Mini Project 2
Productivity Analytics - STRV

CPSC 4720 - Software Development Methodology
Joshua Norman

April 24, 2020

Introduction

For Mini-Project 2, we were tasked with creating a Desktop application for measuring the users productivity. Here we will focus on the process of software production within industry; moving from wire-frames to implementation. While learning about good design practices to make a visually appealing interface. All of the code is available publicly on GitHub at the following link.

https://github.com/jmnorma/Productivity_Analytics

Like any successful project, our productivity analytics application needed a name – meet STRV, your productivity analytics Power tool. STRV is loaded with detailed graphics that helps track your day. With privacy in mind, STRV only keeps track of the applications you tell it to. Built on a simple python engine, STRV is your next efficient work-day Power tool.

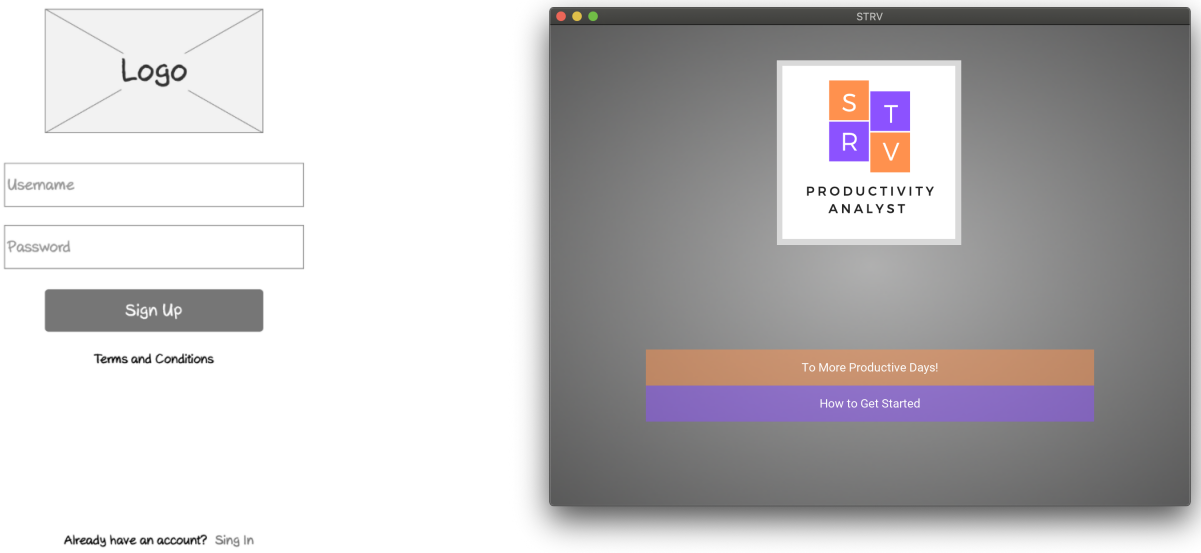
Design

A majority of the design work was done in Mini Project 1, along with the creation of wire frames models. For Mini Project 2, I took a look at design patterns that utilized color. This approach optimized the visual appeal of the application and allowed for our application to be learn-able to new users. As described in Don Norman's book, *The Design of Everyday Things*, learn-ability is a key design principle that is important to consider. I limited the main color pallet to three main colors. All of which having a similar pastel feel, in order to maintain vibrance. Buttons that are important to the user had a background color of green, showing the user to start or take action with the button. Once pressed, some buttons would be greyed to show the user the button is already in use.

In terms of the main dashboard layout, I tried to use different charts to help the user visualize their data. A Pie Chart was used to show all application usage regardless of whether it was productive or not. A Bar graph was used show productive vs unproductive usage with total time along the y-axis. These two visualization were colored to draw the users attention. The Pie Chart, is the most helpful analytic so it was placed in the top left corner of the screen. This was identified as a visual hot spot within eye-tracking heat maps research discussed in class.

Another design aspect considered was word choice. Within ENGL 3070 Business writing we discussed the importance of tone when creating deliverables. I wanted the tone of our application to be stress-free and displayed this through our instructional start-up guide within the application.

The last part of the design process consisted of asset creation. Here I designed the backgrounds and logos that would be used in the implementation. This was done before the implementation stage to expedite the process. Adobe Illustrator and the Canva Web application were use for this part.



Left: *Is the Landing page from Mini Project 1. This is the first thing the user will see when they open the application.* ***Right:*** *is the landing page from Mini Project 2, implementing the design principles of color and tone.*

Implementation

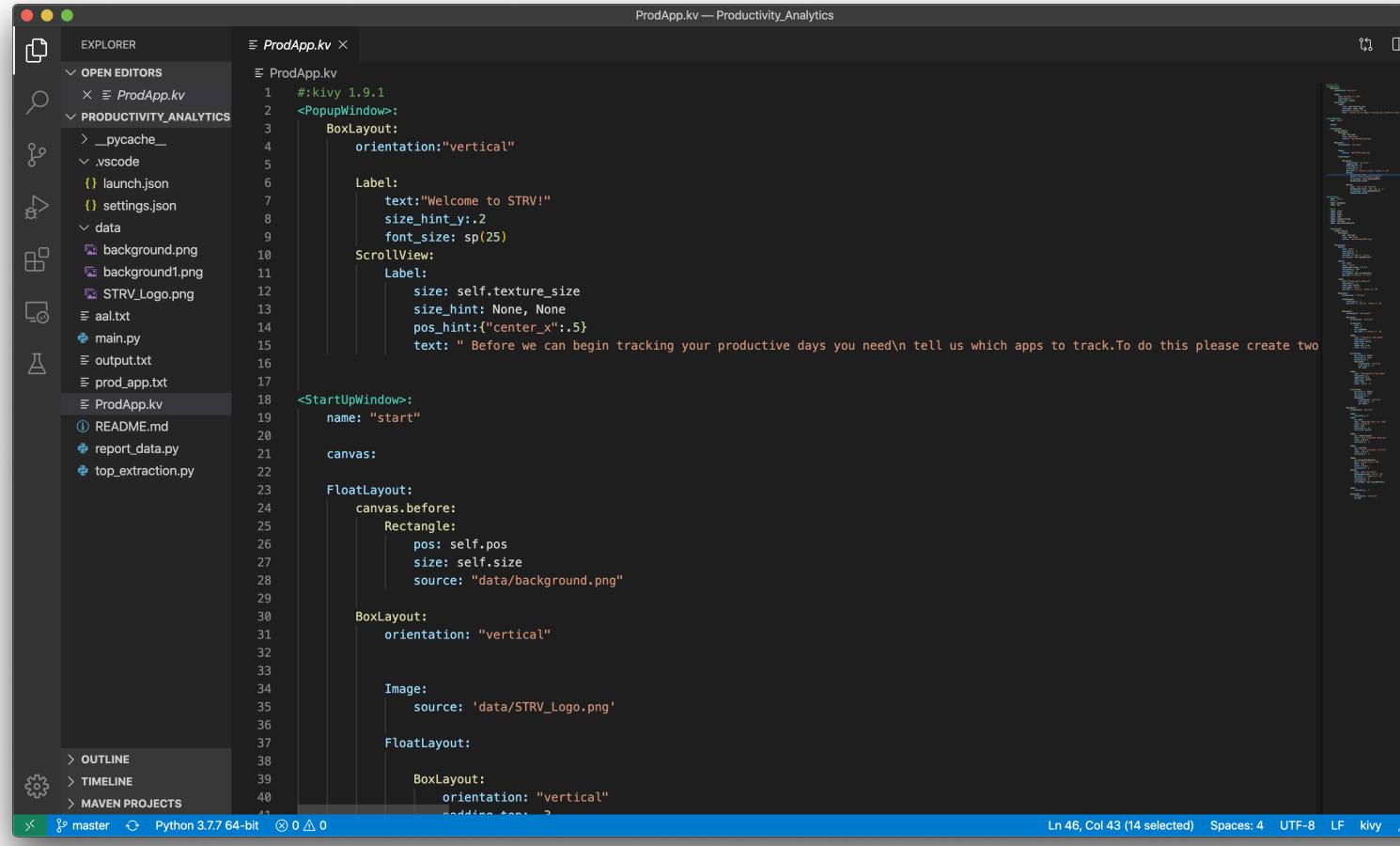
The application was suppose to measure keyboard and mouse metadata however I found it more interesting to track CPU usage to determine productivity. This was so the application could run on the new Mac OS (Catalina). With the new update, third party collection of keyboard and mouse metadata was restricted and there weren't any available libraries to collect this information. Since this is a Mini Project, I couldn't justify writing one from scratch, given I have no prior knowledge with such information. Which is why I choose to look at CPU data instead. This change was cleared by Dr. Rodeghero via Slack on Sunday April 19, 2020.

The implementation process of this project took about 62 hours to complete. This process was started on March 16, 2020 with the back end python engine for data collection. A large part of why this project took ample time to complete was the research that was involved to find and use a library for mouse and keyboard metadata data collection, a GUI library, and a graph plotter library. The STRV files on the GitHub can be categorized into three main sections:

- **Data Engine**
  - top_extraction.py
  - report_data.py
- **GUI Source**
  - main.py
  - ProdApp.kv
  - Data (Assets Folder)
- **User Genrated**
  - aal.txt
  - prod_app.txt

The Data Collection engine was build from scratch using python and its regular built in libraries. The Data Engine files work in tandem to collect a list of running applications on the computer using the command line "top" function and pythons subroutine library. These applications are then parsed by two fields: Important and Productive. The data of application names deemed important are stored within a text file created by the user. The file must be named aal.txt (acceptable application list). Then the list of productive apps are in a file names prod_app.txt. While this does require an extra step for the user, once set the user doesn't have to change anything.

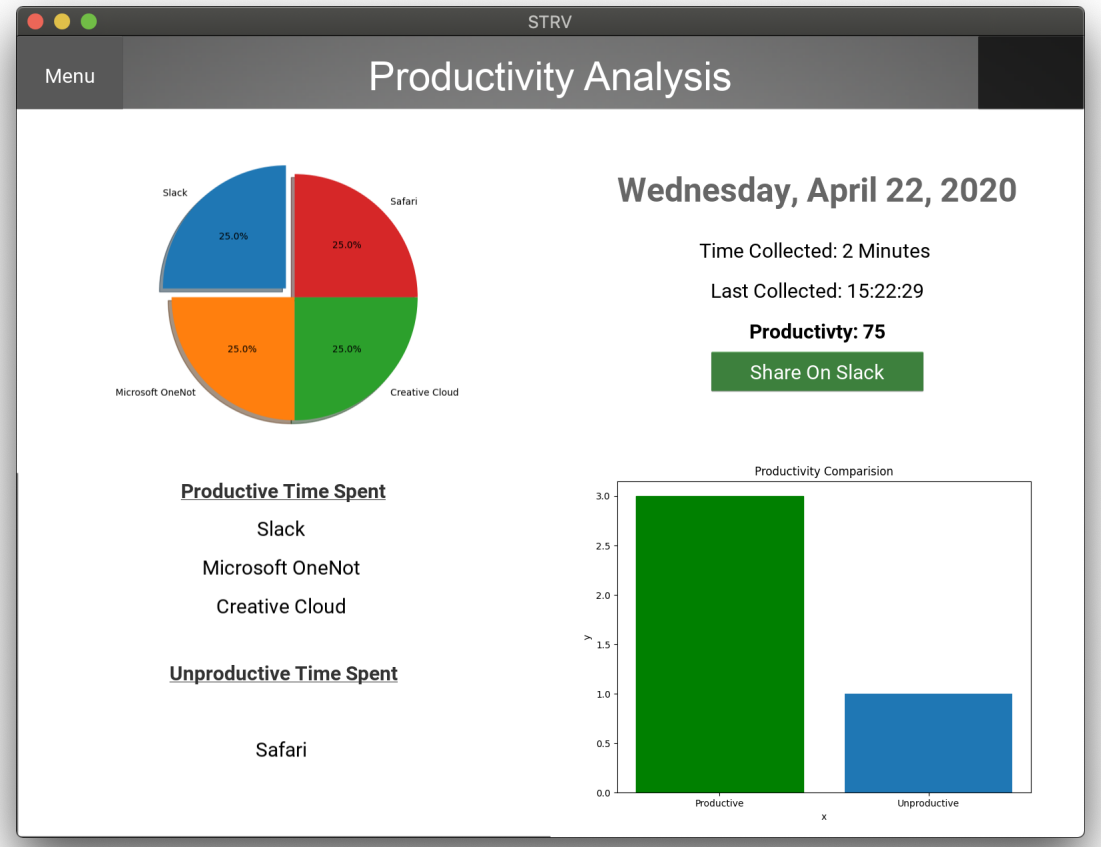
The Graphical User Interface or GUI was also created in python using a library called Kivy. Prior to starting this project I had no experience in creating GUIs in python. I chose Kivy because one of my professors suggested it. Kivy is a library much like Python's tkinker but supports kv files. This is Kivy's version of an XML file. I have worked as an Android Mobile Developer this pass summer and have lots of exposure to XML. Below you'll find a snippet of kv lang code.



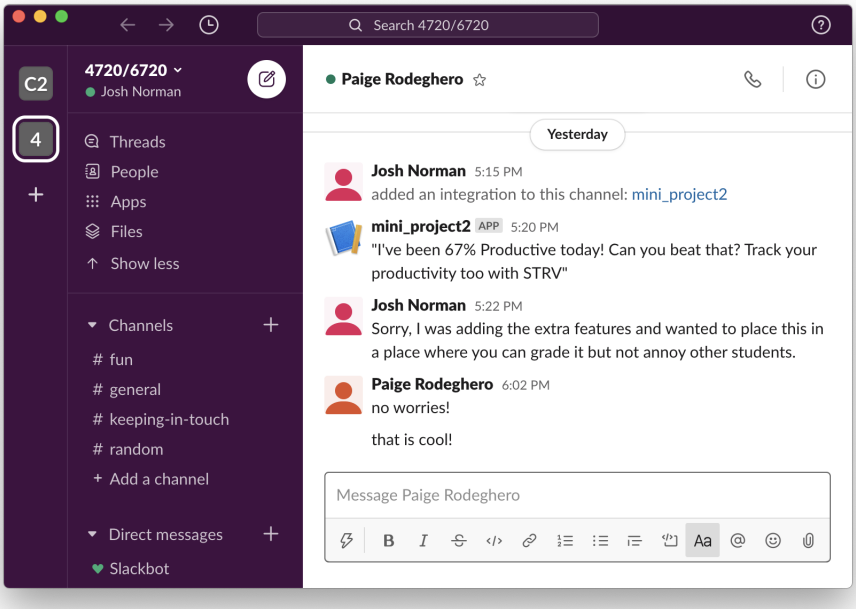
Underneath the hood of a Kivy GUI is a powerful engine which has scheduling capabilities. However, I only discovered this feature after 4 hours of trying to make a multi-threaded version of the app. The native scheduler allows the application to run the data collection engine every 60 seconds in the backgrounds. This makes the app still usable while collecting data. The scheduling feature only starts when the user presses a Green "start" button located at the top of the screen.

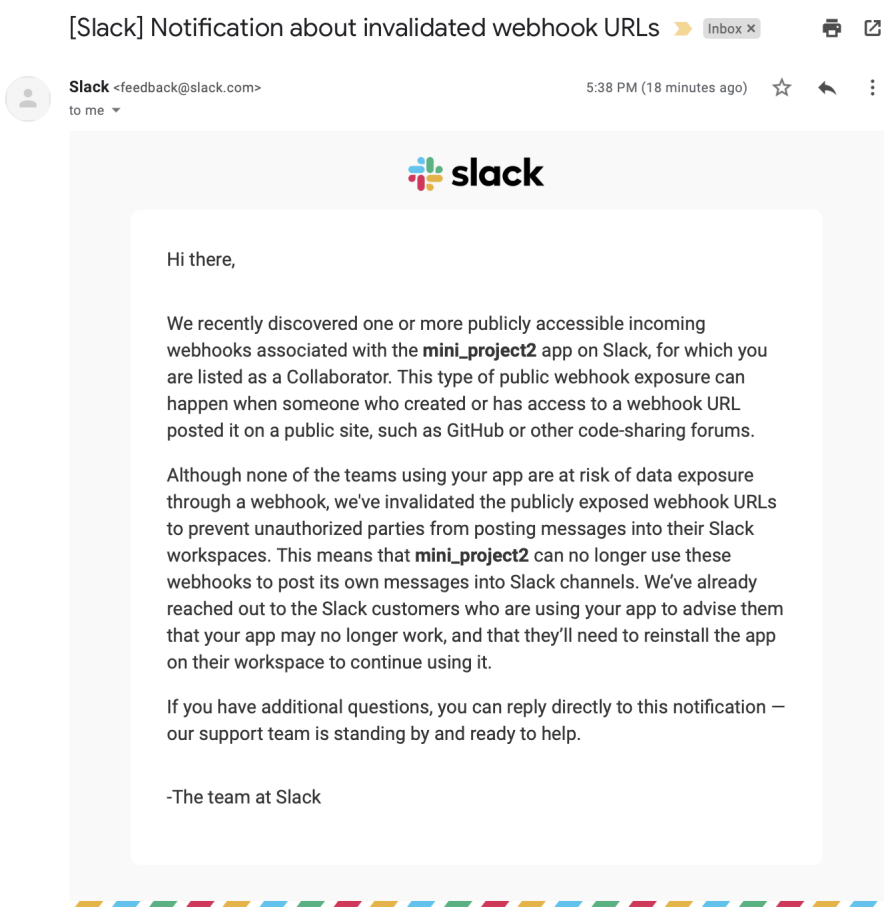
For the chart and graph visualizations, a python library called matlibplot was used. The library creates and an object of itself which you can then generate a gcf file from. These gfc files are ported directly to the GUI to reduce the Data usage and memory dependencies. The data for the graphs is collected from an instance the report_data object from the engine.

Below: *The main dash board is shown while in use to display the layout and graphs.*



For the additional feature work, I chose to implemented a Slack notification that will post to a slack channel with a specified web-hook. This web-hook was obtained from the slack developer site and has been tested (shown below). However since this project is required to be posted publicly on GitHub the current web-hook was automatically deactivated by slack as soon as it was pushed to GitHub to protect the users. If you would like to use this feature please create your own Slack App (using web-hooks) and put the URL for the API call in the shareButton function of the MainWindow class located in main.py. Also shown below, is the email from Slack saying the web-hook has been Automatically disabled.




[Slack] Notification about invalidated webhook URLs Inbox ×

Slack <feedback@slack.com> 5:38 PM (18 minutes ago)
to me

slack

Hi there,

We recently discovered one or more publicly accessible incoming webhooks associated with the **mini_project2** app on Slack, for which you are listed as a Collaborator. This type of public webhook exposure can happen when someone who created or has access to a webhook URL posted it on a public site, such as GitHub or other code-sharing forums.

Although none of the teams using your app are at risk of data exposure through a webhook, we've invalidated the publicly exposed webhook URLs to prevent unauthorized parties from posting messages into their Slack workspaces. This means that **mini_project2** can no longer use these webhooks to post its own messages into Slack channels. We've already reached out to the Slack customers who are using your app to advise them that your app may no longer work, and that they'll need to reinstall the app on their workspace to continue using it.

If you have additional questions, you can reply directly to this notification — our support team is standing by and ready to help.

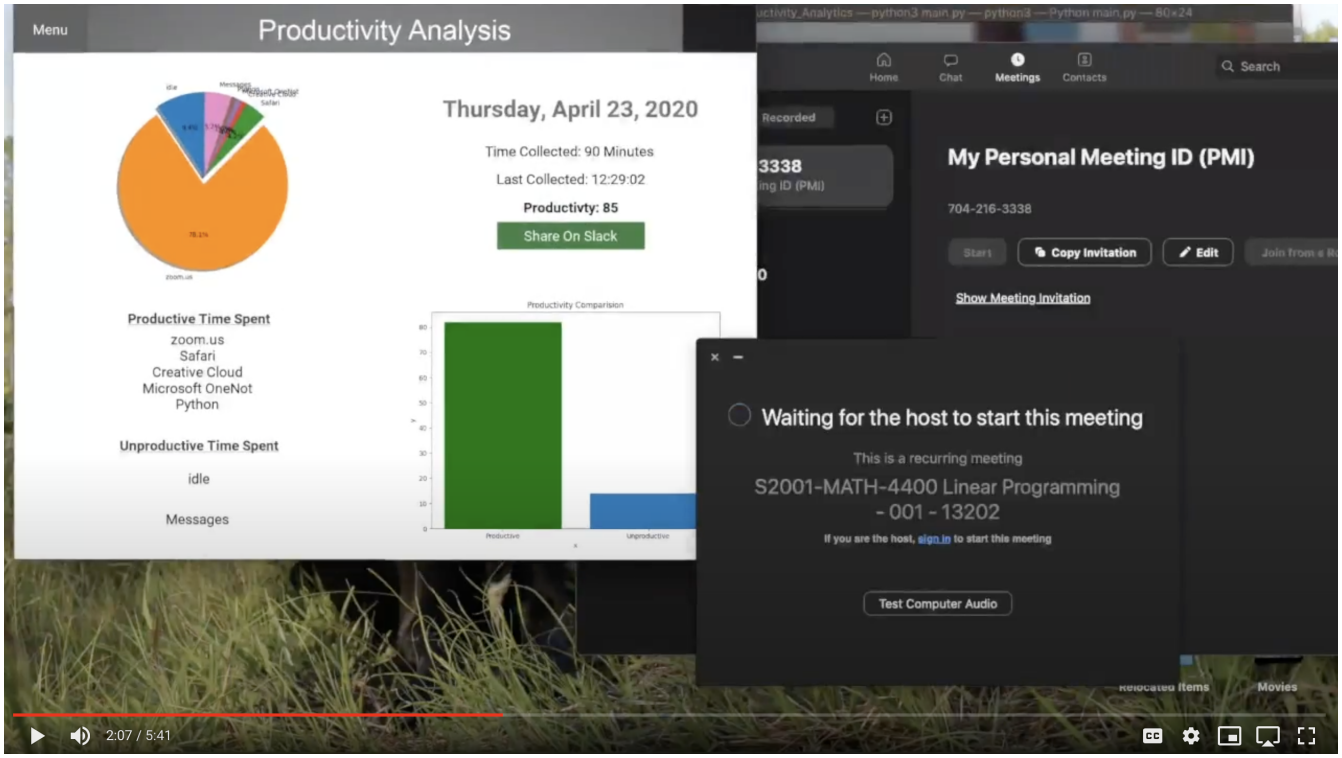
-The team at Slack

How to Get Started Using STRV

1. First you need to install the necessary python libraires. This can be done using python's pip terminal command. Denote you must be using python 3.
   - python -m pip install kivy
   - python -m pip install ffpyplayer
   - pip install -U matplotlib
2. Then you need to tell STRV the applications you wish to track. This can be done by creating two text files listing all of the applications you wish to track (pressing enter between each application).
   - Place all of the applications you wish to track in a file names **aal.txt**. This stands for acceptable application list.
   - Place all of the application from aal.txt that you deem as productive into a file name **prod_app.txt**.
3. Once complete, places the txt files in the root project directory (i.e. in the same folders main.py).
4. Open your terminal to the root project directory and execute ***python3*** **main.py**
5. Then the application will open and you will be faced with the landing page. You may either continue to the main dashboard or read the in-app set-up guide that will direct the user to complete steps 2-4 again.
6. When you are finally ready to start tracking, enter the main dash board and press the green start button in the upper right hand corner of the screen.
7. Finally, continue about your day leaving STRV running in the background. Your data will automatically populate on the main dashboard as it happens. Now go reach your Stride with STRV!

Analysis of my 4-hour workday

In order to complete this part of the project, I chose to pick a school day where I would be actively using my computer for a majority of the day (either on productive applications or not). The app worked considerably well and highlighted certain edge cases that I didn't think about during the implementation stage and allowed for me to fix them before my final submission. However, since it was a school day my productivity was mainly zoom.



Here is the link to the full video:
https://drive.google.com/file/d/1_VHl8I8n5SnnWoYGcQsXES2rxjjd8OxT/view?usp=sharing

Sources and References

"17.1. Subprocess - Subprocess Management." 17.1. Subprocess - Subprocess Management - Python 2.7.18 Documentation, docs.python.org/2/library/subprocess.html.

"Clock Object." Clock Object - Kivy 1.11.1 Documentation, kivy.org/doc/stable/api-kivy.clock.html.

Dossetto, Fio. "Understanding Heatmaps for Better UI Design: Inside Design Blog." Invisionapp, Inc., www.invisionapp.com/inside-design/heatmaps-for-ui-design/ .

"Kivy Framework." Kivy Framework - Kivy 1.11.1 Documentation, kivy.org/doc/stable/api-kivy.html.

"Labeling a Pie and a Donut." Labeling a Pie and a Donut - Matplotlib 3.2.1 Documentation, matplotlib.org/gallery/pie_and_polar_charts/pie_and_donut_labels.html#sphx-glr-gallery-pie-and-polar-charts-pie-and-donut-labels-py.

Norman, Donald A. The Design of Everyday Things. Basic Books, 2013.

Slack. "Slack API." Slack, api.slack.com/.

"Visualization with Python¶." Matplotlib, matplotlib.org/.