GraphKey

A Microbiology Lab Information Management and Visualization System

Senior Design Project: Final Report
November 2020

Team 29: Brittany McPeek, Benjamin Vogel, Rob Reinhard, Kyle Gansen, Ben Alexander, and Samuel Jungman

Technical Advisor: Thomas Daniels

Client: Karrie Daniels

Team Website: https://sddec20-29.sd.ece.iastate.edu

Executive Summary

Development Standards & Practices Used

- Detailed Use Case diagrams and User Stories
- Simple Design
- Pair Programming
- Heavy documentation in both code and manuals, updated regularly
- Single coding standard (will use PEP-8, a coding style for Python)
- Frequent refactoring
- Simple APIs
- Construct and maintain a user manual for non-technical clients

Summary of Requirements

- System should support importation of large amounts of data in a reasonable amount of time from sources such as CSV and Excel
- System should be able to abstract the data and place them into modifiable graphs
- System should support exporting those graphs to be published into research papers
- System should be able to maintain past data and support modifications and the addition of new data
- System will be written in Python
- System should be easy to understand and use by users with little-to-no background in programming
- System should be maintainable by 1 to 2 IT personnel

Applicable Courses from Iowa State University Curriculum

- COMS 309
- COMS 227
- COMS 228
- COMS 363

New Skills/Knowledge acquired that was not taught in courses

- Graphing and graph visualization
- Python and Python Libraries
- Data management

Table of Contents

Final Report

User Guide

Copy of Design Document From Last Semester

Introduction

Acknowledgement

Our group would like to acknowledge and thank Thomas Daniels for his guidance, support, and technical advice throughout this project. Our group would also like to acknowledge and thank Karrie Daniels for providing us information and requirements for this project, as well as acting as a sample user of our end product. We appreciate the time and commitment these two individuals have donated towards this project

Problem and Project Statement

Many scientists and researchers dedicate large amounts of time towards organizing, maintaining, and visualizing the data they collect. The purpose of this project is to find a solution to this problem. The solution should be able to automate the process of organizing, maintaining, and visualizing data. It is important that scientists and researchers have more time to collect and analyze their data, especially in time-sensitive experiments; thus resolving the issue of organizing, maintaining, and visualizing their data will be beneficial to scientists and researchers.

Our group proposes creating an application named GraphKey that allows the user to import pre-existing data from Excel and visualizes the data. The application will allow users to select data elements and a type of graph/statistical analysis and visualize the resulting information in the form of a graph or another type of visual. The graphing utilities will allow the user to customize the appearance of the graphs and will meet scientific publication standards. Additionally, the application will allow the data and visuals to be exported and shared with another person. Our hope is to create an easy-to-use application that does not require too much maintenance and allows scientists and researchers an easier method to organize, maintain, and visualize their data.

On a more personal level, our client has often mentioned that she spends large amounts of time going through and formatting data so that her current software can process the data without issues. On top of that, she manually creates graphs one at a time to ensure no problems arose from the generation of these graphs. Her primary goal is to lessen the amount of time it takes her to analyze the data and instead focus that time on research.

End Goal

Our end goal for this project is to create a free, easy to maintain, and powerful tool that allows researchers to import data from sheets such as Excel and create detailed graphs with relative ease. This application will be self-contained, and allow for the user to run and operate the application with little-to-no knowledge of the underlying code and mechanics.

Over the last year, this goal has changed very little, as our client has been adamant about the core problems this application solves. On top of that, in order to keep the code maintainable, complex additions to the project were limited. This allowed us to focus our undivided attention towards implementing the core issues that this application addresses with quality.

Intended Users and Uses

GraphKey is intended to be used by scientists and researchers for organizing and visualizing large amounts of data efficiently and effectively. The visualizations created by the product should meet scientific publication standards so that these visualizations can be used in published scientific reports and papers. Non-technical users should be able to use the end product with ease. This places high importance on the intuition and cleanliness of the User Interface of our software. Furthermore, this software will not have a maintainer or IT team after it is delivered, so the project will need to be clean and bug free to match the lack of maintenance staff.

Operational Environment

GraphKey is purely software based. Thus, GraphKey's environment is simply any computer that has Python installed. Our end users will most likely be in a climate-controlled, indoor location, which means that GraphKey will be able to run on our user's computers without any problems.

Assumptions and Limitations

Assumptions:

- The maximum number of users per instance of the product will be one
- The solution will not be distributed outside of Iowa State
- Python and the Python Interpreter will be the primary development tool used
- The end user will require an instruction set about the end product

Limitations:

- The end product shall be able to be maintained by 1-2 IT workers on a minimal time basis
- The end product shall be free (less than the cost of the client's current solution, adjusted given the final product does not use any proprietary technology)
- The end product shall be easy to run and navigate with little-to-no programming experience
- The end product shall be based on, and released for, the ISU research department

Related Products and Literature

The previous work we are basing GraphKey off of is the solution our client is currently using, GraphPad. This technology is designed specifically to take data and organize it similarly to a spreadsheet, and then provide graphing utilities to help visualize the data. GraphPad itself does not operate data entry as a spreadsheet, but a more specialized version where they offer special data tables that can be catered to how the client wishes to organize the data.

One of the current major downsides to this technology is the price. GraphPad can be extremely expensive on a yearly subscription, especially when more than one person needs to have a license for it[1]. Another downside of GraphPad is the lack of options for a robust suite of graphs. Currently, GraphPad can only create bar graphs. While our client uses bar graphs, she would also like to be able to work with more varied graphs such as scatter plots and box plots. Additionally, GraphPad can only create one graph at time. This can be very time consuming for our client who makes several graphs every day.

GraphKey is designed to address all the major issues our client has with GraphPad. Mainly, GraphKey will be free to use, will have a more robust suite of graphs, and will allow our client the ability to create multiple graphs at once.

Project Design

Our final design for GraphKey is a revised version and implementation of the previous project design we proposed in April 2020. The specifics of those design implementations have been worked out, and the process we took to get to the final design changed over the course of the semester. This section will go over our requirements and design from last semester, as well as the new design implementation that GraphKey leverages.

Requirements

Our client's overarching requirement is to be able to use this software to manage data from microbiology experiments.

The following is a list of functional requirements given by the client:

- Ability to import data from Excel
- Customizable data visualization based on specified data elements
    - Supporting bar graphs, box plots, and scatter plots
    - Supporting statistical analysis such as correlation and p-value computation
    - Multiple graphs can be created at the same time
- Data sets and graphs should be able to be saved to the file system, as well as exported and shared with coworkers
- Supports the creation of projects
    - Collation of multiple graphs from similarly based experiments

Our non-functional requirements include:

- Ability for the system to be maintained by one or two people
- Secure enough so that research data can't be seen by anyone else
- Utilizes Python libraries for data visualization
- Data must be parsed after it is imported
- User Interface should be intuitive and easy-to-use

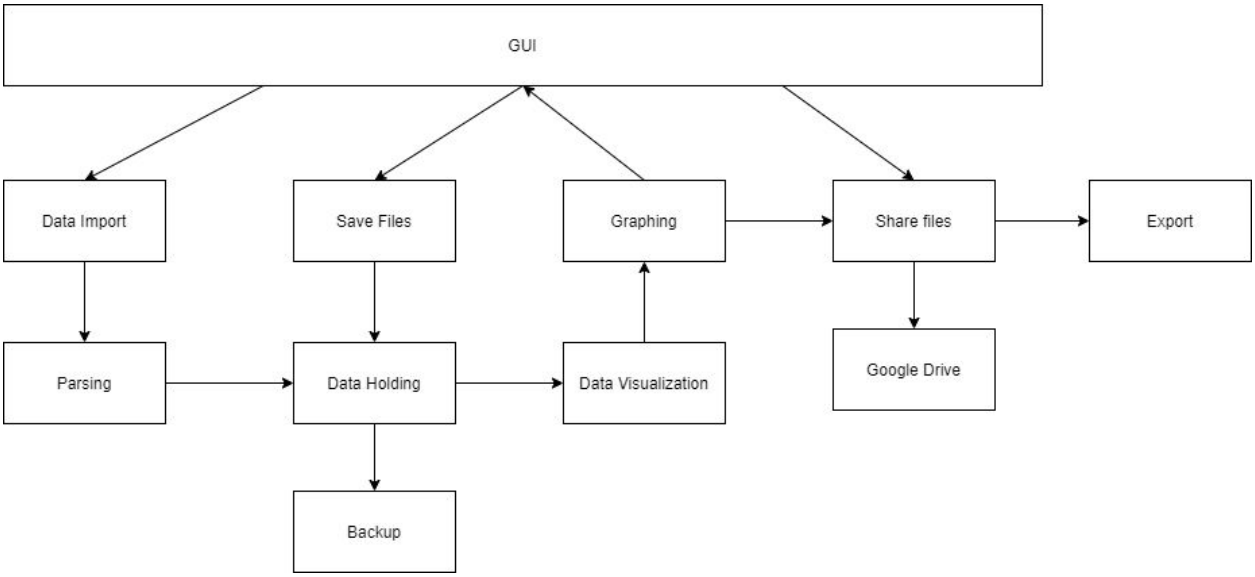
Engineering Constraints:

- A desktop application is required as the end product
    - Must be able to run on Windows, Linux, and Mac OS
    - Will not require more than 200MB of space
- Python was the desired development language
- Plotly is the required Python graphing library

Old Design Approach

For each element in the diagram below our application would have an interface that abstracts each component and thus, simplifies how the elements interact. The benefits of using interfaces include better collaboration. This means that each team member does not have to know everything about another component in order to use it, they can just use the interface.

Figure 1: Initial conceptual sketch of the application and interactions between modules



The thought process behind this conceptual sketch was that the GUI would have limited access to the backend systems and limited knowledge of how those systems interacted with each other. Underneath the User Interface, the backend modules would communicate and interact with each other through function calls and inheritance to allow for a modular approach to development and interaction. The backend was designed to be doing the heavy lifting, supporting all functional requirements from data importation and parsing, to containing the data, to visualizing the data and then presenting the visualizations to the GUI. The GUI was supposed to be as lightweight as possible and only call functions that were absolutely necessary to keep the process moving. Otherwise, the backend would control the pipeline.

One of our core values when designing the architecture was to make sure that neither backend or frontend would modify or manipulate any of the data stored without making an appropriate API call. Doing this would allow us to make absolutely sure that the path

the data takes from start to finish is well documented and understood from those outside of the system. Users of our application should be able to read a user's guide and understand the structure of the system.

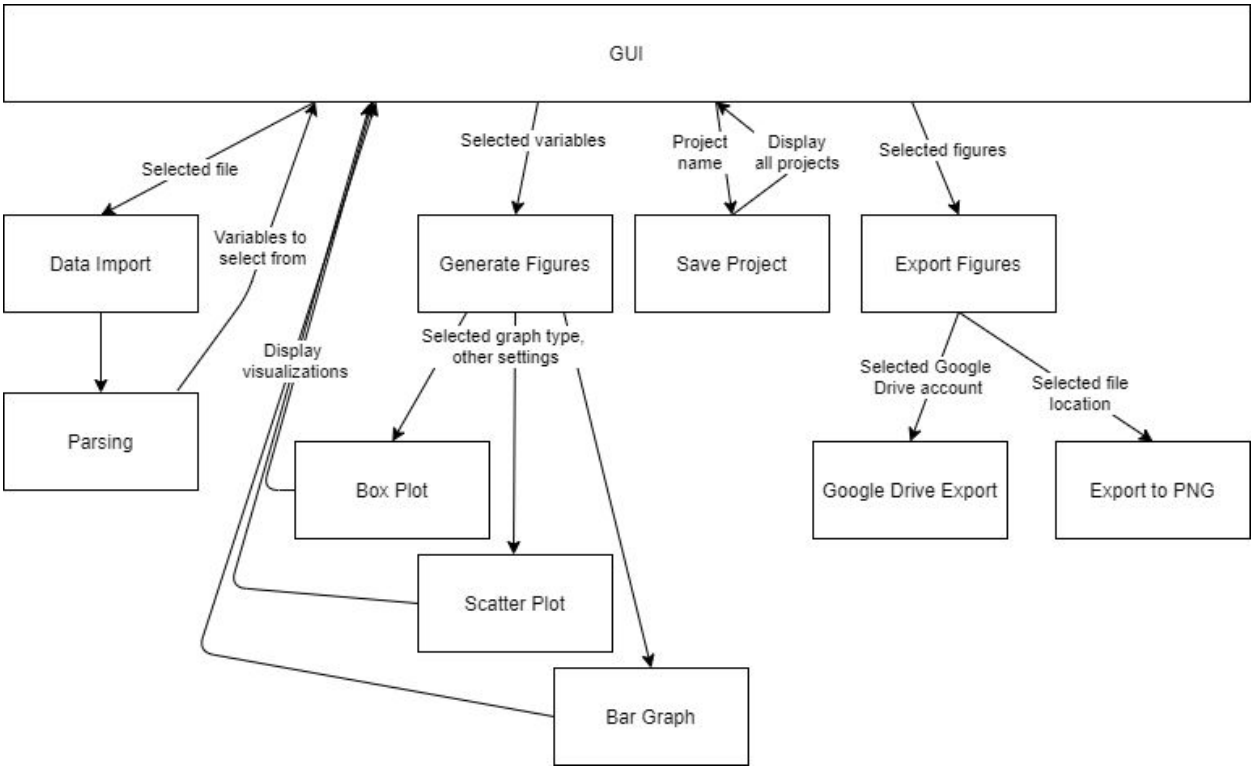
To see more design ideas our team had at the very beginning of the project, please see Appendix 1.

New (Final) Design Approach

While the final design did change "dramatically" between the initial design approach and the final design approach, the main core value of manipulating the data through the use of API calls to the backend was maintained. Doing so allowed us to swap out GUI implementations on the fly with little worry that the backend would respond poorly to the new GUI and reject it.

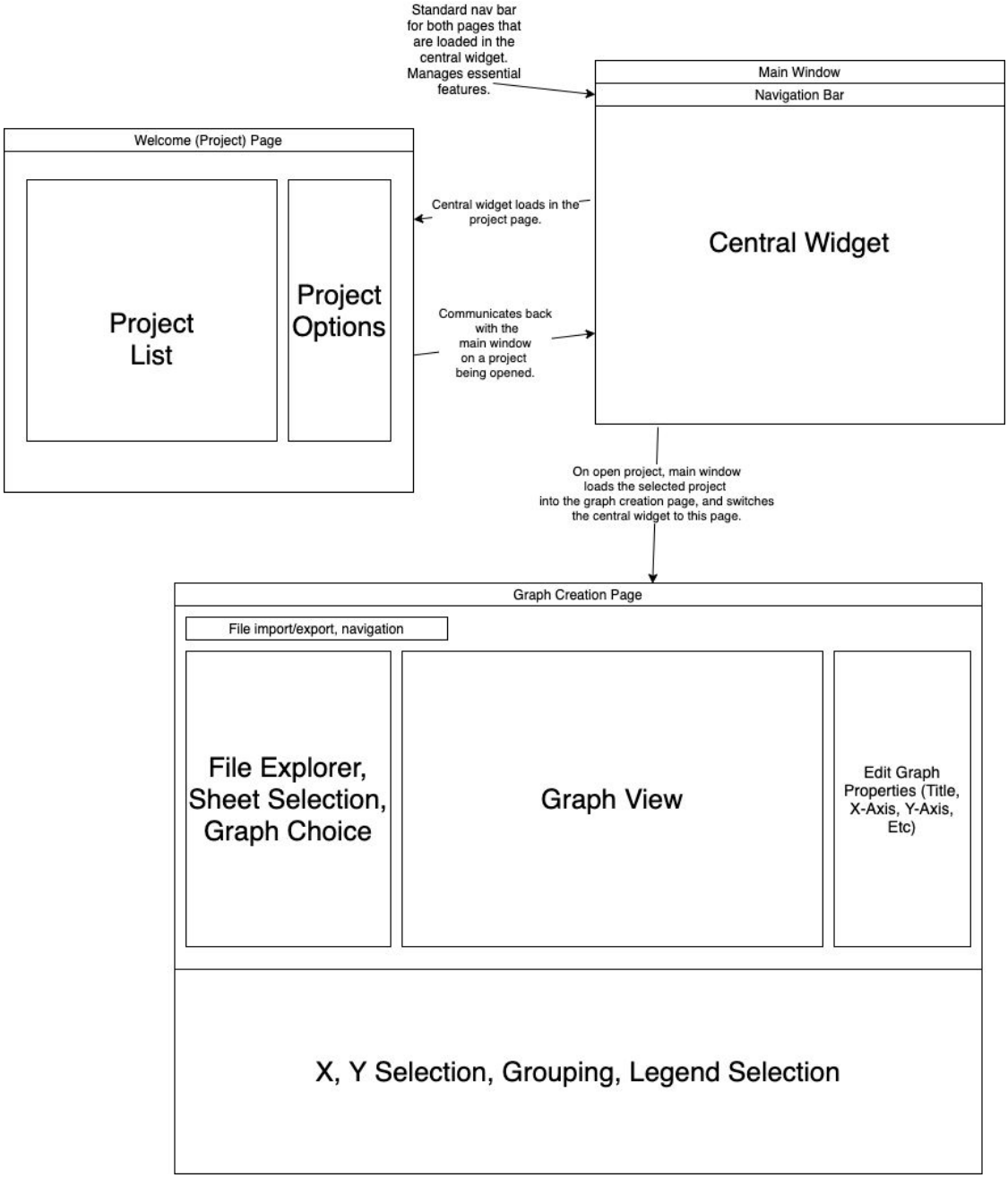
Outside of keeping our modular, interface-style approach to the backend, we did alter our design philosophy greatly in regards to the frontend's role in dealing with data. We decided that the frontend would make API calls to the backend to get certain information it needed (such as all the data variables in an Excel sheet). Then, the frontend would make another call to the backend with new data (such as the variables to be graphed, selected by the application user) to get the final product it desired. Doing so shifted the burden of the data flow to the frontend, keeping the backend completely segmented with no reliance on any other modules within the backend. We thought this would be the best approach so that the core building blocks of our product, located in the backend, could be completed faster than the frontend, which was going to continue being developed on until the end of the project. With this design adjustment, our new conceptual sketch looks like the following:

Figure 2: Final conceptual sketch of the application and interactions between modules



Frontend (Graphical User Interface)

Figure 3: Final frontend GUI flow



The GUI has a main window that manages what page is loaded in. This main window also manages a unified navigation bar and any communication that occurs between the two main pages. When something calls for a page to be loaded in, it sets that view as the central widget.

The first page is the project page, which is primarily used for project creation and selection. Projects allow the user to have specific projects for different data sets, as well as generate graph templates that are unique to a specific project. Doing this allows us to keep the user's data persistent from session to session, and allows the user to save and quit and be able to come back later and continue working on the same graphs and files.

The second page is the graph creation page, which has several different elements throughout it. On the left, users can select a certain workbook, workbook edition, and sheet that they would like to generate graphs from. Below this, there is a graph explorer which contains the graphs already generated from the specific sheet selected. From this, the user can select a graph to view. At the bottom of this segment of the window, users can select the type of graph they wish to generate. The bottom-half of this screen contains our mass-graph generation. In this portion of the page, a user can select the data they want to be graphed as well as how they would like the data to be grouped. Users can select multiple variables at a time to generate multiple graphs at the same time. Finally, on the right side of this screen holds a section where the graph's properties can be edited after being generated. This includes the title, axis names, text font and size, dimensions of the graph, the min and max values of the x-axis and y-axis, as well as the marker colors and symbols. The main window's navigation bar holds buttons for file importation and exportation, default graph preferences, and graph template selection. In the center of the screen is where a selected graph generates a preview for the user.

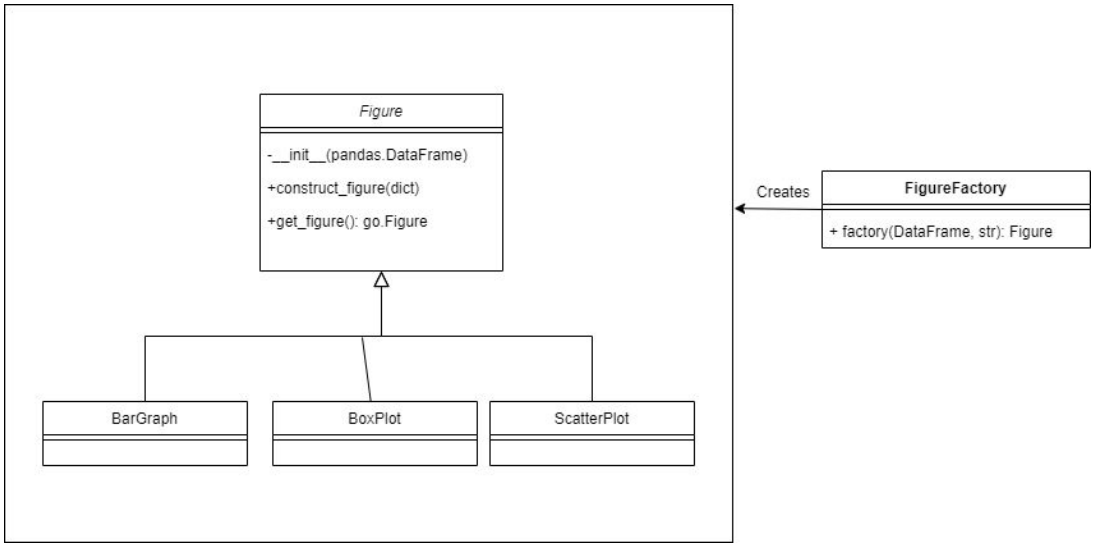
We thought this approach worked the best as it simplified the GUI to two different windows, while also keeping all the relevant information present to the user. Instead of having the GUI broken up into many smaller and more simplistic windows, we are able to have the user control all the tools needed to generate graphs without having to navigate through multiple windows and frustrating UI designs.

Backend

As a result of the frontend being so involved in how we control the flow of data, the backend was kept extremely simple in its structure (aside from one portion which will be talked about later in this section). We designed the backend such that it did not care which type of UI (if there even was one at that point) was calling the function, so long as it made a proper function call. The backend would take the input, process that input into an output, and return it back to whoever made that call. This essentially kept not only the backend as a whole very self-contained, but also made the modules within the backend self-contained as well. Doing this allowed us to make our backend testable, portable, and stable as we moved to developing the frontend more.

The only part of the backend that required a good amount of development time and thought process was how to process the data into an object the GUI can graph. Because each method of graphing (box plot, scatter plot, bar graph) are different in the underlying Plotly engine we use, the backend needs to know which one the frontend wanted, while also being generic enough such that the frontend would not have to manage independent object types within its code. This is why we decided to use the Factory method (pictured below) as well as abstract classes.

Figure 4: Figure Factory implementation



The GUI calls the `FigureFactory.factory()` method with a `DataFrame` object and a `string` (or enum) corresponding to the type of graph it wants generated. The `FigureFactory` then creates the specific `Figure` object for the desired plot and returns it to the frontend. The frontend simply knows that it now has a `Figure` object with two methods it can call (`construct_figure()` and `get_figure()`), which are utilized to display the graph to the user.

Using this approach allows the frontend to not have to be refactored every time a new graph needs to be generated or when the backend needs to fix an issue with a specific plot. Instead, there is only one point that needs to be changed when adding or removing supported graphs: the `FigureFactory` class. This design also allows us to avoid the frontend having to manage different object types for different graphs as everything can just be considered as a `Figure` object. Finally, by having a dictionary as the set of arguments for constructing the figure, we are able to keep templates and configurations

separate from the instance of the `Figure` itself. We now can use the exact same arguments on multiple graphs without having to require the user to input the arguments over and over. The dictionary can be saved as its own object and then loaded in whenever the settings it contains are wanted.

Another design challenge was how to manage graphs. Over time, we realized we wanted to import revisions to a workbook while still utilizing the same graphs already generated. We also foresaw issues with file size, as each graph would save the data it needed into a pickle file. Our solution to this was to separate the data used by a graph from a graph's configuration file. This greatly reduces a project's size, and prevents duplicate data. It also allows us to import revisions. Each "edition" of a workbook saves into a project's workbook, and the client can now select which edition they want to use. Using that selected edition, it grabs the data from the edition selected, combines it with the graph's configurations, and loads in a graph.

Implementation Details

This project was completed entirely using Python and Python libraries. The following is summarized list of the Python libraries we used:

- Plotly
- Pandas
- PyQt5
- PyDrive
- Unittest

Plotly

The first software library we interfaced with is Plotly, a graphing based module. This was the engine of our graph generation. We performed rigorous input and data manipulation testing with the module to make sure that it can handle the large and complex amounts of data that will be inputted to the application at one time. We found that it was able to handle the data we were giving it, as well as provide enough options and flexibility to let the user define their own constraints and customization on the graph being produced.

Pandas

Another software library we interfaced with is pandas, a data analysis and manipulation tool for Python. This was our primary module behind data importation and parsing. With the use of pandas, we were able to handle wrong files, large files, different types of files, while also providing a unified and correct output so as to keep the logic between importing data and manipulating/graphing the data as simple as possible. Additionally, we were able to use some of the built in functions for pandas to perform statistical analysis on our data.

PyQt5

We also used PyQt5, a Python binding of the cross-platform GUI toolkit Qt. We utilized this toolkit to develop our Graphical User Interface. With this library, we were able to create a GUI that was simple and easy to use, while also looking clean and professional. We were able to use File Trees, multiple screens, pop-up windows, scrolling windows, sectional dividers, and a helpful menu bar (with dropdowns) to streamline the user's experience.

PyDrive

We also utilized PyDrive, a wrapper library that simplifies many common Google Drive API tasks, to export and upload generated graphs to Google Drive. PyDrive made Google Drive authentication simple and also allowed us to maintain access to multiple Google Drive accounts at the same time. Additionally, with the use of this library, our users can not only upload their generated graphs to Google Drive, but also create new folders in their Google Drive all the while staying within our application.

Unittest

Unittest is Python's own unit testing library. This library helped us to create and run robust unit testing required for each of the modules within our project.

Implementation Standards

Throughout our implementation process, we kept the following standards in mind:

- Simple, modular design: This makes it easier for each team member to interface with the code of another team member.
- Single coding standard (specifically PEP-8, a style guide for Python coding): This ensures that we all use the same coding standard and utilize best practices when developing our Python code.
- Frequent refactoring: This forces us to retrospectively review the work we have completed and make it better. It also helps us clean up our code and program structure, getting rid of anything we no longer use, and cleaning up anything that has been made too complex.
- Heavy documentation in both code and manuals, updated regularly: This will make it easier for our intended users to understand and use the application. It will also help anyone who decides to continue working off our project in the future.

Source Code and Documentation

To see our source code and the corresponding source code documentation, please go to our Senior Design Website (https://sddec20-29.sd.ece.iastate.edu/). Here, you will find a link to our GitLab Repository, as well as a set of HTML files documenting our source code.

Testing Process

Due to GraphKey being designed entirely as a software solution, we were able to benefit in our testing as our software would not have to be integrated with hardware, keeping the amount of failure points in our product relatively low. By running our software on a cross-platform language and interpreter such as Python, we were also able to ensure that our solution could work for Mac, Linux, and Windows type machines without any concern of outlier behaviors.

Thanks to these constraints and design decisions by the client and the team, we focused on two specific forms of testing, unit testing and integration testing.

Unit Testing

For unit testing, we used Python's unittest framework which allowed us to create fast and simple tests for the backend. Take the following test as an example:

Figure 5: Example unit test

```
import os
import unittest
from GraphKey.app.modules.data_import.edit_data import DataEdit


class TestDataEdit(unittest.TestCase):
    def test_raises_error_when_given_invalid_file(self):
        invalid_file = 'C:\\dummyfile.txt'
        with self.assertRaises(FileNotFoundError):
            DataEdit(invalid_file)

    def test_raises_error_when_given_directory(self):
        working_directory = os.getcwd()
        with self.assertRaises(FileNotFoundError):
            DataEdit(working_directory)
```

Doing this allowed us to use Python's built-in support for testing to make sure that our backend was stable enough for the frontend's tasks. On top of these specific automated unit tests, we also hand tested modules within the backend, particularly in the case where they required hooking into different services (like uploading images to Google Drive).

Along with these unit tests, early in the development process we had our GitLab repository set up for CI/CD (Continuous Integration and Continuous Deployment). On each branch of development, whenever a developer pushed, the GitLab repository would run the unit tests within the test folder and notify the owner of the branch if the branch failed testing. We also prohibited failing branches to be merged into master. Doing so allowed us to make sure that master was as stable as possible and any changes to master would not affect the stability of that branch. Unfortunately, as the frontend was getting more developed and new PyQt pages were introduced into the project, the pipelines would begin to fail because of certain dynamically created files that PyQt required. Due to this, we decided to disable CI/CD and trust developers to run the unit tests on their machines (which still worked) to make sure they passed before merging into master.

Our main challenge in the testing process was being diligent in writing tests for our code. Due to the shorter semester and lack of unit testing training and experience, developers on the team tended to test things by hand instead of creating detailed unit tests. Continuous testing became harder and harder to do as technical debt built up, and if we had more time, we would have liked to flesh out our unit testing more.

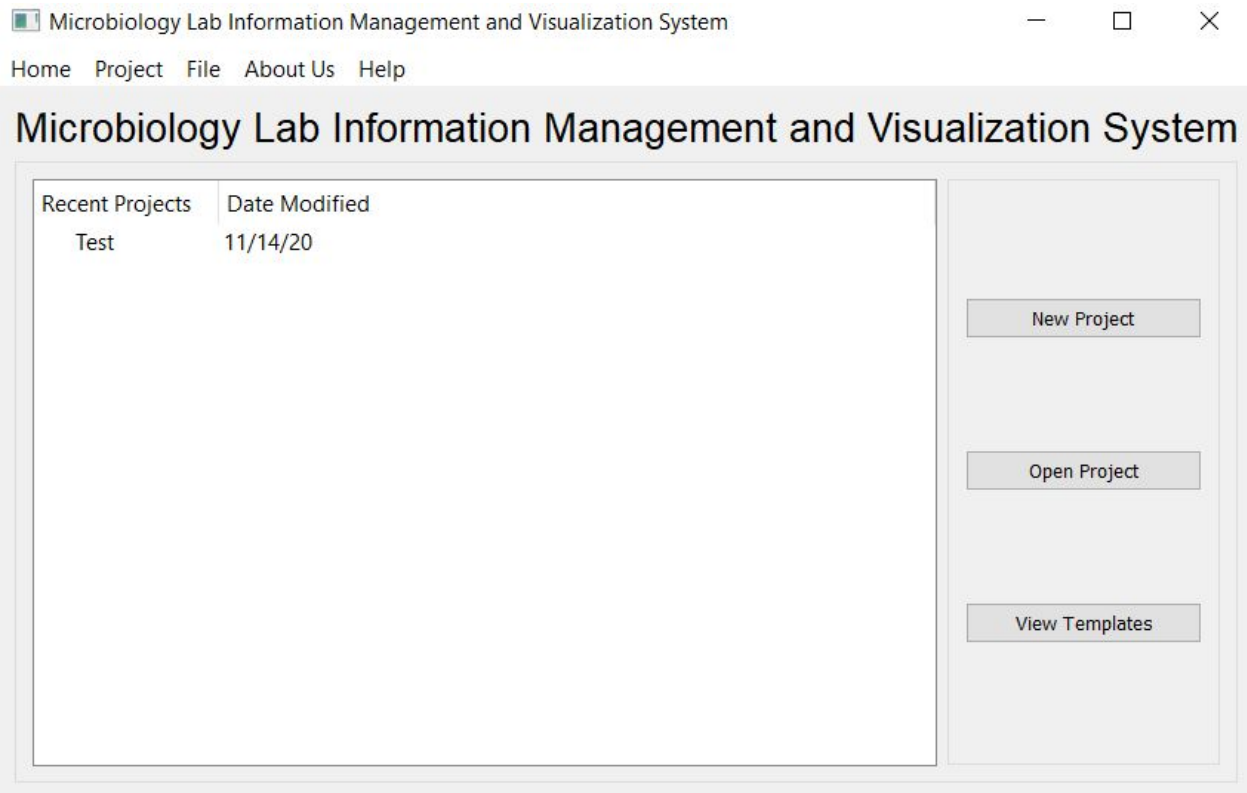
Integration Testing

We mostly completed integration testing by hand and within the meetings with our client. This type of testing was primarily focused on how the frontend operated and how the GUI looked and performed. This was done primarily by hand and eye due to the team's lack of experience with automated GUI testing and development. We would also consult the client on the layout and format of the GUI so that we could develop an easy-to-use application that supported all the features that the client desired.

Results

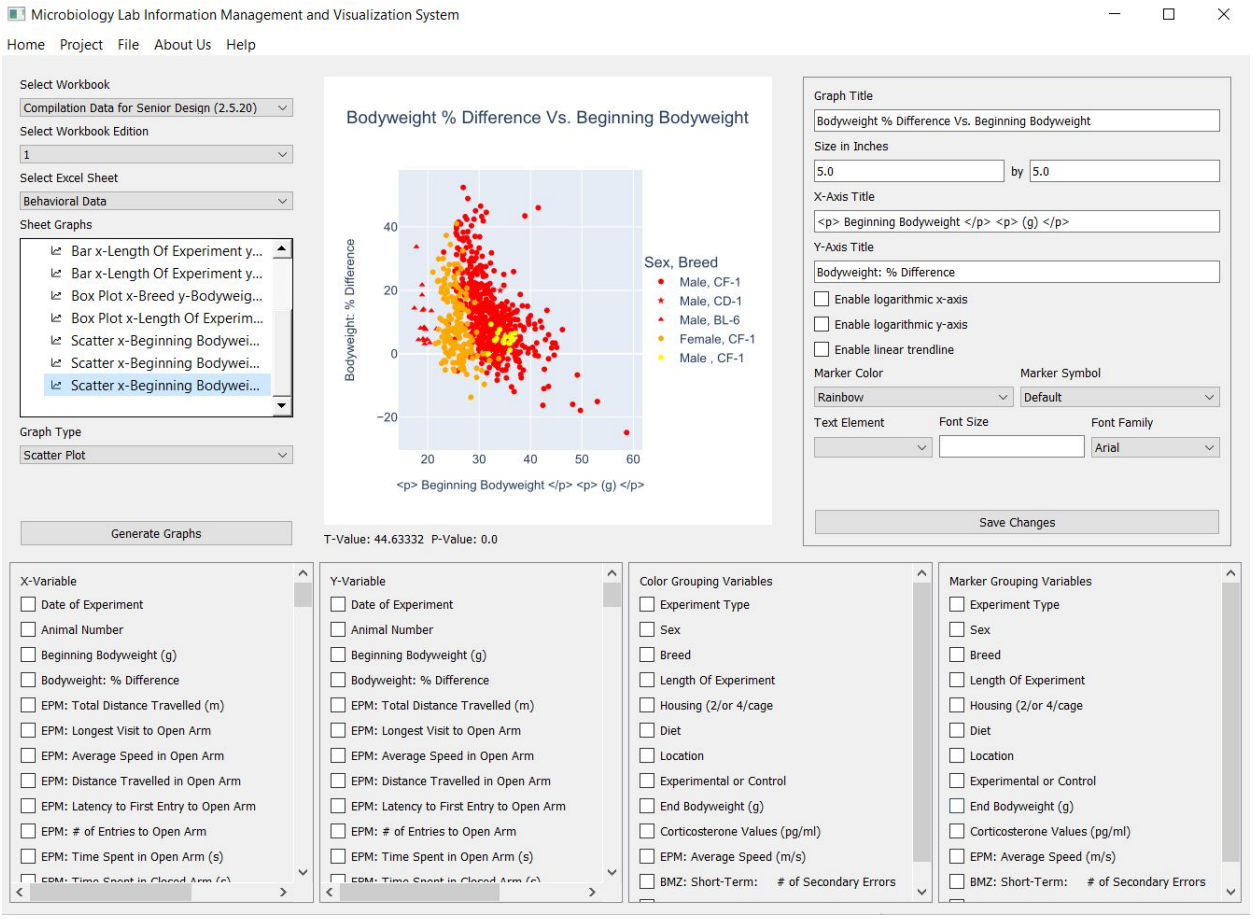
After a year of design and development, we have produced an application called GraphKey that can import scientific data from Excel sheets into custom projects (that can be saved and loaded between sessions) and generate multiple customizable graphs at the same time. All of this is done in a program that can be run on any machine so long as Python is installed. Thus we have developed an application that can be used for scientific research papers. The first screen (for creating and selecting projects) is featured in the figure below:

Figure 6: GraphKey Projects Screen



The second primary screen of our application (for generating and customizing the graphs) is featured in the figure below:

Figure 7: GraphKey Graphing Screen



Along with the delivery of our product, we have strived to maintain simplicity and modularity in the application in order for easy expansion and maintenance after the development team is finished. We have also delivered a user guide (see the Appendix) and documentation (see our Senior Design Website) for those who would like to work on the project after we are done.

For a short demo video of our application, please go to our Senior Design Website (https://sddec20-29.sd.ece.iastate.edu/).

Conclusion

Throughout the year, our team has been continuously creating, implementing, and improving our GraphKey design to meet the requirements of our client. Our end goal was to produce an application that allows the user to import pre-existing data from Excel and to manage and visualize the data. We also wanted the user to be able to customize the data variables used to make-up the graphs, as well as the appearance of the graphs. Additionally, we also wanted the user to be able to export and share generated graphs with another person.

We believe the product we have come up with meets all of the goals we set out for ourselves. In addition, the product is an easy-to-use application that does not require much maintenance, and allows our target client demographic, scientists and researchers, an easier method to organize, maintain, and visualize their data. We have tested and experimented with the base components of the application to guarantee our design works and meets our client's needs. We believe that we have completed an application that confirms the exceptionality of our design choice and solves our client's problem.

Appendices

Appendix 1 - Previous/Alternate Design Versions

Throughout the design process, we had several different versions of the project that would have either been alternative solutions or additions to the current solution that would have dramatically changed how the application was run and developed. The common reason as to why these designs were scrapped were due to a combination of being less efficient, more difficult to maintain after completion, and time consuming for development.

Alternate Design 1: Machine Learning Integration

At the very beginning of last semester, we had a stretch goal to implement TensorFlow machine learning into the application as a sort of assistant to the user. It would comb through the important data and point out any irregularities within the data that may have come from entering the data wrong (such as adding an extra zero) as well as giving suggestions as to which data would be graphed and how. Doing so would ease up the burden of graph creation on the user immensely and would increase the speed of graph generation. As can be expected, this was scrapped pretty quickly into development as the time it would take to integrate the machine learning algorithm into the application, let alone training it to spot erroneous data and provide graphing suggestions, was way out of the scope of this project. Not only would have it been difficult to implement correctly within the app, but it being in the application creates a massive headache for maintainers after the team's departure. In order to keep the application small and manageable, this design was left on the drawing board.

Alternate Design 2: Web Application + Database

When coming up with different designs for the application, another solution was to have a web application and database create the graph. Advantages for this would be the removal of a local app on the user's machine and ease of access for anyone who would want to use it. Another bonus is that members of the team had more knowledge in web development and database management (to a degree) over Python, which would keep the cost of learning a new language down. This idea was also moved on as the question of maintainability and costs arose. In order to keep costs essentially free and make it so only one person would need to make small changes in their free time, having a complex web application and database relationship would have been too much for that person to devote resources into maintenance.

Alternate Design 3: Raspberry Pi

The last application design that was considered before moving to our final design was to have the entire thing contained on a Raspberry Pi. The thought process here was to have it plugged into the computer of the user and run and store data within the pi and give the user their graphs if they so desired. This was designed as more of a middle ground between our final design and the web application, as it could have easily been converted to a small server that the user could start up and connect to for processing data and creating graphs.

The reason why we moved away from this design was because of wanting to keep the application simple and easy to maintain, as well as for computational reasons. The first reason was fairly simple, keeping the application on the users and developers machines would make for easier and more rapid deployment of changes and updates rather than pulling in the Pi and making changes locally on that machine. While the difficulty of maintenance is not that bad when it comes to modifying an application on top of a Pi, it is only exacerbated by the second reason.

Lack of a powerful processor on the Pi is the primary setback to this solution. Because pulling in, processing, and graphing large amounts of data is CPU intensive, having a small amount of processing power that the Pi has can make the experience of graphing with the application miserable for the user. With long load and render times, we did not want the client to feel as if they were just waiting for something that would be much faster to do manually. While the power of the new Raspberry Pi 4 has increased, it is still much less powerful than modern computers that can run our application faster.

Appendix 2 - Other Considerations

Throughout our project our team suffered a lack of cohesion and a large architecture shift, mostly a result of lack of experience in development as well as having to work entirely remotely due to the ongoing global pandemic. While some of us have had some work experience through internships and co-ops, we as a team just did not have enough experience as a whole to prevent these problems from arising throughout our development.

Independent Projects (Lack of cohesion)

At the beginning of our project, we made a huge push to be diligent in our git practices. This meant having independent branches for each feature, and only merge into master once the feature was done and reviewed by peers. We kept this value core throughout

development and it helped a lot when it came to keeping master the most stable branch. Unfortunately the side-effect of that was in the beginning when we did not have a solid foundation and backend to build the GUI. This resulted in each person having a different frontend design that would work with their code. When they merged into master, master became a collection of different individual projects that each worked for only one feature. It essentially became a bunch of "hello world" mini-projects rather than one application. This was further exacerbated by the global pandemic, which forced us to work remotely, severely reducing our abilities to collaborate.

This issue forced us to spend a week or two of development time to sit down and collect all these independent features under one GUI. All branches were forced into master and then collected into one comprehensive frontend and one comprehensive backend. After this was done, development and integration into the project was much easier, as we had a foundation to expand upon and work with on each branch, while still keeping master as the stable branch. Unfortunately, had we done this much earlier than we did (or had this plan from the start), the entire rebase and refactor could have been mitigated or completely avoided, speeding up development time.

Architecture Shift (Subsequent Refactor)

The second problem we had was intrinsically linked to the first problem above as well as the outlined design change between the initial design approach and the final design approach. We made this major shift in design later in development in order to support the inevitable problem that was the independent projects, but we had originally designed the backend as the one that was the side manipulating the data. In order to fix these issues, we had to spend time evaluating the design philosophy of the backend and its role within the application. Finally we came to our final design approach and implemented it into GraphKey while the frontend was coming together. We believe that the design choice of our final project is a good solution. We only regret that we made the shift so late in the developing process.

References

[1]“How to Buy Prism,” GraphPad. [Online]. Available: https://www.graphpad.com/how-to-buy/. [Accessed: 26-Apr-2020].

User Guide

(Please Use the Table of Contents on Page 4)

GraphKey User Guide

A Microbiology Lab Information Management and Visualization System

Senior Design Project

2020

Team 29: Brittany McPeek, Benjamin Vogel, Rob Reinhard, Kyle Gansen, Ben Alexander, and Samuel Jungman

Team Website: https://sddec20-29.sd.ece.iastate.edu

1. Overview

The purpose of this User Guide is to provide a user of GraphKey all the details needed to use the GraphKey application. This Guide will walk the user through generating Projects within the application, importing data, generating various types of graphs, creating graph templates, and exporting graphs. All available options within the application will be explained.

2. Getting Started

This product is a software application that allows the user to import pre-existing data from Excel and manage and visualize the data. The application supports the importation of data in CSV and JSON format. Users can select the type of graph they would like to generate along with the data they would like to be graphed. Three graph types are supported: scatter plots, bar graphs, and box plots. Multiple graphs can be generated at once if the users selects more than one data grouping for each graph variable at a time. Specifically, the application will generate all permutations of the selected data grouping. Depending on the graph type, the user can also select to use colors and symbol markers to visually group the data on the graph. The user can also customize various graph settings, including but not limited to, the dimensions of the graph, the axis labels, and the title of the graph. Lastly, the application allows the user to export the generated graphs to a specified location on the local machine as well as their Google Drive account.

2.1. Obtaining the Application

A copy of the application can be found on the Senior Design website for Team 29.[1] On the **Home Page**, download **GraphKey.zip**.

2.2. Running the Application

2.2.1. Requirements

- Python (at least version 3.7)[2]
- GraphKey.zip (see Section 2.1)

2.2.2. Installing Requirements

To install Python, please see Python's Website[2].

To install GraphKey, please follow the instructions below:

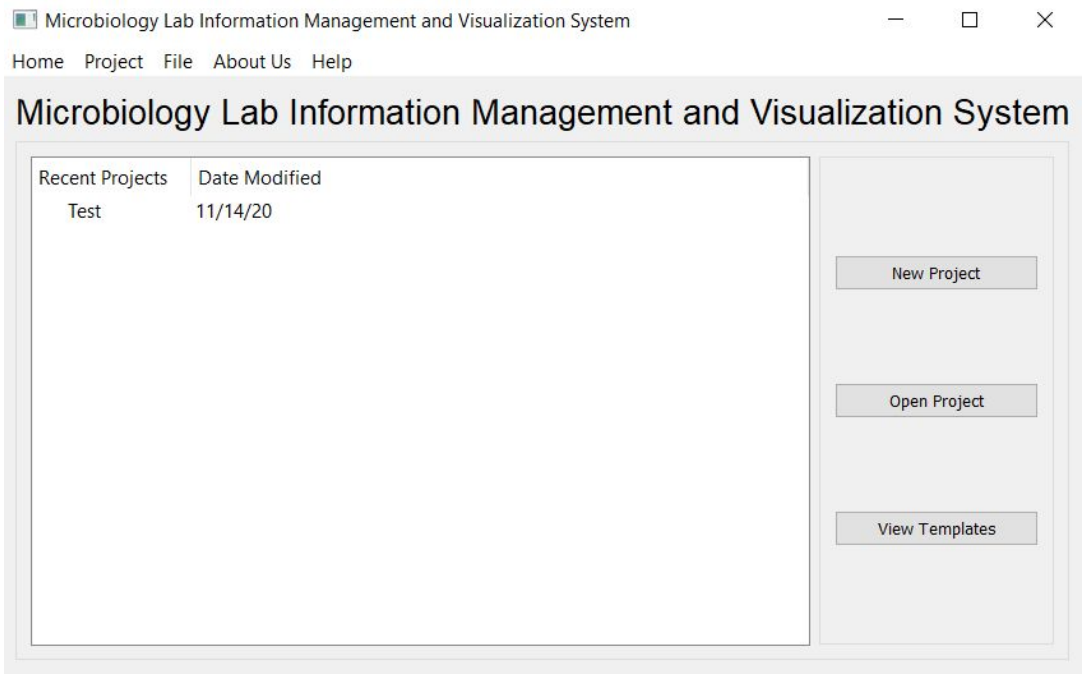
1. Unpack the **GraphKey.zip** file into a desired directory
2. Verify that the **requirements.txt** file is within that directory
3. Start up a command prompt and navigate to the directory where **GraphKey.zip** was unpacked
4. Type `python -m pip install -r requirements.txt` either in a Virtual Environment or your native Python installation

2.2.3. Starting the Application

There are two ways to start the application.

1. Find **GraphKey.py** in your file system and double click on it.
2. On a command line interface, type `python GraphKey.py`.

Please note that it can take up to 30 seconds for the application to launch, especially on the first time launching the application. Once the application has launched, you should see this window (recent projects may be blank if this is the first time launching the application):

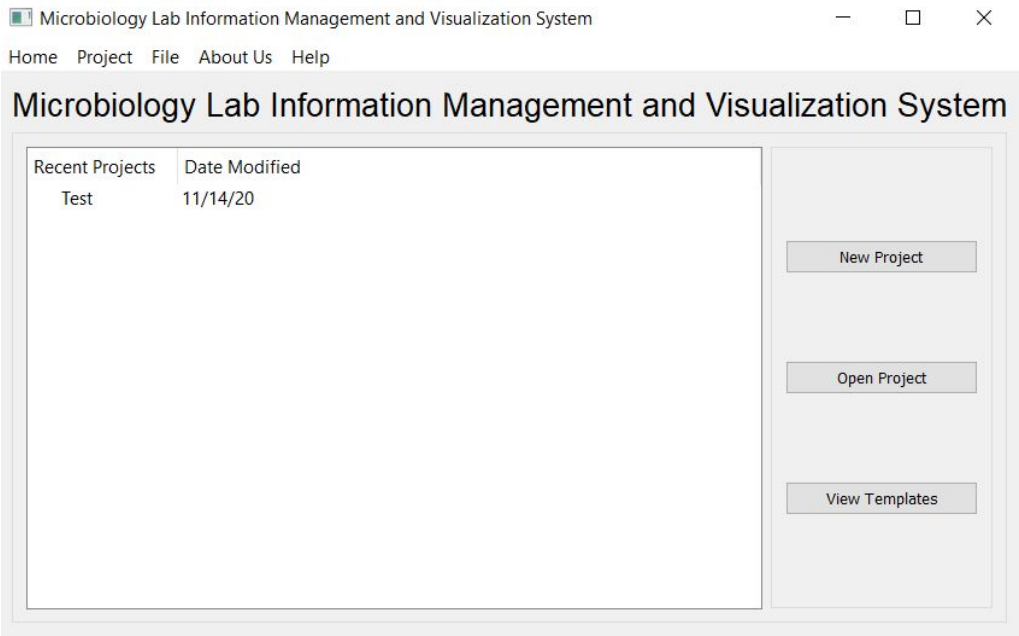


3. Projects

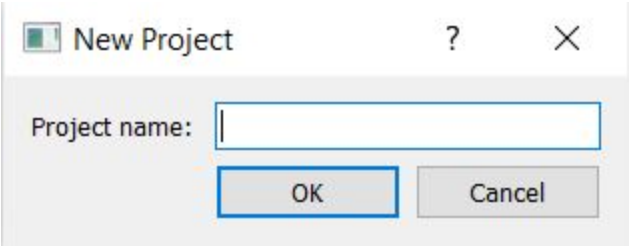
Projects serve as an easy way to keep imported data and generated graphs organized. It is recommended to keep the data and graphs of one experiment separated from another experiment through the use of Projects.

3.1. Creating a New Project

The **Home Screen** of the application looks like the following:



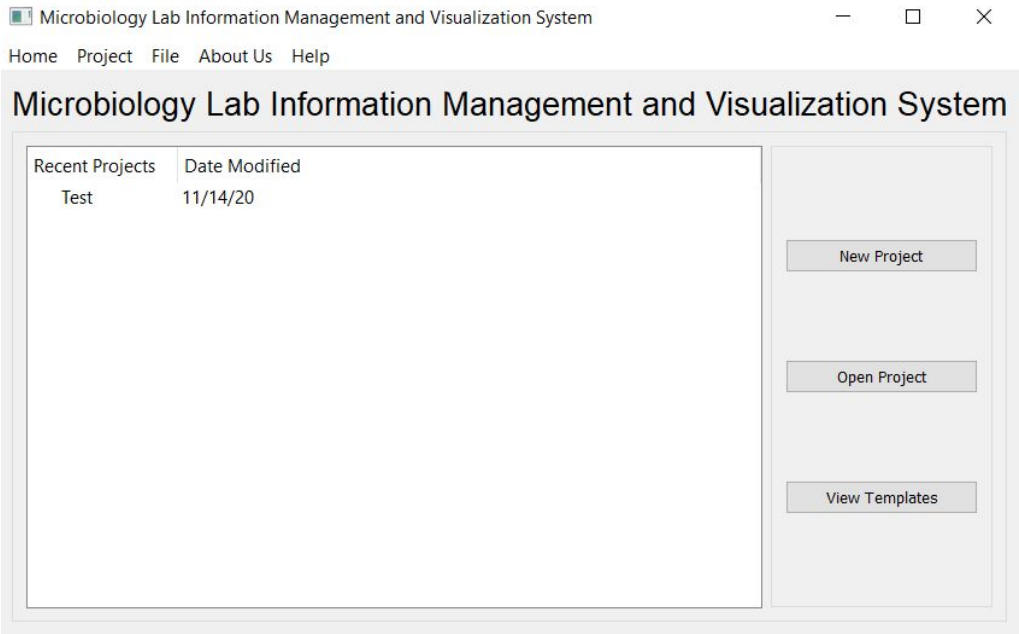
To create a new project using this window, click on the **New Project** button in the right column of the window. A **New Project Pop-Up** window will be displayed.



Enter in a name for the new project and click **OK**. The **New Project Pop-Up** will close, and the project will show up under **Recent Projects** on the **Home Screen**.

3.2. Opening a Project

The **Home Screen** of the application looks like the following:



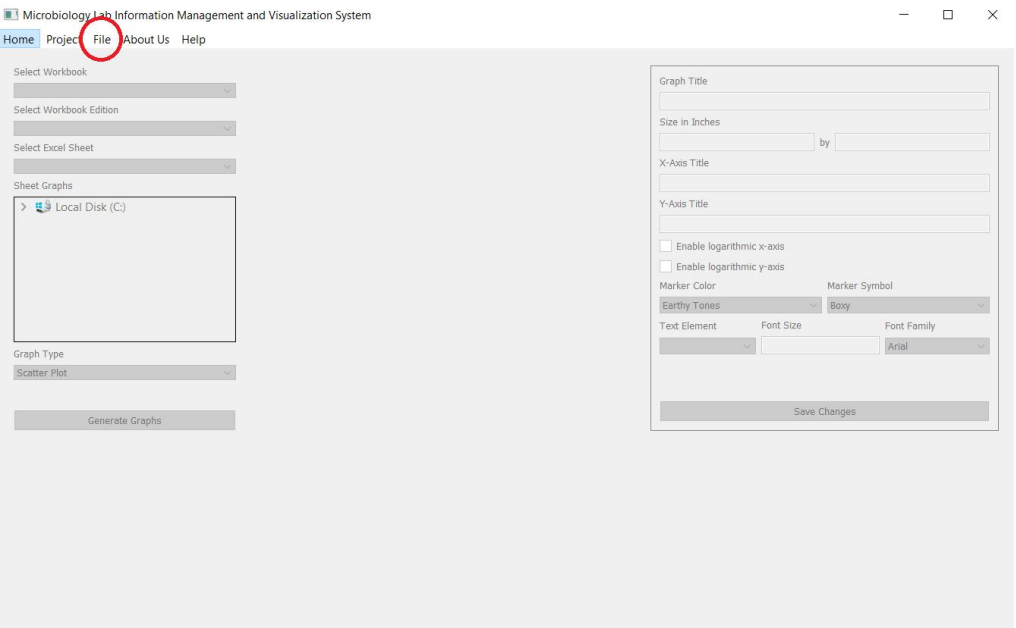
To open a project using this window, either double-click on the project to be opened in the Recent Projects window, or single-click on the project to be opened in the Recent Projects window and then click on the **Open Projec**t button in the right column of the window. The screen will change into the **Graphing Screen.**

4. Importing Data

Data can only be imported while on the **Graphing Screen**. There are two types of data that can be imported: a new data set and a revised version of an older data set (i.e. a data set that has been imported before). Allowing the importation of a revised version of a data set allows a user to keep old versions of the data within the application as well as older versions of graphs. This allows for easy data and graph comparison between data versions.

4.1. Importing New Data

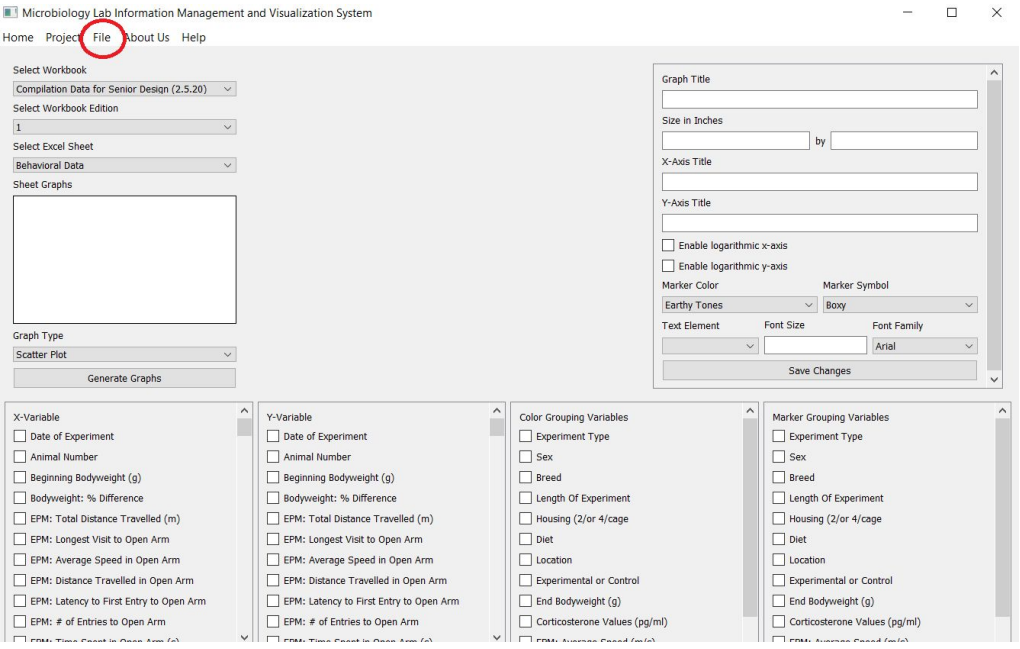
Click on the **File Dropdown Menu** in the upper left of the window.



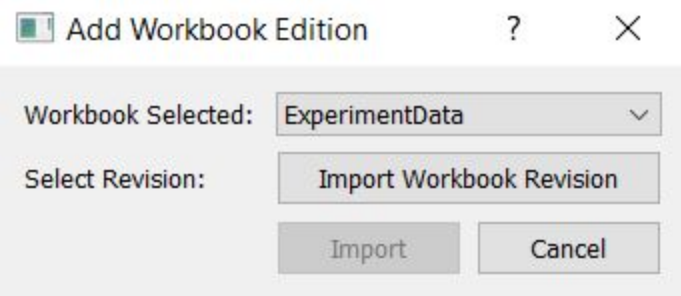
From the dropdown menu, click on **Import File**. A file system browser will pop-up. Browse to the file to be imported. Select the file and click **Open**. If the file is large, a pop-up window explaining to please wait may appear. Once the file is done importing, it will become available to be selected in the **Select Workbook Dropdown Menu**.

4.2. Importing Revised Data

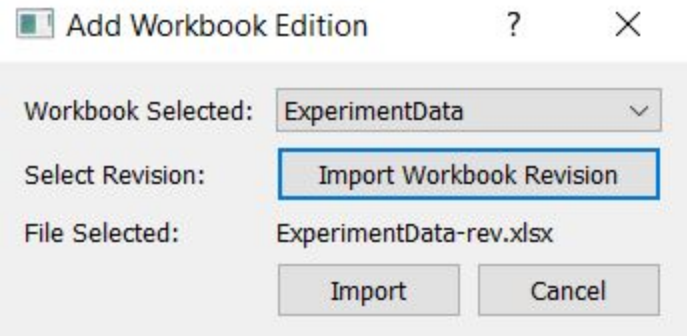
Click on the **File Dropdown Menu** in the upper left of the window.



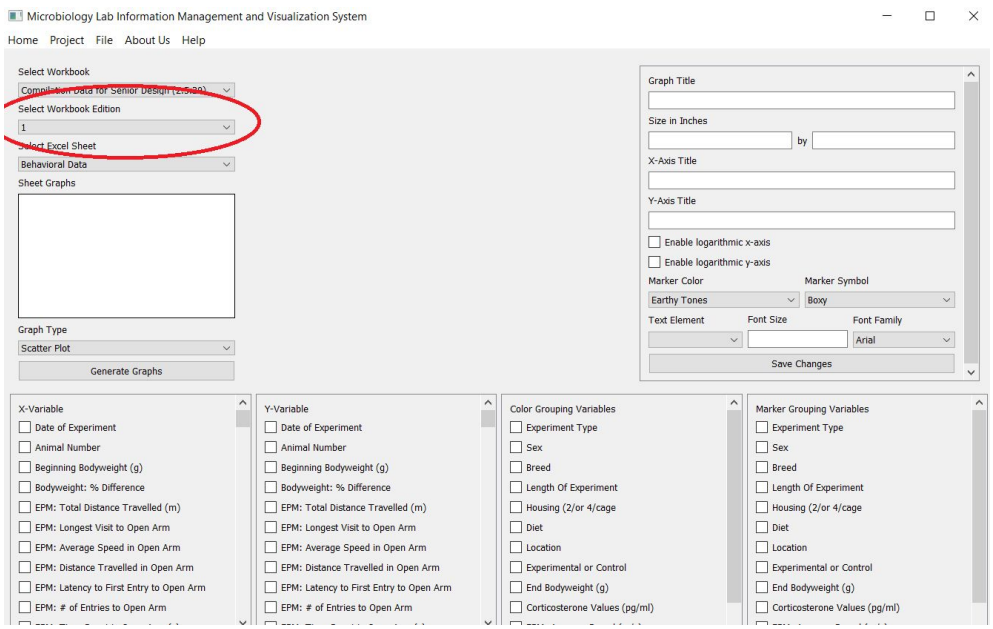
From the dropdown menu, click on **Import Revision**. The **Add Workbook Edition Pop-Up** will appear.



Select the Workbook that has a revised version of its data. Then select the **Import Workbook Revision** button. A file system browser will pop-up. Browse to the file to be imported. Select the file and click **Open**. The **Add Workbook Edition Pop-Up** will now display the select file.

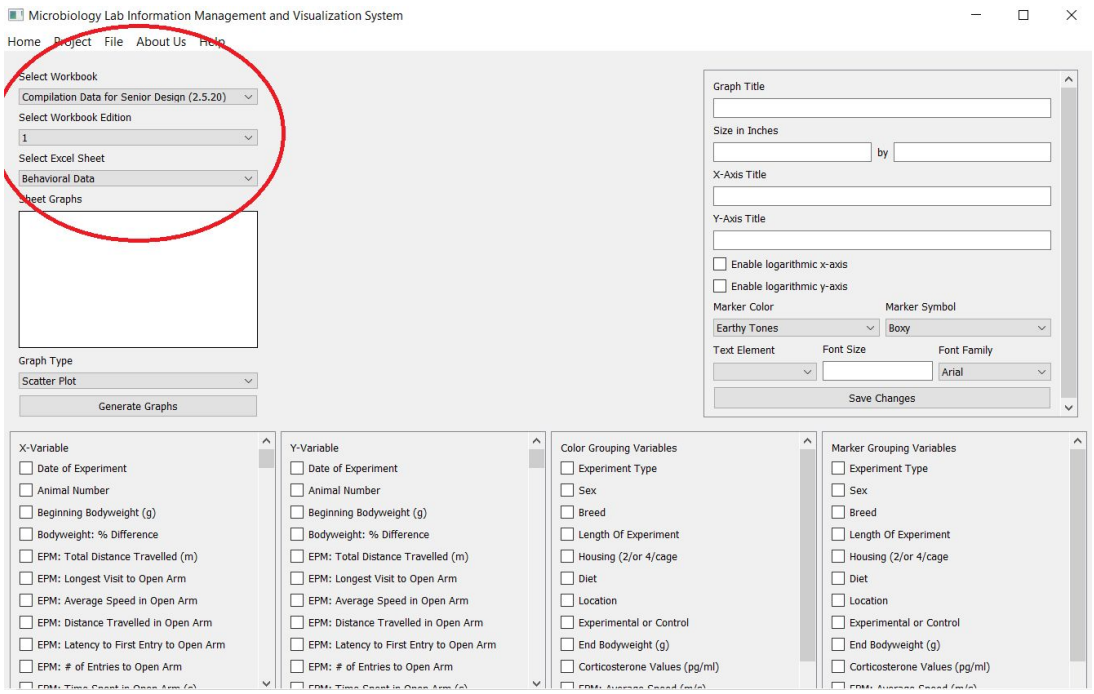


Select the **Import** button to begin importation. If the file is large, a pop-up window explaining to please wait may appear. Once the file is done importing, it will become available to be selected in the **Select Edition Dropdown Menu**.



5. Selecting Workbook, Edition, and Sheet

On the Graphing Screen for a project, the selected workbook, workbook edition, and Excel sheet can all be changed in the upper left hand corner of the screen.



The workbook can be selected from the first dropdown menu. To import another workbook, see Section 4.

The workbook edition can be selected from the next dropdown menu. This reflects any revisions that may have been imported for the selected workbook. So, if the current workbook has no other versions, there will only be "1" to choose from in this dropdown.

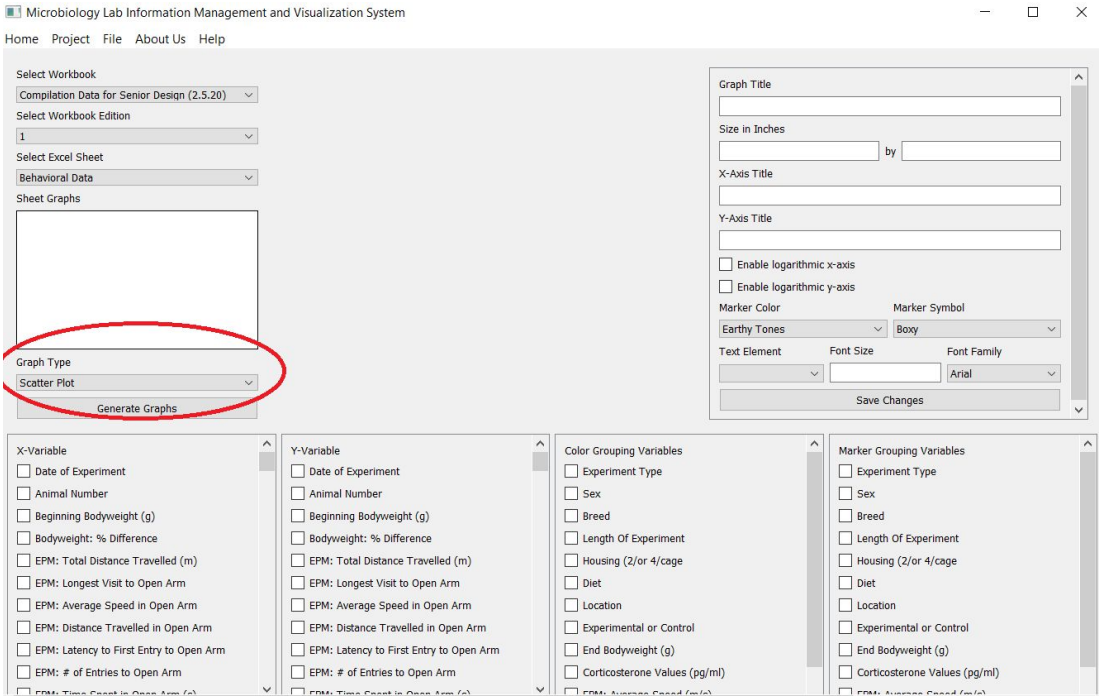
The next dropdown is for selecting a specific Excel sheet in the specified workbook. The dropdown options will simply include the names given to the sheets in the specified workbook.

6. Generating Graphs

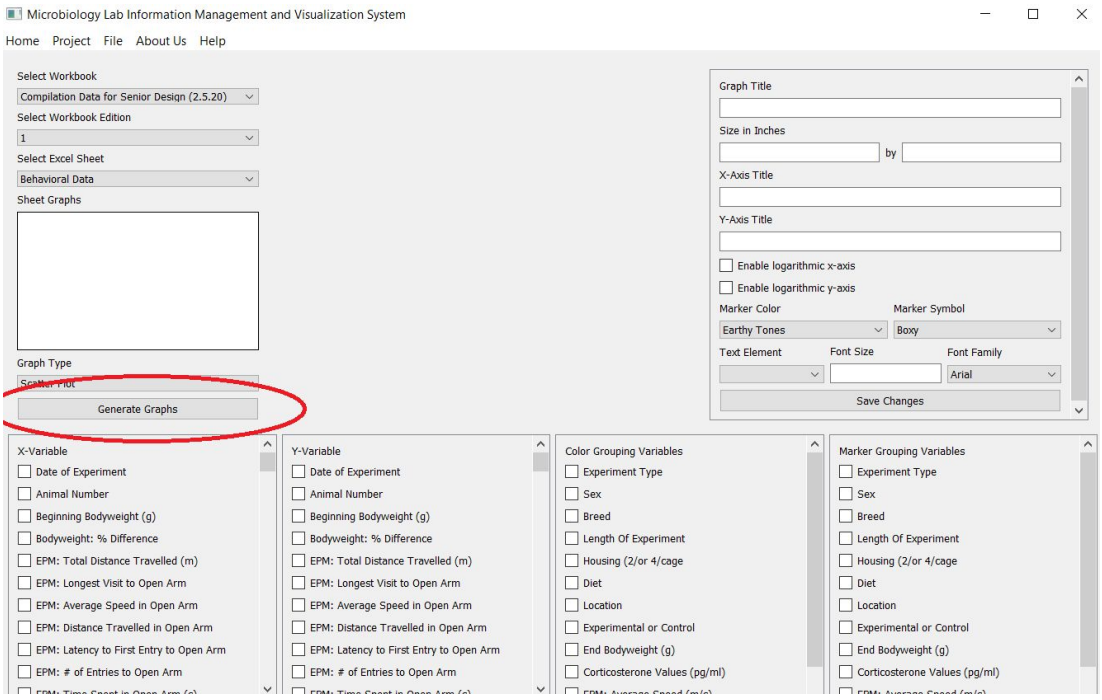
Before generating the graphs, please ensure that the correct **Graph Preferences** are selected. Please see Section 11 for details on **Graph Preferences**.

Additionally, select the **Workbook**, **Workbook Edition**, and **Excel Sheet** where the data to be graphed is located. Please see Section 5 for details.

Then, select the desired **Graph Type**. Scatter Plots, Bar Graphs, and Box Plots can all be generated. See Sections 8 and 9 for details on these graph types and how to select data for the given type.



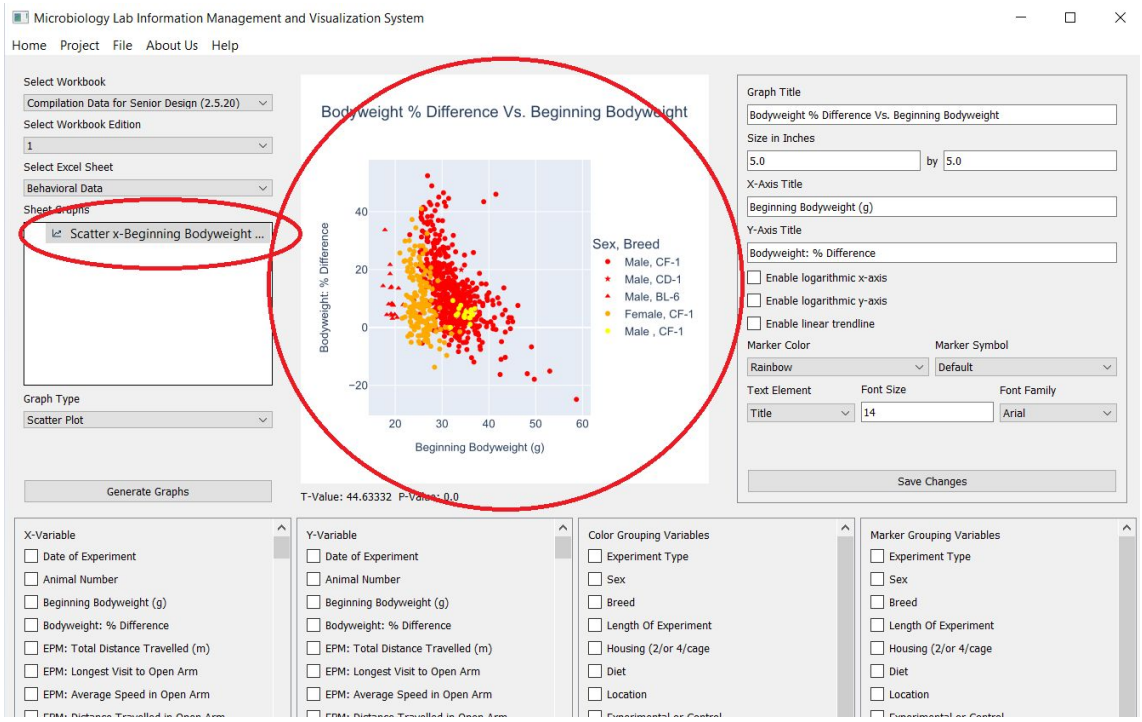
After the data and Graph Preferences have been selected, click on the **Generate Graphs** button located midway down the screen on the left side.



If the graph generation is expected to take a long time, a pop-up window explaining to please wait may appear. Once graph generation is complete, the graphs will appear under the **Sheet Graphs** window section on the left side of the **Graphing Screen**.

7. Viewing Graphs

Graphs available for viewing will be located in the **Sheet Graphs** window section on the left side of the **Graphing Screen**. To view a graph, simply click on the name in this window section. The graph will then appear in the center of the **Graphing Screen**. Note that large graphs may take a while to appear.

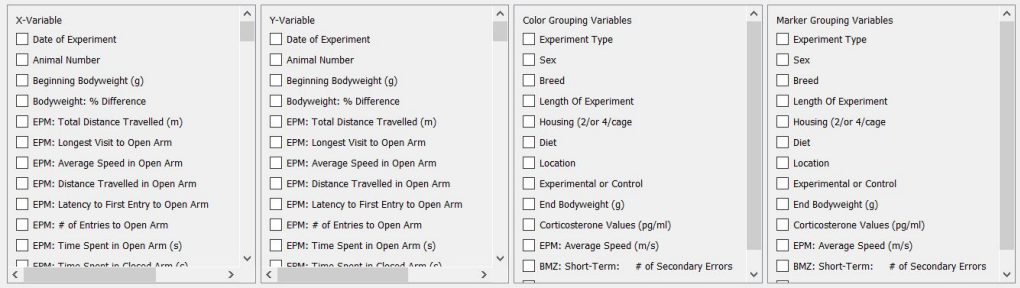


Please note that only the graphs for the currently selected sheet are available for viewing. To view a graph from another sheet, workbook, or edition, change the corresponding selections. See Section 5 for more information.

8. Scatter Plots

8.1. Selecting Data

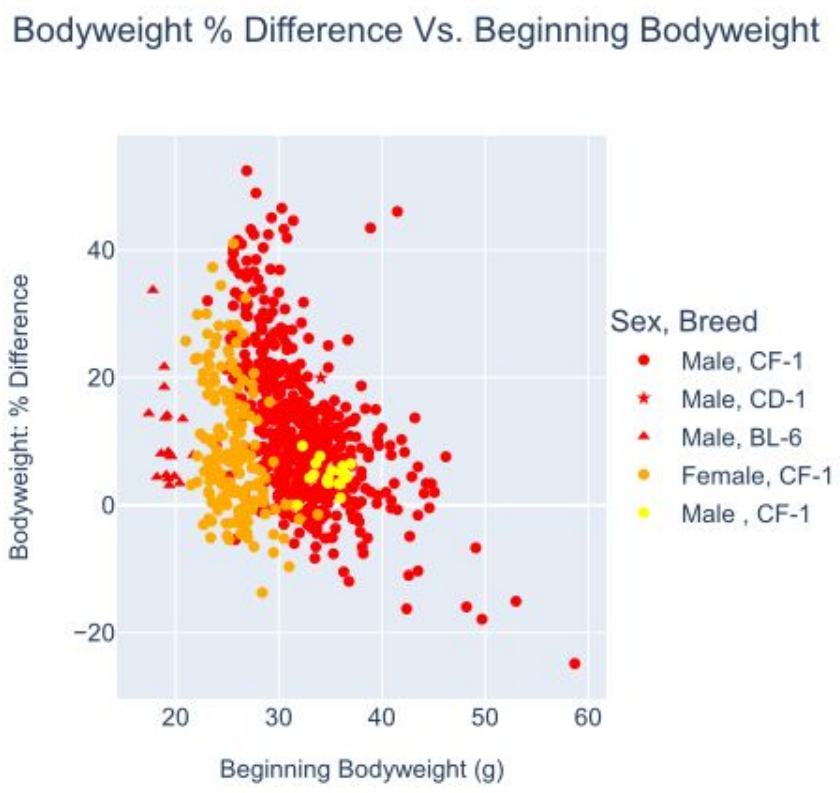
Data groupings can be selected on the bottom-half of the **Graphing Screen**.



For Scatter Plots, the user will be required to select at least one **X-Variable** and at least one **Y-Variable**. For multiple graphs to be generated, select more than one **X-Variable** and/or **Y-Variable**.

Optionally, the user can select **Color Grouping Variables** and **Marker Grouping Variables**. This will group the scatter points by color and symbol markers respectively, based on the groupings selected. To generate multiple graphs with different groupings of the scatter points, select more than one **Color Grouping Variables** and/or **Marker Grouping Variables**.

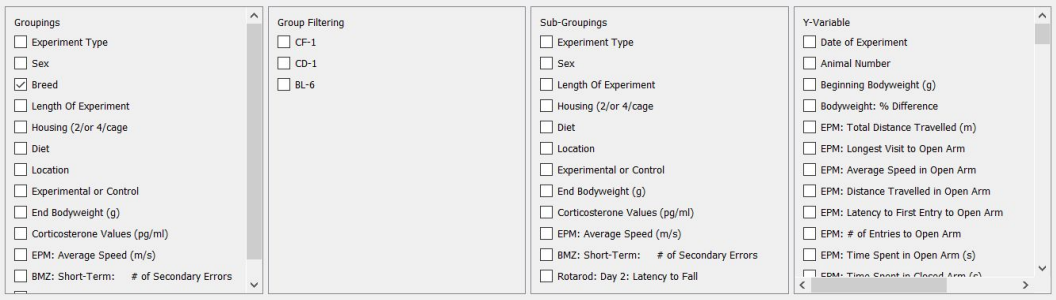
8.2. Example Scatter Plot



9. Bar Graphs and Box Plots

9.1. Selecting Data

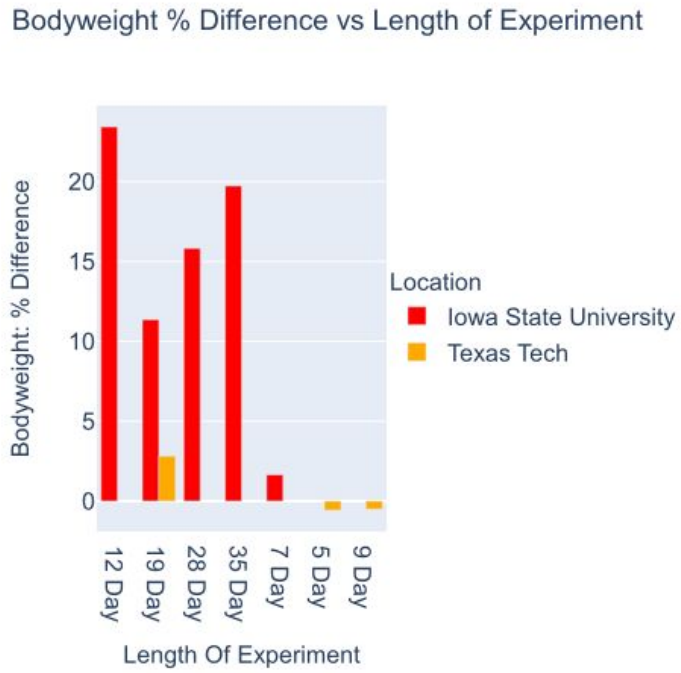
Data groupings can be selected on the bottom-half of the **Graphing Screen**.



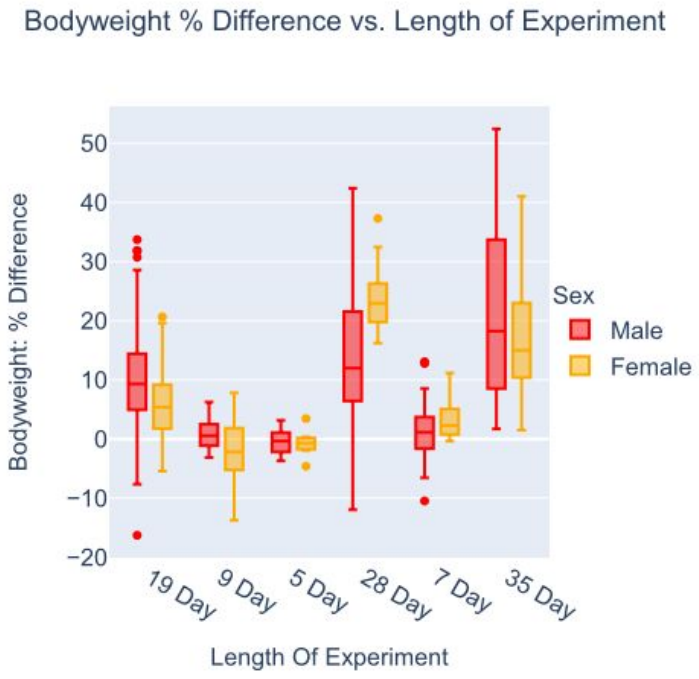
For Box Plots and Bar Graphs, the user will be required to select at least one **Grouping** and at least one **Y-Variable**. For multiple graphs to be generated, select more than one **Grouping** and/or **Y-Variable**.

Optionally, the user can filter the groups shown, as well as select **Sub-Groupings** (for Bar Graphs) and **Color Grouping Variables** (for Box Plots). This will group the bars and boxes by color based on the groupings selected. To generate multiple graphs with different groupings of the bars and boxes, select more than one **Color Grouping Variables** or **Sub-Groupings**.

9.2. Example Bar Graph

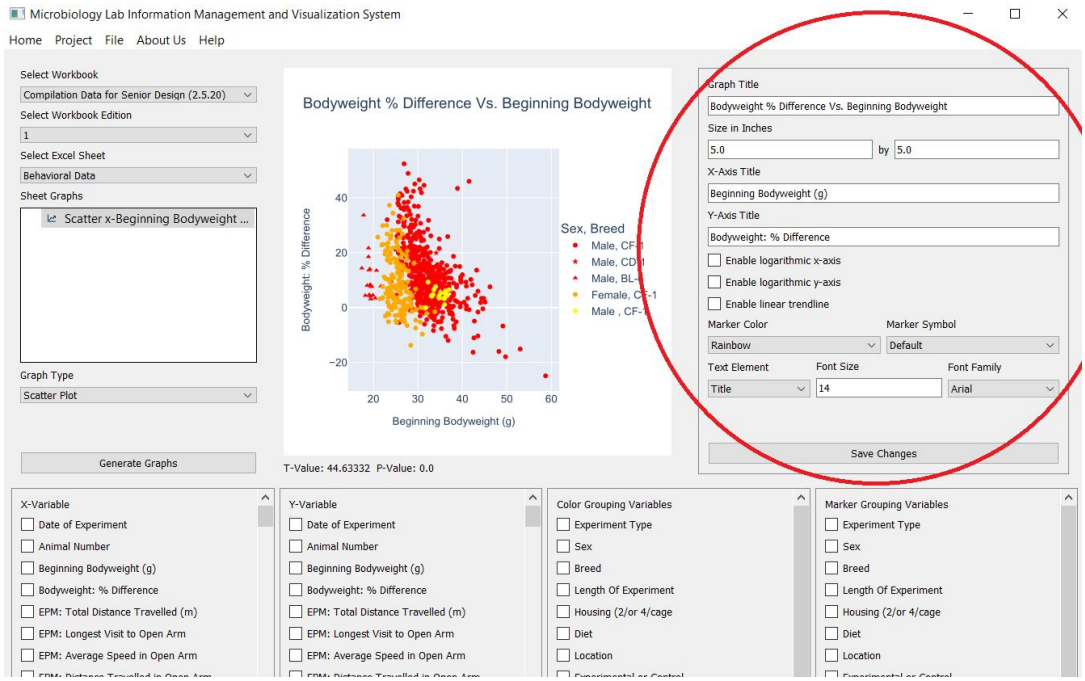


9.3. Example Box Plot



10. Editing Graphs

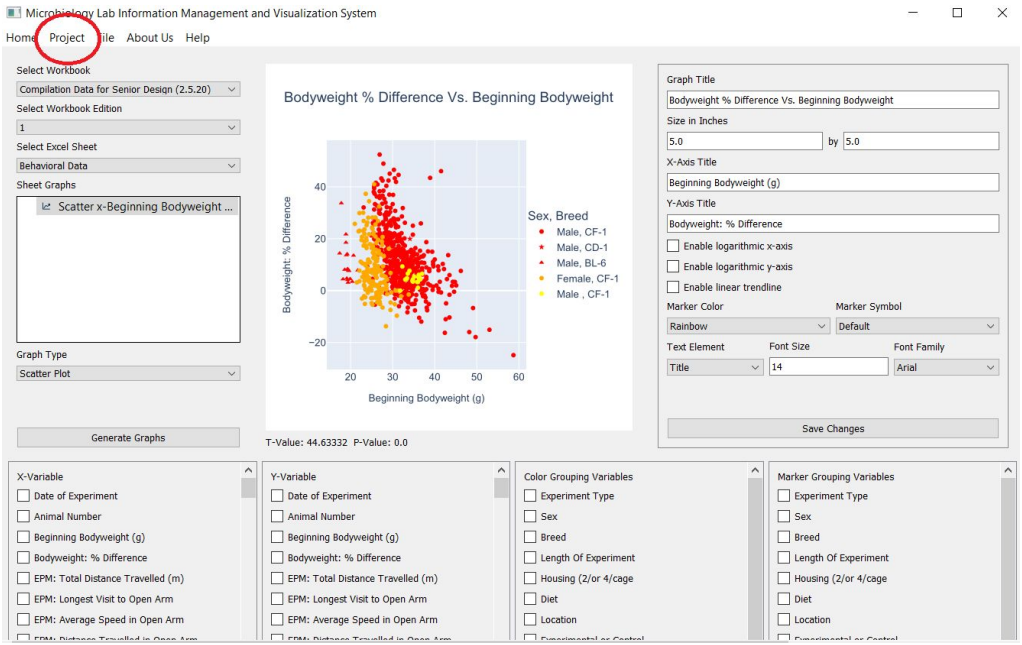
Graphs can be edited after they have been generated. To edit a graph, first select it from the **Sheet Graphs** window. Then, the upper right window of the Graphing Screen will contain information above the settings of the graph.



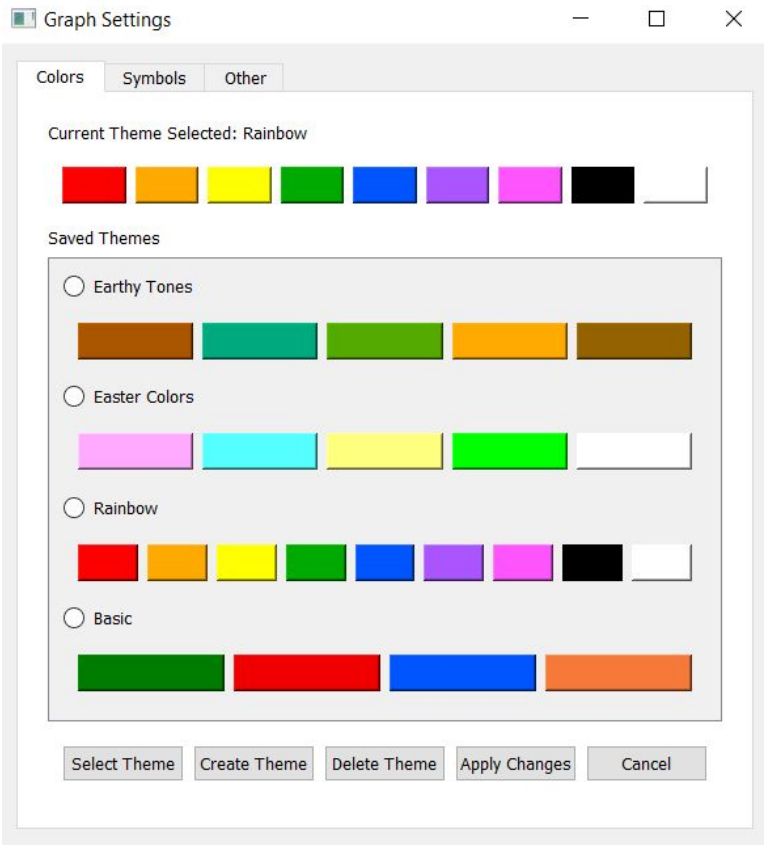
From this window, the graph title, size, x and y-axis labels, x and y-axis scales, marker color, and marker symbol can be changed. Additionally, the x and y-axis ranges can be manually adjusted, and a trendline can be selected to appear on the graph. After making the necessary adjustments, click **Save Changes**.

11. Graph Preferences

To view and/or change the **Graph Preferences**, click on the **File Dropdown Menu** in the upper left of the window.



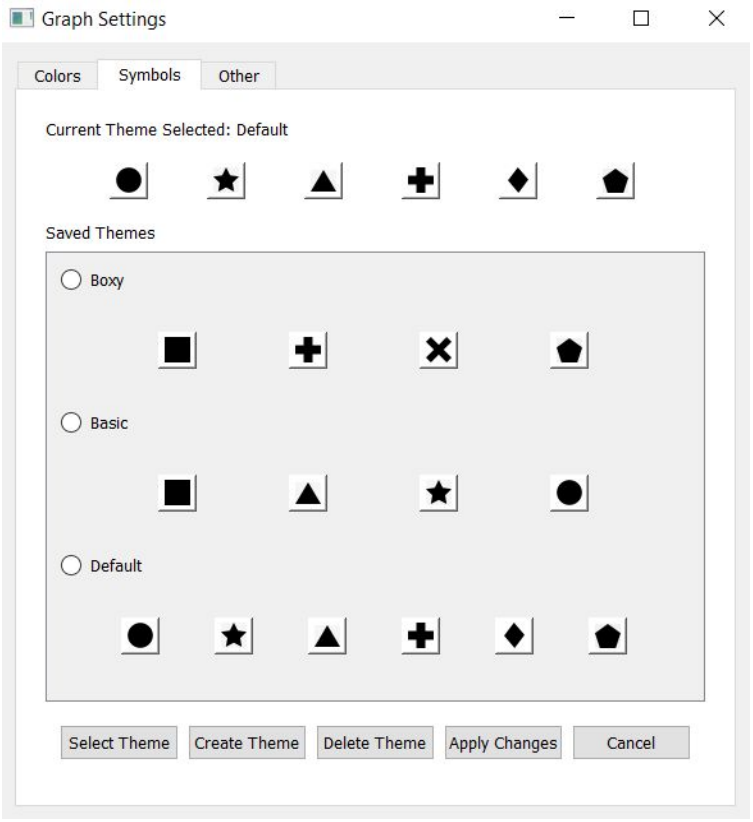
From the dropdown menu, click on **Graph Preferences**. The **Graph Settings Pop-Up** will appear.



On this settings window, there are multiple preferences to view/change: **Color Theme**, **Symbol Theme**, and **Other**. To switch between them, click on the corresponding tab.

The **Color Scheme** represents the colors used in the graphs generated. The tab to change this preference looks like the above picture. If a **Color Grouping Variable** was selected, the graph will rotate through the selected **Color Scheme**. If the **Color Grouping Variable** has more categories than the number of colors in the selected **Color Scheme**, the graph will rotate through all the colors in the selected **Color Scheme** and then start again with the first color in the scheme. Users can select, create, and delete themes on this tab.

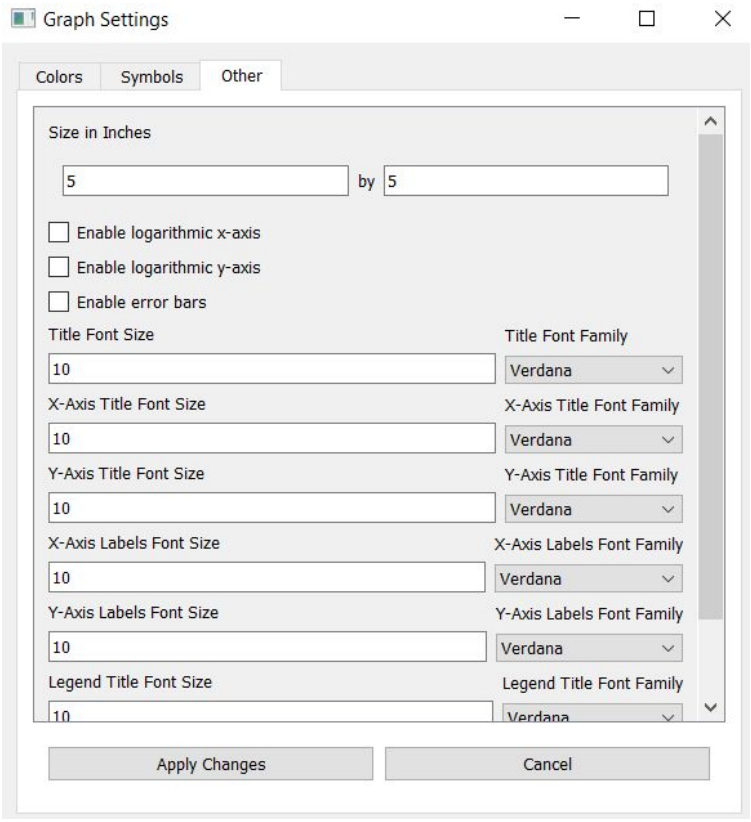
The **Symbol Scheme** represents the marker symbols used in the graphs generated. The tab to change this preference looks like this:



If a **Symbol Grouping Variable** was selected, the graph will rotate through the selected **Symbol Scheme**. If the **Symbol Grouping Variable** has more categories than the number of symbols in the selected **Symbol Scheme**, the graph will rotate through all the symbols in the selected **Symbol Scheme** and then start again with the first symbol in the scheme. Users can select, create, and delete themes on this

tab.

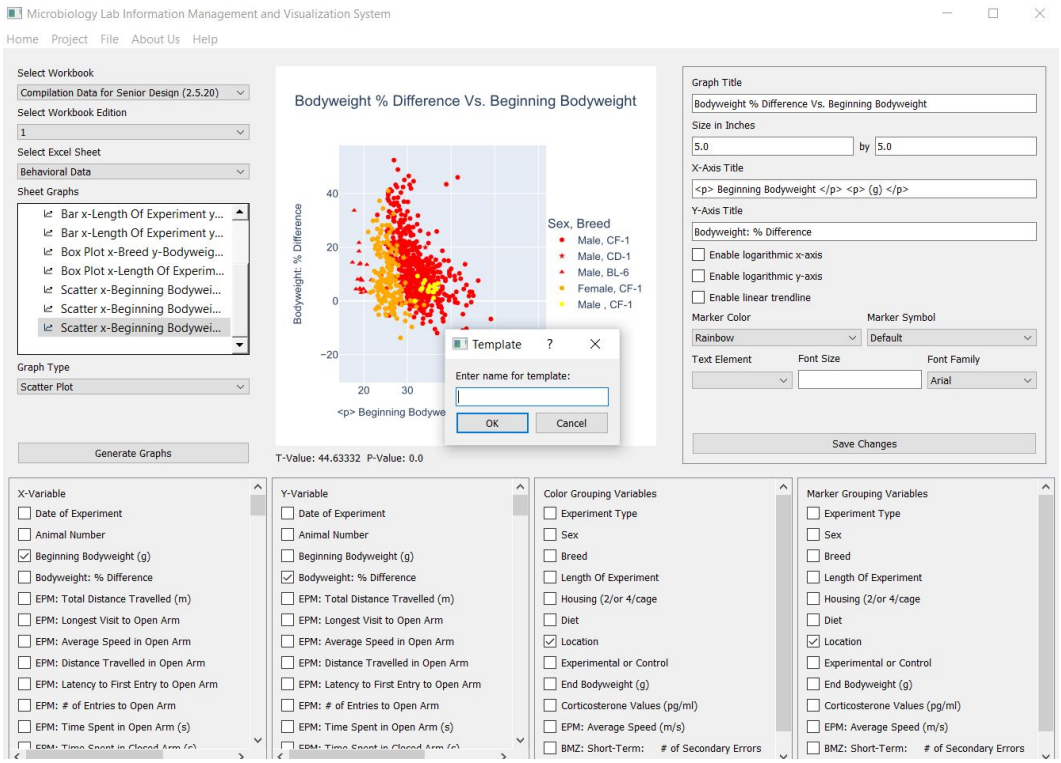
The **Other** tab has additional graph settings, including: the dimensions of the generated graph image, logarithmic capabilities for x and y-axis, error bar capabilities, as well as font Size and Type for the graph title and x and y-axis labels.



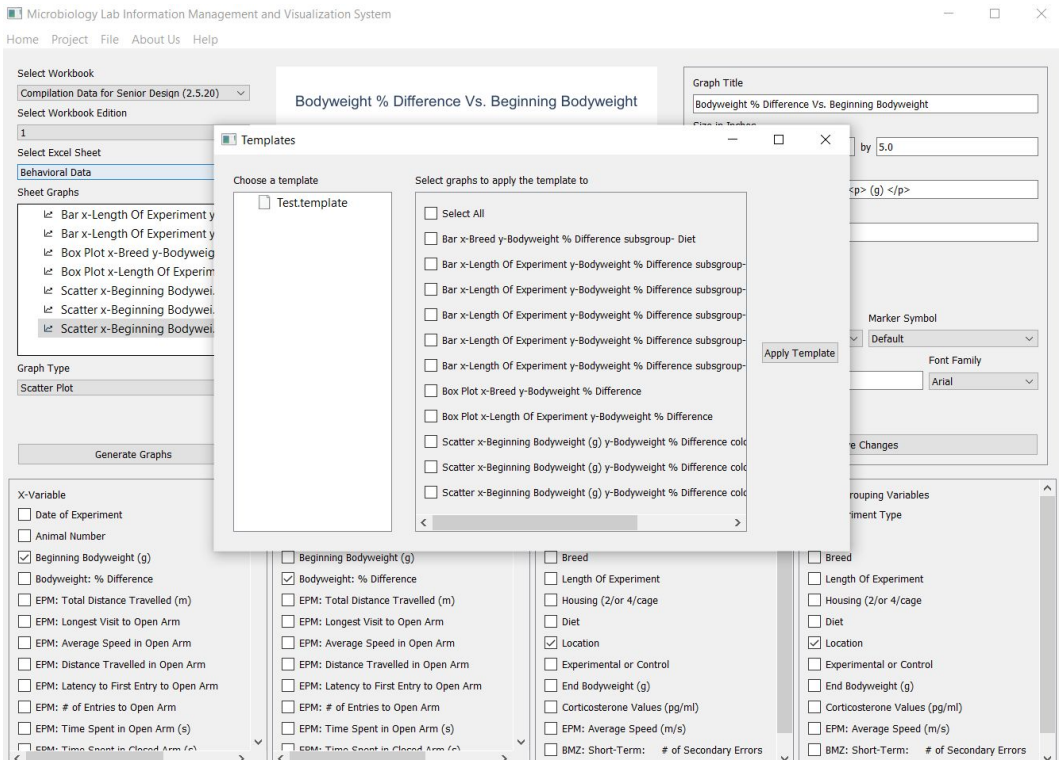
12. Graph Templates

A graph template is used to save the configuration of a graph so that the same configuration can be applied to other graphs later on.

To create a template of a graph, click on a graph file in the **Sheet Graphs** section that was previously generated. When the graph appears, on the **Project Dropdown** select **Create Template**. An alert will appear prompting for a name for the template. Enter a name for the template and select **OK**.

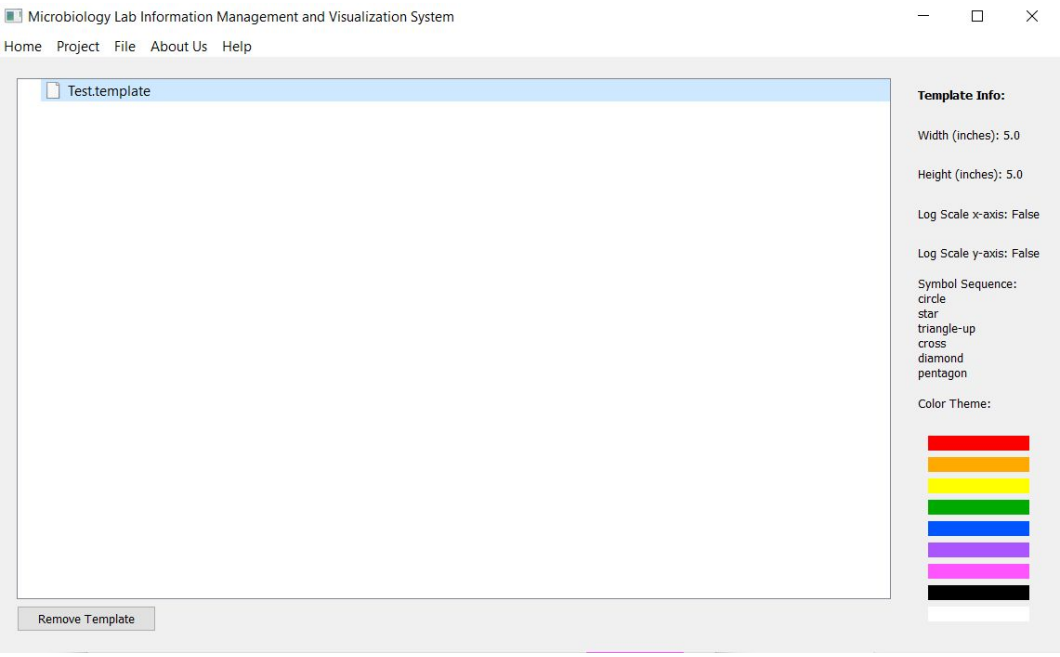


To apply the template to one or multiple graphs, on the **Project Dropdown** select **Apply Template.**



A new window will appear. Select a template from the panel on the left of this window and select the graphs the template will apply to. After **Apply Template** is clicked, the height, width, color theme, shape theme, and the logarithmic scale states of each axis are copied to the graphs that have been selected.

To view the contents of a template and delete templates, click on the **View Templates** button on the **Home Screen**.



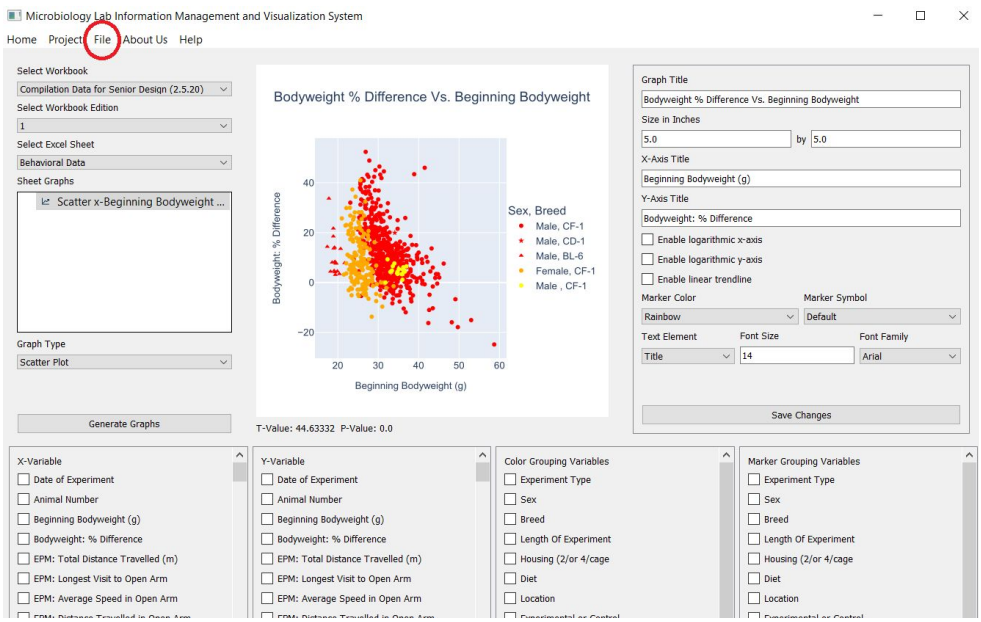
After selecting a template, the contents of it will appear on the right side of the window. The template can also be removed by selecting **Remove Template**.

13. Exporting Graphs

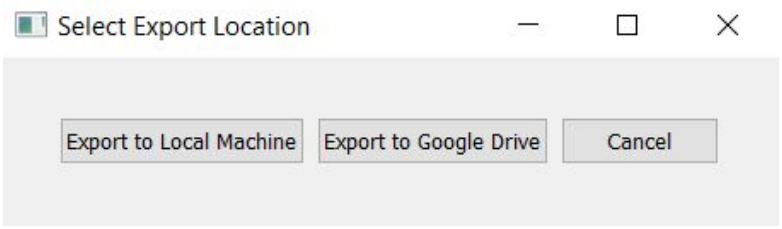
Please note that Exporting will only work if graphs have been generated and the **Sheet Graphs** window is populated.

13.1. Exporting to the Local Machine

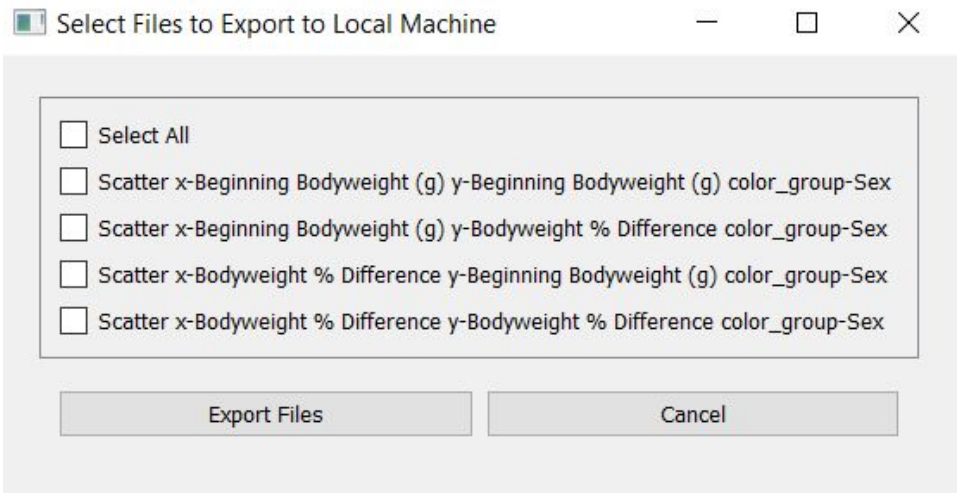
First, click on the **File** tab, and select **Export File(s)**.



A small pop-up window will appear.



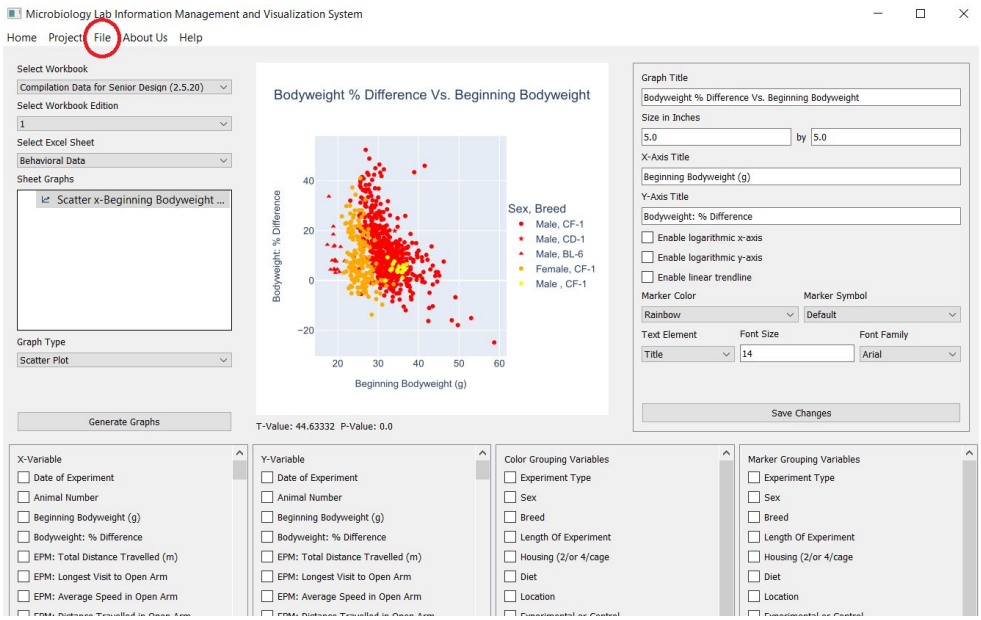
Choose **Export to Local Machine**.



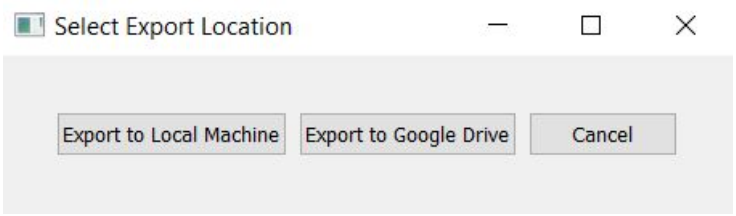
A window populated with the generated graphs will appear and look similar to the above screenshot. Select the files that are to be exported and click **Export Files**. A file system browser will appear. Navigate to the location the files are to be saved, and click **OK**.

13.2. Exporting to Google Drive

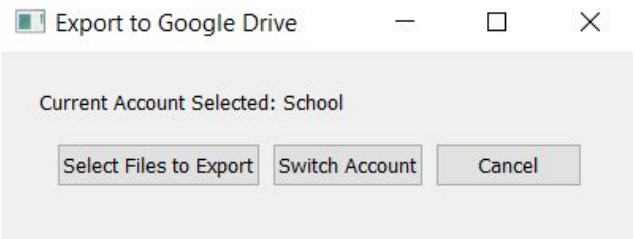
First, click on the **File** tab, and select **Export File(s)**.



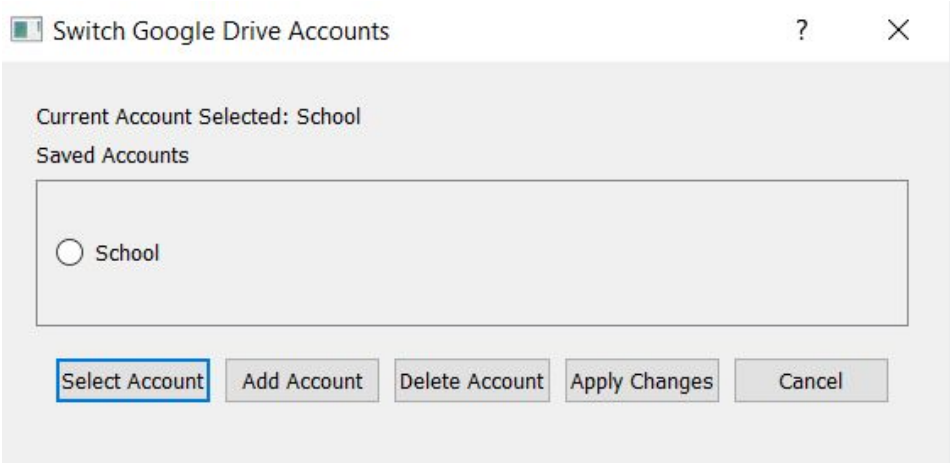
A small pop-up window will appear.



Choose **Export to Google Drive**. The following window will appear.



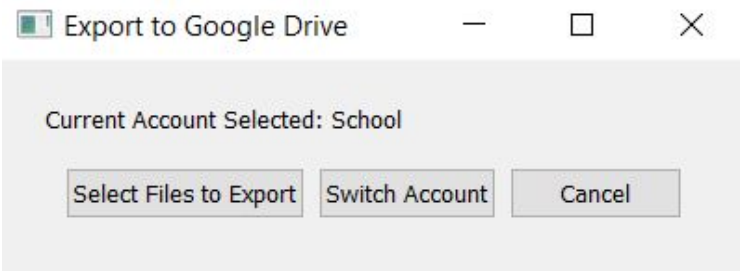
For new users, **None** will be the current account selected. If that is the case, **Switch Account** can be used to add a new Google Drive account. The following window will appear when **Switch Account** is clicked.



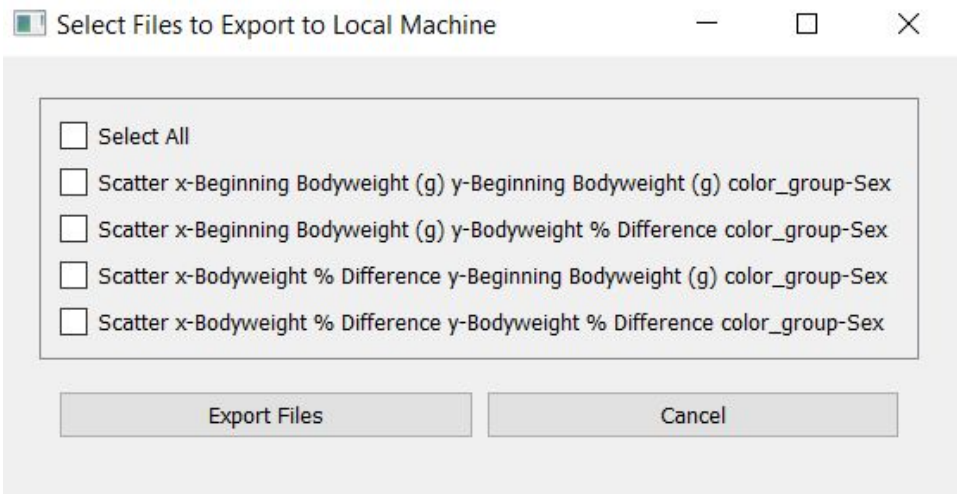
From this window, new Google Drive accounts can be added by clicking on the **Add Account** button. After clicking on the **Add Accoun**t button, go through the Google Drive authentication flow (a browser window will appear). Allow **Quickstart** to have access. There is a 30-second timeout in place, waiting for a successful authentication response. If no response is given within 30 seconds, the process will be cancelled. If an authentication error occurred, please try restarting the **GraphKey** application.

Additionally, Google Drive accounts can be deleted by clicking on the **Delete Account** button.

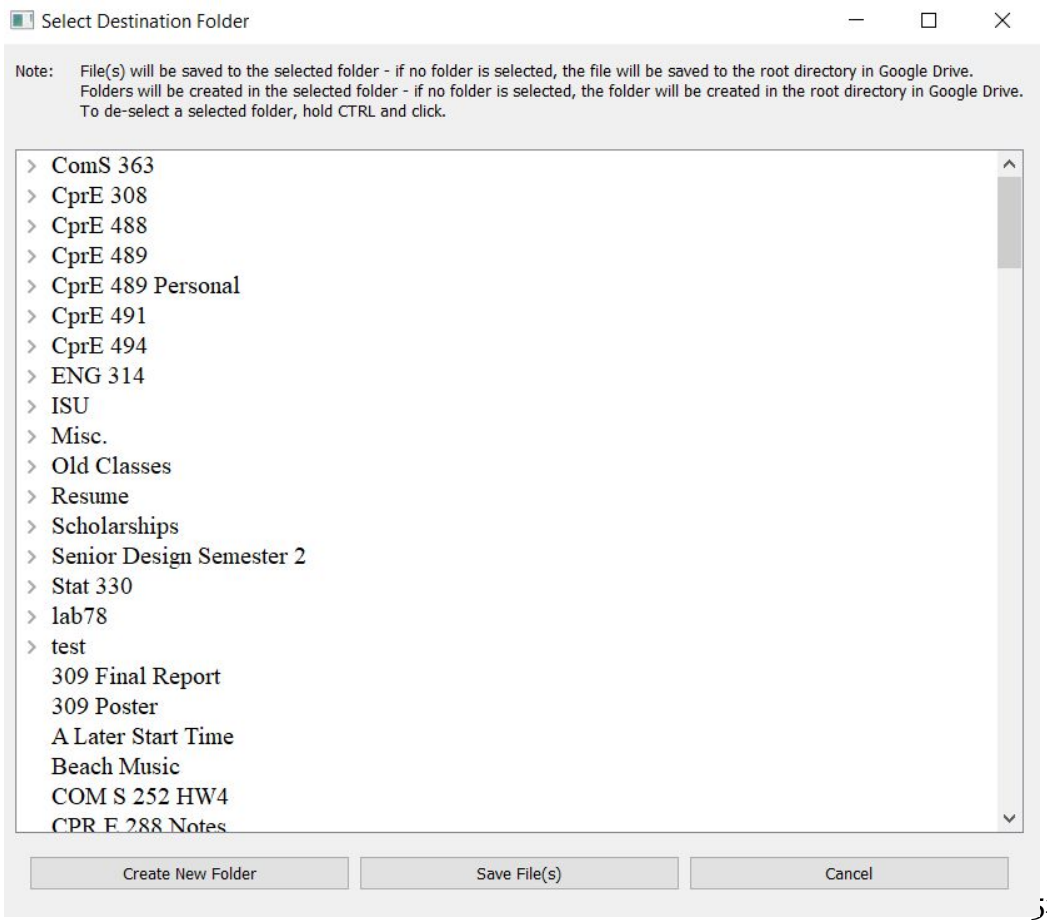
Once at least one Google Drive account is hooked up to **GraphKey**, use the **Select Account** button to select it. Then, click **Apply Changes**. The new Google Drive account should be selected in the following window:



To export files, click on **Select Files to Export**.



A window populated with the generated graphs will appear and look similar to the above screenshot. Select the files that are to be exported and click **Export Files**. The application will now retrieve the file structure of the selected Google Drive account. Depending on how much content is on the Google Drive Account, this may take a while. Eventually, the following window will appear (with the contents of the selected Google Drive of course).



Navigate to the folder the graphs are to be saved to, and click **Save File(s)**.

From the window above, new folders in the Google Drive account can also be created. Navigate to where the new folder should be created, and click **Create New Folder**. Enter in a name at the prompt window and press **OK**. The application will then take some more time to create the new folder and re-retrieve the contents of the selected Google Drive account.

Tip: Hold the **CTRL** button down on the keyboard and click on a selection to unselect it.

14. Appendix

14.1. Reference Links

- [1]Team 29 Senior Design Website: https://sddec20-29.sd.ece.iastate.edu/

  Contains information about our project including presentations, reports, and further links to source code and code documentation. Also contains an executable of the GraphKey application that can be downloaded and run.

- [2]Python Downloads: https://www.python.org/downloads/

  The Python website where Python can be downloaded and installed. Python is required for our application to run.

14.2. About Us

We are the 29th Senior Design team of the graduating class of Fall 2020. Our group is made up of Software Engineering students as well as one Computer Engineering student. Please go to https://sddec20-29.sd.ece.iastate.edu/team.html for more information.

To contact us, please email us at sddec20-29@iastate.edu.

Copy of Design Document from Last Semester

(Please Use the Table of Contents on Page 6)

Microbiology Lab Information Management and Visualization System

DESIGN DOCUMENT

Team 29

Client: Karrie Daniels
Adviser: Thomas Daniels

Brittany McPeek: Report/Documentation Manager
Benjamin Vogel: Team Manager
Rob Reinhard: Meeting Facilitator
Kyle Gansen: Meeting Scribe
Ben Alexander: Progress Manager
Samuel Jungman: Chief Engineer

Team Email: sddec20-29@iastate.edu
Team Website: https:/sddec20-29.sd.ece.iastate.edu

Revised: 4/26/2020

Executive Summary

Development Standards & Practices Used

- Detailed Use Case diagrams and User Stories
- Simple Design
- Pair Programming
- Heavy documentation in both code and manuals, updated regularly
- Single coding standard (Will use PEP-8)
- Frequent refactoring
- Simple APIs
- Don't write code you might need in the future, but don't need yet
- Construct and maintain a user manual for non-technical clients

Summary of Requirements

- System should support importation of large amounts of data in a reasonable amount of time from sources such as CSV and Excel
- System should be able to abstract the data and place them into modifiable graphs
- System should support exporting those graphs to be published into research papers
- System should be able to maintain past data and support modifications and the addition of new data
- System will be written in Python
- System should be easy to understand and use by users with little-to-no background in programming
- System should be maintainable by 1 to 2 IT personnel

Applicable Courses from Iowa State University Curriculum

- COMS 309

- COMS 227
- COMS 228
- COMS 363

New Skills/Knowledge acquired that was not taught in courses

- Graphing and graph visualization
- Python
- Data management/backup

1 Introduction

1.1 Acknowledgement

Our group would like to acknowledge and thank Thomas Daniels for his guidance, support, and technical advice throughout this project. Our group would also like to acknowledge and thank Karrie Daniels for providing us information and requirements for this project, as well as acting as a sample user of our end product. We appreciate the time and commitment these two individuals have donated towards this project.

1.2 Problem and Project Statement

Many scientists and researchers dedicate large amounts of time towards organizing, maintaining, and visualizing the data they collect. The purpose of this project is to find a solution to this problem. The solution should be able to automate the process of organizing, maintaining, and visualizing data. It is important that scientists and researchers have more time to collect and analyze their data, especially in time-sensitive experiments; thus resolving the issue of organizing, maintaining, and visualizing their data will be beneficial to scientists and researchers.

Our group proposes creating an application that allows the user to import pre-existing data from Excel and manages and visualizes the data. The application will allow users to select data elements and a type of graph/statistical analysis and visualize the resulting information in the form of a graph or another type of visual. The graphing utilities will allow the user to customize the appearance of the graphs to be created and will meet scientific publication standards. Additionally, the application will save backups of the data and allow the data and visuals to be exported and shared with another person. Our hope is to create an easy-to-use application that does not require too much maintenance and allows scientists and researchers an easier method to organize, maintain, and visualize their data.

1.3 Operational Environment

Our end product will be only software based. The software's environment is any computer that it runs on. Our end users will most likely be in a climate-controlled, indoor location which means that a computer will be able to operate without any problems.

1.4 Requirements

Our client wants to be able to use this software to log and manage data about microbiology experiments. The following is a list of functional requirements that the client wants:

- Data import from Excel
- Data modification after import
- Custom data visualization based on specified data elements
- Data sets and graphs should be saved to the file system
- Data should be backed-up in a separate location
- Function to export the data and be shared with someone else

- Creation of projects that contain multiple experiments
    - Collation of multiple graphs from similarly based experiments

Our non-functional requirements are:

- Possible for the system to be maintained by one or two people
- Secure enough so that research data can't be seen by anyone else
- Use libraries in Python for data visualization
- Data must be parsed after it is imported

1.5 Intended Users and Uses

This project's end product is intended to be used by scientists and researchers for inputting, organizing, and visualizing large amounts of data efficiently and effectively. The visualizations created by the product should meet scientific publication standards so that these visualizations can be used in published reports, papers, etc. Non-technical persons should be able to utilize the end product with ease. Many of these users are not particularly 'tech-savvy' and are not accustomed to complicated and unintuitive software. This experience places high importance in the intuition and cleanliness of the UI of our software. Furthermore, this software will not have a real maintainer or IT team after it is delivered, so the project will need to be clean and bug free to match the lack of maintenance staff.

1.6 Assumptions and Limitations

Assumptions:

- The maximum number of users per instance of the product will be one
    - The client prefers the data to be modifiable by one person, with additions of simultaneous editors as a stretch goal. Given the size and complexity of the data, having simultaneous users modifying and graphing data from one set could prove to be a daunting task.
- The solution will not be distributed outside of Iowa State
    - Due to the specific solution the end product brings to the research department, the use of the product (at least in its early stages of development) will not be useful outside of the university.
- Python and the Python Interpreter will be used for the development
    - By the request of the client, using Python with the product should provide easy maintenance if needed by people who may be unfamiliar with the development of the product itself. This added with powerful graphing utilities can provide a solid solution to the client's problem
- The end user will require an instruction set about the end product
    - Due to the technical knowledge of the end users, a simplified instruction manual will be required to help convey the usages and information needed to operate the solution.

Limitations:

- The end product shall be able to be maintained by 1-2 IT workers on a minimal basis (Client Requirement)
- The end product shall cost the end user no more than $200 (less than cost of current client's solution)
- The end product shall be easy to run and navigate with little-to-no programming experience (Client Requirement)
- The end product shall be based on, and released for, the ISU research department (Geographical Constraint)

1.7 Expected End Product and Deliverables

The final product will parse excel files and output graphs based on the data provided. The end-user will also be able to customize these graph layouts as well as select the variables in the graphs so the project's use extends outside of the provided data-set. Finally, the end product's components will be decoupled to allow further customization if the needs of the end-product evolve following delivery (i.e. new types of data or graphs are needed).

The design of the project will be completed by the end of April, and the end product and documentation will be finalized by the end of December 2020. Documentation code will be ongoing as we meet each date. A look below shows a list of our deliverables over the course of the next two semesters.

Table 1: Projected deadlines for the project

ID	Date	Deliverable
D1	April 30th	Framework for each component of our project will have been created.
D2	September 30th	Prototype of each component of our project.
D3	October 15th	Functional Product
D4	November 15th	Final Product
D5	November 30th	Testing
D6	Ongoing	Documentation for above deliverables

Table 2: A description of the deliverables for the project

ID	Description
D1	By this point, "Hello World" programs will be built on the technological stack we have chosen. In addition, the preliminary frameworks for importation, graphing, exportation, and the GUI binding them together will have been created.
D2	Essential features of each component will have been implemented. Data can be imported from an Excel file. Then, you will be able to select data to create at least one type of graph. The graph will generate, and then you will be able to adjust the graph if needed. Finally, you will be able to export a high-quality image of the graph. In addition, data will be saved.
D3	Each of the core features will be fully implemented. Every type of graph requested will be available for creation, and the importation and exportation settings will be available. More types of files will be available for importation and exportation. In addition, backup snapshots will be implemented in case data is lost. Finally, the product will be able to calculate correlation and perform T-Tests and P-Tests from the graphs.
D4	Modifications requested by our clients looking at the functional product will be taken into account, and stretch-goals, if possible, will be implemented. Bugs that occur in normal boundaries will be squashed.
D5	While testing will occur as we develop the product to ensure proper functionality in normal circumstances, these two weeks the testing will be done in attempts to "break the product." This way, our final product we created for the client will continue to properly function even in scenarios we didn't account for.
D6	Two types of documentation will be provided: one for the end-user, and one documenting the code behind our end-product. The end-user documentation will consist of tutorials for the end-product. The code documentation will provide a detailed look at all classes and functions making up the back-end. The documentation will also outline how these classes and functions relate to each other. This will be demonstrated by providing "under-the-hood" looks at the tutorials in the end-user documentation. Finally, the documentation will show how classes and functions can be extended.

2. Specifications and Analysis

2.1 Proposed Approach

Our proposed approach is to create a local application using Python. The PEP-8 style guide for Python Code will be used during development. The user will interact with the program through a GUI made using the PyQt5 Python library. The various functions of the application will then be split into components which interact with each other through interfaces. This should allow for independent development of the various components of the software. A figure showing how these components relate to each other is available in section 2.4.

The design addresses the functional and non-functional requirements from section 1.4 in the following ways:

Functional Requirements:

- Data import and parsing from Excel will be handled through a data import component of the software. Python has a module “pandas” which can be used to extract data from Excel spreadsheets.
- Data modification after import will be handled through a data holding component of the software. Python also has a module “xlutils” which can be used to modify an existing Excel sheet.
- Custom data visualization of the data will be handled through data visualization and graphing components of the software. The open source Python library “Plotly” is one potential library to use for these components.
- The graphs and data will be saved through a save files component of the software. Plotly allows static images of plots to be saved, and the data can be saved to new Excel files using the “xlwt” Python module.
- Data will be backed up to a separate location through a data backup component of the software. After data has been imported into the software, it can be written to a file in a separate location.
- A share files component of the software will allow for files created by the application to be uploaded to Google drive.

Non-Functional Requirements:

- To ensure the system could be maintained by one or two people, a local application approach was chosen. Excluding a database from the design of the software eliminates the need for database maintenance.
- The research data will be secure due to the software being a local application.
- It was requested that Python libraries be used to visualize the data, and the Python library “Plotly” is robust enough to cover the functional requirements.
- The various Excel modules mentioned in the functional requirements allow for parsing the data after it has been imported.

2.2 Design Analysis

One of the strengths of the design in theory is the modularity. Breaking the software into separate components should allow for team members to work fairly independently of each other on the software. Another strength of the design is that it allows for new functionality to be added to the software without having to overhaul the design. New functions can be implemented as new components in the software.

One potential weakness is that all processing of the data will be done locally. The speed at which this occurs will depend on the processing power of the computer running the application, whereas a client-server application could allow for processing of data on the server side.

2.3 Development Process

The development process will be an agile based development. We will use 1-2 week iterations to produce predefined goals. These goals will be written out in the form of story cards that will encapsulate use cases with diagrams to allow for easy to follow documentation and maintenance after the project is delivered. We will use a Kanban board of some kind to track the progress of story cards. The reasoning behind this process is that it will allow us constant feedback from our client as to the specifications and implementations of the project. This is a project that will require input as the design and intuition of the UI has high importance in the overall quality of the final deliverable.

2.4 Conceptual Sketch

For each element in the diagram we would have an interface that abstracts each component and thus, simplify how the elements interact. The benefits from using interfaces here is better collaboration. Each team member doesn't have to know everything about another component in order to use it, they can just use the interface.

Figure 1: Conceptual sketch of the application and interactions between modules

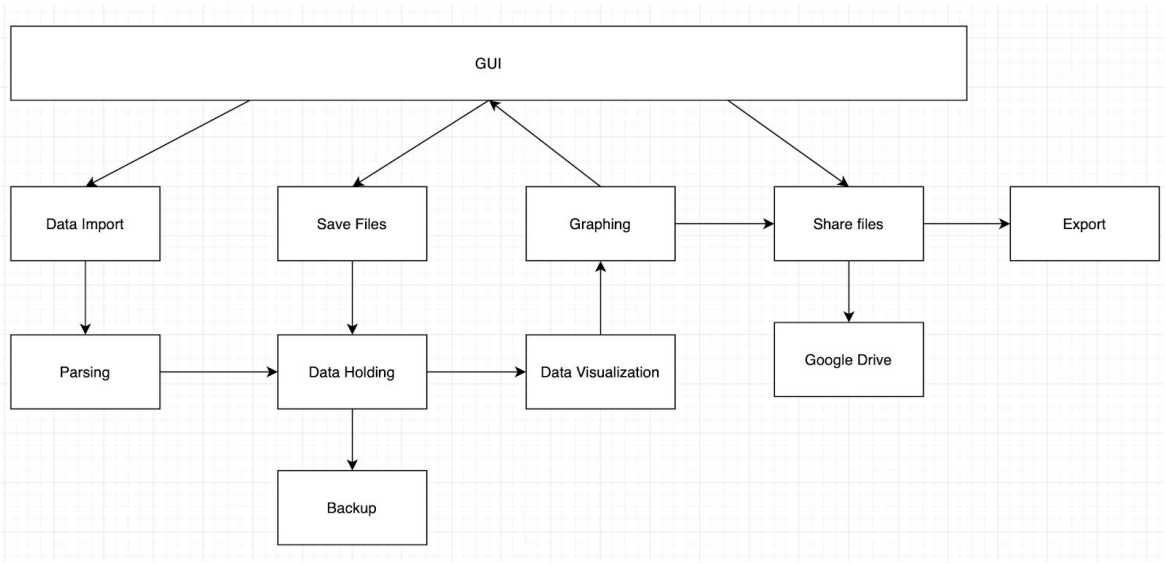
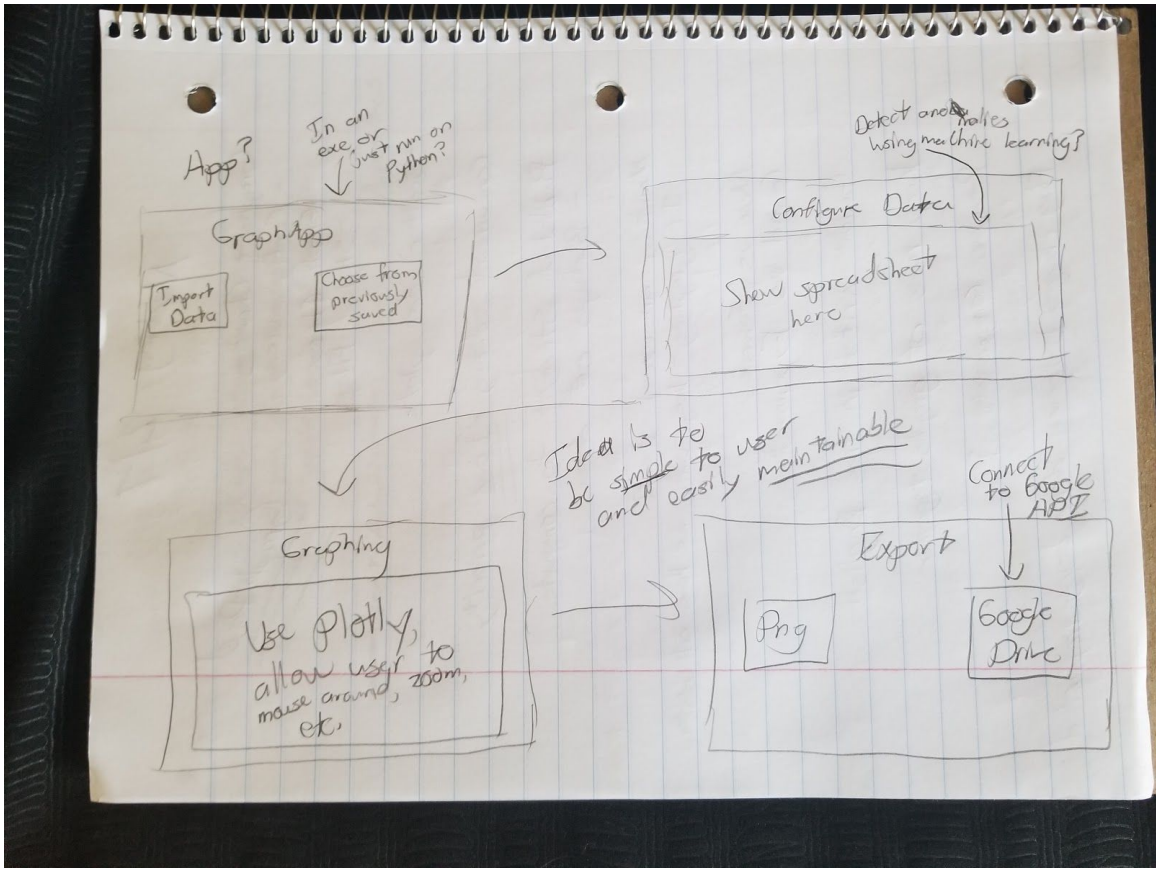


Figure 2: Conceptual sketch of the user interaction of the UI


Apgp?
In an exe, or just run on Python?
GraphApgp
Import Data
Choose from previously saved
Detect anomalies using machine learning?
Configure Data
Show spreadsheet here
Graphing
Use Plotly, allow user to mouse around, zoom, etc.
Idea is to be simple to user and easily maintainable
Export
Connect to Google API
Png
Google Drive

3. Statement of Work

3.1 Previous Work And Literature

The previous work we are basing ourselves off of is the solution our client is currently using, GraphPad. This technology is designed specifically to take data and organize it similarly to a spreadsheet and then provide graphing utilities to help visualize the data. GraphPad itself does not operate data entry as a spreadsheet, but a more specialized version where they offer special data tables that can be catered to how the client wishes to organize.

One of the current major downsides to this technology is the price. GraphPad can be extremely expensive on a yearly subscription, especially when more than one person needs to have a license for it [1]. The other downside that the client has told us is the lack of options for a robust suite of graphs. Currently, GraphPad only can visualize with bar graphs, which the client may use, but would rather work with more varied graphs such as scatter dot plot graphs and correlation graphs.

3.2 Technology Considerations

The possible solution we are bringing to the table would provide a free variant to GraphPad. It would handle data in a similar way since that was valued by the client, but also utilize the power of Plotly (a Python library specialized with advanced graphing utilities) to provide a wider variety of graphs to use.

One strength of this solution would be the ability to use a free, open-source library on top of a free, open-source language and interpreter, which will be able to drive costs down on the development, which can transfer over to the user. Python has a massive amount of packages that can be easily installed and incorporated into our project to help us not only provide a robust product for the client, but relieve a lot of the behind the scenes work for the developers (for example, importing and parsing large amounts of data). Another strength of this solution is due to the nature of Python and our structure, we will be able to make an end product that should be much easier to maintain and fix for people who are not as experienced in software development. By making the front end and the back end as accessible as possible for our end clients, we can ensure that the product will be able to live beyond the Senior Design class.

A downside to this possible solution is the nature of Python itself. Because Python runs on an interpreter, creating an EXE or application will be harder. There are packages and libraries to help with the process, but it just doesn't work as naturally as some other languages or solutions might. Another downside is the current solution is a local solution. The client, while making it a low-priority stretch goal, wanted to have simultaneous users. By keeping, modifying, and visualizing the data on a local machine, it will make a solution with simultaneous users difficult if not impossible.

3.3 Task Decomposition

Our tasks can be decomposed into the modules similar to the structure of our solution in Figure 1:

1. Graphical User Interface
    a. Styling
        i. Create a visually appealing front end that also shows all relevant data
        ii. User should be able to edit styles to their own liking

b. Layout

i. Layout should be easily understandable by the end user

ii. Graphing should be the primary focus of the layout

2. Data Import and Parsing

a. The client should be able to import CSV or EXCEL files to be analyzed and graphed

b. The solution should be able to parse data from the files and sort them into data structures for better visualization options

3. Data Visualization/Graphing

a. Create a system that integrates with Plotly to create and show graphs to the end user

b. The system should be able to handle the following types of graphs and computations: dots plots, scatter plots, bar graphs, t-test plots, p-value computation, and correlation computation

c. The graphs should be customizable in regards to the types of graphs, which variables are graphed, what color schemes are used, and what data point shapes are used

4. Saving Files

a. The user should be able to save a current file within the application

b. The system should store backups of the imported data in a hidden folder for data recovery

5. Sharing Files

a. Create an export tool that can either share to Google Docs through an external API or save it as a picture to their local machine.

3.4 Possible Risks And Risk Management

Unfamiliarity with both Python and some of the needed packages will become a possible risk for the group, though we are already taking steps to remedy the problem with research and practice before beginning on developing the actual solution.

In order to avoid most of the risks that come from developing this solution, we will be communicating with one other and the client frequently in order to sort out misunderstandings and confusion as fast as possible. By doing so, we are able to mitigate risks faster and more effectively as they come up.

3.5 Project Proposed Milestones and Evaluation Criteria

The key milestones in our proposed project are the framework, prototype, the functional product, and final product stages.

The framework stage is a key milestone because it will require us to build a theoretical model to implement. Since each of us are focusing on individual components of the project, the frameworks section is also critical because it will require us to have discussions on how data is passed between each component. The test for our frameworks are not programmatic, the test is that the frameworks we have built handle a consistent type of data and could work together.

The prototype stage is a key milestone because it will be the first time a product is delivered. By implementing core-features with the frameworks laid out in the previous stage, we will have our first demonstration of the product's components not only working, but working together. Testing will be done by passing in data through each component in a few key scenarios.

The functional product stage is a key milestone because it will be an implementation of all features we have been asked to deliver. Testing for these phases are done similar to the prototype stage. Using the same data, we test the features of that component, and then test the passing of the data to other components.

Finally, the final product stage is a key milestone because it will be a final tailoring of what we presented in the functional product stage for our client. Feedback will be taken into consideration, and stretch goals, if possible will be met. Testing is now for small adjustments, and similar to the functional product stage.

3.6 Project Tracking Procedures

Our group is currently using Trello to manage and track progress for this and next semester. We also meet with the client regularly to discuss our progress and gain feedback so we know what portions of the project need correction and to narrow our next steps.

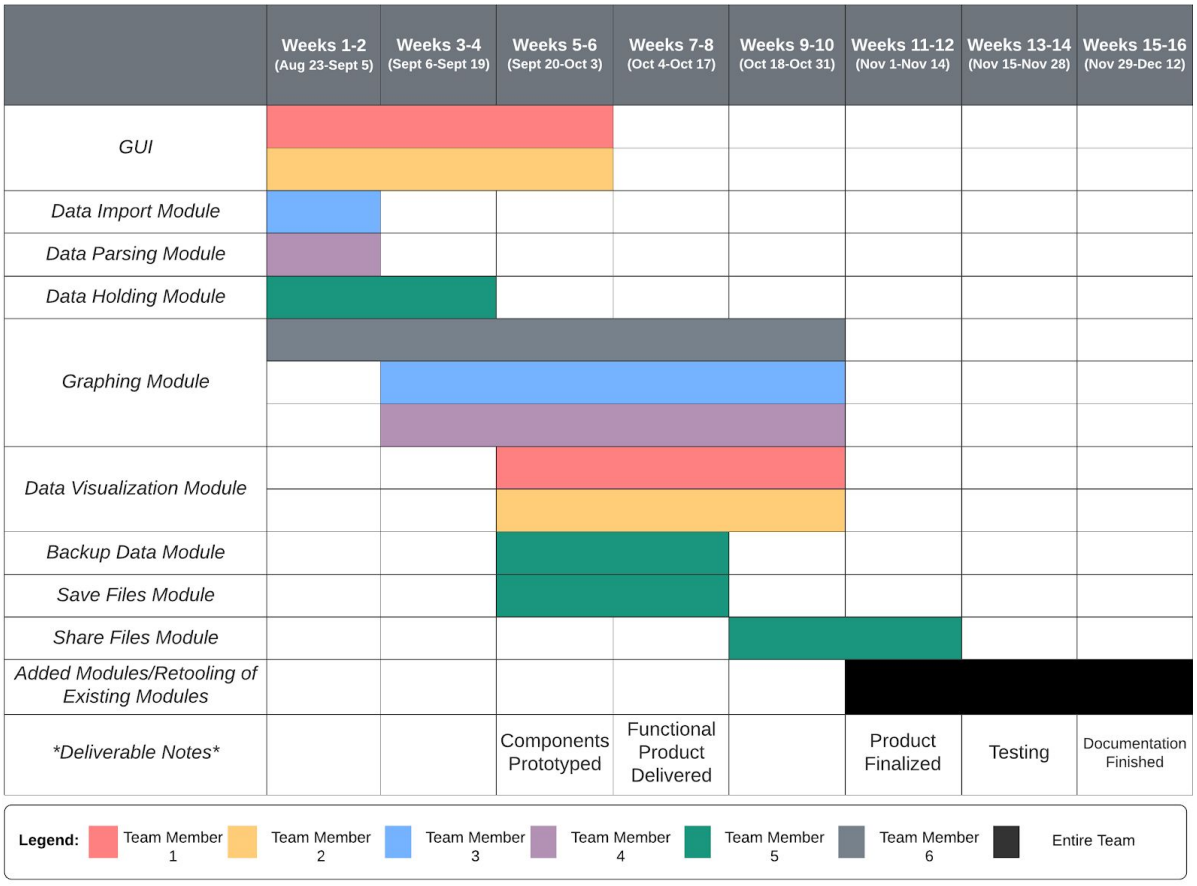
3.7 Expected Results and Validation

The desired outcome is a program that encompasses all of the tasks layed out in 3.3. Each of these tasks must behave properly within themselves as well as interact properly with one another. This can be verified at a **high level** using full integration tests in which individuals separate from our team will be asked to perform various tasks in a black box testing fashion that incorporate all of our key milestones and features that have been laid out in this document. It should be noted that this testing will occur only after rigorous Moq testing and unit testing on a low level as well as small scale integration testing at various points of development.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

Figure 3: Project Timeline



Work on the project on this proposed timeline begins with the GUI, data import module, data parsing module, data holding module, and graph module. These components set the groundwork for the project. For the future modules to be functional in the scope of the program, it is necessary that these are finished first. The GUI is vital for this project as it is the only way for a user to interact with the modules, so two team members are set aside for work on it for the first six weeks of the fall.

After significant progress has been made on the GUI and data can be imported into the program, the majority of the team is scheduled to work on the graph and data visualization modules. Because these are the key features of the programs, the graphing and data visualization modules will have five of the team's six members dedicated to working on them. After the product has been finalized, rigorous testing will be performed on the product. Testing will have already occured throughout development, but this portion of the testing will be done to ensure the finished product is airtight.

4.2 Feasibility Assessment

The project will be a standalone executable with its own files kept in the program location. It will encapsulate and allow for all of the main features previously discussed in this document. However, there

are some foreseen challenges with this predicted project that have been laid out in the following paragraph as well as some preliminary plans to relive these challenges and improve the feasibility of this project.

The first main challenge will be the formatting and creation of the graphs themselves. Graphs will require some statistical knowledge and practice to be able to generate the graphs from the data. There will need to be some team research to accomplish this aspect of the project. The second is the variety of Python libraries that will be needed to complete this project. Due to the immensity of Python and its libraries, our group has varying experience with Python and certain libraries and will need to conduct experiments and research to learn how to develop the project using these libraries. These present themselves as the main foreseen challenges of our project which is why this project seems to be well within the realm of feasible.

4.3 Personnel Effort Requirements

Table 3: Personal Effort Requirements

Task	**Description**	**Time Estimate**	**Difficulty Estimate** (1 = Easy, 5 = very hard)
GUI Styling	Create a desktop application that is aesthetically appealing and intuitive.	15 hours over 2 weeks	2
GUI Layout	The layout should be easy to navigate and the app functions easy to understand	15 hours over 2 weeks	2
Data Import	Import data into the app from an excel or csv file	10 hours over 2 weeks	1
Parsing	Analyze the imported data to find and sort relevant variables and measurements	25 hours over 3 weeks	3
Data Holding	The parsed data should be temporarily kept in the program to be used later	5 hours over 1 week	2

Saving Files	Parsed data will be saved to the local storage in a csv file	10 hours over 2 weeks	1
Backup	Versions of program files will be backed up in the file system once every set time interval	15 hours over 2 weeks	3
Data Visualization	Use Plotly to create various kinds of graphs. User can customize data inputs, colors and shapes of data points	40 hours over 4-5 weeks	4
Sharing Files	Export files to Google Drive through API to share with other users	20 hours over 2 weeks	2

4.4 Other Resource Requirements

This project will be completed entirely using Python and Python libraries such as Plotly. This means that the project will not require any physical materials or equipment, other than the computers we use to complete the project. The end product, an application, should not take up much space, and will mostly likely end up around 10MB. Furthermore, the application will not require a very powerful computer to run. This means no additional equipment will be required for our client to use the product since the client has computers available to them in their lab.

4.5 Financial Requirements

This project will only require the use of Python and some Python libraries. A Python install is completely free, and the libraries we will use, such as Plotly, are free and open-source. We also plan on using our own PCs to complete the project, so there is no additional cost in that regard. The client will not need to buy any special equipment to use our end product, as their computers in their lab will suffice. Therefore our project will not require any financial resources.

5. Testing and Implementation

5.1 Interface Specifications

Our project is primarily software in nature with no real hardware to interface with. However, there is some software interfacing that will need to be tested, both from our own creation and from the libraries we opted to utilize for this project.

The first software library we will be interfacing with is Plotly, a graphing based module which will be the engine of our graph generation. We will need to do rigorous input and data manipulation testing with the module to make sure that it can handle the large and complex amounts of data that will be inputted to the application at one time, as well as provide enough options and flexibility to let the user define their own constraints on the graph being produced.

The other software library we will be interfacing with is pandas, a data analysis and manipulation tool for Python, and will be our primary module behind data importation. This interface should be able to handle wrong files, large files, different types of files (as discussed above and in Lightning Talk 3), while also providing a unified and correct output so as to keep the logic between importing data and manipulating/graphing the data as simple as possible.

We will also be utilizing the Google Drive API to export graphing, which will require us to test not only the API, but the module we will be writing to connect the user to the API, which includes providing boundary and exception testing within our module and its connection to other modules within the system.

5.2 Hardware and software

Since our project does not include any hardware, this section will focus on only software we will leverage for our testing.

- PyCharm and Python's unit testing: PyCharm (a python IDE) and Python's own unit testing library will provide a sufficient enough software platform to create and run robust unit testing required for each of the modules within our project
- behave[2] is a Behavior Driven Development module within Python that will be a powerful tool within integration testing. It takes documents written in Gherkin[3] (which is a plain-english language made up of Given-When-Then statements) and translates them into tests to run. This will allow us to create easy to read and understandable tests for anyone not familiar with the project which will be vital as we are not the ones to maintain the code after we graduate.

5.3 Functional Testing

Functional testing will play a critical role in determining the success of our project. The following is an overview of the types of different functional tests that will be conducted for this project.

5.3.1 Unit Tests

For unit testing, the acceptance criteria is that ALL tests pass in order for the module to be considered "valid" and be integrated to the main branch of the repo and included in the project

NOTE: The testing plan will evolve and expand over time as things become more stable. This is the testing plan we have currently with the modules we have planned.

Table 4: Unit Testing Plan

Module Under Unit Test	**Testing Plan**
Data Importing (this will be done for each individual file module)	Test for "gold value" (normal behavior) Test when given no file path (NULL value) Test when given an invalid path Test when given a path to wrong type of file Test when given an empty file Test when given a directory Test when data within file is invalid
Data Visualization (Graphing, Plotly)	Test for "gold value" (normal behavior) Test when given a NULL value Test when given a non-supported type (not a pandas data type) Test when data has "none" type Test when conflicting plots are selected
Data Export	Test for "gold value" (normal behavior) Test for when given a NULL value Test for when given a non-Plotly graph Test for when user asks for a non-supported file type to export to Test for when user asks for an upscaled resolution Test for when user provides invalid Google Drive credentials

5.3.2 Integration Tests

For integration testing, we will utilize behave and the power it has to test the interactions between the two

modules. The project itself will not be considered a stable release until it has passed ALL the integration tests.

An example of an integration test using behave and Gherkin is as follows:

Given the user provides a valid .csv file

And has selected the “bar graph” option

When the user clicks “export to .png”

Then the system will create a .png file for the user

This will give us readability and testability between multiple modules.

5.4 Non-Functional Testing

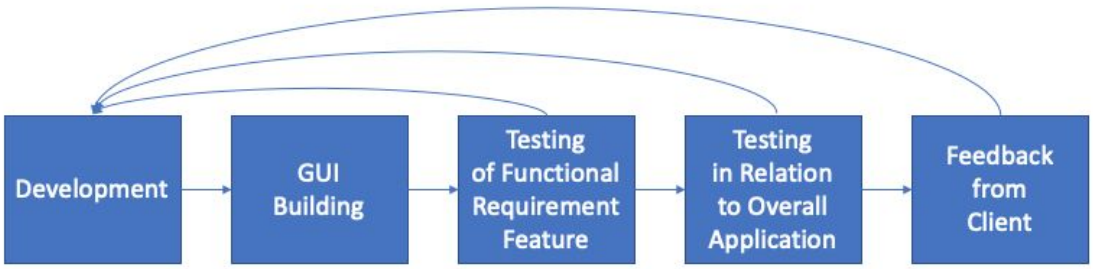
While non-functional testing is not of the highest priority for us in our first semester of the project, it is still on our minds. For compatibility, we have a desire to make sure that the program we develop is not only easy to download and use, but also can be used across multiple platforms. While Windows is the primary target, and having the system run natively with Python, we run into errors when we start requiring the end-user to download and install Python for the sole use of our program. In order to test these certain issues such as performance, usability, and compatibility, we will be heavily relying on field testing with the stakeholders to make adjustments based on what they need. Because the field of research our stakeholders are in is relatively foreign to the team, we will try to accommodate the standards that we know about, but will be looking for feedback in order to make sure our product matches the desires of the users and fits our non-functional requirements.

Part of our non-functional requirements stated above in the design document will be covered through our use of the unit testing and integration testing which will help reduce the burden on the stakeholders to provide large amounts of feedback to the team.

5.5 Process

Each functional and nonfunctional requirement feature in Section 2 undergoes the same testing process. In each iteration, code is written for that requirement. The code should be independent on other functional requirements besides what is passed in as parameters. It is also independent of the GUI. After this, the building of a GUI interface for these functions is performed. Once the GUI is completed, testing begins to ensure quality of the implementation of a feature of that functional requirement begins, as well as testing of the feature working with other features and requirements. Tests that fail are rewritten, and the process begins again. After testing is completed, the feature is presented to the client, and feedback is implemented into the design. Once there are no revisions, the implementation of the feature is complete.

Figure 4: Flow Chart of Testing



5.6 Results

While the plan for testing has been set out in the few sections preceding this, it has not yet been implemented for the project and thus there are no results from it at the moment. A simple demo has been created for the software and no testing has occurred for it outside of simple verification that the demo software compiles and runs as expected.

6. Closing Material

6.1 Conclusion

Throughout this first semester of our project, the team has been working towards creating a design that meets the requirements of our client. We needed to design an application that allows the user to import pre-existing data from Excel and manages and visualizes the data. The design also needed to consider that the user must be able to customize the data variables used to make-up the graphs as well as the appearance of the graphs. Additionally, the design needed to address the need for saving backups of the data and allowing the data and visuals to be exported and shared with another person.

The design we have come up with will result in an easy-to-use application that does not require much maintenance, and allows our target client demographic, scientists and researchers, an easier method to organize, maintain, and visualize their data. We have worked on and experimented with the base components of the application to guarantee our design is feasible and meets our client's needs. A video demonstration of these initially completed basic components can be found in the Final Presentation Slides on our team's website. The video shows that the completion of these components went smoothly and indicates that our design is feasible and satisfactory. This design was the best choice for our project and we will continue to implement it next semester. We believe that by the end of next semester, we will have completed an application that confirms the exceptionality of our design choice and satisfies our client immensely.

6.2 References

[1]"How to Buy Prism," *GraphPad*. [Online]. Available: https://www.graphpad.com/how-to-buy/. [Accessed: 26-Apr-2020].

[2]"Welcome to behave!," *Welcome to behave! - behave 1.2.7.dev1 documentation*. [Online]. Available: https://behave.readthedocs.io/en/latest/. [Accessed: 18-Apr-2020].

[3]"Cucumber," *Introduction - Cucumber Documentation*, 2019. [Online]. Available: https://cucumber.netlify.app/docs/guides/overview/. [Accessed: 18-Apr-2020].