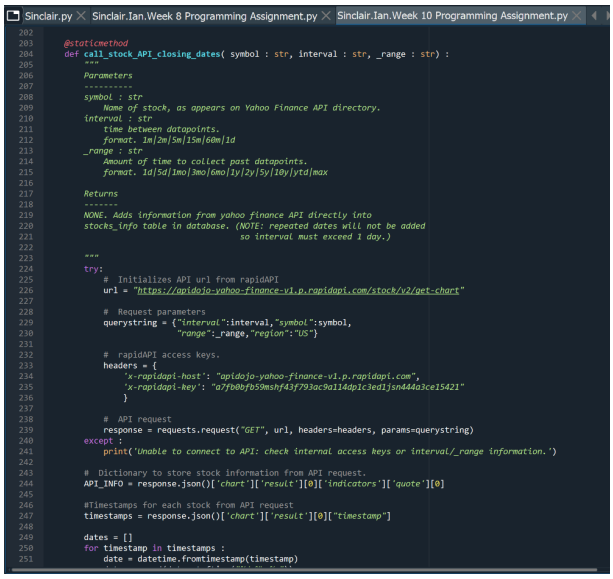
Week 10 Programming Assignment Reflection:
PORTFOLIO PROGRAMMING ASSIGNMENT - IMPROVING THE STOCK PROBLEM WITH ADDITIONAL FUNCTIONALITY
ICT 4370. Python Programming
Dr. Nirav Shah
Ian Sinclair

I am fairly happy with how my project turned out. For my functionality I created a method to request information from the yahoo finance API that would be added to the existing stock information table in the stored database. I also added a method to run linear regression analysis on the stock data to attempt to predict closing prices for different stocks in the portfolio. Finally, I moved many of the previous analysis and plots to functions/methods to improve readability and versatility of the codebase. In general, the program is now capable of storing investor portfolio information, including stock purchase reports. Along with storing large amounts of general stock information and running analysis and creating visualizations. Lastly, the program is able to quickly update general stock data with new information from yahoo finance. This all seems to be a good framework for a functional and user friendly stock portfolio management program that is easy to add new features too.

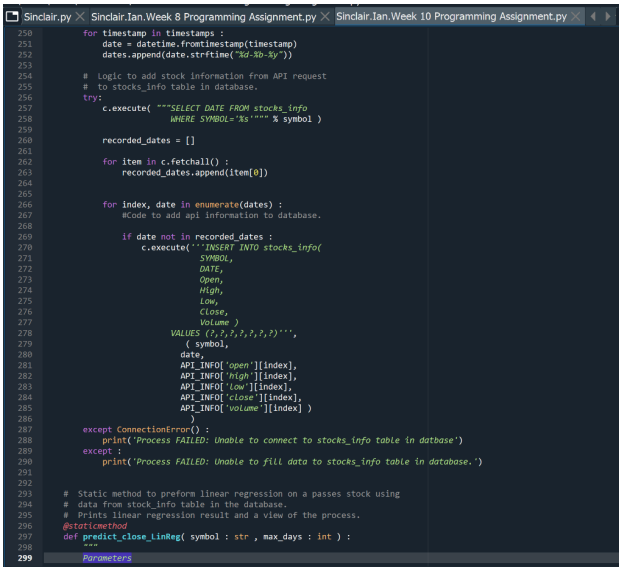
My goal with the new functionality was to develop adaptive code, so unique visualizations and data updates can be done just by passing in different parameters. This was mostly successful; however, I would have likely to create frameworks for complicated forms of analysis, beyond linear regression. Still, with how the classes are set up I don't think this would be terribly difficult to implement. In general I had the most trouble creating the method to connect to the yahoo finance API and retrieve the correct information. I would end up getting long streams of nested dictionaries that were difficult to decipher. I was able to find a lot of documentation about connecting to APIs but less about parsing the data. Overall, I really enjoyed the assignment and I'm happy with the outcome. I do think there is a lot more functionality that could be added, like better visualizations specifically tied to the investor class, more integration between the classes, and better automation to load new data from files.

Examples:

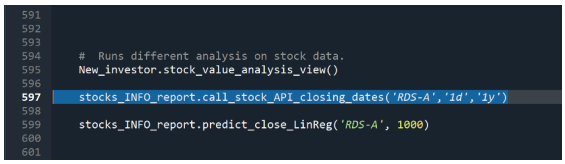
Calling the API:




```
    @staticmethod
    def call_stock_API_closing_dates( symbol : str, interval : str, _range : str) :
        """
        Parameters
        ----------
        symbol : str
            Name of stock, as appears on Yahoo Finance API directory.
        interval : str
            time between datapoints.
            format. 1m|2m|5m|15m|60m|1d
        _range : str
            Amount of time to collect past datapoints.
            format. 1d|5d|1mo|3mo|6mo|1y|2y|5y|10y|ytd|max

        Returns
        -------
        NONE. Adds information from yahoo finance API directly into
        stocks_info table in database. (NOTE: repeated dates will not be added
                                        so interval must exceed 1 day.)

        """
        try:
            #  Initializes API url from rapidAPI
            url = "https://apidojo-yahoo-finance-v1.p.rapidapi.com/stock/v2/get-chart"

            #  Request parameters
            querystring = {"interval":interval,"symbol":symbol,
                           "range":_range,"region":"US"}

            #  rapidAPI access keys.
            headers = {
                'x-rapidapi-host': "apidojo-yahoo-finance-v1.p.rapidapi.com",
                'x-rapidapi-key': "a7fb0bfb59mshf43f793ac9a114dp1c3ed1jsn444a3ce15421"
                }

            #  API request
            response = requests.request("GET", url, headers=headers, params=querystring)
        except :
            print('Unable to connect to API: check internal access keys or interval/_range information.')

        #  Dictionary to store stock information from API request.
        API_INFO = response.json()['chart']['result'][0]['indicators']['quote'][0]

        #Timestamps for each stock from API request
        timestamps = response.json()['chart']['result'][0]["timestamp"]

        dates = []
        for timestamp in timestamps :
            date = datetime.fromtimestamp(timestamp)
```




```
        for timestamp in timestamps :
            date = datetime.fromtimestamp(timestamp)
            dates.append(date.strftime("%d-%b-%y"))

        #  Logic to add stock information from API request
        #  to stocks_info table in database.
        try:
            c.execute( """SELECT DATE FROM stocks_info
                          WHERE SYMBOL='%s'""" % symbol )

            recorded_dates = []

            for item in c.fetchall() :
                recorded_dates.append(item[0])


            for index, date in enumerate(dates) :
                #Code to add api information to database.

                if date not in recorded_dates :
                    c.execute('''INSERT INTO stocks_info(
                                SYMBOL,
                                DATE,
                                Open,
                                High,
                                Low,
                                Close,
                                Volume )
                            VALUES (?,?,?,?,?,?,?)''',
                              ( symbol,
                              date,
                              API_INFO['open'][index],
                              API_INFO['high'][index],
                              API_INFO['low'][index],
                              API_INFO['close'][index],
                              API_INFO['volume'][index] )
                              )
        except ConnectionError() :
            print('Process FAILED: Unable to connect to stocks_info table in datbase')
        except :
            print('Process FAILED: Unable to fill data to stocks_info table in database.')


    #  Static method to preform linear regression on a passes stock using
    #  data from stock_info table in the database.
    #  Prints linear regression result and a view of the process.
    @staticmethod
    def predict_close_LinReg( symbol : str , max_days : int ) :
        """
        Parameters
```



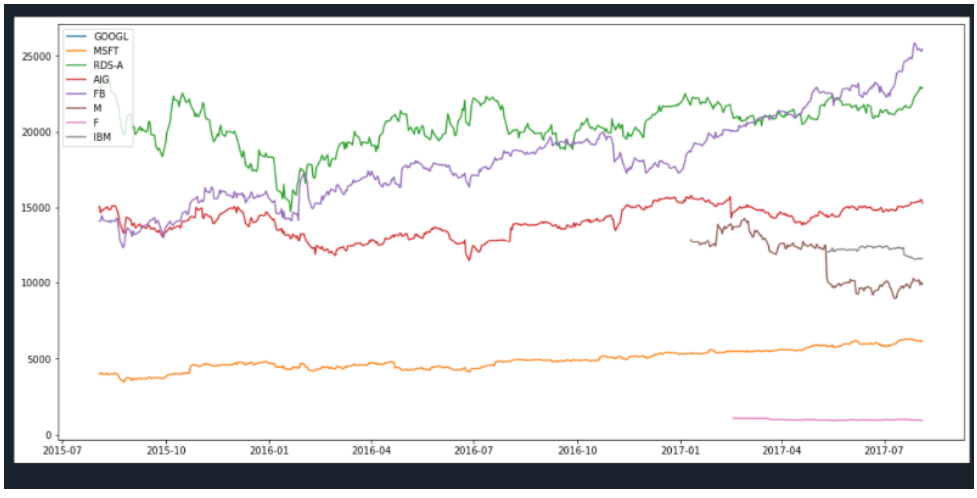
```
591
592
593
594     #  Runs different analysis on stock data.
595     New_investor.stock_value_analysis_view()
596
597     stocks_INFO_report.call_stock_API_closing_dates('RDS-A','1d','1y')
598
599     stocks_INFO_report.predict_close_LinReg('RDS-A', 1000)
600
601
```

The Idea here is that the only requirement to request data from the yahoo finance API is to call the static function and pass the stock name, and interval between data points, and the total time range from present day.

Creating Investor Stock Value Visualizations

```
73
74      def stock_value_analysis_view ( self ) :
75          """
76          Returns
77          -------
78          Prints to png a diagram/view of the value of each stock in the
79          investors portfolio over time (spans entire database).
80
81          """
82          #  Retrieves information from stocks_info table in database
83          #  returns a tuple in the form (SYMBOL, DATE, Close)
84          try:
85              c.execute("SELECT SYMBOL, DATE, Close FROM stocks_info")
86              stock_information_update = c.fetchall()
87
88
89              #  Retrieves information from stocks_MASTER table in database
90              #  for a particular investor.
91              #  Returns a tuple in the form (SYMBOL, No_SHARES, PURCHASE_DATE)
92              c.execute("""SELECT SYMBOL, No_SHARES, PURCHASE_DATE
93                           FROM stocks_MASTER
94                           WHERE investor_ID is '%s'""" % self.investor_ID)
95              stock_in_portfolio = c.fetchall()
96          except :
97              print('Process FAILED: unable to retrieve from table stocks_info.')
98
99
100         plt.figure(figsize=(17,8))
101
102         #  Developes view for stock report for portfolio under investor object.
103         #  NOTE: stock = (SYMBOL, No_SHARES, PURCHASE_DATE)
104         #        stock_update = (SYMBOL, DATE, Close)
105         for stock in stock_in_portfolio :
106             stock_value_y_axis = []
107             dates_x_axis = []
108
109             for stock_update in stock_information_update :
110
111                 if stock[0] == stock_update[0] :
112                     purchase_date = datetime.strptime(stock[2], "%m/%d/%Y")
113                     x_axis_date = datetime.strptime(stock_update[1], "%d-%b-%y")
114                     if (purchase_date - x_axis_date).days <= 0 :
115                         current_stock_value = float(stock[1])*float(stock_update[2])
116                         stock_value_y_axis.append(round(current_stock_value,2))
117                         dates_x_axis.append(x_axis_date)
118             plt.plot(dates_x_axis, stock_value_y_axis, label=stock[0])
119
120         plt.legend(loc="upper left")
121
122         plt.savefig('simplePlot.png')
```

```
591
592
593
594     #  Runs different analysis on stock data.
595     New_investor.stock_value_analysis_view()
596
597     stocks_INFO_report.call_stock_API_closing_dates('RDS-A','1d','1y')
598
599     stocks_INFO_report.predict_close_LinReg('RDS-A', 1000)
600
601
```

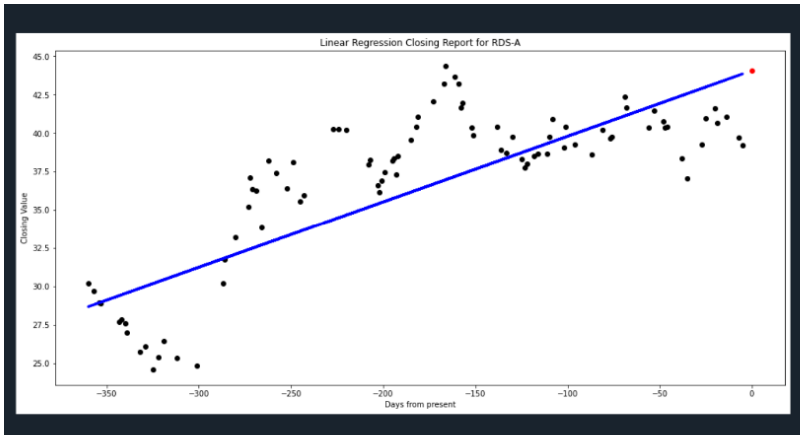


Running Linear Regression

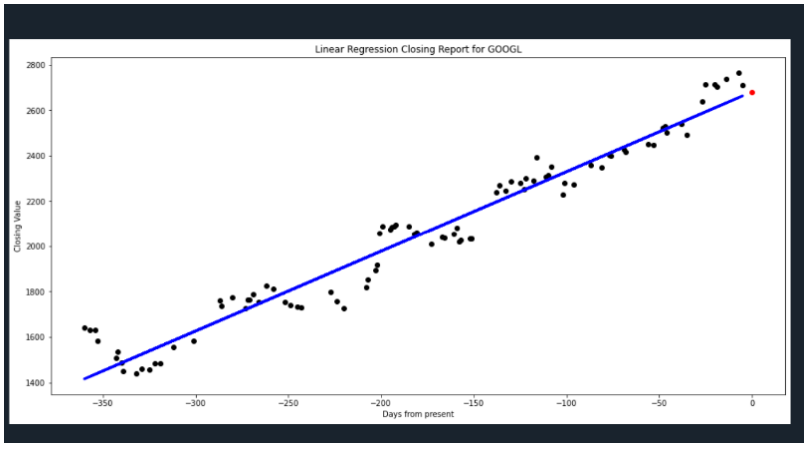
Here, to run linear regression on the closing prices of a stock, all you have to do is call the static function, then pass the name of the stock and the number of previous days stock information to collect as data points. Here are a few cool examples,

The black dots are the testing set, the blue line is the regression line, and the red dot is today's closing price estimate for that stock.

```
#  Runs different analysis on stock data.
New_investor.stock_value_analysis_view()

stocks_INFO_report.call_stock_API_closing_dates('RDS-A','1d','1y')

stocks_INFO_report.predict_close_LinReg('RDS-A', 1000)
```




```
#  Runs different analysis on stock data.
New_investor.stock_value_analysis_view()

stocks_INFO_report.call_stock_API_closing_dates('GOOGL','1d','1y')

stocks_INFO_report.predict_close_LinReg('GOOGL', 1000)
```




```
#  Runs different analysis on stock data.
New_investor.stock_value_analysis_view()

stocks_INFO_report.call_stock_API_closing_dates('AIG','1d','1y')

stocks_INFO_report.predict_close_LinReg('AIG', 10000)
```