Group 29 – Microbiology Lab Information Management and Visualization System (GraphKey)

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

Technical Advisor: Thomas Daniels
Client: Karrie Daniels

Introduction

Problem Statement

- Many scientists and researchers dedicate large amounts of time towards organizing, maintaining, and visualizing the data they collect.
- The solution should be able to automate the process of organizing, maintaining, and visualizing data.

Project Goal

- Free and easy to maintain app
- Import data and create graphs with ease
- Won’t require knowledge of underlying mechanics to use

System Design

Requirements (Functional)

- Ability to import Excel data
- Support bar graphs, box plots, and scatter plots
- Allow basic statistical analysis of data
- Multiple graphs can be created simultaneously
- Graphs will be saved to file system and be exportable as images
- Supports the creation of projects
    - Collation of multiple graphs from similar data sets

Requirements (Non-Functional)

- System will be easily maintainable
- Data should be secure
- Utilizes Python libraries for data visualization
- User Interface should be intuitive and easy-to-use

Related Products

GraphPad

- Expensive
- Lack of options for a robust suite of graphs
- Can only create one graph at time

GraphKey

- Free to use
- More robust suite of graphs (bar graphs, box plots, and scatter graphs)
- Ability to create multiple graphs at once

Resources/Cost Estimate

- No physical materials or equipment are required to complete the project
- No additional equipment will be required for our client to use the end product
- Project does require the use of Python and some Python libraries

Bottomline: Project ultimately did not require any financial resources

Risks and Mitigation

Risk: Unfamiliarity with Python and some of the needed packages.

Mitigation:

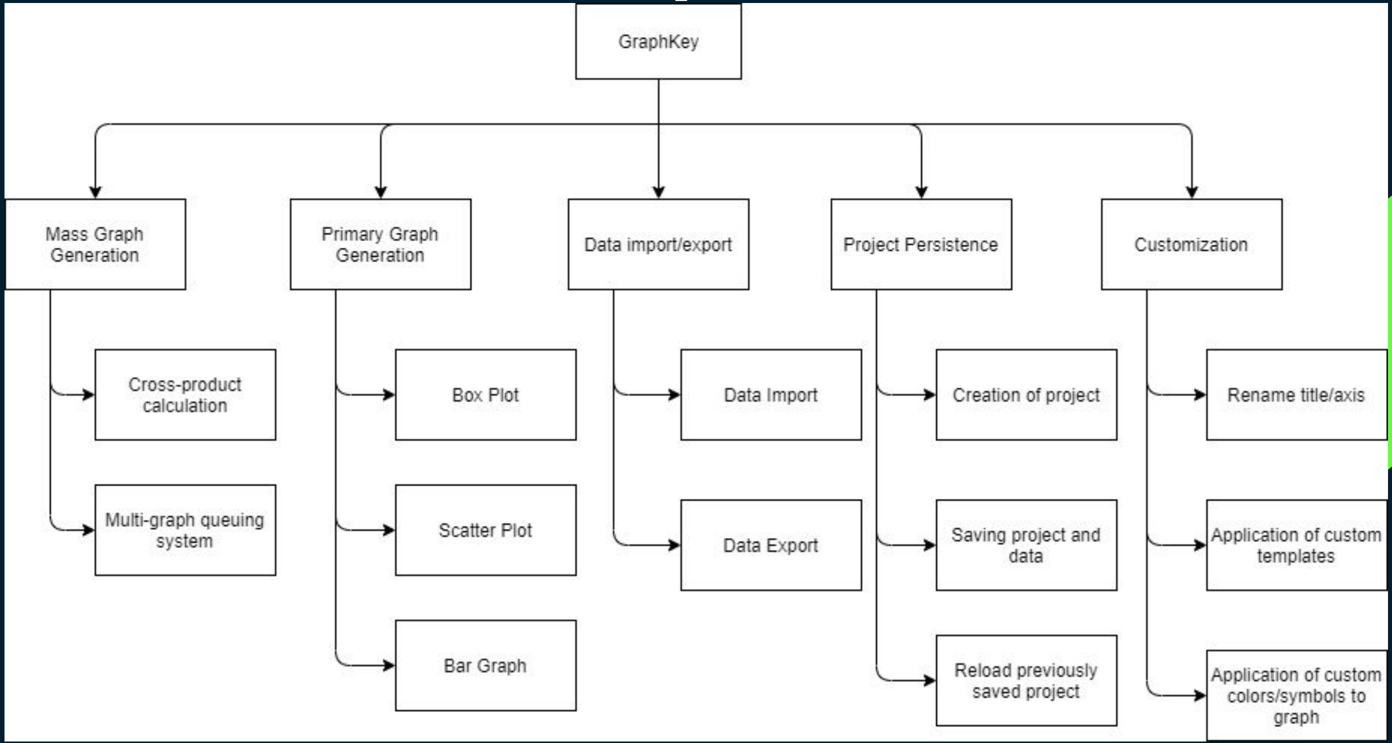
- Practice coding in Python and using the libraries before beginning on developing the solution
- Communicate with each other and the client to identify misunderstandings as soon as possible

Risk: COVID-19 forces us to work remotely.

Mitigation:

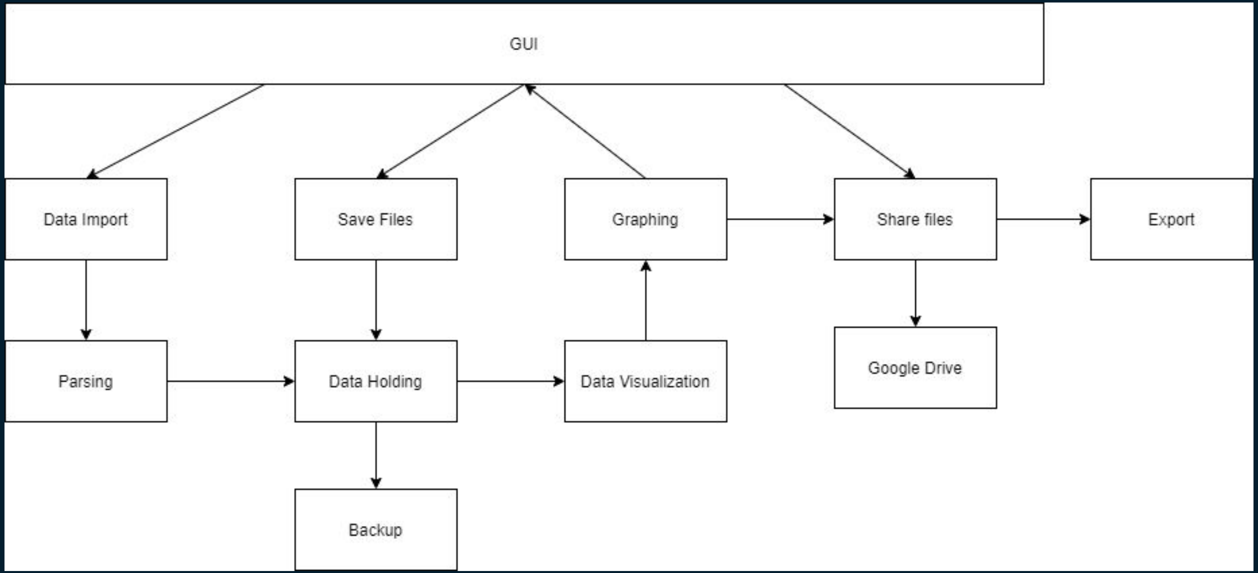
- Hold virtual meetings frequently
- Communicate with each other often

Functional Decomposition



Detailed Design

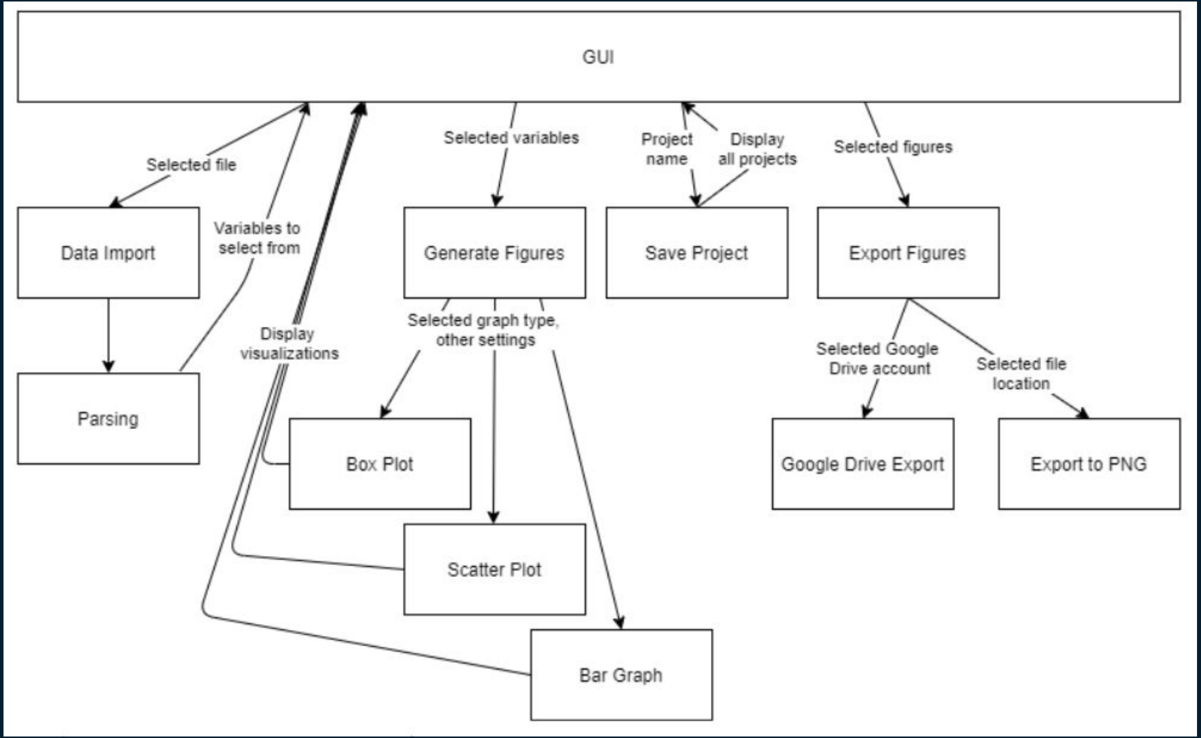
System Design (Old)



System Designs (Alternatives)

- Machine Learning Implementation
  - Extremely complex
- Web Application + Database
  - Storage and calculations on a server
- Raspberry Pi
  - Small, local device that contains the code
  - Pros:
    - Independent of client’s machine
    - Data is localized into the RPi
  - Cons:
    - Lower computing power
    - Costs for RPi

System Design (Current/Final)



Implementation Details

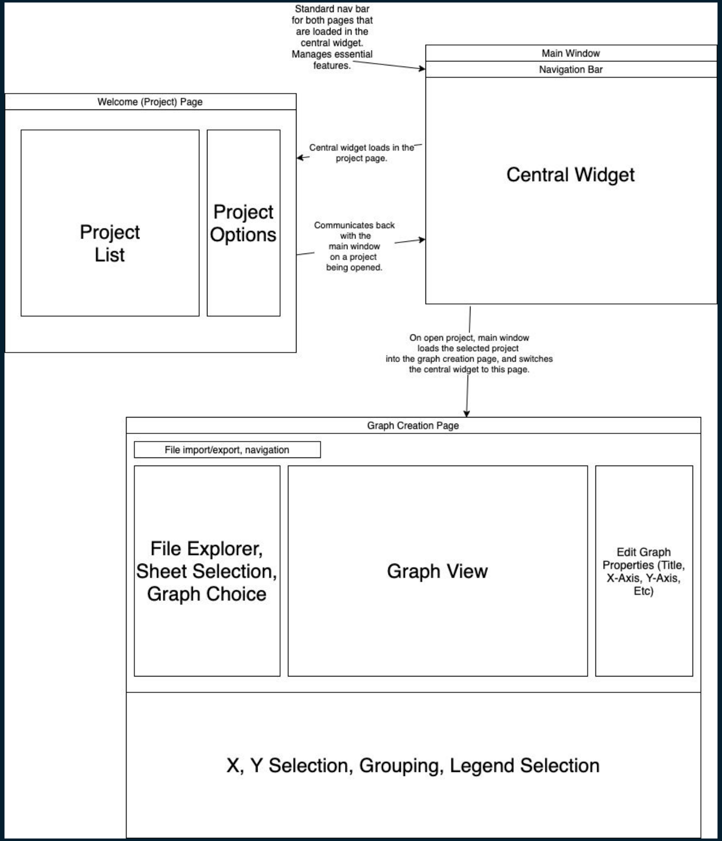
Modules/Environment

- Python 3.8 development environment
- Plotly Express for graph generation
- PyQt5 for user interface design
- Python Pickle files
- Pandas for data importing and manipulation
- PyDrive for interacting with Google Drive API
- Unittest for testing in Python

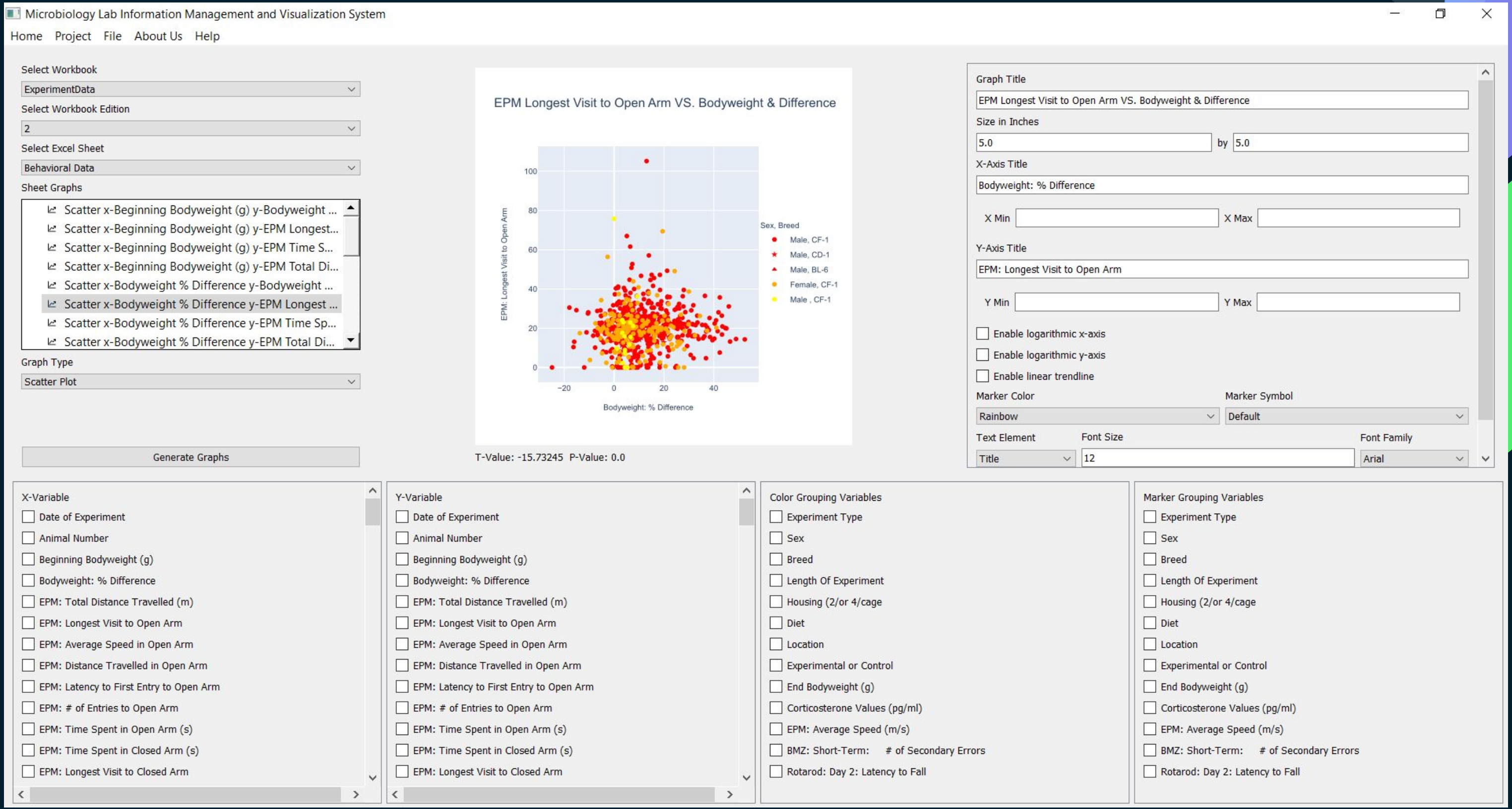
Reworking the Frontend

- We previously had several different prototypes of the UI with different features implemented on each one
  - Disjointed UI windows from prototypes are streamlined and managed by window manager; additionally, a standardized menu bar also toggles actions
  - Needed to rework the UI so it isn't so cluttered
- We also wanted the user to have the ability to create projects
  - So the data and graphs generated for one experiment wouldn't get mixed up with another experiment
  - So the user could save a project and then open it back up later

GUI Page Manager


Standard nav bar for both pages that are loaded in the central widget. Manages essential features.
Main Window
Navigation Bar
Central Widget
Welcome (Project) Page
Project List
Project Options
Central widget loads in the project page.
Communicates back with the main window on a project being opened.
On open project, main window loads the selected project into the graph creation page, and switches the central widget to this page.
Graph Creation Page
File import/export, navigation
File Explorer, Sheet Selection, Graph Choice
Graph View
Edit Graph Properties (Title, X-Axis, Y-Axis, Etc)
X, Y Selection, Grouping, Legend Selection

Reworking the Frontend - Result



Microbiology Lab Information Management and Visualization System
Home Project File About Us Help
EPM Longest Visit to Open Arm VS. Bodyweight & Difference
EPM: Longest Visit to Open Arm
Bodyweight: % Difference
100
80
60
40
20
0
−20
0
20
40
Sex, Breed
● Male, CF-1
★ Male, CD-1
▲ Male, BL-6
● Female, CF-1
● Male , CF-1
Generate Graphs

Select Workbook: ExperimentData

Select Workbook Edition: 2

Select Excel Sheet: Behavioral Data

Sheet Graphs

- Scatter x-Beginning Bodyweight (g) y-Bodyweight ...
- Scatter x-Beginning Bodyweight (g) y-EPM Longest...
- Scatter x-Beginning Bodyweight (g) y-EPM Time S...
- Scatter x-Beginning Bodyweight (g) y-EPM Total Di...
- Scatter x-Bodyweight % Difference y-Bodyweight ...
- Scatter x-Bodyweight % Difference y-EPM Longest ...
- Scatter x-Bodyweight % Difference y-EPM Time Sp...
- Scatter x-Bodyweight % Difference y-EPM Total Di...

Graph Type: Scatter Plot

T-Value: -15.73245 P-Value: 0.0

Graph Title: EPM Longest Visit to Open Arm VS. Bodyweight & Difference

Size in Inches: 5.0 by 5.0

X-Axis Title: Bodyweight: % Difference

X Min: X Max:

Y-Axis Title: EPM: Longest Visit to Open Arm

Y Min: Y Max:

- [ ] Enable logarithmic x-axis
- [ ] Enable logarithmic y-axis
- [ ] Enable linear trendline

Marker Color: Rainbow Marker Symbol: Default

Text Element: Title Font Size: 12 Font Family: Arial

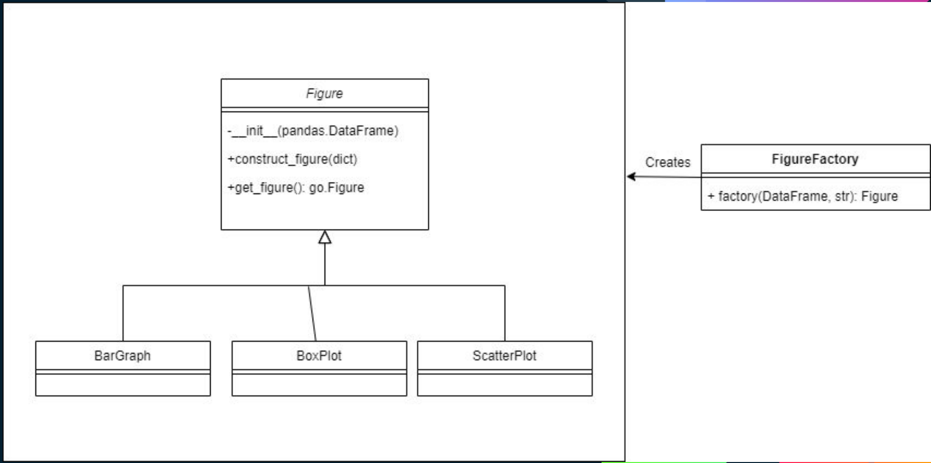
X-Variable	Y-Variable	Color Grouping Variables	Marker Grouping Variables
☐ Date of Experiment	☐ Date of Experiment	☐ Experiment Type	☐ Experiment Type
☐ Animal Number	☐ Animal Number	☐ Sex	☐ Sex
☐ Beginning Bodyweight (g)	☐ Beginning Bodyweight (g)	☐ Breed	☐ Breed
☐ Bodyweight: % Difference	☐ Bodyweight: % Difference	☐ Length Of Experiment	☐ Length Of Experiment
☐ EPM: Total Distance Travelled (m)	☐ EPM: Total Distance Travelled (m)	☐ Housing (2/or 4/cage	☐ Housing (2/or 4/cage
☐ EPM: Longest Visit to Open Arm	☐ EPM: Longest Visit to Open Arm	☐ Diet	☐ Diet
☐ EPM: Average Speed in Open Arm	☐ EPM: Average Speed in Open Arm	☐ Location	☐ Location
☐ EPM: Distance Travelled in Open Arm	☐ EPM: Distance Travelled in Open Arm	☐ Experimental or Control	☐ Experimental or Control
☐ EPM: Latency to First Entry to Open Arm	☐ EPM: Latency to First Entry to Open Arm	☐ End Bodyweight (g)	☐ End Bodyweight (g)
☐ EPM: # of Entries to Open Arm	☐ EPM: # of Entries to Open Arm	☐ Corticosterone Values (pg/ml)	☐ Corticosterone Values (pg/ml)
☐ EPM: Time Spent in Open Arm (s)	☐ EPM: Time Spent in Open Arm (s)	☐ EPM: Average Speed (m/s)	☐ EPM: Average Speed (m/s)
☐ EPM: Time Spent in Closed Arm (s)	☐ EPM: Time Spent in Closed Arm (s)	☐ BMZ: Short-Term: # of Secondary Errors	☐ BMZ: Short-Term: # of Secondary Errors
☐ EPM: Longest Visit to Closed Arm	☐ EPM: Longest Visit to Closed Arm	☐ Rotarod: Day 2: Latency to Fall	☐ Rotarod: Day 2: Latency to Fall

Data/Project Persistence

- Projects
  - Creating/Importing/Exporting Projects
  - Loading Projects from session to session
- Pickle files
  - Graph configurations and settings
  - Imported data
- By separating imported data from a graph's configurations, it allows us to import revisions to a project workbook as a data-set grows while using same graph configurations.
- Also significantly reduces a project's size.

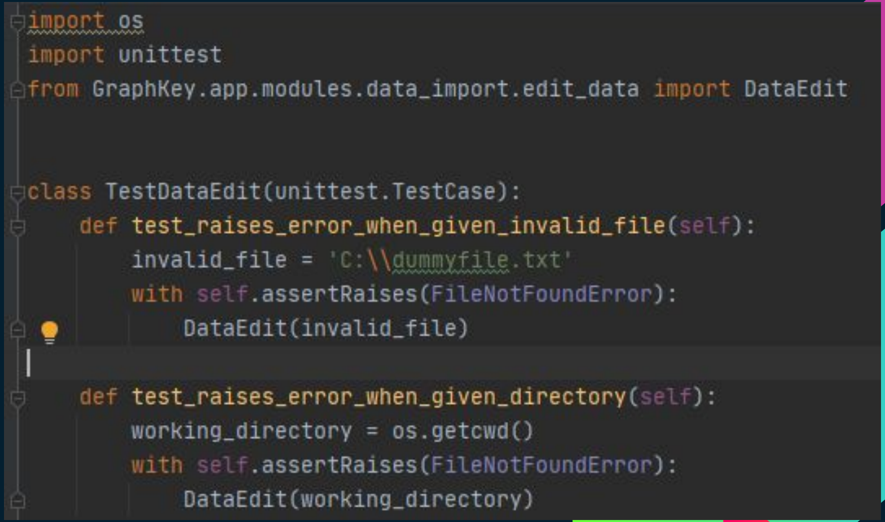
FigureFactory - Abstract Graphing

- Figure - Abstract class each graph function will implement
- Individual graphs - Implement a construct_figure() method
  - User passes a dictionary that holds parameters such as names, data, colors, shapes, etc.) into the method
- Factory method - GUI calls the FigureFactory with their desired figure and gets a Figure object
  - No more recursive changes throughout the GUI, only in the Factory class



Testing and Results

Testing



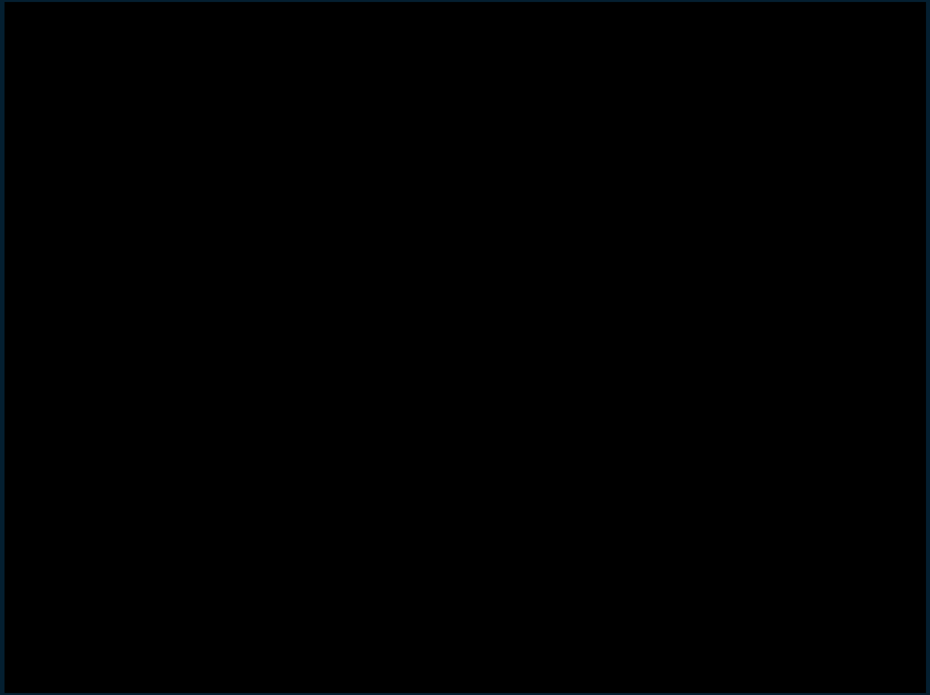
```python
import os
import unittest
from GraphKey.app.modules.data_import.edit_data import DataEdit


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)
```

- Unit Testing
  - Using Python’s unittest
  - Ensure stability on back-end
- Integration Testing
  - Hand-testing - verification of connections to front-end and back-end
  - Done by developers and the client
- CI/CD
  - Set up in GitLab
  - Pipelines could not be merged unless unit tests passed

Results/Demo



Conclusion

Challenges and Lessons Learned

- Original work was very segmented between team members, each was working on their own project
- Should have spent more time on determining specific requirements before development
- User interface diagram could have been more refined and detailed
- Needed to create more unittests throughout the second semester
- Finishing a project remotely with a compressed time schedule due to COVID-19

Things We Couldn’t Get To

- More security
- Fixing executable generation
  - Currently we have “support”, but executable can get to over ~500 mb and the paths for certain files breaks when generating
  - Decided to leave the hooks in, but not support it at launch do to time constraints
- Automatic data backups
- Dark Mode

Conclusion

- GraphKey meets all of the required goals we set out for ourselves
- The product is an easy-to-use application
- Through testing we guarantee our design works and meets our client’s needs
- Overall, we believe that we have completed an application that confirms the exceptionality of our design choice and solves our client’s problem

Questions?