1L4-ControlStructures.ipynb	FALSE
9a2fe931-ef64-11e9-87f3-98039b8f6bd0			46409	10/15/19 10:57	usr.1111	PSDS2100OP3-3	1	D1L3-Functions.ipynb	FALSE
9d423248-ef64-11e9-87f3-98039b8f6bd0	bd90f0f4-55c8-416a-8ad6-7e049cf16e5c	7073		10/15/19 10:58	usr.1111				
e5c6f461-ef64-11e9-87f3-98039b8f6bd0			46410	10/15/19 11:00	usr.1111	PSDS2100OP3-3	1	D1L3-Functions.ipynb	FALSE
2d3a6d91-ef65-11e9-87f3-98039b8f6bd0			46411	10/15/19 11:02	usr.1111	PSDS2100OP3-3	1	D1L3-Functions.ipynb	FALSE
7444c4b0-ef65-11e9-87f3-98039b8f6bd0			46412	10/15/19 11:04	usr.1111	PSDS2100OP3-3	1	D1L3-Functions.ipynb	FALSE
bc852ad1-ef65-11e9-87f3-98039b8f6bd0			46413	10/15/19 11:06	usr.1111	PSDS2100OP3-3	1	D1L3-Functions.ipynb	FALSE
e9e2efd1-ef65-11e9-87f3-98039b8f6bd0			46414	10/15/19 11:07	usr.1111	PSDS2100OP3-3	1	D1L3-Functions.ipynb	FALSE
a83b7691-ef67-11e9-87f3-98039b8f6bd0	811b99c1-bb27-43a8-9e71-307254254a1b	7074		10/15/19 11:19	usr.1111				
0324e2c1-f036-11e9-87f3-98039b8f6bd0			46415	10/16/19 11:56	usr.1111	PSDS2100OP3-3	-1	Start_Here.ipynb	FALSE
043f5371-f036-11e9-87f3-98039b8f6bd0	a5e60ec2-5b6f-43b0-a54f-a295196f9fa2	7075		10/16/19 11:57	usr.1111				
0a17c161-f036-11e9-87f3-98039b8f6bd0			**46416**	**10/16/19 11:57**	**usr.1111**	**PSDS2100OP3-3**	**1**	**Schedule.ipynb**	**FALSE**
23071e01-f036-11e9-87f3-98039b8f6bd0			46417	10/16/19 11:57	usr.1111	PSDS2100OP3-3	2	Schedule.ipynb	FALSE
23888301-f036-11e9-87f3-98039b8f6bd0	cbbc8833-e059-454a-b8db-9334ca2d5a64	7076		10/16/19 11:57	usr.1111				
ab338c91-f046-11e9-87f3-98039b8f6bd0			46418	10/16/19 13:56	usr.1111	PSDS2100OP3-3	1	D1L4-ControlStructures.ipynb	FALSE
ac35ba50-f046-11e9-87f3-98039b8f6bd0	3c977dad-f82b-46c0-bf3e-e13fcfa33774	7077		10/16/19 13:56	usr.1111				
b2e1f301-f046-11e9-87f3-98039b8f6bd0			46419	10/16/19 13:56	usr.1111	PSDS2100OP3-3	2	FilesIO.ipynb	FALSE
b3b632f1-f046-11e9-87f3-98039b8f6bd0	51ab18e6-ead4-4c7c-a459-1e3bf8b960cf	7078		10/16/19 13:56	usr.1111				
fb5b1c63-f046-11e9-87f3-98039b8f6bd0			46420	10/16/19 13:58	usr.1111	PSDS2100OP3-3	2	FilesIO.ipynb	FALSE
1b9dc721-f047-11e9-87f3-98039b8f6bd0			46421	10/16/19 13:59	usr.1111	PSDS2100OP3-3	2	FilesIO.ipynb	FALSE
215b3301-f047-11e9-87f3-98039b8f6bd0			46422	10/16/19 13:59	usr.1111	PSDS2100OP3-3	2	FilesIO.ipynb	FALSE

Figure 16. Ambiguous matches and orphan notebooks.
The matching process is not perfect. The green rows are ambiguous matches, with the same **timestamp**. The row in with bold black text is an orphan **notebook** with no matching **kernel_id**.

Chapter 5: PSDS Case Study

5.1. Selected Courses

For our case study of applying learning analytics to the Jupyter environment in the University of Missouri's PSDS, several class instances of Boot Camp courses were selected on the recommendation of Dr. Grant Scott, the MU Program Manager of PSDS. Course instances are named for the contract period they occur in (in this case, OP3) followed by an instance number. Complete log data are available beginning May 24, 2019, so only courses offered after that date were selected. Not all analyses are presented for each course, but example visualizations are selected to show interesting data or to contrast patterns of activity between different classes or students.

Courses and classes selected for analysis include:

- Python Boot Camp - PSDS 2100
    - OP3-3 Beginning Oct 15
    - OP3-4 Beginning Oct 28
- DB/SQL Boot Camp - PSDS 2110
    - OP3-1 Beginning June 3
    - OP3-2 Beginning June 17
    - OP3-3 Beginning Sep 16 *
- R Programming Boot Camp - PSDS 2120
    - OP3-1 Beginning July 1
    - OP3-2 Beginning July 8
    - OP3-3 Beginning Nov 12

Note that PSDS 2110 OP3-3 occurred in September, which had a one to two-day Graylog outage at some point during the month. This could affect analysis on that course, as discussed previously in Section 4.1.2.

5.2. Use Case Descriptions

The PSDS learning analytics use cases explored include administrative, class level, and high-level student views of the data, in line with what is often provided in an LMS. Additionally, finer-grained analysis of student engagement at both the class and student level shows the potential for what can be achieved using Jupyter learning analytics with a carefully structured course and in-depth logging mechanisms.

Two use cases were developed initially to meet pressing administrative needs. As mentioned in Section 1.3, the accelerated nature of the program presents logistical issues for accurate and timely university enrollment. Because the sequence follows *"NGA enroll" → attend → assess → "MU enroll" → pay*; it is useful to evaluate the participation of students who have indicated they intend to apply for university credit, to see if they are following through with their plans and should have their paperwork routed to MU enrollment. If their engagement with the course material falls below a minimum level of interactive participation, it is assumed they have decided to not complete the course and therefore not enroll for MU credit. This is the *Routing Use Case*, discussed in Section 5.4.1.

The second administrative use case focuses on the instruction hours contracted between PSDS and NGA. PSDS faculty and staff provide support during the NGA professional workday, and PSDS students are expected to do all or most of the work during that time. A class view of whether students are completing their course work during or outside instruction hours is informative to both PSDS and NGA administrators to know whether complaints about lack of support are warranted and whether adjustments should be made to the contract. The *Contracted Instruction Use Case* is also discussed in Section 5.4.1.

A high-level view of individual student engagement was developed next, useful for the instructor, program administrator, and the students. This plots daily instruction and non-instruction hours for an individual student to provide insights as to how much time a student is investing in course activities and whether the time spent falls outside the hours of support. The instructor can see whether a student who is struggling is spending a reasonable amount of time engaged with the material, and whether they are working during support hours. This visualization can also be provided to the student to enable them to track how much time they are investing in the course. PSDS and NGA administrators can check an individual's engagement patterns to see if perhaps the student's workplace manager is providing time for study during the working day, as is expected. *The Student Engagement - Daily Hours Use Case* is found in Section 5.4.2.

Three views of aggregated student engagement begin to dive into student activity patterns related to course design and pedagogy. Section 5.4.3 shows *Class Engagement by Activity Type Use Case* (i.e., whether the **notebook** is a *LAB*, *PRACTICE*, resource, solution, etc.), Section 5.4.4 shows *Class Engagement by Activity Type Within a Module Use Case*, and Section 5.4.5 shows *Class Engagement by Module/Notebook Use Case* (that is, the **notebooks** within a given module.) Instructors will find these views enlightening, with the possibility of highlighting **notebooks** that are too hard or too easy, and whether the students are following the course design.

The *Individual Student Engagement by Module/Notebook Use Case* is discussed in Section 5.4.6, allowing the instructor or students to clearly see where individual students are spending their time.

Plotting time spent in *LABS* versus time spent in *PRACTICES* provides a different view into how students are engaging with the pedagogical design of the course. The *Practice vs Lab Balance Use Case* is shown in Section 5.4.7. The selected courses are all boot camps, which only have *LABS* and *PRACTICES*; other courses also have *EXERCISES*. Similar plots could be constructed for the relationship between *LABS* plus *PRACTICES* compared to *EXERCISES* in future analysis.

5.3. Data Preparation for Reporting and Visualization

The log records previously ingested into the database as described in Section 4.2 are selected and prepared for reporting and visualization using Python in Jupyter notebooks. This process is depicted in Figure 17 and described below.

Instruction dates and times, as well as the time zone of the course, are also stored in the database and retrieved for reporting (represented by the brown DataFrame). Records from the psds.log_entry table are retrieved if they fall within the time period for the report, if they have a **user**, and if they are

event_type EXECUTE, or **log_type** PROXY_WEB and **event_type** SAVE (shown as the green DataFrame.)

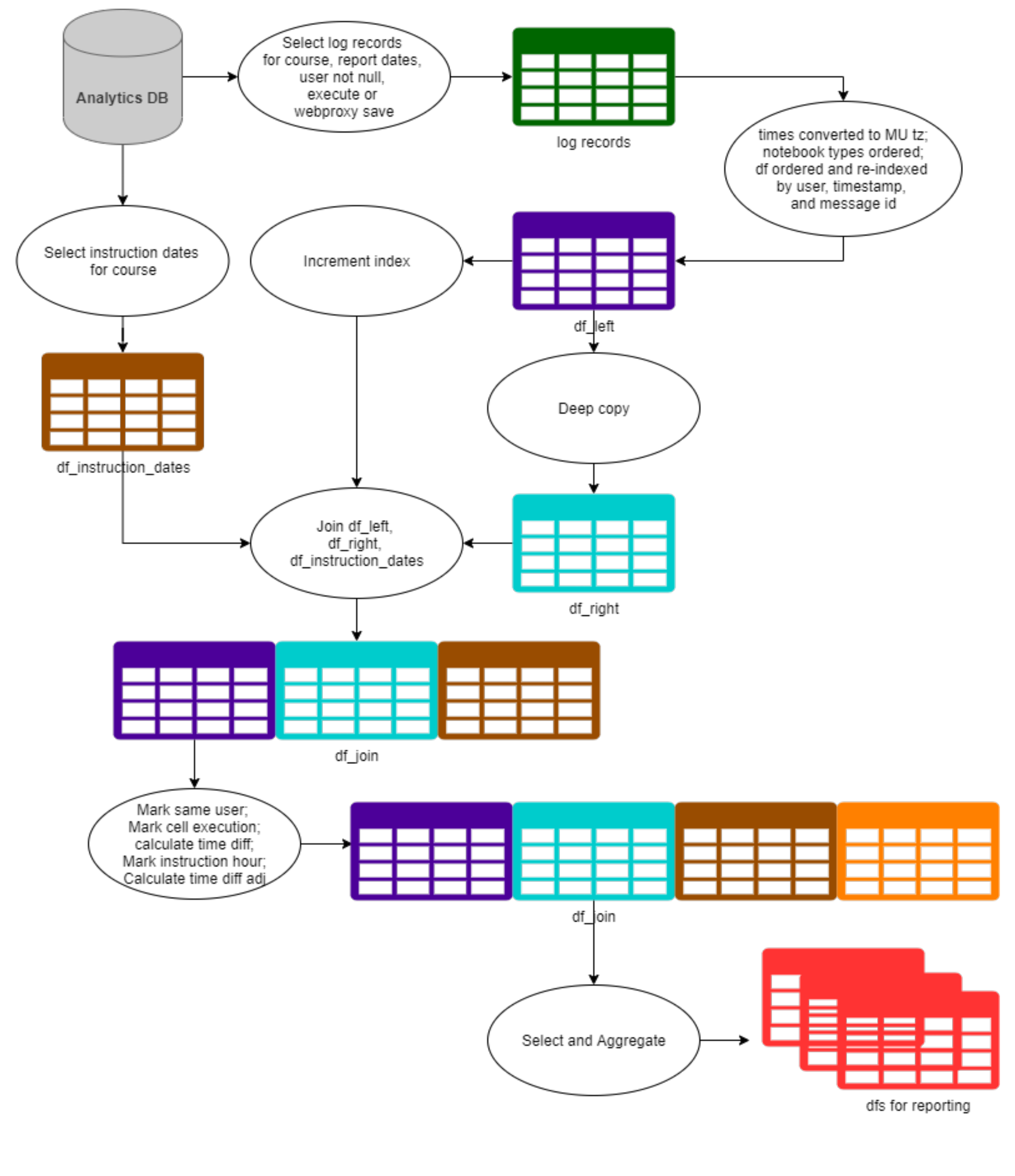
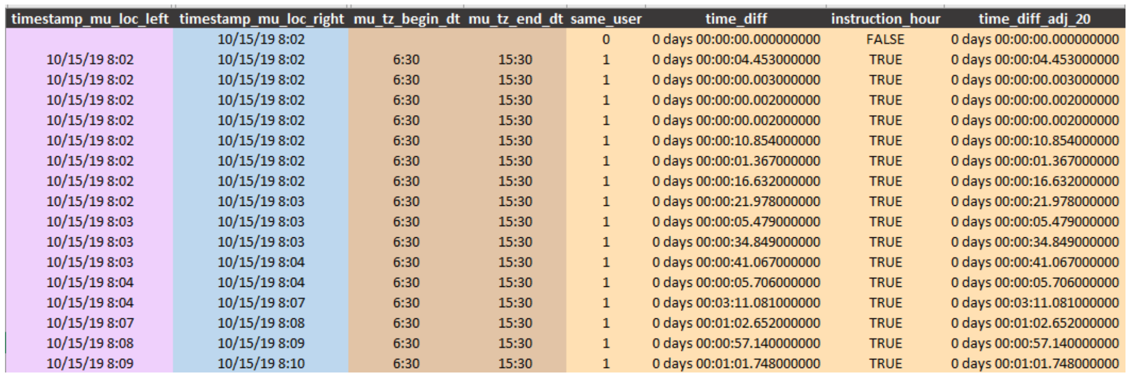

Analytics DB
Select log records for course, report dates, user not null, execute or webproxy save
log records
times converted to MU tz; notebook types ordered; df ordered and re-indexed by user, timestamp, and message id
Select instruction dates for course
Increment index
df_left
df_instruction_dates
Deep copy
Join df_left, df_right, df_instruction_dates
df_right
df_join
Mark same user; Mark cell execution; calculate time diff; Mark instruction hour; Calculate time diff adj
df_join
Select and Aggregate
dfs for reporting

Figure 17. Data preparation for reporting and visualizations.

The beginning time of the report is the start of the business day in the class location on the first day of class; the end time for most reports is the deadline for collecting work, which is 12:00 am of the fifth business day after the last day of instruction in the class location. The Routing Report discussed in Section 5.2, has an earlier end time - the close of business on the 5th business day of the course. Restricting log records to the course time frame eliminates counting interactions when students refer to course material after completing the course.

Some general preparation is done: all times are converted to the MU time zone for consistent calculation; **notebook_types** are ordered for later visualization using the pandas.series.map function; and the DataFrame is ordered and re-indexed by **user**, **timestamp**, and message id (depicted as the purple DataFrame.)

To determine elapsed time between student interactions, a deep copy of the DataFrame is made (depicted as the aqua DataFrame). The copy is named df_right, and the original DataFrame is df_left. The index of df_left is incremented by 1. The two DataFrames are joined on the index, and then joined to a DataFrame of instruction dates on the date. This means each log record is paired with the subsequent record in the same DataFrame row, with information about instruction times for that date. A spreadsheet view of selected columns from this large DataFrame is seen in Figure 18.



timestamp_mu_loc_left	timestamp_mu_loc_right	mu_tz_begin_dt	mu_tz_end_dt	same_user	time_diff	instruction_hour	time_diff_adj_20
	10/15/19 8:02			0	0 days 00:00:00.000000000	FALSE	0 days 00:00:00.000000000
10/15/19 8:02	10/15/19 8:02	6:30	15:30	1	0 days 00:00:04.453000000	TRUE	0 days 00:00:04.453000000
10/15/19 8:02	10/15/19 8:02	6:30	15:30	1	0 days 00:00:00.003000000	TRUE	0 days 00:00:00.003000000
10/15/19 8:02	10/15/19 8:02	6:30	15:30	1	0 days 00:00:00.002000000	TRUE	0 days 00:00:00.002000000
10/15/19 8:02	10/15/19 8:02	6:30	15:30	1	0 days 00:00:00.002000000	TRUE	0 days 00:00:00.002000000
10/15/19 8:02	10/15/19 8:02	6:30	15:30	1	0 days 00:00:10.854000000	TRUE	0 days 00:00:10.854000000
10/15/19 8:02	10/15/19 8:02	6:30	15:30	1	0 days 00:00:01.367000000	TRUE	0 days 00:00:01.367000000
10/15/19 8:02	10/15/19 8:02	6:30	15:30	1	0 days 00:00:16.632000000	TRUE	0 days 00:00:16.632000000
10/15/19 8:02	10/15/19 8:03	6:30	15:30	1	0 days 00:00:21.978000000	TRUE	0 days 00:00:21.978000000
10/15/19 8:03	10/15/19 8:03	6:30	15:30	1	0 days 00:00:05.479000000	TRUE	0 days 00:00:05.479000000
10/15/19 8:03	10/15/19 8:03	6:30	15:30	1	0 days 00:00:34.849000000	TRUE	0 days 00:00:34.849000000
10/15/19 8:03	10/15/19 8:04	6:30	15:30	1	0 days 00:00:41.067000000	TRUE	0 days 00:00:41.067000000
10/15/19 8:04	10/15/19 8:04	6:30	15:30	1	0 days 00:00:05.706000000	TRUE	0 days 00:00:05.706000000
10/15/19 8:04	10/15/19 8:07	6:30	15:30	1	0 days 00:03:11.081000000	TRUE	0 days 00:03:11.081000000
10/15/19 8:07	10/15/19 8:08	6:30	15:30	1	0 days 00:01:02.652000000	TRUE	0 days 00:01:02.652000000
10/15/19 8:08	10/15/19 8:09	6:30	15:30	1	0 days 00:00:57.140000000	TRUE	0 days 00:00:57.140000000
10/15/19 8:09	10/15/19 8:10	6:30	15:30	1	0 days 00:01:01.748000000	TRUE	0 days 00:01:01.748000000

Figure 18. Calculating the time difference between events in df_join.
Selected columns from df_join showing the **timestamp** from df_right and df_left, start and end times from df_instruction_hours, and the calculated fields. The record on the top row from df_right is the same record as the record in the second row from df_left. This allows us to calculate the time difference between each interaction and sum the times for the same **user**.

Additional columns are added to this large DataFrame (represented in orange). A Boolean column 'same_user' is marked true if the two joined log records are from the same **user**. For each row that has the same **user**, the time difference between the log entries on that row is calculated. If the log record on the left is within the instruction day, a Boolean field instruction_day is marked true.

Because students could leave the computer with the notebook open and running, time differences over 20 minutes are set to zero (in a new column, time_diff_adj_20) to not count idle time as activity.

For some visualizations we count the number of cell EXECUTES, so a Boolean column indicates a log entry for an execution (as opposed to a SAVE).

For the various reports and visualizations, the relevant columns are selected and aggregated as appropriate, using the log information on the left as the **notebook** engaged with. For per-notebook reports, it should be recognized that students may have different notebooks open at the same time and switch between them. The time between interactions will be counted as activity in the **notebook** from the first interaction.

5.4. Use Case Visualizations

5.4.1. Routing and Contracted Instruction Hours

This use case answers the question, "Which students are investing enough time in the course to confirm their intention to complete the course, warranting their routing to MU enrollment? The total hours per student are summed up and the students are ranked in order from the most time spent to the least and plotted by rank on the x-axis. Student names can be associated with plot points for ease of instructor interpretation, but that has been left off these plots for privacy. On the fifth business day of the course, the students have had three days with contracted instruction hours, totaling 24 hours of supported class time. This line is marked in red on the plots in Figure 19. [1]

All time spent by students will not be counted, as there is time spent reading or viewing materials outside of the notebooks. Also, with only selecting EXECUTE and SAVE log messages, we do not know

1 Visualizations use matplotlib.pyplot and associated libraries.

when the notebook was opened, so notebooks that are only read or with reading at the beginning will not contribute the time spent reading to our total. Some courses have a significant portion of work done outside of Jupyter notebooks. For instance, in the case of the DB/SQL Boot Camp - PSDS 2110, work is done in the terminal. Because of the unique nature of each course, the patterns of expected time spent engaged with a **notebook** will vary.

Figure 19 shows the line plot of Ranked User Hours and the associated stacked bar chart for one class of the Python Boot Camp - PSDS 2100 OP3-3, and Table 4 presents the same data in table form. There is one overachiever and a fairly steady decrease in time spent from student to student. There is a drop from around 10.5 hours to around 8 hours at rank 17. Ten hours is the cutoff used for routing students to MU enrollment for this course.

In this class, only two students worked entirely within instructional hours during the first five business days of class. Many students are spending similar amounts of time at work and at home, with one (Rank 2) spending significantly more time at home than at work.

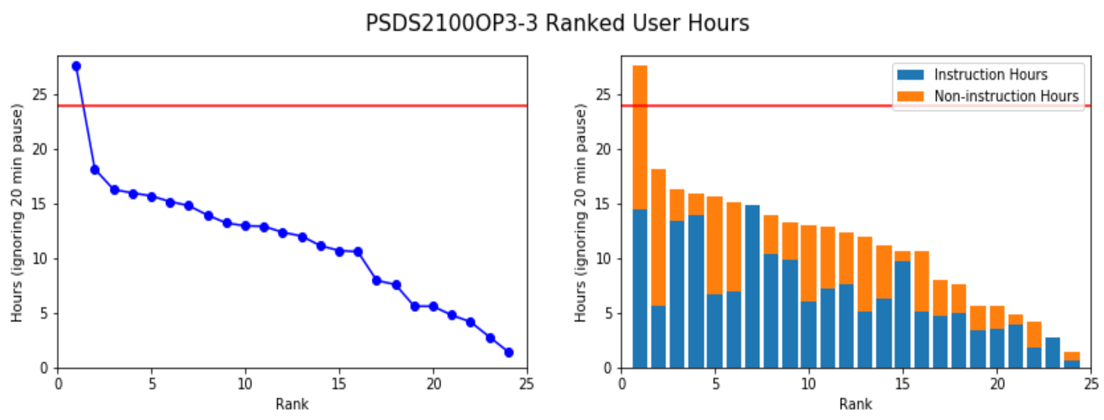
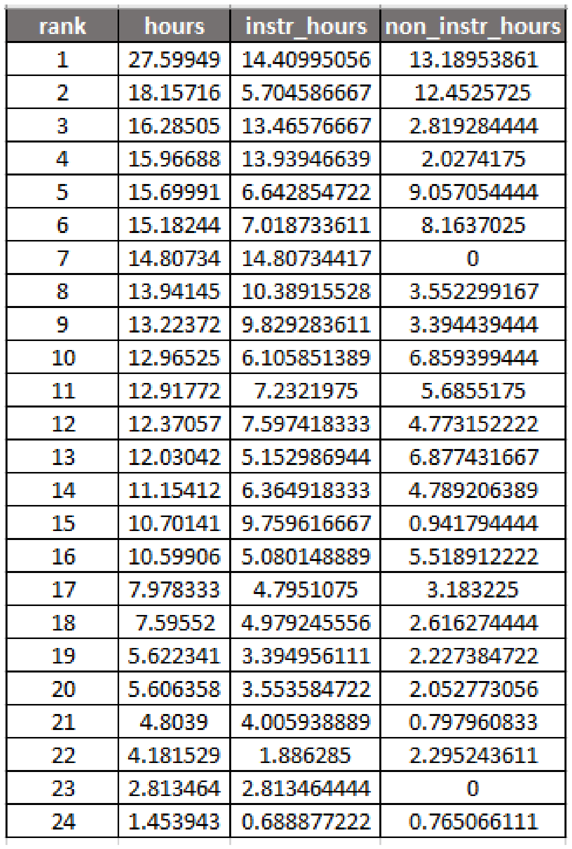


Figure 19. Routing Report - Ranked user hours for PSDS2100OP3-3.
The ranked hours for one Python Boot Camp class are presented as a line plot and a stacked bar chart of instruction and non-instruction hours for quick visual interpretation.

Table 4. Routing Report – Table of ranked user hours for PSDS2100OP3-3.
Ranked user hours for one Python Boot Camp class, presented as a table for precision.



rank	hours	instr_hours	non_instr_hours
1	27.59949	14.40995056	13.18953861
2	18.15716	5.704586667	12.4525725
3	16.28505	13.46576667	2.819284444
4	15.96688	13.93946639	2.0274175
5	15.69991	6.642854722	9.057054444
6	15.18244	7.018733611	8.1637025
7	14.80734	14.80734417	0
8	13.94145	10.38915528	3.552299167
9	13.22372	9.829283611	3.394439444
10	12.96525	6.105851389	6.859399444
11	12.91772	7.2321975	5.6855175
12	12.37057	7.597418333	4.773152222
13	12.03042	5.152986944	6.877431667
14	11.15412	6.364918333	4.789206389
15	10.70141	9.759616667	0.941794444
16	10.59906	5.080148889	5.518912222
17	7.978333	4.7951075	3.183225
18	7.59552	4.979245556	2.616274444
19	5.622341	3.394956111	2.227384722
20	5.606358	3.553584722	2.052773056
21	4.8039	4.005938889	0.797960833
22	4.181529	1.886285	2.295243611
23	2.813464	2.813464444	0
24	1.453943	0.688877222	0.765066111

It was thought that by combining multiple classes (instances) of one course, a clearer cutoff would be seen for future routing of classes of the same course. This is not the case. Figure 20 combines two classes of the R Boot Camp. Again, we see one overachiever, and a fairly steady decrease in the amount of time spent from student to student, with the last student being the only one to show a significant drop from the next highest ranked student.

Figure 21 combines three classes of the DB/SQL Boot Camp. The pattern here is flatter, except for four students putting in much more time than their peers. There is a small dip around 10 hours, and a steeper slope beginning around 7 hours and a steep drop at around 5 hours. Since students in the DB/SQL Bootcamp spend time in the terminal as well as Jupyter, a lower cutoff around 7 or 5 hours is reasonable for this course.

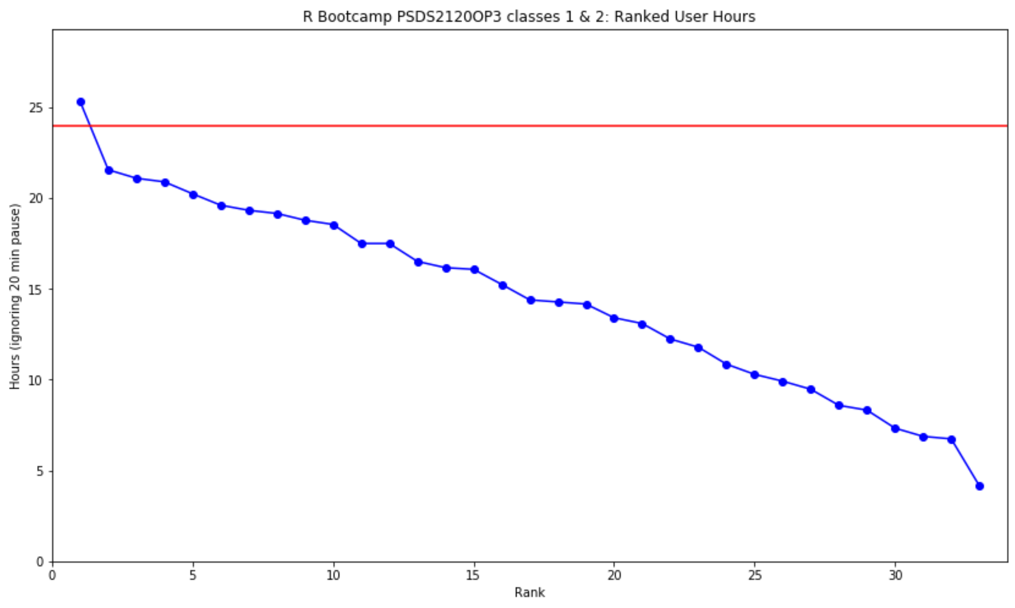


Figure 20. Routing Report - Ranked user hours combined for two class instances of PSDS2120. Rank of the times for all students in two classes of the R Boot Camp during the first five business days of each class. A clear cutoff for Routing is not seen, but around 5 hours might be appropriate.

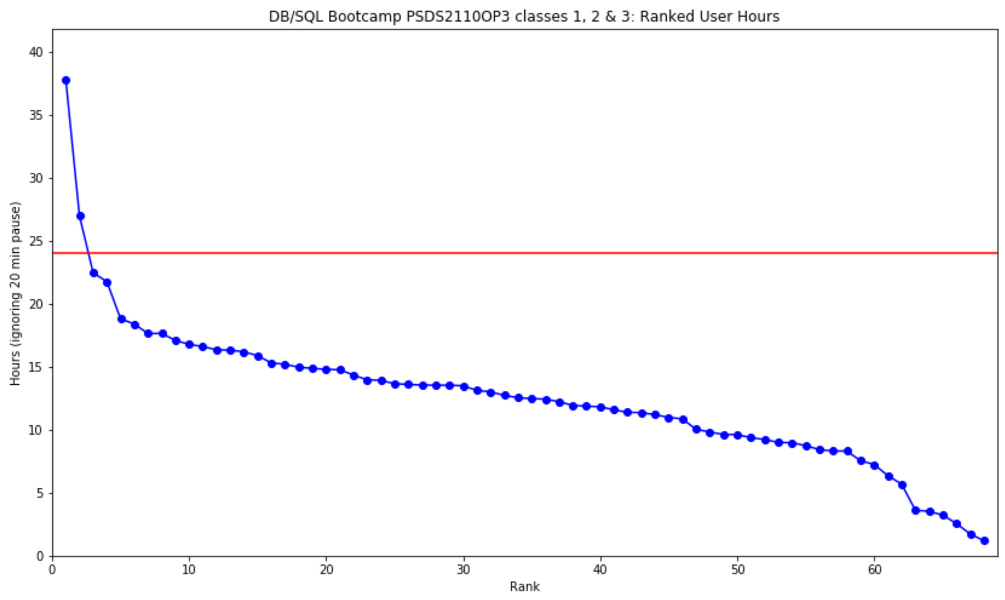


Figure 21. Routing Report - Ranked user hours combined for three class instances of PSDS2110.
Ranking the times for all students in three classes of the DB/SQL Boot Camp during the first five business days of each class. A cutoff for Routing at around 5 hours seems appropriate.

5.4.2. Student Engagement - Daily Hours

How is an individual student spending their time throughout the course? Examining the hours a student spends each day of the course and whether those hours are during instruction time can provide more detailed information regarding the effort put forth and whether instruction hours are utilized. For these plots, the hours are summed for each student-day and plotted as a stacked bar chart of instruction and non-instruction hours and a line chart of the cumulative total.

The charts for four students in Python Boot Camp - PSDS 2100 OP3-3 are provided for comparison. The students are identified by their time rank, ranked by the time spent for the entire course (not just the routing period). For the bar chart, a red line at 8 hours marks the maximum instruction hours possible on a day of instruction. For the cumulative line chart, the reference line is at 40 hours for five days of instruction.

The daily hours of the student ranked 14 are shown in Figure 22. This student worked entirely within the five instruction days, putting in around 4 hours a day. We must remember however, that not all time spent is captured by our methods.

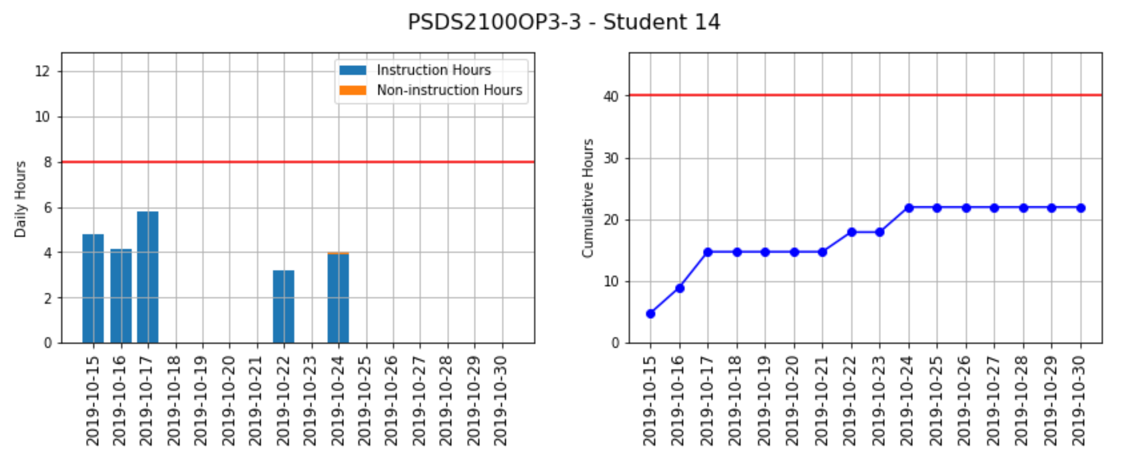


Figure 22. The daily hours of the student hourly-ranked 14 in PDSS2100OP3-3.
This student worked entirely within the five instruction days.

The daily hours of the student ranked 10 are shown in Figure 23. This student worked mostly outside of instruction hours and put in hours at a fairly steady pace, although not every day.

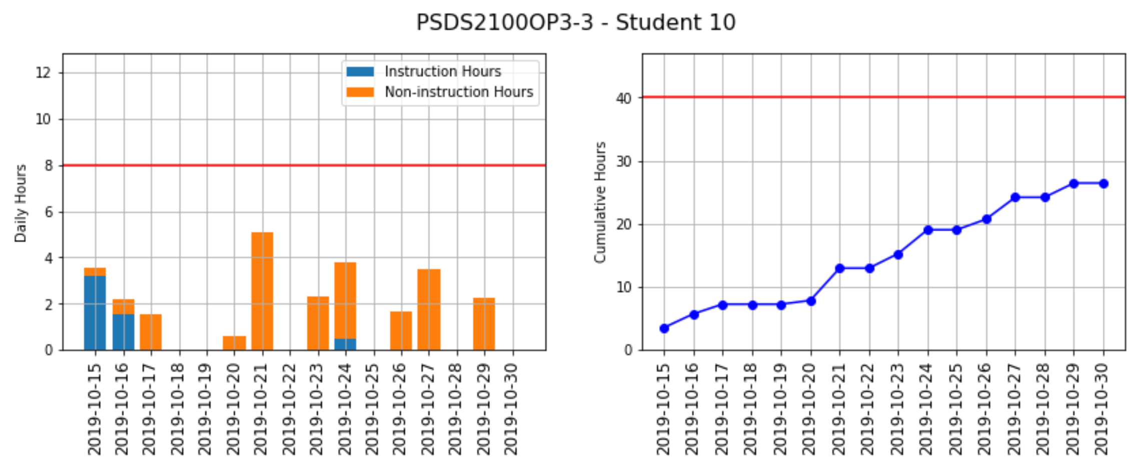


Figure 23. The daily hours of the student hourly-ranked 10 in PDSS2100OP3-3.
This student worked mostly outside of instruction hours and put in hours at a fairly steady pace throughout the course, although not every day.

The daily hours of the student ranked 1 are shown in Figure 24. This student worked both during and outside of instruction hours at a fairly steady pace throughout the first 12 days of the course, finishing early and accumulating about 46 hours.

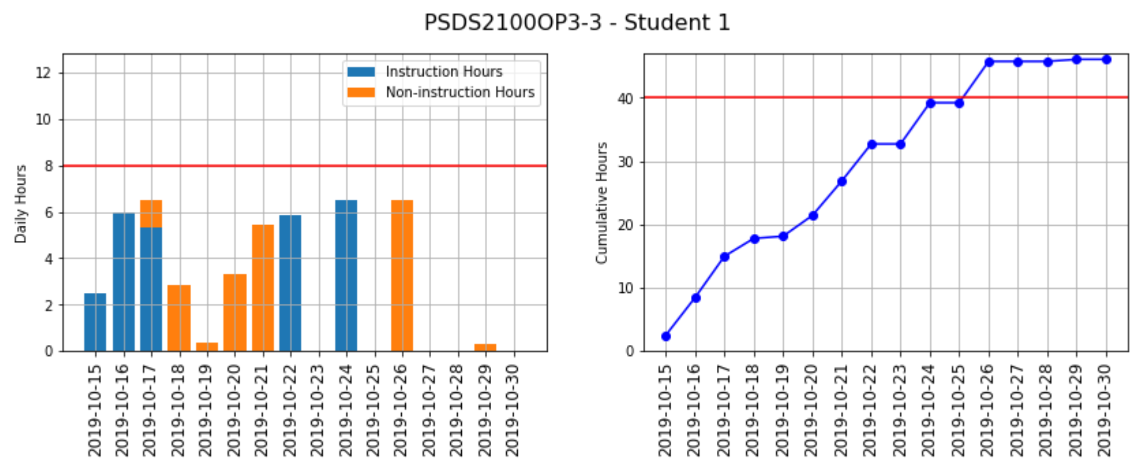


Figure 24. The daily hours of the student hourly-ranked 1 in PDSS2100OP3-3.3.
This student worked both during and outside of instruction hours at a fairly steady pace throughout the first 12 days of the course, finishing early and accumulating about 46 hours.

The daily hours of the student ranked 21 out of 24 are shown in Figure 25. This student put in little time, working less than four hours on the first day, less than two on the second day, and one or less on six other days for a cumulative total of about 9 hours.

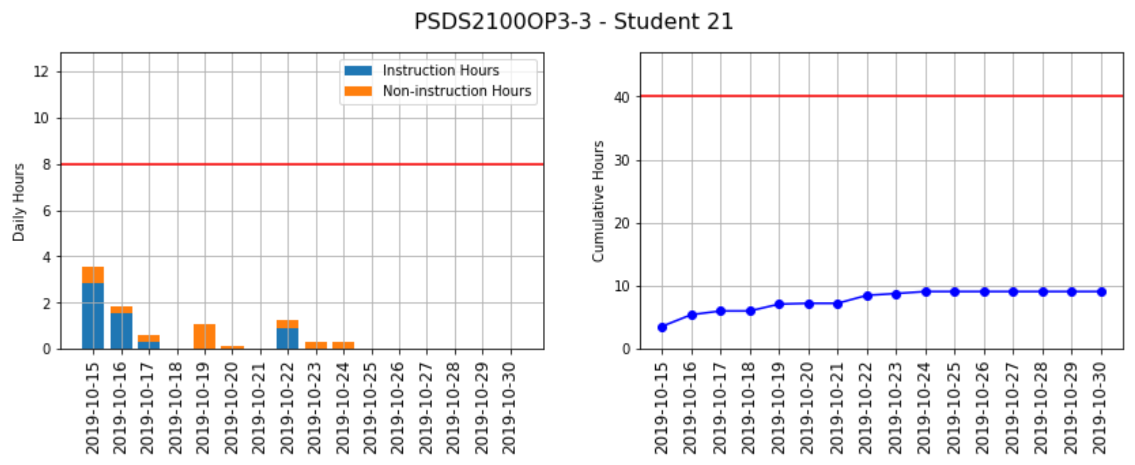


Figure 25. The daily hours of the student hourly-ranked 21 in PDSS2100OP3-3.3.
This student put in little time, working less than four hours on the first day, less than two on the second day, and one or less on six other days for a cumulative total of about 9 hours.

5.4.3. Class Engagement by Activity Type

Which activities (**notebook_types**) is the class spending its time on? How often are they executing code? For these visualizations, the time differences and cell executions are summed, aggregated by **notebook_type** and visualized with a notched box and whisker plot showing outliers (see Figure 26.) Outliers are outside 1.5 times the Interquartile Range (IRQ), and the notch represents the 95% confidence interval for the median. For our example course, PSDS2100 OP3-3, a few students spend a little time with resources, solutions, or other notebooks (such as FAQ notebooks). Again, we must remember that we are only recording time between EXECUTES and SAVES, so not all time is getting logged. Most students are recorded as spending less than an hour on *LABS*, with outliers ranging up to three hours. More time is spent on the *PRACTICES* but still 75% of the values fall around an hour or less. The largest number of hours spent on *PRACTICES* for students not considered outliers is less than 3 hours, and outliers have a great range with the majority from 3 to 7 hours, and one almost at about 10.5 hours. A similar pattern is seen for the number of cell executions.

Both the hours spent and cell executions are heavily skewed toward low values, creating an interquartile range that is very small. This results in many outliers. Especially for the *PRACTICES*, the outliers have a large range. This indicates some students are spending much more time on the *PRACTICES* than many students. Perhaps these students are skipping *LABS*, making the *PRACTICES* harder for them to complete, or perhaps this pattern is due to there being students of widely varying abilities in the course. As a boot camp, this course is offered for students to both learn Python or brush up their skills if they have not used the language in a while. The skew toward low amounts of time and fewer executions are suspected to be students that are brushing up, and the large range for the top half of the box-and-whisker is thought to represent students new to Python. We will look at the balance of time spent between *PRACTICES* and *LABS* in Section 5.4.7 to see if some students are skimping on time spent in the *LABS*.

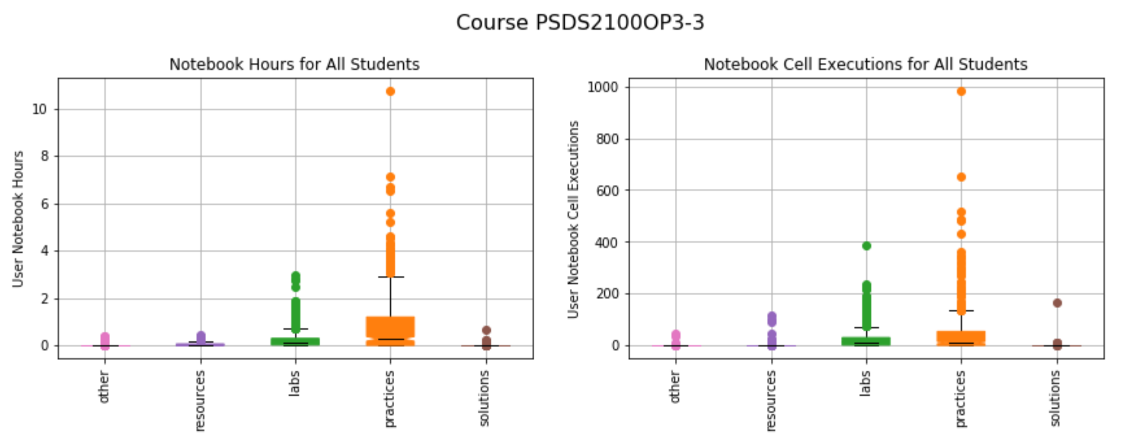


Figure 26. PSDS2100OP3-3 class engagement by activity type.

5.4.4. Class Engagement by Activity Type Within a Module

Because the Interquartile range is so tight for Class Engagement by Activity Type, more information may be apparent by drilling down into a module to get a finer-grained view. Figure 27 shows student engagement by Activity Type for Module 1 in our example course, PSDS2100 OP3-3. As expected from the previous chart, there is little engagement measured with the resources, solutions, or other notebooks. Also as expected, there is more time spent on *PRACTICES* than *LABS*. The distribution of the time spent on *LABS* is tight, with 75% of students spending less than a half hour on the *LABS* and the largest outlier only reaching 1.5 hours. There is a greater range for the *PRACTICES*, especially in the top 50% which ranges from about 0.75 hours to 3.5 hours, with two outliers around 4 hours.

Cell executions are a bit different, with the number of cell executions having a higher median, 3rd quartile value, and upper whisker. It could be however, that there are more cells to execute in the *LAB* notebooks.

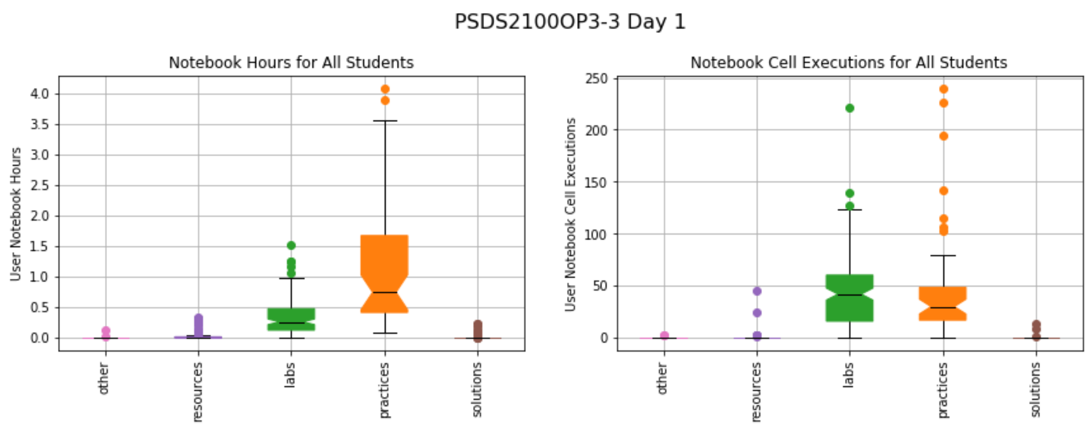


Figure 27. PSDS2100OP3-3 class engagement by module and activity type.

5.4.5. Class Engagement by Module/Notebook

More informative still, is to drill down another level to view Class Engagement by Module/Notebook.

For Day 1, in the plot for Notebook Hours shown in Figure 28 we can see six *LAB* notebooks (shown in green) were provided and two *PRACTICES* (displayed in orange.) The first *PRACTICE* listed occupied most of the students' time and had a large range of time spent. The *LAB* notebooks have a pretty tight range.

The IQR for the first *PRACTICE* notebook extends from 1.75 to 2.6 hours, with the total range extending from almost zero to a bit over four.

The cell executions for Day 1 in Figure 29 show more variability between Lab notebooks, although the time range remains tight for each *LAB* notebook. The same *PRACTICE* notebook has a higher range for the number of cell executions, with half the class tight in the lower range and half the class in the upper range more spread out, with two outliers. The extensions at the bottom of the box represent a 95% confidence interval for the median that extends below the 25th percentile.

An instructor viewing these plots might ask themselves, "Does the range of time spent and number of executions seem reasonable for the course design and the known characteristics of the student cohort, or does more preparation or scaffolding need to be provided for the first *PRACTICE* notebook?"

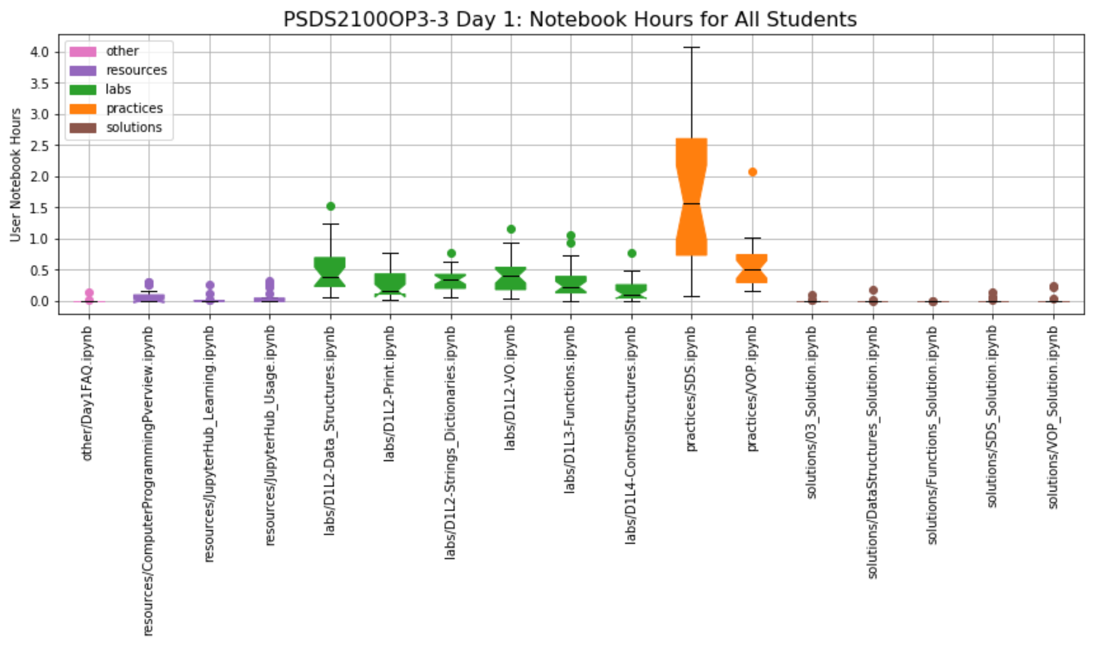


Figure 28. PSDS2100OP3-3 class engagement for Day 1 notebooks - hours.

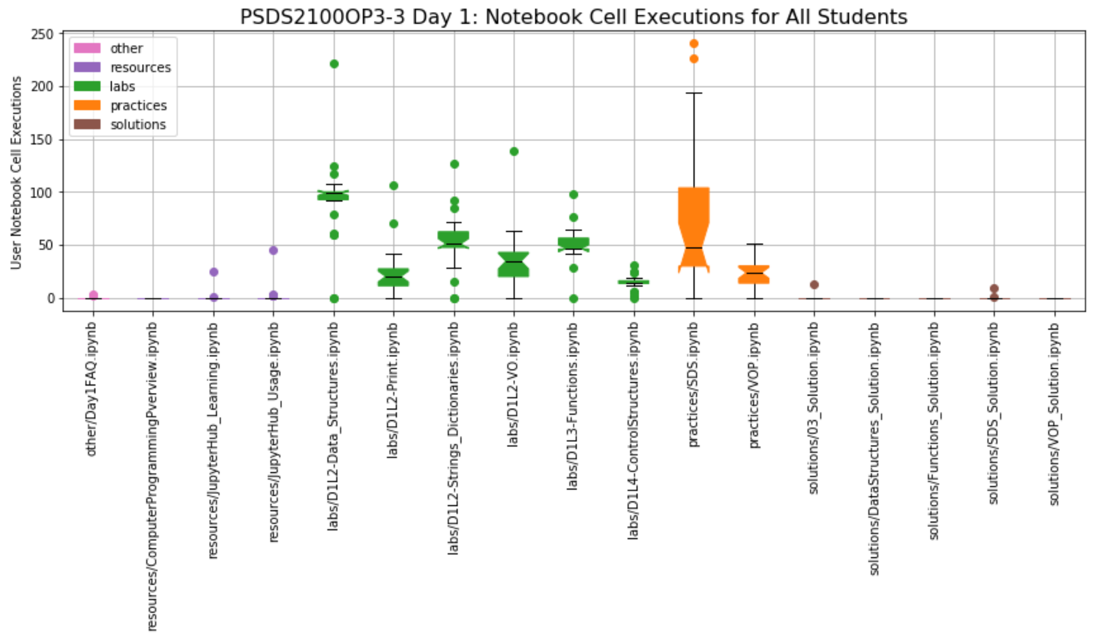


Figure 29. PSDS2100OP3-3 class engagement for Day 1 notebooks – cell executions.

The next four figures show the Notebook Hours for All Students for the remaining modules (**days**.) Figure 30 shows the hours for Day 2. Notice that there are two *PRACTICES* with one dot. These are student-created copies of the *PRACTICE* notebook. A canonical list of the **notebooks** could prevent these from showing up, but they do preserve some information on student engagement. If many students adopted this practice and named their copies differently it would clutter the plot too much. The third *PRACTICE* (other than the copies) is optional and may have a tight range due to low participation.

The first two *PRACTICES* have similar means, but different ranges. The first *PRACTICE* is very tight between the 25th percentile and the median, with a larger range between the median and the 75th percentile. Compared to the second notebook, this first notebook took more time for students between the median and 75th percentile.

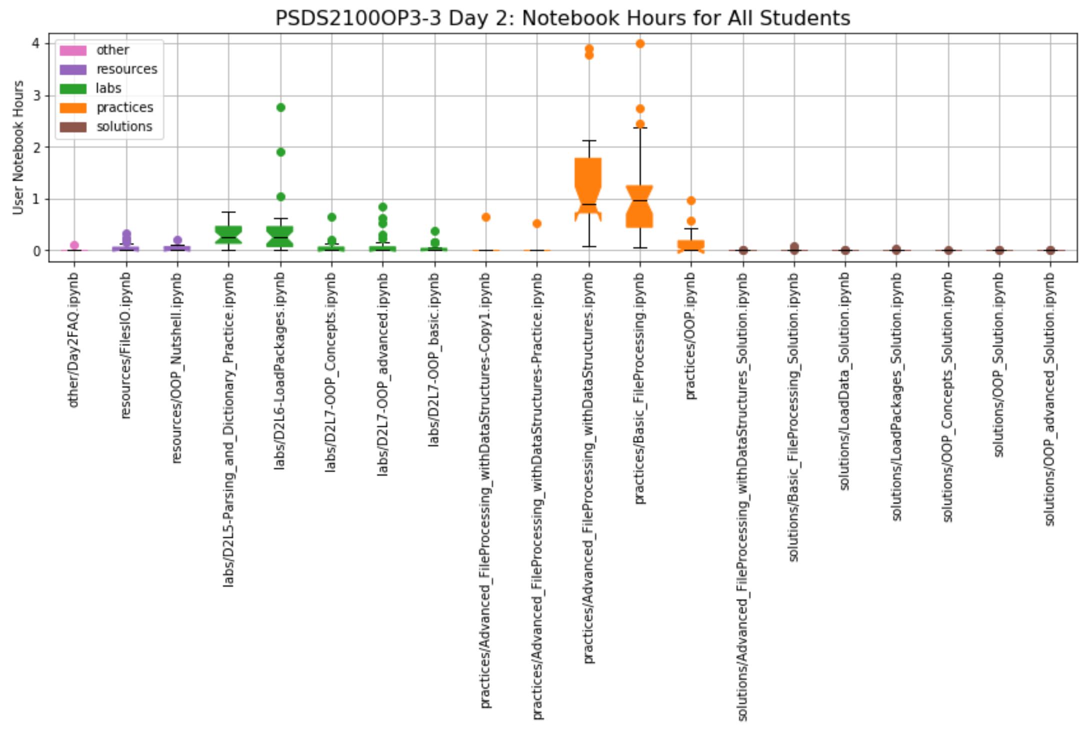


Figure 30. PSDS2100OP3-3 class engagement for Day 2 notebooks - hours.

Figure 31 shows the hours for Day 3. The second *PRACTICE* is optional. We can see that a lot of preparation for the *PRACTICE* for this module occurs with many *LABS*. The IQR for this notebook is about 0.5 to 1.5, with the range between whiskers going from 0 to about 2.5. This is not as severe as our example from Module 1, and there are only two outliers. More students seem well prepared for this *PRACTICE*. Also, the instructor may have considered the many *LABS* to be worked on in planning the expected time for students to invest in the *PRACTICE*.

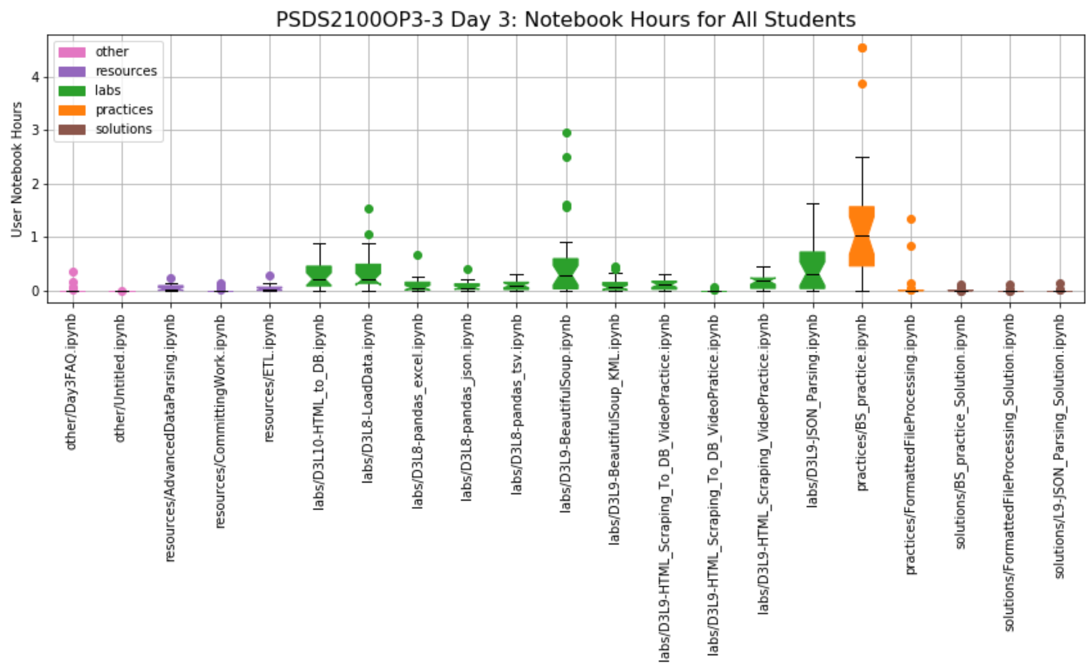


Figure 31. PSDS2100OP3-3 class engagement for Day 3 notebooks - hours.

Figure 32 shows the hours for Day 4. There is one *PRACTICE* copy in this plot. The first *PRACTICE* clearly takes less time, while the next two take similar time for the first 50% of the class. The second 50% find the third *PRACTICE* notebook to take more time (except for one student.) The top of the whisker is around 6.5 hours, which is the most time we've seen spent on a *PRACTICE* so far.

Figure 33 shows the hours for Day 5. There are three *LABS* and the *FINAL PROJECT PRACTICE*. The *FINAL PROJECT* does not take as much time as the third *PRACTICE* in the previous module. Only the instructor can answer if this fits the instructor's intent.

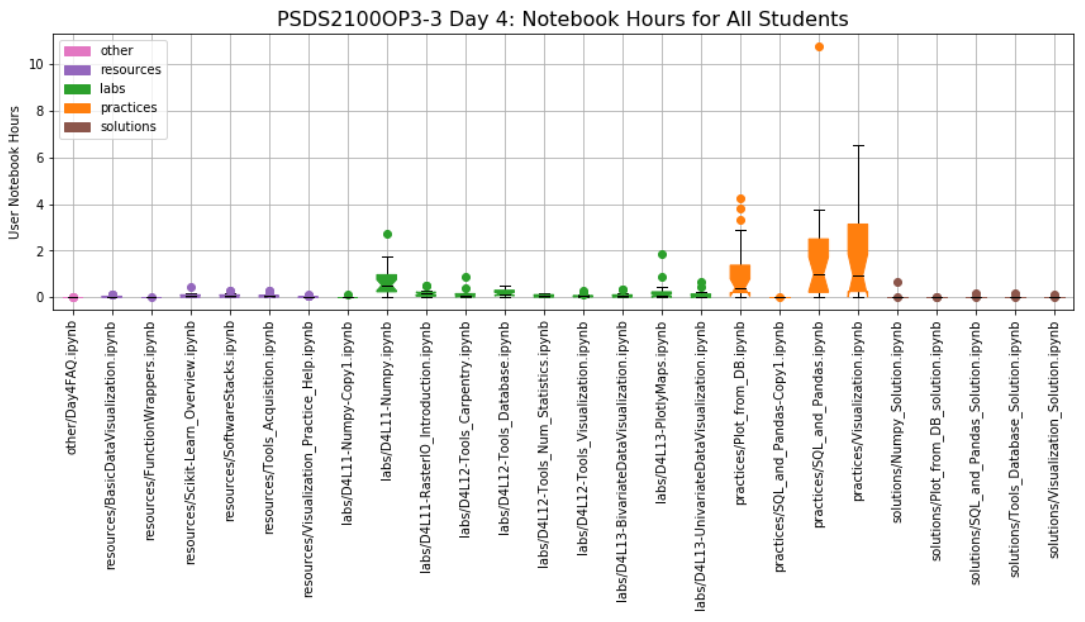


Figure 32. PSDS2100OP3-3 class engagement for Day 4 notebooks - hours.

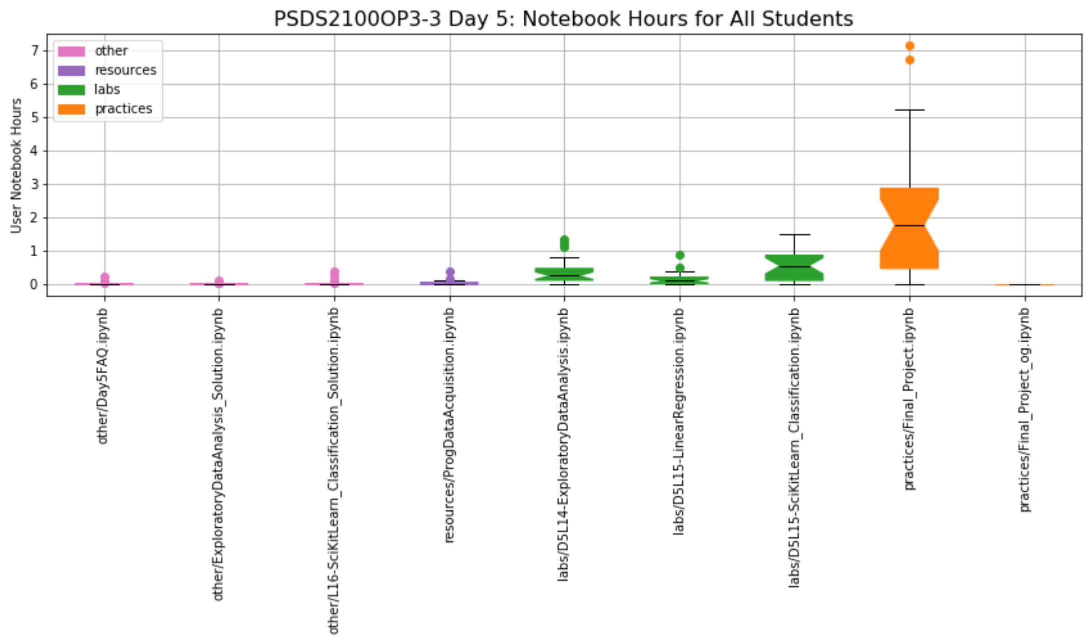


Figure 33. PSDS2100OP3-3 class engagement for Day 5 notebooks - hours.

5.4.6. Student Engagement by Module/Notebook

We saw in the Class Engagement by Module/Notebook plots that the *LABS* typically took little time and the range was tight. The *PRACTICES* showed greater variability, but often had a tight range for the bottom 50%, and a much higher range, sometimes with outliers, for the upper 50%. What can we learn by looking at Individual Student Engagement by Module/Notebook?

Figure 34 shows the hours spent by Student 1 (our overachiever) in each notebook for Day 4. This student is spending time in the *LABS*, especially one *LAB*, but a lot of time in the *PRACTICES*. This plot is for the entire timespan of the course -visualizing *when* the materials were accessed (using a pattern for early, on-time, or late) could be informative. For instance, the brown bars are for the solutions, which were visited after the material was due while working on the next module. Are students visiting other notebooks in earlier modules when working on later ones?

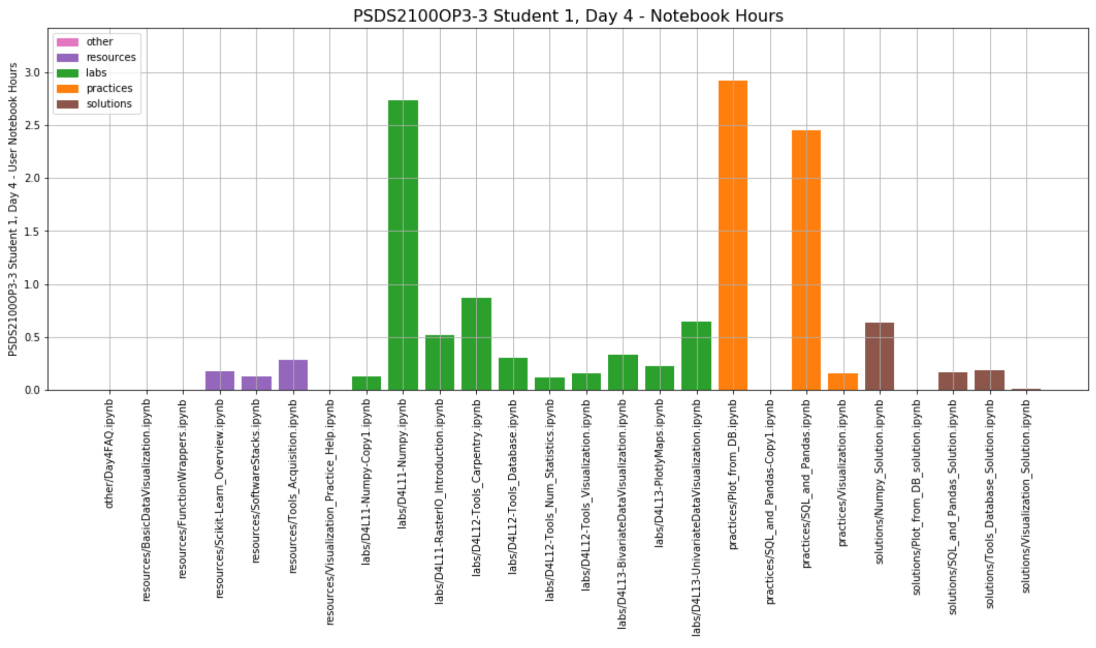


Figure 34. PSDS2100OP3-3 Day 4 notebook hours for Student 1.

Figure 35 shows the hours spent by Student 2 in each notebook for Day 4. For the same day as the previous example, Student 2 is spending most of their time in the *PRACTICES* without much time in the *LAB* and did not visit the solutions at all. This is a poor use of time.

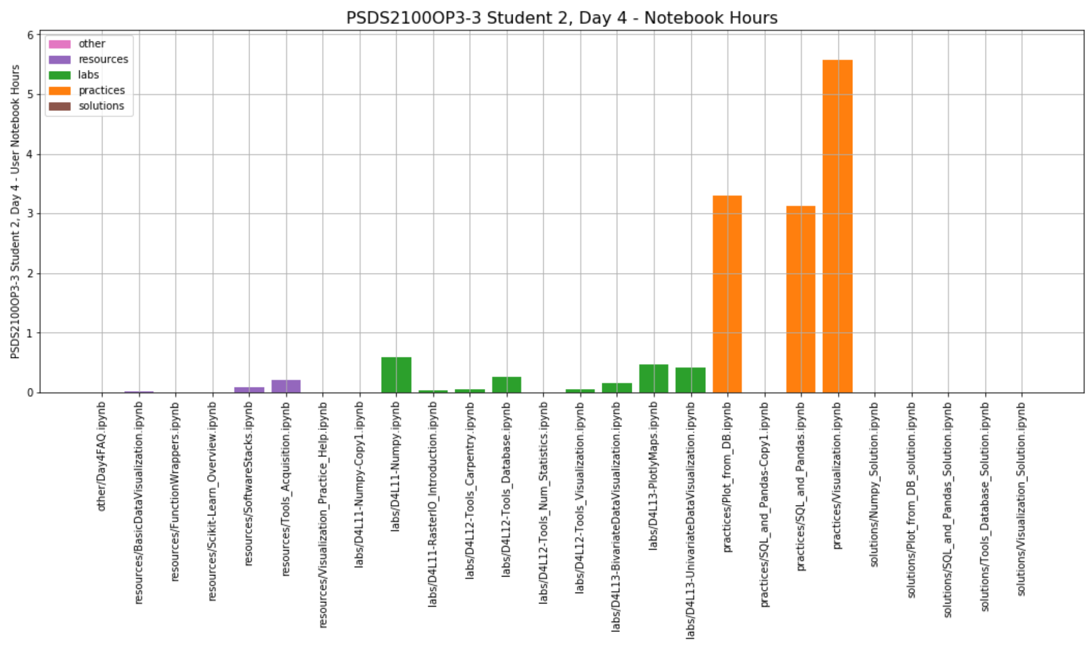


Figure 35. PSDS2100OP3-3 Day 4 notebook hours for Student 2.

Figure 36 shows that on Day 3, Student 7 spent a lot of time in the *LABS*. But we see in Figure 37 that for the next module, Day 4, the same student spent almost no time in the *LABS*, so this student's work pattern was not consistent.

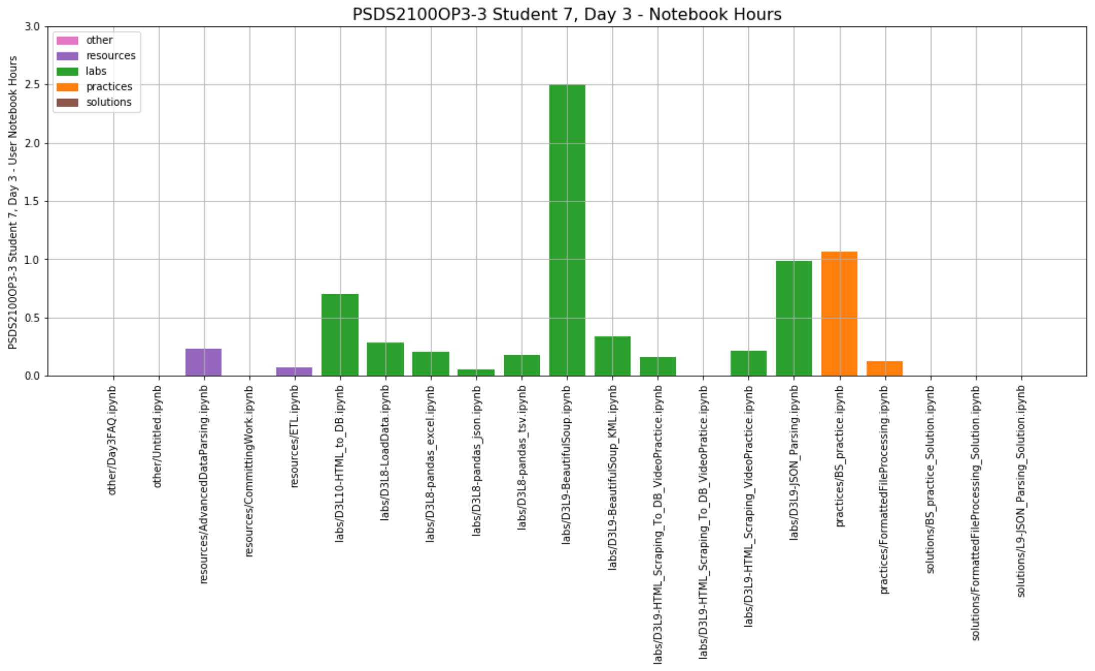


Figure 36. PSDS2100OP3-3 Day 3 notebook hours for Student 7.

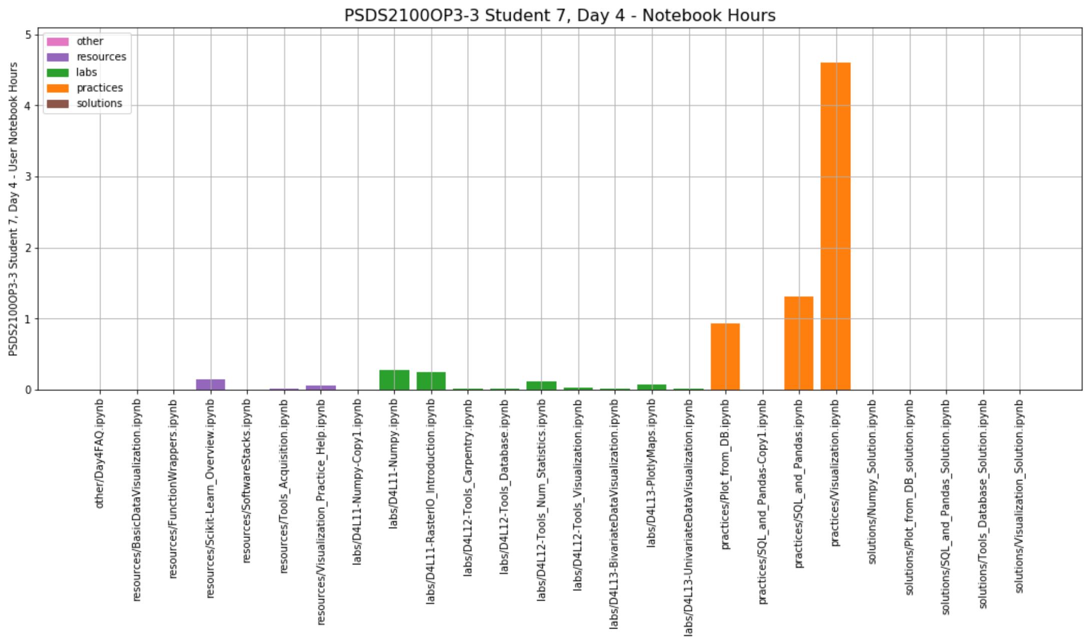


Figure 37. PSDS2100OP3-3 Day 4 notebook hours for Student 7.

5.4.7. Practice vs Lab Balance

In Section 5.4.3 where we visualized Class Engagement by Activity Type, we noted that for the *PRACTICES*, the outliers had a large range of time spent and we considered whether these outlier students were skipping *LABS*, making the *PRACTICES* more difficult. This last chart provides a perspective on that question by plotting engagement in the *LABS* vs engagement in the *PRACTICES*, for both hours and number of executions (see Figure 38.)

For the most part, students spend comparable time and effort in the *PRACTICES* and *LABS* compared to their peers with a few exceptions, indicated in red. Student 1 spent more time and interacted more in the *LABS*, than any other student. Students 2 and 3 spent more time and performed more executions in the *PRACTICES* than in the *LABS*, although the effort spent in the *LABS* was similar to their peers. Perhaps these students are new to Python. Student 14 performed many executions in the *PRACTICES*, but the hours spent was similar to their peers. This student could be characterized as more interactive than their peers for the time spent.

How do the grades for these students compare to their peers? This plot could be color-coded by grade.

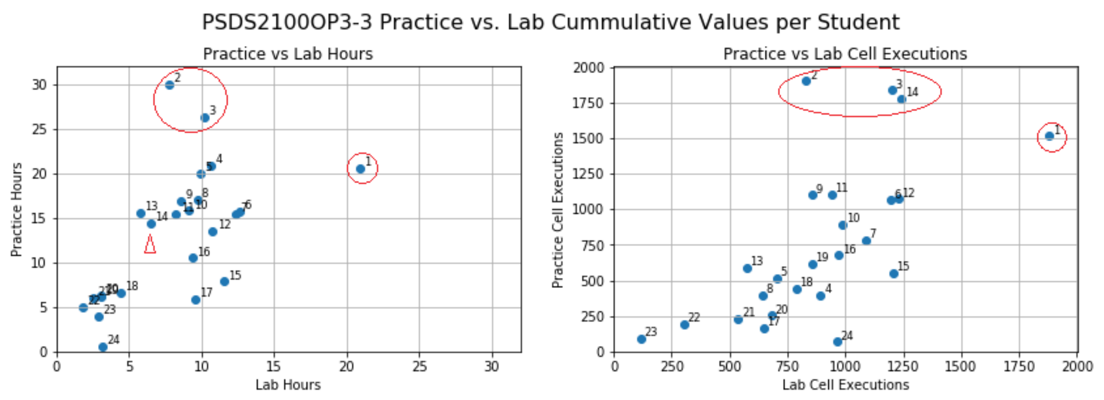


Figure 38. PSDS2100OP3-3 practice vs lab cumulative hours and cell executions for all students.

Chapter 6: Recommendations and Further Research

This thesis has barely scratched the surface of what is available to achieve and the benefits that could be gained from analyzing student activity in Jupyter Notebook.

6.1. A Comprehensive Jupyter Learning Analytics System

The TRAK custom Jupyter logs for EXECUTE_INPUT messages capture additional features such as the **notebook_path** and the executed **code** that are not captured in the regular EXECUTE_INPUT logs, allowing a detailed, efficient, and accurate analysis of EXECUTE_INPUT actions. Custom logging with rich data capture could be extended to other types of messages, to capture a complete picture of student interaction from page load, through code cell edits, cell additions and subtractions, every type of execute, and closing the page. This data could then be mined for more detailed analysis.

Among these custom messages, capturing a cell identifier for activities within a specific cell could provide a very detailed view of executions, errors, and code edits, allowing instructors to fine-tune their materials based on student interactions.

However, as exciting as the potential for rich data capture is, the amount of logging would need to be carefully balanced with performance requirements.

In combination with Jupyter custom logging, configuring additional Graylog extractors for key features on all relevant messages would provide indexing on those features, allowing for efficient, real-time searches for those messages in the Graylog browser as well as more efficient data ingestion for the reports.

The sister program to PSDS, MU's Master's Program in Data Science and Analytics, would benefit greatly from the application of these analyses. An ambitious vision would be to provide a structure for organizing course materials and their timing, tied to custom logging and analytics features, to allow generalizing to other programs as an open source project integrating with Jupyter Notebook and JupyterHub.

Instructor, student, and administrator dashboards would provide easy access to relevant analytics on demand.

6.2. Additional Descriptive Analysis

6.2.1. New Features

The number of executions varies quite a bit between notebooks. Normalizing the number of executions per notebook would allow us to see how repetitively a student executes the cells in a notebook.

There is a command to run all cells in a notebook. This type of engagement is different from working through cell by cell and thinking about the content. This difference could be captured by custom logging the "run all" command and/or calculating the average time between executions.

Tracking the type and number of errors per notebook or cell would give information on common errors, students that are having difficulty, and notebooks or cells that are particularly difficult.

Three other features were found in the literature for learning analytics applied to LMS logs. Timing of engagement is whether the work is early, on-time or late compared to the course design. Nguyen et al. showed that different patterns in this feature were associated with student performance as well as engagement with different types of activities [12]. Login frequency (how many separate logins there are) and regularity of learning interval, (the standard deviation of the average login time) were used in [11] and [13] to identify different time management strategies. These may be less applicable to the PSDS program due to its accelerated pace but would be useful features in other programs.

A PSDS course lead has suggested other features such as whether a course is blended or entirely online; which instructor is assigned to the class instance; and what the NGA location of the class is. These could identify learning environment features affecting student activity patterns or performance.

6.2.2. Additional Visualizations

New features mentioned above would all call for new visualizations, designed to highlight the relevant insights.

Stacked bar charts used to visualize instruction and non-instruction hours for all the students in a class could also be used to visualize other breakdowns: whether the hours spent were early, on-time, or late for the assignment; the hours logged for each type of notebook; and the hours spent in each module.

Student final grades added as color codes to the line chart for Ranked Student Hours and the scatter plot for *PRACTICE* vs *LAB* Balance (or *LAB/PRACTICE* vs *EXERCISE*) would allow patterns that affect performance to be discerned visually.

A Probability Mass Function plot of the average time per student in each type of notebook across all instances of a course would be useful to course leads as another perspective informing course design. A similar plot of the average time per module would likewise be useful.

6.3. Predictive Analytics

Predicting student outcome as shown by final score or even pass/fail, is a natural goal for predictive learning analytics. Calculating correlation coefficients between different features and outcomes would identify which features are independently the most influential.

Unsupervised learning using cluster analysis could identify types of students based on multiple features, which could be useful in describing groups of students which could then be targeted in course design. But clustering could also include outcomes to predict student performance.

Clustering algorithms could be applied to identify patterns of student activity among identified features at different levels of granularity, including time, executions, and outcomes, whether the outcome is final grade or pass/fail. A few examples are:

- Total time, normalized average executions, and outcome.
- Instruction time, non-instruction time, normalized average executions, and outcome.

- Early hours, on-time hours, and late hours spent, normalized average executions, and outcome.
- Early hours, on-time hours, and late hours in *LABS*, *PRACTICES*, and *EXERCISES*, normalized average executions for *LABS*, *PRACTICES*, and *EXERCISES*, and outcome.

6.4. Prescriptive Analytics

Descriptive and predictive analytics form the basis for prescriptive analytics, leading to interventions.

Alerts provide the opportunity to make timely interventions. A notification that a given notebook or module is taking too much time on average, or students in a given notebook are having a lot of errors on average would give the instructor an opportunity to provide additional explanation through a video or other supplemental material, rather than waiting for the next instance of the course to make changes.

Identifying an individual student fitting a pattern or student “type” associated with a poor outcome would give the instructor an opportunity to reach out to the student to provide counsel to help correct their course trajectory.

Retrospectively, poor outcomes that are linked to engagement patterns with certain notebooks or modules, or with student types as determined by clustering could indicate design changes are warranted.

CONCLUSION

Jupyter Notebook is particularly well-suited for STEM instruction and activity-based learning, providing a superior environment for building skills and developing a deep understanding of course material. The logs generated in Jupyter provide a window to view student activity patterns that have previously been hidden in offline courses and courses utilizing traditional LMS systems.

The PSDS program offered by the University of Missouri for NGA is an ideal program for a case study in applying learning analytics to the Jupyter environment. The Jupyter-centered delivery of course material and the design and organization of PSDS courses allow insights into students' active-learning patterns and how interactions with preparatory material influence interactions with subsequent material.

For this thesis, log messages originating in Jupyter were extracted from the Graylog log management system, transformed into useful student-activity features, and loaded into a PostgreSQL database for long-term storage and historical reporting. A variety of visualizations of student activity useful for administrators, instructors, and students have been produced.

New insights have been demonstrated and recommendations made for building a richer and more utilitarian logging and analytics system, with an expansive vision for what can be achieved mining Jupyter logs for learning analytics.

REFERENCES

[1] C. C. Bonwell and J. A. Eison, “Active Learning: Creating Excitement in the Classroom. ERIC Digest,” ASHE-ERIC Higher Education Reports, The George Washington University, One Dupont Circle, Suite 630, Washington, DC 20036-1183, Sep. 1991. Accessed: Apr. 09, 2020. [Online]. Available: https://eric.ed.gov/?id=ED340272.

[2] S. Freeman *et al.*, “Active learning increases student performance in science, engineering, and mathematics,” *Proceedings of the National Academy of Sciences*, vol. 111, no. 23, pp. 8410–8415, Jun. 2014, doi: 10.1073/pnas.1319030111.

[3] “Project Jupyter.” https://www.jupyter.org (accessed Jan. 15, 2020).

[4] C. Rizo Maestre, F. Aznar Gregori, M. Pujol López, and R. Rizo Aldeguer, “Jupyter Notebook: Theory and Practice of Mathematical Models in Engineering and Architecture,” in *Proceedings of ICERI2016 Conference*, Seville, Spain, Nov. 2016, pp. 6523–6530, doi: 10.21125/iceri.2016.0492.

[5] B. W. Weber, “Climbing Bloom’s Taxonomy With Jupyter Notebooks: Experiences in Mechanical Engineering,” in *Proceedings of the ASME International Mechanical Engineering Congress and Exposition*, Salt Lake City, UT, USA, Nov. 2019, vol. 59421, p. V005T07A022.

[6] J. M. Perkel, “Why Jupyter is data scientists’ computational notebook of choice,” *Nature*, vol. 563, no. 7729, pp. 145–146, Oct. 2018, doi: 10.1038/d41586-018-07196-1.

[7] G. Siemens, “Learning analytics and educational data mining: towards communication and collaboration,” in *Proceedings of the 2nd international conference on learning analytics and knowledge*, Vancouver, BC, Canada, May 2012, pp. 252–254, doi: 10.1145/2330601.2330661.

[8] N. Sclater, A. Peasgood, and J. Mullan, “Learning analytics in higher education: a review of UK and international practice,” Jisc, London, United Kingdom, Apr. 2016.

[9] “What are Analytics? | Canvas Basics Guide | Canvas Guides (en).” https://guides.instructure.com/m/67952/l/724559-what-are-analytics (accessed Apr. 10, 2020).

[10] R. Mazza and V. Dimitrova, “CourseVis: A graphical student monitoring tool for supporting instructors in web-based distance courses,” *International Journal of Human-Computer Studies*, vol. 65, no. 2, pp. 125–139, Feb. 2007, doi: 10.1016/j.ijhcs.2006.08.008.

[11] I.-H. Jo, D. Kim, and M. Yoon, “Analyzing the log patterns of adult learners in LMS using learning analytics,” in *Proceedings of the Fourth International Conference on Learning Analytics And Knowledge*, Indianapolis, Indiana, USA, Mar. 2014, pp. 183–187, doi: 10.1145/2567574.2567616.

[12] Q. Nguyen, M. Huptych, and B. Rienties, “Linking students’ timing of engagement to learning design and academic performance,” in *Proceedings of the 8th International Conference on Learning Analytics and Knowledge - LAK ’18*, Sydney, New South Wales, Australia, 2018, pp. 141–150, doi: 10.1145/3170358.3170398.

[13] T. Yu and I.-H. Jo, “Educational technology approach toward learning analytics: relationship between student online behavior and learning performance in higher education,” in *Proceedings of the Fourth International Conference on Learning Analytics And Knowledge*, Indianapolis, Indiana, USA, Mar. 2014, pp. 269–270, doi: 10.1145/2567574.2567594.

[14] “$12 Million Federal Contract to MU Will Establish Education Program for National Intelligence Agency | News Bureau, University of Missouri.” https://munewsarchives.missouri.edu/news-releases/2016/0831-12-million-federal-contract-to-mu-will-establish-education-program-for-national-intelligence-agency/ (accessed Apr. 13, 2020).

[15] “Looking at Logs — The Littlest JupyterHub v0.1 documentation.” http://tljh.jupyter.org/en/latest/troubleshooting/logs.html#journalctl-tips (accessed Apr. 25, 2020).

[16] “Messaging in Jupyter — jupyter_client 6.1.3 documentation.” https://jupyter-client.readthedocs.io/en/stable/messaging.html (accessed Apr. 25, 2020).

[17] “Open Source Log Management for All | Graylog.” https://www.graylog.org/products/open-source (accessed Mar. 03, 2020).

[18] “Managing Centralized Data with Graylog | The Graylog Blog.” https://www.graylog.org/post/managing-centralized-data-with-graylog (accessed Mar. 03, 2020).

[19] “Elasticsearch Reference [6.8] | Elastic.” https://www.elastic.co/guide/en/elasticsearch/reference/6.8/index.html (accessed Apr. 13, 2020).

[20] “Welcome to the Graylog documentation — Graylog 3.2.0 documentation.” http://docs.graylog.org/en/3.2/index.html (accessed Mar. 03, 2020).

[21] A. Craighead, “interputed/grapi,” Jan. 14, 2020. https://github.com/interputed/grapi (accessed Apr. 28, 2020).

[22] “Graylog REST API — Graylog 3.2.0 documentation.” https://docs.graylog.org/en/3.2/pages/configuration/rest_api.html (accessed Apr. 28, 2020).