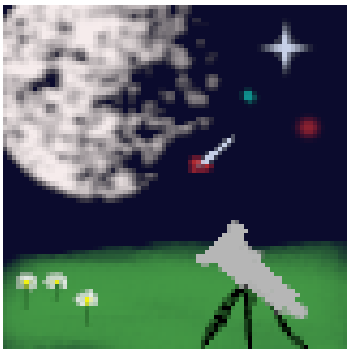
Team First Light



Technological Feasibility

3 November 2021

Sponsors

Dr. David Trilling

Dr. Mike Gowanlock

Faculty Mentor

Felicity Escarzaga

The Team

Matt List- mjl79@nau.edu

Carson Pociask - cmp557@nau.edu

Jakob Nelson - jrn235@nau.edu

William Fuertes- wf69@nau.edu

Jensen Roe - jr2999@nau.edu

Table of Contents

1.0 Introduction

The big data revolution is well underway and there is no end in sight for its role in everyday life. 90% of the world's data has been created in just the last 2 years and sites like Facebook are creating 4 petabytes worth every single day! (Petrov 2021, Roy 2020). Not only does the modern world need resources to be able to send and capture data, but many aspects of the big data revolution also require analysis and processing. Forbes predicts that more than 150 zettabytes will need analysis by 2025. These massive sets of data play an important role in many people's lives. Researchers, businesses, and the average technology user all have a hand in driving this revolution. At the current growth rate, it's important that technologies keep up with the demand of this data.

While this tech-driven era may seem like a seamless trend upwards, not all users of big data creation have kept up with the demand. The field of astronomy is a huge player in the big data revolution yet the tools in place for researchers and scientists to use fall behind and are quite lacking in features. Clean, interactive visualizations of very large data sets for night sky data are far and few between. Instead, the solutions currently in place are difficult to read, time-consuming to understand, and often do not provide graphical interfaces. This hinders anyone dealing with large data in astronomy; there exists a bottleneck of workflow when it comes to manipulating data in a way to provide further insight into the observations. The clients for the project, Dr. David Trilling and Dr. Mike Gowanlock, have experienced this problem first hand. Both of the sponsors are professors at Northern Arizona University (NAU): Dr. Trilling is a professor of Astronomy and Planetary Science and Dr. Gowanlock is an assistant professor in the School of Informatics, Computing, and Cyber Systems. Focusing on the current work of the clients, Trilling and Gowanlock are involved in collecting and observing all-sky observations at the Vera C. Rubin in Chile that will begin in 2024 under the Legacy Survey of Space and Time (LSST). After processing through the Zwicky Transient Facility in San Diego, data related to asteroids will be collected here at NAU's database where it will need to be displayed in an interactive, informative manner. Dr. Gowanlock and Dr. Trilling both have experience dealing with numerous astronomical visualization tools and websites during their research, such as ANTARES (antares.noirlab.edu/loci) and MARS (mars.lco.global/). These two websites encompass some of the biggest problems that the clients and other astronomical scientists face: an overwhelming and unwelcoming home page, a lack of data visualization, and little interactivity among the data presented. More specifically, both of the clients are dealing with an unusable website that is not hosted at NAU and doesn't utilize libraries that are sufficient for continuing development. While the tools exist to capture massive amounts of data that astronomy produces, modern methods to further analyze the data through the use of things like visualization are lacking. With the proper tools and resources, astronomy has the ability to continuously advance what humans know about space while keeping up with the pace of the big data revolution.

The envisioned solution for Dr. Trilling and Dr. Gowanlock is a web-based visualization tool capable of providing a graphical interface that provides clean data representation and interactivity. This solution will need to be efficient while handling millions of data points out of a database here at NAU.

2.0 Technological Challenges

Due to the project primarily building a graphical user interface (GUI) for data visualization of large astronomical data sets, data visualization libraries will be utilized in the solution to be hosted on NAU servers. The solution will:

- Be written in either JavaScript, R, or Python
- Take in a large data set from an NAU database
- Store the large amount of data in memory
- Build data visualizations, from both the larger data set, and smaller data sets
- Make the interactive visualizations available on a web page, or series of web pages.

The viability and performance of technologies at each stage of the list will need to be assessed to determine the best option for use. However, it should be noted that the data science landscape is largely either in Python or R, with a few JavaScript options also available. Options across these languages will be assessed and the best language to use will be determined. From this language choice, the best packages to solve the challenges can be analyzed and selected.

3.0 Technology Analysis

Out of the previously noted technological challenges, the following technological issues and design decisions will be looked further into:

- **Which programming language to use** - To stay consistent through the use of different tools during development, a base language needs to be selected.
- **Interacting with a database** - In order to conduct any analysis or operations in the first place, the data must first be pulled from tables in the NAU database.
- **Storing the data in memory** - Once data is pulled from the database, it will need a way to be interactive within memory.
- **Plotting the data** - The ability to transform data in tabular formats into images, charts, maps, and graphs is critical to the project at hand.
- **Controlling dashboard style** - The visualized and interactive data needs to be accessible online so the clients and other researchers can utilize the technologies.

These play the most important role in the project and when combined, determine the overall viability of the proposed solution. Below is the beginning of the analysis into these challenges and decisions.

4.0 Challenge Ⅰ - Which Programming Language to Use?

The issue that needs to be addressed is which programming language to interact with large datasets will be used for this project. By using a language that has many libraries for versatility, to a language where it focuses more on statistical computing and graphical interfaces or a language that will be more useful for web development for interactivity and visualization. All of these different aspects are needed for this project of displaying large data sets on a website and being able to filter through the data and displaying said data. With the different aspects of the vast programming languages, there is a solution that will provide the best outcome and characteristics that will best fit this project.

Certain characteristics are needed for this project to be successful. For example, one very important characteristic that is needed is a language that will provide adequate speed when displaying the large datasets which will give the users the ability to search whatever they need in the graphs that the website will provide. Since the website will consist of handling large datasets, speed will be extremely important since users do not want to wait a large amount of time for the graphs to load. The speed will also depend on what libraries will be used and how intricate the graph will be. For instance, Python is known for having a multitude of libraries to work with but some libraries have too much when all that is needed is one method or a simpler graph.

Another characteristic involves having a language that the developers and sponsors would be comfortable using and if some are not comfortable with it, then how easy the developers will be able to learn the language in a short amount of time. When learning something, documentation is key. With good documentation, the developers will have an easier time handling the different languages and will provide a better understanding of the different libraries that can be imported. Most of the libraries that this program will be using consist of reading data and plotting said data. By having a library that has many functions, the developers will have to review which functions in the library will be useful and thus the documentation is needed.

The scoring for each desired characteristic is as follows

- 7-10**: Above Average**. Characteristics scored this way will work great for the desired solution
- 4-6: **Average**. Characteristics scored this way will work just fine, but potentially could be an issue for the desired solution
- 0-3: **Below Average**. Characteristics scored this way will not work for the desired solution

4.1 Candidate Solution #1 - Python

For the first solution, Python will be looked into. Python, which was initially designed by Guido van Rossum in 1991 and developed by the Python Software Foundation is known to have a multitude of libraries that can have a variety of functions and therefore have a lot of versatility. Having a language that the clients and current developers are familiar with will provide extra time to come up with solutions and decrease the time to learn something entirely new. The many courses that the project's developers have taken involved using Python which provided an understanding of the language and added convenience to the clients since that is what they normally use for projects like this. Since the developers already have a stable knowledge foundation, they can help each other out more easily than having to research an unknown subject. At this time, Python is mostly known for how easy it is to learn and simple syntax making it a beginner's guide to programming(Ayer et al. 2014). That being said, since the project's developers and clients were more familiar with Python, it will receive a score of 9 out of 10 in the familiarity criteria.

As for the documentation, some libraries provide comments with the code to make it easier for readers to understand. For instance, **Figure 4.1.1** provides documentation on how to create a triple line graph for different equations which will give developers an easier time to either use it or to modify the code. In the documentation, it also gives an image of the outcome so that it is easy to distinguish what the code did. For this reason, Python will receive an 8 out of 10 in the documentation criteria. In total, Python will receive a 17 out of 20 due to the developers and client familiarity with Python and the easy-to-read documentation that Python provides for its graphing libraries.

```
x = np.linspace(0, 2, 100)  # Sample data.

# Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
ax.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend()  # Add a legend.
```

Figure 4.1.1 *Matplotlib Documentation*

4.2 Candidate Solution #2 - R

Another language considered was the programming language R. R is more well known in the statistical computing and graphics area and is open source. Although R is new to the clients and current developers, it provides many libraries for plotting points as well as reading data and creating graphs from it. The R programming language is similar to the S statistical language since most of the S code runs in the R language (Venerables et al. 2013). R has been around for around 28 years meaning that it is constantly being refined and many people have worked on it to create better libraries. Although its main function as a programming language is for statistical analysis and graphing data, the clients and developers do not have much experience in it. This will create a hindrance when trying to create more complicated code in the project. Therefore, R will receive a score of 4 out of 10.

After looking at the documentation of the different programming languages, R seems to have a similar documentation base to Python. Reading the documentation was simple and **Figure 4.2.1** shows the cheat sheets that come with the documentation for ggplot2. Python has many functions that are similar to R and since it is well known, many resources and tutorials can be found. Since the documentation for the graphing libraries is similar to Python and provides cheat sheets to make it easier to develop code, R will receive a score of 9 out of 10 due to its easy-to-read and helpful documentation

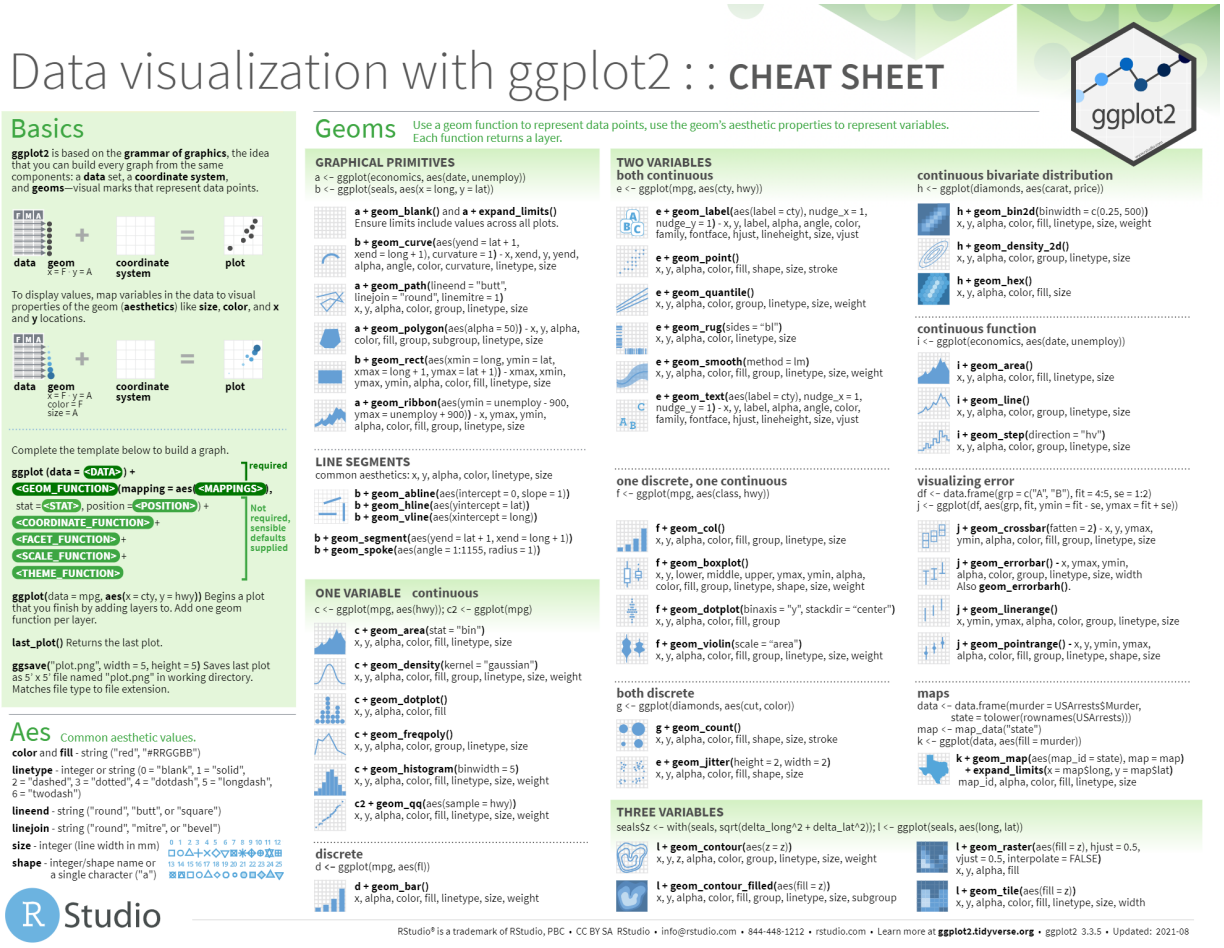


Figure 4.2.1 *ggplot2 Cheat Sheet*

4.3 Candidate Solution #3 - JavaScript

The final solution that will be looked into is JavaScript. Since this project will include the use of a website, displaying the data as a graph will become important and will thus make JavaScript beneficial to use. JavaScript allows developers to create content that can be interactive or more enjoyable by using libraries that help with the user interface or to help the user find results faster with autocomplete features. All of these libraries are made possible due to the 25 years of JavaScript being released and the creator Brendan Eich. Since JavaScript is a language that the current developers have moderate knowledge in, creating graphs or other aspects of the website will take some time since research will be needed. Medium familiarity will cause some time to be put into research and some time developing code but when it comes to more complex functions more time will be needed to be proficient in JavaScript. Therefore, in the familiarity criteria, Javascript will receive a score of 6 out of 10.

The documentation that was viewed was very confusing at first glance and will require more time to look through. Compared to the other languages, this is substandard in the sense that

the clients and developers will struggle when trying to modify the code and since there are easier options for documentation and familiarity, JavaScript will be more difficult to implement into projects. In **Figure 4.3.1,** no comments are available with the code which will make it difficult to interpret and in the website there are paragraphs about the code where single sentences will be sufficient and less complicated. Therefore, JavaScript will receive a score of 4 out of 10.

```
chart = Choropleth(unemployment, {
  id: d => d.id,
  value: d => d.rate,
  scale: d3.scaleQuantize,
  domain: [1, 10],
  range: d3.schemeBlues[9],
  title: (f, d) => `${f.properties.name}, ${statemap.get(f.id.slice(0, 2)).properties.name}\n${d?.rate}%`,
  features: counties,
  borders: statemesh,
  width: 975,
  height: 610
})
```

Figure 4.3.1 *JavaScript Choropleth Code*

4.4 Conclusion

In **Table 4.4.1**, documentation will be weighted more since documentation will be more important than the other criteria.

Table 4.4.1 *Ratings for languages*

	Familiarity	Documentation	Total
Python	9/10	8/10	17/20
R	4/10	9/10	13/20
JavaScript	6/10	4/10	10/20

After reviewing the criteria that need to be met, Python has been chosen as the main programming language. To come to this conclusion, certain criteria were chosen and will be ranked on a scale of 0 (lowest) through 10 (highest). With the familiarity having a bigger weight and Python being the best in the familiarity section, Python will provide a better understanding and will be more efficient when implementing it into the website. The reason for that is because

Python has similar libraries to R which would then make the documentation criteria less helpful. Not to mention that R will have the same difficulty when trying to implement it into the website. JavaScript had low scores on everything since there is not much documentation or libraries to use for large datasets and if the solution were to use JavaScript the developers and clients will have to rely on the little resource that JavaScript provides.

In the future, the developers will provide a demo that will prove how everything will come together and how successful it will be for this project.

5.0 Challenge Ⅱ - Interacting with a Database

Some of the most important aspects to consider regarding the big data revolution are the storage and handling of the data. Databases and database tools are the technologies that can solve these general problems, yet the offered solutions vary in type and purpose. A database stores data in tabular formats that can then be accessed, manipulated, updated, or deleted through the use of database tools. This project requires the ability to interact with a database hosted here at NAU in an organized manner. The amount of data pertaining to asteroids and their related characteristics to be queried and analyzed is large enough to call on solutions capable of handling this size of data in a simplistic nature. Grouping of database points based on the derived or actual characteristics of the observed asteroids is crucial for proceeding to data visualization, thus it must be well thought out with consideration for future operations. Proper and efficient retrieval of asteroid data out of NAU's database has the potential to make visualization an easier process by providing clean data to work with.

For this project, the tools to be analyzed as candidate solutions include the dplyr package from R's Tidyverse, the Python library sqlite, the Python library SQLAlchemy, and raw SQL within the chosen base language. The spread of potential candidate solutions offer language flexibility and a wide variety of tools to consider during the analysis of each.

Some characteristics the project requires regarding database interactions are adaptability to the NAU database, ease of use/readability, and a sufficient documentation base. It is also important that the candidate solution provides ease of use while making an initial connection to the database so efforts can be more focused on data retrieval and analysis. As mentioned before, this challenge is important to handle well as its outcome is responsible for providing data to the next major operation: data visualization. The organization of the data pulled from the database plays an important role in the data visualization's efficiency. Below is an analysis of the candidate solutions to ease the challenge of interacting with a database. Desired characteristics will be scored on a scale out of 10. Score ranges applied to each metric are described as follows:

- 8 - 10: **Excellent**. Candidate works well with project regarding the given metric
- 4 - 7: **Average**. Candidate works for the project but may be lacking in some features and/or compatibility
- 1 - 3: **Insufficient**. Candidate is unable to work with the project without major modifications to the tool and/or project

5.1 Candidate Solution #1 - dplyr

Dplyr is a core package from the extensive and powerful Tidyverse library under the R language and is supported by R Studio. As a continuation of R's plyr package, dplyr is able to tackle some of the most common data manipulation problems developers run into around databases and data frames. The use of verbs in dplyr make working with data in general an easier task by avoiding the need to learn how to use a more complicated query language. There are 5 verbs (functions) that provide a basis to branch off of: select, filter, arrange, mutate, and summarize. Just these 5 alongside the group_by function provide ease of use during development by avoiding large, unnecessary sets of tools to figure out yet still remain powerful enough to retrieve the data for the project. Speaking of ease of use, many of dplyr's functions are easy to implement and understand since many operations can be done in a single line of code, thus it will receive 9 out of 10 points for the readability metric. **Figure 5.1.1** gives an example that arranges and sorts data points into a tibble, or a simple data frame in R's tidyverse. Just a couple of words worth of code can achieve clean, readable output.

```
starwars %>%
  arrange(desc(mass))
#> # A tibble: 87 x 13
#>   name    height  mass hair_color skin_color  eye_color  birth_year gender
#>   <chr>    <int> <dbl> <chr>      <chr>       <chr>           <dbl> <chr>
#> 1 Jabba …    175 1358. <NA>       green-tan,… orange          600.  herma…
#> 2 Grievo…    216  159. none       brown, whi… green, ye…       NA   male
#> 3 IG-88      200  140. none       metal       red              15.0 none
#> 4 Darth …    202  136. none       white       yellow           41.9 male
#> 5 Tarfful    234  136. brown      brown       blue             NA   male
#> # ... with 82 more rows, and 5 more variables: homeworld <chr>,
#> #   species <chr>, films <list>, vehicles <list>, starships <list>
```

Figure 5.1.1 *R Documentation Usage Example - dplyr*

Dplyr's ability to adapt its functionality to different backend databases like MySQL, sqlite, and PostgreSQL is important to consider as the project requires more information about the database workings at NAU to figure out which tool will be a best fit for both the clients and the development process while being able to work with NAU's database. Due to this wide variety

of connectivity, dplyr will receive 9/10 points when considering adaptability. Full points were not quite awarded here because while the candidate does have wide adaptability to some of the most common backends, it still does have a closed set of backends it can work with. Dplyr utilizes functionality to interact with databases through dbplyr, the central database tool in the package. Connections with databases are made as if the data was loaded into a data frame locally and can be made with a single line of dbplyr code, thus this tool gets close to full points (9/10) for the database initial connection metric. This option also provides flexibility in use and performance as loading in data frames is typically a quicker operation than querying a database and provides output that may be more suitable for certain situations. It is worth noting that dplyr only generates SELECT SQL statements, not that the project calls for database manipulation but rather an observation that more focus exists on the retrieval of data which suits the project better. However, if the queries to NAU's database are simplistic, the dplyr package may prove mostly unuseful. There is no point in having such a large toolkit and not utilizing the vast majority of what it can do. The documentation for dplyr is extensive and provides many examples that would be helpful while conducting database operations. Therefore, it will receive 7/10 points for the documentation metric, which is on the higher end of average. This is due to the fact that dplyr is a newer tool and hasn't had the time to have it's documentation expand to the reaches of more time-tested tools.

5.2 Candidate Solution #2 - sqlite

Sqlite is a C library that uses a disk-based database that does not require a separate server process and provides database access through a nonstandard variant of SQL. Since sqlite does not abstract much of the base SQL code, it lacks readability as SQL queries can get quite complicated. Due to this, sqlite will score a 6/10 in readability, or average. The readability of SQL is not all that great, so sqlite and other candidates that use it will receive lower readability scores. A connection can be made to a database with only one line of code, so sqlite will score a 9/10 in this category. Sqlite is the most used database engine in the world and is built into many of life's everyday technologies (SQLite.org 2021). A major plus side to sqlite is its flexibility in languages. Sqlite's functionality is adaptable to both R and Python, which is an important consideration. While sqlite comes built into Python, there are available adapters for R. When considering adaptability, sqlite fails to meet the mark. Sqlite.org states "a good rule of thumb is to avoid using SQLite in situations where the same database will be accessed directly and simultaneously from many computers over a network". Therefore, sqlite will receive a 3/10 for the adaptability metric. However, the combined use of sqlite and Python provides for a nicer experience as the documentation is stronger than that of R's solution, so it will receive an 8/10 given the underlying language, SQL, has been around for so long.

One of the main points that differentiates sqlite from other database management systems is that it operates in a standalone fashion rather than operating as client-server. This plays a big role in why sqlite is so efficient and fast, yet it does not meet the project's requirements.

Avoiding a separate server function and operating on the end system means that data can be accessed very quickly yet this will not prove effective on a deployed website used by many. Sqlite almost meets the mark as it satisfies the requirements for querying the database, just not in the fashion the project requires.

5.3 Candidate Solution #3 - SQLAlchemy

SQLAlchemy is "the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL" (sqlalchemy.org). The main idea behind SQLAlchemy is to compose a larger structure made up of different SQL query statements like SELECT, JOIN, and OTHER. SQLAlchemy is best known for providing an object-relational mapper (ORM), a technique that maps database tables into objects. This leads to an abstraction of the underlying SQL code, thus providing for great readability and a 7/10 score for the metric. SQLAlchemy's ORM is an optional component of the toolkit and raw SQL may also be used, providing flexibility and transparency for database operations. Thus, the readability metric is balanced between excellent and average due to the fact that plain SQL can be used instead of the simpler, SQLAlchemy syntax, bumping the score down. **Figure 5.3.1** gives an example of some SQLAlchemy code that makes objects out of database tables.

When connecting to different databases, SQLAlchemy includes different functionality to adapt to different databases like Oracle, MySQL, or PostgreSQL. Again, since this candidate offers just SQL to be used as well, there is the ability to query more databases than just the built in options. Thus for the adaptability metric, SQLAlchemy scores 10/10. Relating to database connections, SQLAlchemy is able to make a connection in only a couple of lines of code, thus it will receive a 9/10 for the metric. Documentation is thorough on SQLAlchemy's website alone, there even exists dialects for each of the different types of databases, so it will receive 9/10 points in the metric.

5.4 Candidate Solution #4 - SQL

SQL is an older, time-tested query language that is used to communicate with databases through statements that retrieve, add, update, and delete data from databases.

Using SQL to retrieve project data from NAU's database is a solution that is raw and low-level. Without any abstraction, database access operations will have to be more thought out yet they can be tailored to meet the project's exact needs if done properly. While SQL is a fantastic query language used throughout computing science, the language syntax is cluttered and oftentimes harder to read when in larger blocks. So, SQL will score a 5/10 for the readability metric. Using SQL is definitely doable, it is just that the readability of the code may slow down the development process as careful thought must be put into the queries. If the queries to be executed for the project are simplistic, SQL can prove beneficial. On the other hand, code for the project can get quite messy and unreadable if the queries end up being demanding. **Figure 5.4.1** is an example of a larger query using SQL where one can observe how the code feels cluttered and hard to follow.

```
SELECT tbla.NAME
        ,tbla.LstName
        ,tblB.PhoneNum
        ,tblC.Address
FROM tblUsrName tblA
INNER JOIN tblUsrPhone tblB ON tblA.usrID = tblB.usrID
LEFT JOIN tblUsrAddr tblC ON tblC.usrID = tblA.usrID
INNER JOIN (
        SELECT workplace
                ,workadr
                ,workrole
        FROM tblOfice
        ) z
INNER JOIN tblB ON z.ID = tblB.OfficeID
WHERE tblC.Stane IN (
                'GA'
                ,'AL'
                ,'NY'
                ,'CA'
                )
```

Figure 5.4.1 *Sample SQL Code Snippet*

The documentation for SQL, however, is fantastic. Almost 50 years of existence means there has been plenty of time for the language to be groomed and experimented on, so it will receive a 9/10 for this metric. Regarding the ease of connecting to a database, not too much analysis can be done. This connection, when considering raw SQL, will be made with a combination of both Python and SQL code, potentially alongside another library. So, SQL will score a 7/10 for this metric given some extra work may have to be done to ensure proper functionality. SQL would be used almost solely for retrieving data from the database and has the ability to do this on almost any type of database. This is because almost all databases made are able to understand SQL code, given that SQL is the main language of databases in the world. So, SQL will receive the full 10 points for the adaptability metric.

5.5 Conclusion

The analysis of the candidate solutions now comes to a close and **table 5.5.1** sums the total scores for each candidate solution. Starting off, all candidate solutions provided a general ease of use when it came to making a connection to the database and if anything, dplyr provides better readability here as well since so little code is needed. Dplyr, the candidate solution geared towards R was first to be analyzed. While dplyr scored the highest in half of the categories, much of the toolkit's wide range of functionality may go unused, thus creating unwanted complexities. The documentation for dplyr is lacking relative to the other candidates given how long each has been around. Yet, because dplyr is part of the R Tidyverse, it's documentation is evolving at a steady rate given its use in data science. Considering simplicity and lacking documentation for database operations, dplyr is not the correct fit for the project. Moving to sqlite, **table 5.5.1**

shows that this library scored lowest in most categories. While it does provide SQL in a simplistic nature, sqlite can be safely disregarded as the main goal of the library differs from the client-server environment of the project. The underlying commands accomplish what is needed for database retrieval but the library was not created to handle many client connections to the same database. SQLAlchemy, like sqlite, accomplishes database retrieval in a simple manner. However, SQLAlchemy is geared more towards the type of use this project calls for, many users utilizing tools that access the same database. SQLAlchemy also provides the ability to use raw SQL statements which can lead to simplistic, tailored solutions to development issues. Lastly, SQL as a query language is considered. The documentation for SQL, and thus sqlite and SQLAlchemy since they build on top of the language, is superior to dplyr due to SQL's age. SQL is a great candidate solution as it is able to interact with the database in a simple fashion given the project's queries are not too demanding. **Table 5.5.1** contains the candidate solutions and the scores they received in the various metric categories.

Table 5.5.1 *Ratings for database access tools*

	Adaptability	Ease of use/readability	Ease of database connection	Documentation	Total
dplyr	9/10	9/10	9/10	7/10	34/40
sqlite	3/10	6/10	9/10	8/10	27/40
SQLAlchemy	10/10	7/10	9/10	9/10	35/40
SQL	10/10	5/10	7/10	9/10	31/40

The descriptions throughout the candidate analysis is well-reflected in **Table 5.4.1**. Since the candidates' adaptability and the ease of use/readability are most crucial to the future development of the project, those metrics carried heavier weights while considering scores. Secondly, the connection to the database is a smaller operation in the bigger picture of interacting with the database, so it received less consideration in the scoring process.

SQLAlchemy is the chosen candidate solution for this challenge because of how it's adaptability and documentation stacked against the other candidates. While the technology is relatively newer than the other solutions, it's documentation and support found online held up well. The ability to use raw SQL statements within this tool made it attractive to use if the queries end up being simplistic. That way, specific needs can be met whenever needed. Keeping things simple while still being able to conduct the needed database operations is something that SQLAlchemy accomplishes well. Another candidate solution that is always a viable option due to its flexibility and commonality in databases is SQL. Should something not work to plan with

SQLAlchemy, the plain SQL query language is a great solution to keep handy throughout the project.

Before January of 2022, initial tests and a demo of the project's database operations will have been conducted using SQLAlchemy. For the time being, a sample database 3 gigabytes in size will have to suffice for conducting tests. The development plan is to either set up a sample database using the provided data on a local machine or connect directly to NAU's server and database. In either scenario, the abilities of SQLAlchemy can be tested.

6.0 Challenge Ⅲ - Using the data in memory

The application will be handling large data sets. A database sample provided by the clients is 3.11GB in size and the live version is much larger, especially as the Legacy Survey of Space and Time telescope comes online. Once the application retrieves the data from the database, it will need to store it in memory, in a data frame. A data frame is a two-dimensional array-like structure, analogous to a matrix, with rows representing individual data points, and columns representing the various measurements. While this application will look for strategies to limit the in-memory size of the data, due to the size of the data sets presented, this project will need to be aware of the performance of the libraries, especially as the size of the data set increases.

There exist several data frame libraries that handle the data in memory. In this section, the paper will analyse the pandas and Vaex libraries for Python, and the data.frame library for R. This analysis also considered the Tables function in the Datascience Python package, the data.table program, as well as the Tibbles library both in R. The Tables function is marketed as a package to teach data science rather than a production library, and therefore less capable than pandas. The data.table library offers much less performance than the data.frame, and does not appear to offer any benefit over the data.frame library. Finally, the tibble library appears to act as an abstraction layer for the data.frame library, offering some syntactic differences. Since tibble does not appear to offer anything better than data.frame aside from syntactic difference, it will not be considered at this time.

For the testing, this paper will look at how popular and well documented the libraries are, looking at how many books are available, seeing how often the library is used in tutorials, and get a sense of how often the library appears when searching for it, such as questions answered for common issues, and looking at documentation provided freely on the libraries website. The scoring for documentation will be as follows:

- 10 - 6: **Good to great documentation**. Either on the website for the library, or externally. Additionally, if the library is used a lot in tutorials and examples, a less well documented library can score much higher.

- 5 - 0: **Poor documentation**. The documentation is fairly basic or limited.

Additionally, the solution presented will need a lot of speed, as the performance of the application is a critical aspect of the usability of the website. This paper will test the performance by loading two files in csv format, both having 11 columns, with the smaller having 330k rows and the larger 660k rows, and seeing how long it takes for the system to display the first and last five rows of the data, over five trials per file. Scoring for performance will be as follows

- 15 - 10: **Good to great performance**. Less than 0.600 seconds to load the 660k row file
- 9 - 5: **Potentially acceptable performance**. Less than 1 second to load the 660k row file
- 4 - 0: **Unacceptable performance**. Greater than 1 second to load the 660k row file.

6.1 Candidate Solution #1 - pandas

For the Python based solutions, the most popular and well documented data frame library is pandas. To give a sense of the popularity of the pandas library, the GitHub repository for pandas currently has 31.3k stars. There is a seemingly uncountable amount of documentation, examples, books, StackOverflow questions, and tutorials that use pandas. In fact, when researching, almost all tutorials for other libraries that rely on a data frame in Python use pandas as the default, giving pandas full marks for documentation. However, the pandas documentation page named “Scaling to large datasets” states “pandas provides data structures for in memory analytics, which makes using pandas to analyze datasets that are larger than memory datasets somewhat tricky. Even datasets that are a sizable fraction of memory become unwieldy, as some pandas operations need to make intermediate copies.” However, pandas provides several tools and strategies to deal with overly large datasets, such as chunking. When testing pandas ability to load two separate data frames from csv files, one being 11 columns and 330k rows, and the other being the same file, but the rows duplicated. As shown in **Table 6.1.1**, pandas took on average 0.280 seconds over 5 trials to load the 330k row file, and took on average 0.541 seconds over 5 trials to load the 660k row file. Due to the high speed, pandas will get full points for performance.

Table 6.1.1 *Performance of pandas*

Number of rows in CSV file	Time averaged over 5 trials
330,000	0.280 seconds
660,000	0.541 seconds

6.2 Candidate Solution #2 - Vaex

An alternative Python based solution is to use the Vaex library. Vaex is marketed as a high-performance alternative to pandas. The Vaex website states that it supports more than 109 rows per second and is memory efficient due to “no memory copies when doing filtering/selections/subsets”. However, Vaex is not nearly as well documented. There were no books utilizing Vaex found, and only a small subset of tutorials for Vaex, therefore Vaex can only get half points, 5 out of 10, in documentation. When checking the performance of Vaex, Vaex did not appear to provide a substantive performance boost over pandas. As shown in **Table 6.2.1**, Vaex took an average of 0.292 seconds to load the 11 column, 330k row file into a data frame, and 0.556 seconds to load the 11 column and 660k row file over 5 trials each. Due to the slightly slower speeds than the other options, Vaex will lose one point in performance.

Table 6.2.1 *Performance of Vaex*

Number of rows in CSV file	Time averaged over 5 trials
330,000	0.292 seconds
660,000	0.556 seconds

6.3 Candidate Solution #3 - data.frame

For the R based solution, the built in R data frame library, using data.frame, is the best and default option. The data.frame does have a limit of 231 – 1 items, and there exists reports of performance penalties for the data.frame library well before reaching the size limit. However, R does have the data.table library, which is described in the CRAN documentation as “an R package that provides an enhanced version of data.frames”. The data.table documentation states that it provides “low-level parallelism” and “fast and scalable aggregations; e.g. 100GB in RAM”, two major considerations for the website. Additionally, the data.frame syntax is nearly identical to the data.table syntax, which increases the available documentation for data.frame, as the documentation for data.table is likely to be applicable. Searching for “data.frame R” provided a wealth of discussion, and searching for books for data.frame suggests a healthy ecosystem of books available, both free and paid. Therefore, data.frame will get full points for documentation. When testing the data.frame library, as shown in **Table 6.3.1,** data.frame was able to load an 11 column, 330k row csv into a data frame, and print out the head and tail in 0.214 seconds on average over 5 trials. When loading the 11 column, 660k row csv file into a data frame, data.frame averaged 0.427 seconds over 5 trials. Due to data.frame having the quickest load times, it will get full points for performance.

Table 6.3.1 *Performance of data.frame*

Number of rows in CSV file	Time averaged over 5 trials
330,000	0.214 seconds
660,000	0.427 seconds

6.4 Conclusion

Table 6.4.1 *Ratings for data frame libraries*

	Performance	Documentation	Total
pandas	15/15	10/10	25/25
Vaex	14/15	5/10	19/25
data.frame	15/15	10/10	25/25

As shown in **Table 6.4.1**, the wealth of documentation and the great performance for both data.frame and pandas suggest both appear to be very good solutions as a data frame library. Therefore, if this project is using Python pandas will be the choice and if the project is using R data.frame is the choice.

Prior to the end of the semester, the developers plan on building an application that utilizes pandas with either the live database or the slice of the database that was provided to us. This way the team can determine the performance of pandas, and see if the performance correctly scales.

7.0 Challenge IV - Plotting the data

Not only is the application responsible for handling a large data set in memory, but it will need to pass that data to a plotting library to render the plots in order to generate useful graphs for these large datasets. When working with a large dataset in an interactive GUI, the application will need to be responsive and quick, therefore the performance of the library is essential. For

this paper, the viability of ggplot2, Plotly, matplotlib, and Seaborn will be analysed. The Bokeh library was also considered, but due to the much more limited documentation, and the fact that Bokeh does not appear to offer much more performance means that ultimately the Bokeh library will not be considered.

For testing the libraries, this paper will first consider the developers ability to learn the libraries. The availability of documentation, examples, and tutorials on the website of each library will be weighed. Additionally, availability of external documentation, such as availability of books and the availability of forums such as StackOverflow that answer questions on the libraries and tutorials should be considered. This should provide a good idea of how easily it will be to get help on issues relating to use of the library. To score the documentation, this paper will use the following scale:

- 10 - 8: **Good to great documentation**. It is easy to understand and find how to use the graphing library. Additionally, external resources will be considered, and the more easily it is to find, the more likely a library will get a higher score.
- 7 - 5: **Acceptable documentation**. Documentation can be found, and it might be easy to access, but quality may be much more difficult to read. External documentation may be poor or non-existent.
- 4 - 0: **Poor documentation**. The documentation may be poorly managed, hard to read, or not exist. External documentation might be limited to unanswered questions or be non-existent.

A scatter plot with 100,000 or more data points will generate a large mass of points that will make it hard for a researcher to get a sense of the shape of the data, therefore plotting libraries that provide support for a variety of graphs suited for large data sets, such as heat maps or 2d histograms. In addition to graphing large datasets, the website will need to draw graphs and figures for individual asteroids. Therefore, having a graphing library that can draw a wide variety of interesting graphs is important. When scoring for graph selection:

- 5: **Great selection**. A wide variety of graphs, including a selection of graphs that are well suited to large datasets
- 4 - 2: **Average selection**. Graph selection might be limited, or the library may not have an acceptable option for large data sets.
- 1 - 0: **Poor selection**. Graph selection is very limited and the library does not have graphs for large datasets.

Finally, this paper will test the performance for each challenger. For the analysis, the libraries will draw a 2d histogram or other similar graph, utilizing a data frame that contains 330k rows of data, and time to draw will be the metric. The Python based libraries will use pandas as the data frame inside a Jupyter notebook and timed by taking the difference between calls to time.time(). For ggplot2, an R script that drew in the data into a data.frame, and timed it by taking the difference between calls to proc.time(). When scoring performance, the following scale will be used:

- 10 - 8: **Good to great performance**. The graph will either print out in less than 0.75 seconds, or if an interactive graph is produced, less than 2 seconds.
- 7 - 5: **Acceptable performance**. The graph will either print out in less than 1.5 seconds, or if an interactive graph is produced, less than 3.5 seconds.
- 4 - 0: **Poor performance**. The graph will either print out in greater than 1.5 seconds, or if an interactive graph is produced, greater than 3.5 seconds

7.1 Candidate Solution #1 - ggplot2

Within R and data science in general, one of the most well-known and supported choices is ggplot2. A search of Amazon books for ggplot2 gives 96 books, which suggests a wide variety of books available to learn the ggplot2 API. Additionally, the ggplot2 Tidyverse site provides links to vetted tutorials and books for learning ggplot2, such as a free online version of Hadley Wickham's *ggplot2: Elegant Graphics for Data Analysis* book. ggplot2's Tidyverse documentation is impressive, with a list of each graph type that is a link to further documentation for that specific function and a two page "cheat sheet" for ggplot2. Ggplot2 provides a wide variety of graphs, with multiple graphs for one, two, and three variable graphs, and further options to modify the graph. The ggplot2 GitHub states "You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details." ggplot2 provides support for building both square and hexagonal heatmaps with the geom_bin_2d() and geom_hex() functions. This healthy selection of graphs means ggplot2 gets full marks for graph selection. When looking at how ggplot2 handles large data sets, most discussion appears to be focused on making useful graphs with the data, instead of how to make ggplot2 handle the large dataset. While profiling ggplot2, using geom_bin_2d() on two columns of 330k rows took 0.838 seconds to generate the graph shown in **Figure 7.1.1**, giving it 8 out of 10 points for performance. The generated geom_bin_2d() graph does a much better job of differentiating empty bins, showing the underlying graph, and bins with one or two items, showing a dark box. Overall, ggplot2 has a lot of very good documentation and tutorials to learn, and a great set of graphing options.

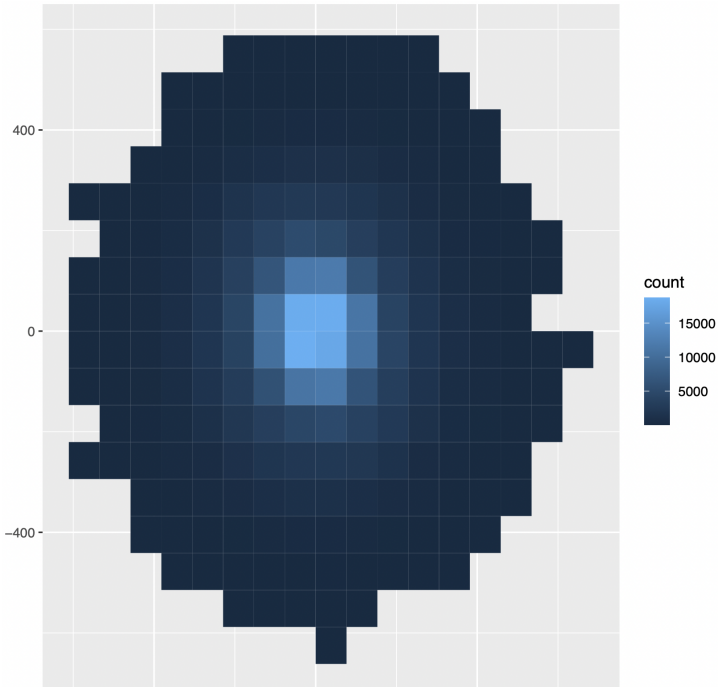


***Figure** 7.1.1 Plot generated by ggplot2 using the sample dataset.*

7.2 Candidate Solution #2 - Plotly

Another plotting library to consider is the Plotly library. Plotly is supported in both Python and R and can act as both a plotting library and an interactive layer on top of another plotting library, such as ggplot2. For this section, this paper will only consider Plotly's ability to act as a plotting library. A search of Amazon for books related to Plotly provides 25 results, and a search of Google for Plotly provides around 2,560,000 results. This indicates that Plotly is not as popular as ggplot2, and tutorials and examples may be less available than ggplot2 but should not be seen as hard to find. However, the Plotly documentation on plotly.com is great, providing examples of how to use Plotly and its various functions. Additionally, as Plotly is a potential solution for making the data visualization interactive, having Plotly be the graphing library means that the application will only need to use one graphing API. Plotly supports square based heatmaps with density_heatmap() function, among others, giving Plotly 5 out 5 for graph selection. Finally, Plotly supports graphing with large datasets, including an interactive example with 1 million data points. When doing a basic heatmap of the 330k row dataset using pandas, Plotly took 2.745 seconds to draw the heatmap shown in **Figure 7.2.1**. While this was slower than the other libraries, it drew an interactive plot, which should explain much of the difference, therefore Plotly will get 8 out of 10 points, to make up for not needing to call a secondary library to make interactive. Plotly is a solid option, allowing for good documentation, and a large selection of graphs that are built as interactive, with options to support large data sets.

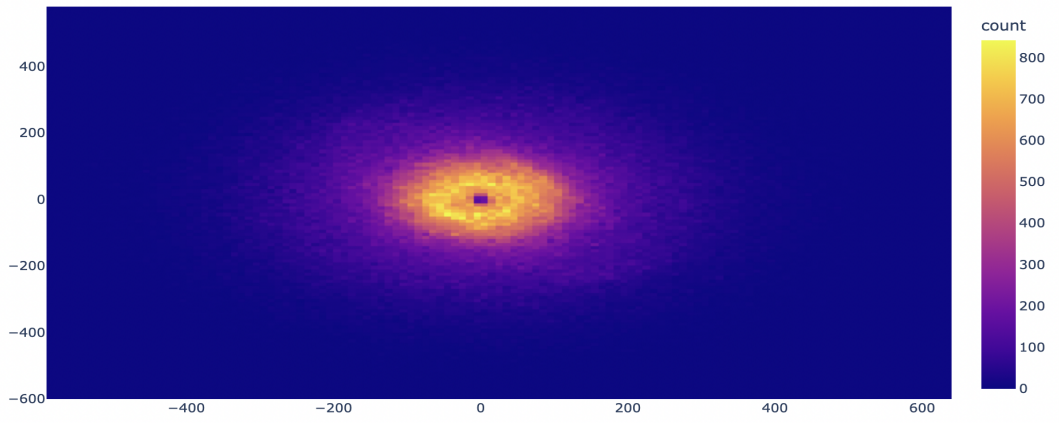


***Figure** 7.2.1 Plot generated by Plotly using the sample data*

7.3 Candidate Solution #3 - matplotlib

When using Python, matplotlib is a very popular solution. A search of Amazon books for matplotlib provides 209 potential books to learn from, and a search of Google for matplotlib provides around 10,800,000 results, nearly as much as ggplot2, indicating a large selection of tutorials to learn from. The documentation for matplotlib is okay, providing tutorials and examples to learn from and an API documentation, but it was noted that it was not as well polished as the other options, therefore matplotlib can only get 7 out of 10 points for documentation. Matplotlib provides a large selection of potential plots, including line plots, histograms, and 3D graphs, and heatmaps, which means matplotlib gets all 5 points for graph selection. When profiling matplotlib with a 2d histogram using 330k data points per axis, matplotlib took on average 0.095 seconds to draw **Figure 7.3.1**, a very fast time. Because of the fast time, matplotlib gets full points for performance. Matplotlib is a solid plotting library that is very mature.

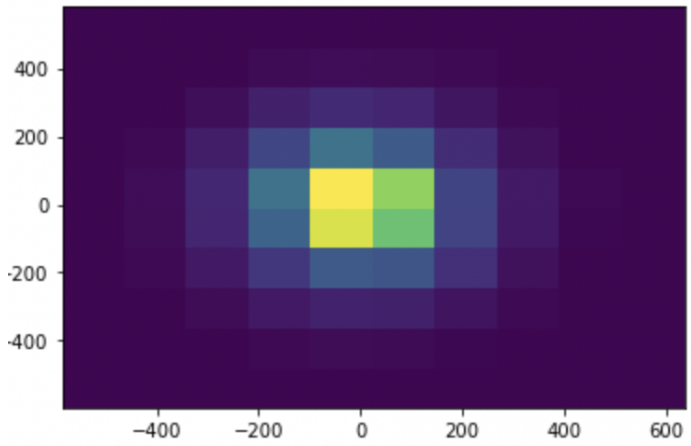


Figure 7.3.1 *Plot generated by matplotlib using the sample data*

7.4 Candidate Solution #4 - Seaborn

Finally, the Seaborn library will be considered. Seaborn appears to have good documentation on its website, but external documentation appears to be much more limited than the other libraries. Therefore Seaborn will get 7 out of 10 points for documentation. Seaborn appears to have a good selection of graphs, including plots suited for large data sets, which means Seaborn gets 5 out 5 points for graph selection. When profiling Seaborn using 330k data

points per axis, Seaborn built the displot() shown in **Figure 7.4.1** in 0.210 seconds, providing 9 out of 10 points.

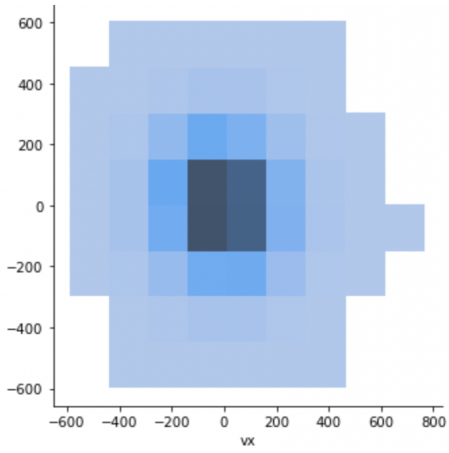


Figure 7.4.1 *Plot generated by Seaborn using the sample data.*

7.5 Conclusion

Table 7.5.1 *Ratings for all plotting libraries*

	Documentation	Graph selection	Performance
ggplot2	10/10	5/5	7/10
Plotly	9/10	5/5	7/10
matplotlib	7/10	5/5	10/10
Seaborn	7/10	5/5	9/10

Overall, all four options provide a solid selection, with the difference between the best and the worst being two points. Therefore, all the libraries would be acceptable as a solution. However, ggplot2 with its great documentation and great looking graphs is the choice if the project uses R. But since this project will use Python, the matplotlib library is the choice, as the great performance makes it a great choice for plotting multiple terabytes of data.

Prior to the end of the semester, the developers plan on building a test application, where they will be able to test the plotting libraries with a much larger data set. They plan on utilizing matplotlib first to determine if it is the correct plotting library.

8.0 Challenge V - Controlling Dashboard Style

Providing a simple, yet elegant way to display astronomical data is a major issue for astronomers. The terabytes of data that telescopes capture and stream are crucial in finding patterns, outliers, and other information about our Solar System. The way to display all this data is not currently up to technological standards. Astronomy has fallen way behind in keeping up with major technological advancements, most notably in data exploration and visualization. Taking those terabytes of data and transforming it into a clear and concise visual for all astronomers to be able to look at and formulate conclusions using the population of objects in the database is the main motivation for this capstone project. A dashboard that can display these visualizations without looking cluttered and to have the visualizations easily identifiable is a challenge for this solution.

The main desired characteristics for what the ideal solution would include is simplicity of the dashboard, ease of maintenance, documentation, ease of login/account creation, and maturity of the technology. Simplicity of the dashboard is a very important characteristic of the solution . because the data will be visualized in a creative and interactive way, but displaying and organizing those visualizations is also something that is needed before the dashboard gets too cluttered and hard to be navigated by the users of this solution. Ease of maintenance is an important desired characteristic as well because there will be bugs in the future that are nearly unavoidable, but being able to identify the bugs in the code and to fix them within a reasonable amount of time is something that is needed.

Maintenance is one area of software engineering that many people can struggle with then using a new product. Documentation is also another desired characteristic that is important because the developers want to be able to learn how to read and write code easily for this solution if a programming language or framework that is unfamiliar to the developers is chosen. Consulting the documentation and determining how to create a way to organize the data in a simple dashboard in a reasonable timeline while also being able to debug with ease and prevent many of the errors and bugs that come along with writing code is significant for this solution. Being able to create an account within the framework and have easy login / logout capabilities is another one of the desired characteristics that were discussed with the client(s). This is an important characteristic because it would allow any user to create an account and save certain asteroids that interest them. This would allow for the users to track important data about the individual asteroids that the users find interesting and prevent the unnecessary challenge of manually finding each individual asteroid each time they visit the website.

The last desired characteristic that was discussed would be the maturity of the technology that the solution will have. This is important to the project because there would be a less chance of issues arising in the product, therefore improving the user experience. This will allow the

client(s) to maintain the website more easily than if the technology that is used was not as mature as it would not be as tested and proven to be a reliable framework.

The scoring for each desired characteristic is as follows

- 7-10**: Above Average**. Characteristics scored this way will work great for the desired solution
- 4-6: **Average**. Characteristics scored this way will work just fine, but potentially could be an issue for the desired solution
- 0-3: **Below Average**. Characteristics scored this way will not work for the desired solution

8.1 Candidate Solution #1 - Dash

The first candidate is Dash, a dashboard framework that allows for easy styling of the webpage to visualize data analytics that are processed in Python or R. Dash is written on plotly.js and React.js and uses patterns to build a full-stack web app for data visualization and this would be perfect for the project as the GUI for the data exploration and visualization of the astronomical data gathered and sent to the database here at NAU's Monsoon. It is an open-source library that was released under the MIT license. Dash was first publicly released in 2017 and is one of the most used front-end frameworks for providing a binding for data analysis code and the user interface. Dash is a unique framework in the sense that it abstracts all the technologies used to build the app which allows it to be a simple solution to the user's needs, that being a simple GUI to visualize and explore large amounts of data.

Upon evaluating Dash based on the criteria that was set for the solution, it appears to be a viable solution for the project. The documentation is extensive to where the project's developers would be able to confidently learn the tools needed to create a simple dashboard with an interactive GUI with all the specifications that the client(s) have set for us to achieve, example in **Figure 8.1.1** Therefore the documentation category gets an 8/10 and gives the developers confidence in using Dash. In hand with the extensive documentation is the maturity of Dash. Since it is an open-source program, anyone can look for bugs and fix them as needed, this results in a score of 7/10 just as a baseline for the maturity of the tech. Creating a login feature would be simple under Dash as well, as it is very customizable and has a feature for the authorization of a user, this gets it a score of 6/10 just because the login feature was not as detailed as the Shiny login feature. Beyond that is the issue of maintenance, since Dash is extremely well documented, any user would be able to consult the documentation when maintaining the web app to preserve the data, interactivity, etc. The ease of maintenance score is a 10/10 when considering the extensive documentation and the clients familiarity with python. Referring to **Figure 8.1.1**, Dash can be used to create dashboards that are simple and easy for locating graphs and other visualizd data. The simplicity allows the dashboard to be explored without the user struggling to locate any of the data and it will be very helpful for the solution. Simplicity is scored at 9/10 using **Figure 8.1.1** as a baseline of a dashboard. Using Python means that the readability is way higher, and the

language is easier to learn regarding tools that may need to be used to develop the solution and make it viable for the users. The other upside is that it has been recommended by not only the client(s), but those that have worked on the project before as well.

Figure 8.1.1 *This graph above is an example of a Dash dashboard in Python. Simplicity is something that the client(s) have reiterated and Dash has the tools for a simple and useable dashboard*

8.2 Candidate Solution #2 - Shiny

The next candidate that has been considered is Shiny from RStudio. Shiny is a data visualization framework built using R that allows web apps to be hosted and interactive with simple dashboards to visualize the data. CSS themes, HTML Widgets, and JavaScript Actions. Shiny is typically used to take data science applications and provide an interactive way for users to view the data. Shiny was introduced in 2012 and has been at the forefront of interactive dashboard data visualization. Shiny was recommended by the clients to look at and determine if it would be a feasible solution to the problem that this solution will solve. R has been proven to be a reliable language for data science and the developers are intrigued to experiment with it to conclude which programming language would be best for the solution to the problem of data exploration and visualization.

Shiny has the potential to be the final solution that this product will use. **Figure 8.2.1** shows just how simple and elegant the dashboards that shiny can create, can be. The score for this characteristic is a 9/10. The organization of the data and the titles given to the visuals for users to refer to and understand what that specific visual is displaying is just what the solution is looking to add and implement. The documentation is very well written and provided for any user. Learning R and using shiny to create a simple interactive web app GUI for this project is helped by strong documentation that allows more time to work on a solution, rather than spent manually

testing out different Shiny tools. This characteristic will be scored a 9/10 for the additional support by the data science community. Building a login feature is also another characteristic that can be developed in Shiny using the shinyauthr R package. This is important for the users to be able to save and track the data that interests them automatically without manual observation. This results in a 7/10 score, this is the highest login score given to a candidate. Shiny has been around the data science community since 2012 and the data visualization that Shiny allows would be very useful. Ease of Maintenance gets a score of 9/10 because maintenance of Shiny R code would be an easy task given that the documentation is strong, and the community is large and helpful for those who struggle with an error or a question, from what has been gathered. Maturity of the tech gets a 8/10 because of the strong documentation and ease of maintenance of R code.

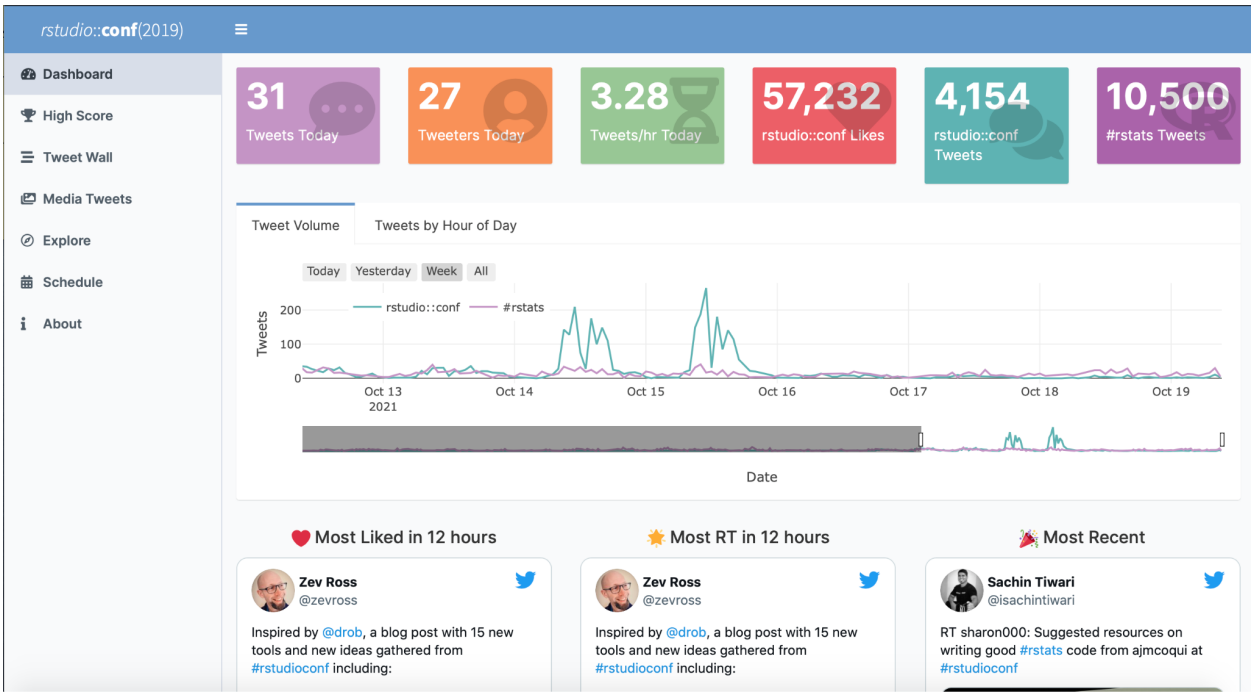


Figure 8.2.1 *This dashboard is an example of how Shiny can create simple, interactive dashboards. The simplicity of the dashboard is very critical for the users and this example showcases the layout and how easy it is to be followed and explored.*

8.3 Candidate Solution #3 - Streamlit

The final candidate that satisfies the project requirements is Streamlit, a Python library. Streamlit is used by hundreds of international corporations to visualize data. Streamlit has a large repository of “widgets” that can be used in conjunction to speed up the process of development. It also allows for full development of a web application without the use of html/javascript/CSS which can help streamline the final product into a single coding language. The product is open source which means it has a large community that keeps the library well documented. Streamlit

provides caching abilities which would help with preloading much of the large datasets the team will need to access and have available to the client. The login creation is a widget that can be added to the library, or can be made from scratch.

Streamlit has the potential to be a great choice with the metrics that are being considered. It can create simple and interactive dashboards to hold and display the data visualizations for this solution and is Python-based, so if Python is chosen as the language solution, then it is a viable candidate for creating the dashboard required. As seen in **Figure 8.3.1**, the dashboard example is simple and gets straight to the point of displaying the share price for top 5 gainers and losers. This simplicity is what the client(s) have advocated for, as a cluttered dashboard would be not easy to navigate and conduct research. Therefore it gets a score of 8/10, losing out on a point because Dash is a better candidate at the moment. Streamlit has a solid community behind it that makes documentation and future development easier, resulting in a 8/10 for documentation. The login creation abilities of the library are easy to use and can be fast-tracked with the “widgets" described above, this gets the login feature a 7/10 for its ease due to the “widgets”. Having everything in Python could be nice, but with the developer’s experience in front-end development, it could prove to be a learning curve that would take additional time. The score of 6/10 for ease of maintenance is because of the potential learning curve of streamlit. Since streamlit is a python-based platform, it gets a score of 7/10 for maturity of the tech just like Dash.

Figure 8.3.1 *Example Streamlit dashboard that shows the simplicity and functionality of what Streamlit can provide*

8.4 Conclusion

Both approaches to the solution provide the same features. The main tiebreaker is whether the product will work best with using Python or R as the main language to develop the

project solution. Dash is a framework that would be a better choice to use because of the recommendation by the clients to use Python, but Shiny is intriguing and very possible to be chosen due to the reputation of R for being a clear choice of tool for data science professionals.

Table 8.4.1 *Ratings for the dashboard frameworks*

	Simplicity	Ease of Maintenance	Login creation	Documentation	Maturity
Dash	9/10	10/10	6/10	8/10	7/10
Shiny	9/10	8/10	7/10	9/10	7/10
Streamlit	8/10	6/10	7/10	8/10	7/10

As seen in **Table 8.4.1**, both Dash and Shiny tie in overall capability to succeed and work for what was envisioned for the solution. Dash came out with a higher score on Ease of Maintenance because of the developer's and client's familiarity with Python. It was above average in every other category just for being a Python framework, but is average in login creation. Shiny edges Dash out in Login creation and Documentation due to the data science community contributing to the documentation heavily. Shiny loses points in Ease of Maintenance due to the learning curve the clients and developers will have to make if this framework is chosen to be the choice for developing the solution. Streamlit being a Python framework also has the simplicity of Dash. Dashboard simplicity is critical to the client(s) needs with interacting and navigating the dashboard.

Overall, Dash will be the better choice for developing the solution due to the metrics discussed above and how it compares to Shiny and streamlit. Shiny is still a great framework and could be of use in the future, but the recommendation by the client(s) and familiarity with Python is something that should be unchanged. Further testing would include the creation of an actual login feature and how well data can be saved and tracked within each separate account. The creation of simple dashboards is also another way that Dash can be demonstrated as the correct tool for this product to organize and visualize the data that will be displayed and show the simplicity of the GUI dashboard to the users, which can be demonstrated in the end of the year demo for the clients).

9.0 Technology Integration

Now that the major challenges have been identified and solved, there exists a plan on how everything will come together. Python is going to serve as the base language for this project and thus the supporting tools will revolve around this choice. The first step in creating this

project will be to interact with the database since it will hold the data that will be needed. The ITS team at NAU is responsible for hosting the database that will be used as a center point in this project. Should assistance be needed for the operations around the database, the clients are able to provide the contact information of ITS faculty. SQLAlchemy will provide the means for data retrieval from the NAU database. Since SQLAlchemy will be used to interact with the database, the pandas library can then be used to utilize the information in memory that was queried from the database. Once the data has been stored locally, dash, the chosen data visualization framework, will create the layout for how the interactive graphs will be portrayed on the website. Once there is a template for displaying the project's data, matplotlib can be used for plotting the data. The use of matplotlib will allow users to select a part of the graph to inspect the smaller data points and find whatever information the user needs about said point. The interactivity of matplotlib provides a solution to an important challenge in the project. Overall, the chosen candidate solutions for the listed technological challenges and design decisions work well with each other to provide a needed solution.

10.0 Conclusion

The project at hand centers around building an application that will help researchers visualize astronomical data from two telescopes that will deliver large amounts of data, upwards of 10TB every day. Currently the data flows from the telescopes to NAU via intermediaries. At NAU, the data is ingested, and further data is derived. All of that data is then stored in the database, where researchers must manually query the data to use. As of right now, the data that is ingested and derived at NAU is sitting in a database. A researcher must take in the data from a database, and manually work with it to create meaningful research. The envisioned solution is to turn these inefficiencies researchers experience while working with data about the universe, and create a platform that allows researchers to quickly and easily create hypotheses and conduct meaningful research. The application will be built to automatically query the database at NAU and produce useful graphics that researchers can use in their research. The application must query the database, store the queried information in memory, generate useful graphs and figures, and provide a useful and attractive user interface for researchers. The solution will need to have high performance in turning that data into useful graphics, as the platform should be pleasant to use, not frustrating or confusing. After analyzing and comparing solutions for the challenges that this project presents, Python will serve as the main language for development, supported by Dash as a web application framework, the pandas and matplotlib libraries for data manipulation and visualization, and SQLAlchemy as the tool for interacting with the database. The simplicity of Python is impressive, and the way all the various supporting tools work together to provide functionality at each step of the project path helps solve the client's problems. The use of Python and supporting tools will also integrate smoothly with their future research plans. Combine that with the great documentation for Python and the related packages, both internally and externally,

the project developers are confident they will be able to quickly begin working toward a useful solution for the clients.