Portfolio Programming Assignment: Improving the Stock Problem with Additional Functionality

Master of Science

Information & Communication Technology

Rich Fallat

University of Denver University College

June 4, 2021

Faculty: Sherri Maciosek, PhD

Director: Michael Batty, PhD

Dean: Michael J. McGuire, MLS

Reflection

The portfolio assignment provided a means to improve the code, implement objects, implement exceptions, and create a code repository on Github (Fallat 2021). The code required refactoring, and I decided to create the Github repository first. Github version controls the code with each push commit, and provides a means to roll back the code, which gave me a safety net, during code refactoring.

Figure one displays the portfolio assignment github page.

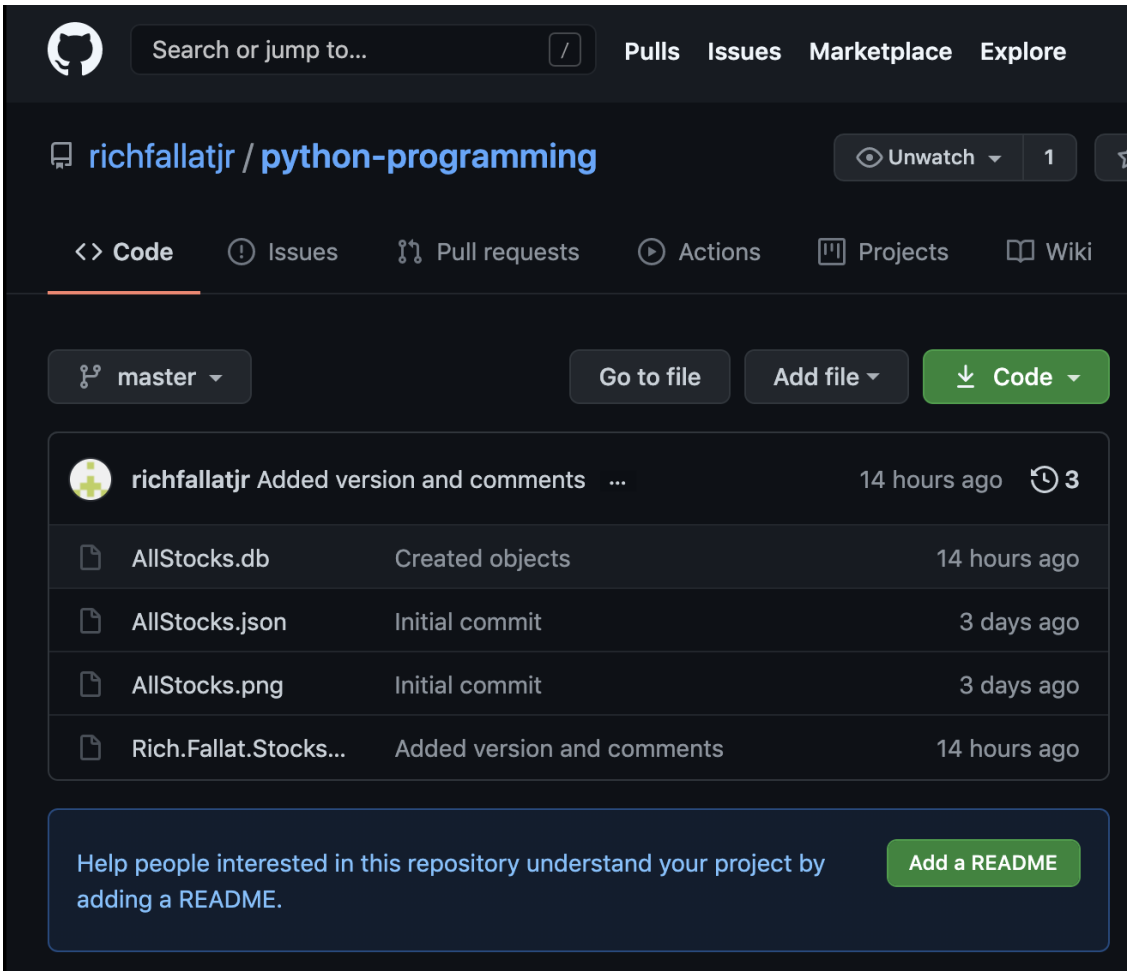

Figure 1. Github Repository. Source: (Fallat 2021).

In the previous assignment, I opted to write the program without objects. The portfolio assignment leverages objects, which eliminates the copying and pasting of lines of code. This made the code simplified, and provided a more generic interface that could handle new datasets. Figure 2 displays the previous methods that transformed and created plots for matplotlib.

```
# Get subplots
goog_date, goog_close = getSubplot("GOOG", symbol, date, close)
msft_date, msft_close = getSubplot("MSFT", symbol, date, close)
rdsa_date, rdsa_close = getSubplot("RDS-A", symbol, date, close)
aig_date, aig_close = getSubplot("AIG", symbol, date, close)
fb_date, fb_close = getSubplot("FB", symbol, date, close)
m_date, m_close = getSubplot("M", symbol, date, close)
f_date, f_close = getSubplot("F", symbol, date, close)
ibm_date, ibm_close = getSubplot("IBM", symbol, date, close)

# Run the graph
plt.style.use("seaborn")
fig, ax = plt.subplots()

ax.plot(goog_date, goog_close, c="#66D9EF", label="GOOG")
ax.plot(msft_date, msft_close, c="#E6DB74", label="MSFT")
ax.plot(rdsa_date, rdsa_close, c="#A6E22E", label="RDS-A")
ax.plot(aig_date, aig_close, c="#F92672", label="AIG")
ax.plot(fb_date, fb_close, c="#AE81FF", label="FB")
ax.plot(m_date, m_close, c="#FD971F", label="M")
ax.plot(f_date, f_close, c="#272822", label="F")
ax.plot(ibm_date, ibm_close, c="#7C806C", label="IBM")
```

Figure 2. Previous Coding Assignment Snippet.

Figure 3 shows how the code was simplified via objects, metadata dictionary, and functions.

```
def makePlots(self):
        """Create subplots based on symbols"""
        plt.style.use("seaborn")
        fig, ax = plt.subplots()

        for i in self.getStocks().keys():
```

```
            date, close = self.getSubplot(i)
            ax.plot(date, close, c=self.getStocks()[i]["color"], label=i)
```

Figure 3. Improved Code Using Objects.

Refactoring was difficult to implement because the original stock problem had several moving parts. These included:

- json file handling
- data transformation
- data calculation given number of shares owned
- data preparation for matplotlib
- matplotlib graphing
- database insertion

The refactoring allowed these parts to talk to each other in a more generic fashion. While this simplified code in some places, it took some time to disentangle the pieces into functions. The code ran into a massive inefficiency. When inserting data into the database, an earlier iteration recalculated all of the data transformation to every row being inserted. At that rate, it would have taken several minutes to store the data into the sqlite database. The solution involved storing the data into initialized variables so the transformed data could be generated once, and then shared between several functions.

Finally, I added the option for a user to input the stocks that get displayed in the graph. The user can experiment with different numbers of shares owned and colors displayed in the graph. This required some exception handling at various places in the code. Figure 4 displays the function that provides an option for the user to input the stocks displayed on the graph, the number of shares owned, and the color of the line used for the stock symbol.

```
def setStocks(self):
      """Set the stocks via user input."""
      active = "n"
```

```
active = input("Would you like to input the stock data? y/n ")

# reset init stocks dictionary
if active == "y":
    self._stocks = {}

while active == "y":
    symbol = input("Which symbol? ")
    num_shares = float(input("How many shares? "))
    color = input("What color? ")
    self._stocks[symbol] = {
        "num_shares": num_shares,
        "color": color
    }
    active = input("Would you like to add another stock? y/n ")
```

Figure 4. User Input Addition.

Conclusion

The portfolio assignment built on previous assignments by adding object functionality and user input to further customize the graph. The refactoring provided several challenges, but made for more reusable code. In the future, I would add to the user input functionality by limiting the stock symbol possibilities from what is contained in the json data. Currently, if the symbol does not exist in the dataset, the graph displays empty without explanation. The additional functionalities helped tie together knowledge learned throughout the course.

References

Fallat, Rich. 2021. “Python Programming.” In github.com/richfallatjr. Access (June): https://github.com/richfallatjr/python-programming.