DELHI PUBLIC BANGALORE SOUTH

TIME-SERIES FORECASTING

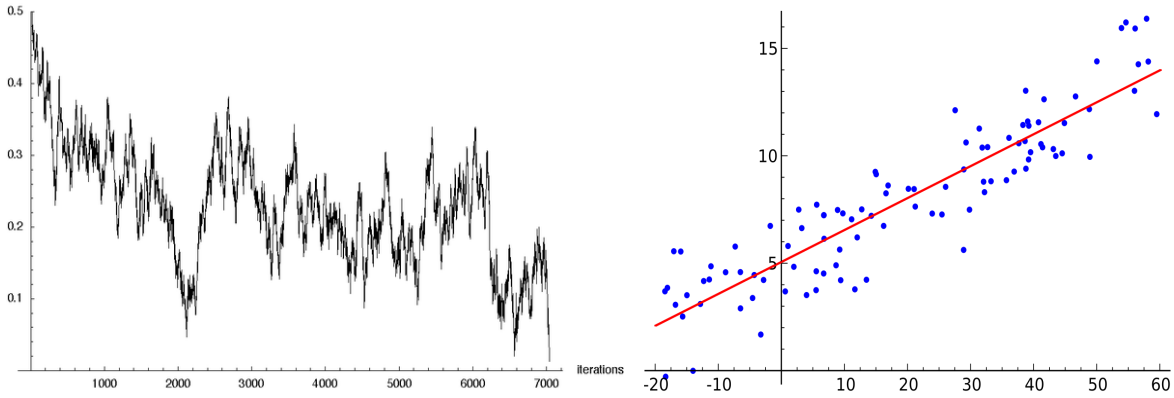
COMPUTER SCIENCE PROJECT

XII-B

By: Ekanshh Praveen, Kaustubh Mukherjee, R Rohit Bharadwaj

Introduction

A time-series is a type of series where the data points are graphed in time order (value vs time graph) at equal successions of time. Now because it is in time order, the values are discrete and there generally isn't a pattern or equation for the entirety of the graph. This is the main reason why techniques like linear regression won't be able to forecast the data and give accurate predictions. So for analyzing a time-series, we need a specific type of model that will be able to give accurate predictions.



The ARIMA Model

The Auto-Regressive Integrated Moving Average (ARIMA) model is one such model that can give accurate predictions while analyzing a time-series. It comprises of three main parts which are :

- Auto-Regression (AR)
- Differencing (D) / Integrated (I)
- Moving Average (MA)

Using the aforementioned processes, we would be able to get a 'future' value of the time-series based on its own past values. Now let us understand each process individually.

1. **Auto Regression (AR)**
   Auto-regression is a time series model that uses observations from previous time steps as input to a regression equation to predict the value at the next time step.

It can only be used for linear (stationary graphs) and cannot be directly used in a time-series.

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} + \epsilon_1$$

Sample Equation for Auto-Regression

2. **Integrated (I) / Differenced (D)**
   As the series needs to be stationary (linear) for Auto-Regression to work, we subtract the (n-1)th values from the nth values. This is known as First-Order Differenced Data which will make the graph stationary.

3. **Moving Average (MA)**
   The moving average (MA) is a simple technical analysis tool that smooths out price data by creating a constantly updated average price. The average is taken over a specific period of time. The Moving Average in the ARIMA model depends on the forecast errors of the Auto-Regressive Model.

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$

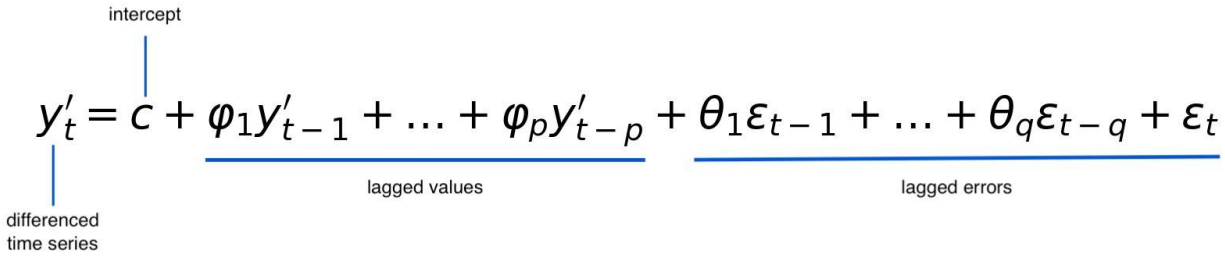
Sample Equation for Moving Average

How the ARIMA Model Works?

An ARIMA model is characterized by 3 terms: p, d, q where,

- p is the order of the AR term
- q is the order of the MA term
- d is the number of differencing required to make the time series stationary

The ARIMA model takes different values of (p,d,q) to give out accurate results for a time-series.



The ARIMA Equation (a combination of three processes)

What we used:

Pandas Module

Pandas is an open-source library in python that specializes in Data Analysis. It allows importing data from various file formats such as CSV, JSON, SQL, Microsoft Excel. Pandas also allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

Pandas Functions and Methods Used in The Project:

- panda.read_csv(file) - reads a comma-separated values (csv) file into DataFrame
- pandas.plotting.lag_plot(series,lag) - plots a lag plot for time series

- pandas.DataFrame(data) - converts data into a two-dimensional, size-mutable form that is tabular and can have arithmetic operations
- pandas.DataFrame.diff(period=1) - calculates the difference of a Dataframe element compared with another element in the Dataframe (default is element in previous row)
- pandas.DataFrame.to_numpy() - converts the DataFrame to a NumPy array

Matplotlib and PyPlot Module

Matplotlib is a plotting library for Python and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython and Qt.

PyPlot is a state-based interface to Matplotlib. It provides a MATLAB-like way of plotting. PyPlot is mainly intended for interactive plots and simple cases of programmatic plot generation

Matplotlib and PyPlot Functions Used in The Project:

- matplotlib.pyplot - module that is mainly intended for interactive plots and simple cases of programmatic plot generation
- pyplot.figure() - creates a new figure/graph
- pyplot.plot(data) - plots data into the figure
- pyplot.show() - displays an existing graph
- pyplot.xticks([ticks,data]) - sets the current tick values and labels for the x-axis

Statsmodels Module

Statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. The results are tested against existing statistical packages to ensure that they are correct.

The ARIMA Statistical Model used in the project was a standard class from the statsmodels library.

statsmodels.tsa.arima_model.ARIMA(data,[p,d,q]) - gives the module for the ARIMA model
ARIMA.fit() - used to estimate different parameters for the model
ARIMA.fit.forecast() - used to predict values for the model

YFinance Module

Ever since Yahoo! finance decommissioned their historical data API
yfinance aims to solve this problem by offering a reliable, threaded, and Pythonic way to download historical market data from Yahoo! Finance.

YFinance Functions and Methods Used:

- yfinance.Ticker(<stock name on Yahoo! finance>) - creates a ticker object for the particular stock from the yahoo finance page so data can be extracted
- <yfinance.Ticker.ticker>.history(<start date>, <end date>) - getting the stock data for the particular dates

KIVY

Kivy is a free and open source Python framework for developing mobile apps and other multitouch application software with a natural user interface. Using this module we were able to create an app for our project.

Classes Used in The Project:

- Class MainWindow(Screen) - used to create and add all values in the main window of the app
- Class SecondWindow(Screen) - used to create and add all values in the second window of the app
- WindowManager(ScreenManager) - used to arrange and take care of the transition between the two screens
- Class MyMainApp(App) - used to run the whole program

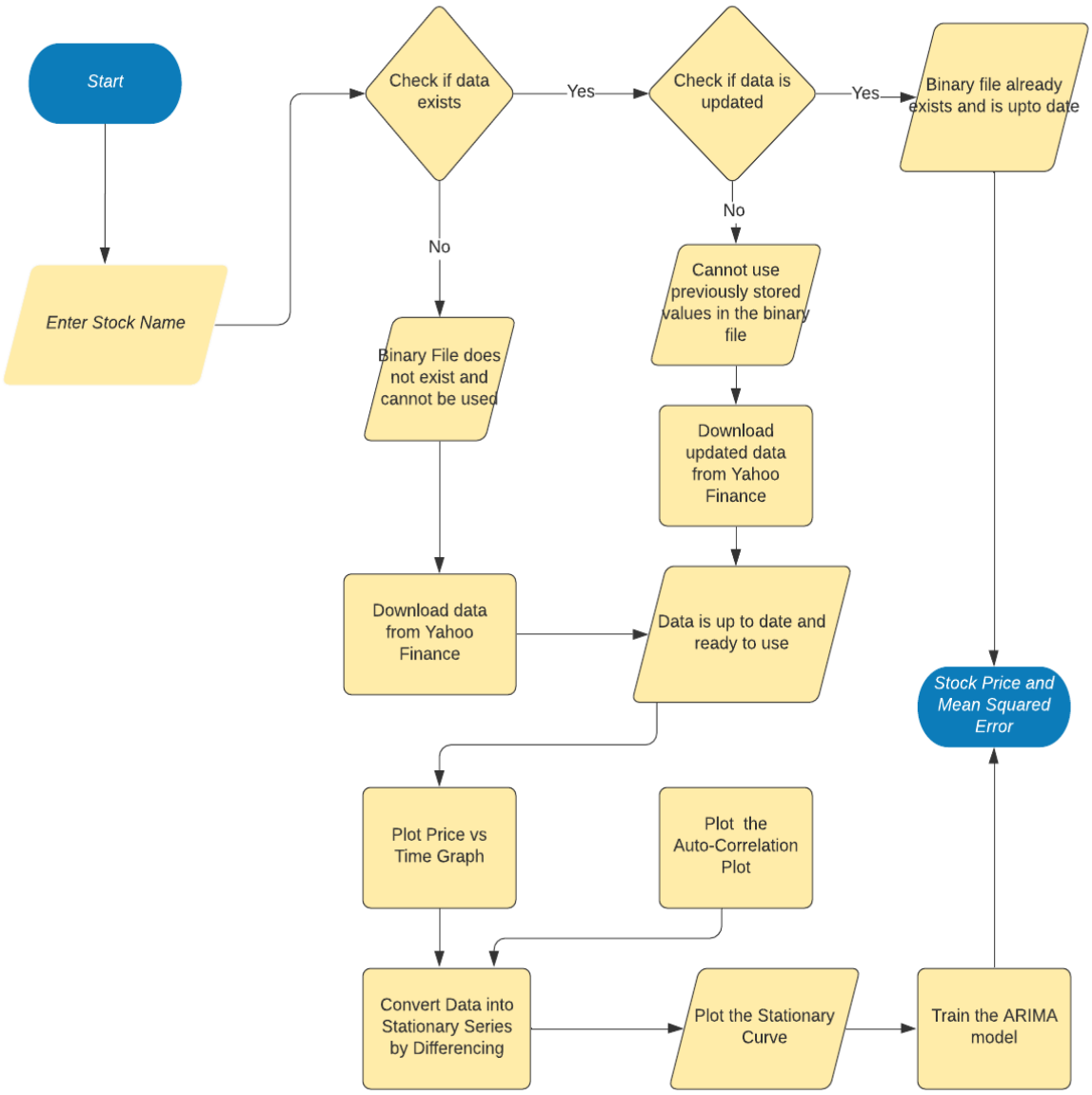
The Kv Language

The Kv language is a language dedicated to describing user interface and interactions in Kivy framework. As with other interface markup languages, it is possible to easily create a whole UI and attach interaction.

Kv Language Commands Used in The Project:

- root.press() - executes the function if widget is pressed
- app.root.current - sets the app to move to the desired screen
- Root.manager.transition.direction - makes a basic animation/transition in the mentioned direction
- root.clear() - clears the screen

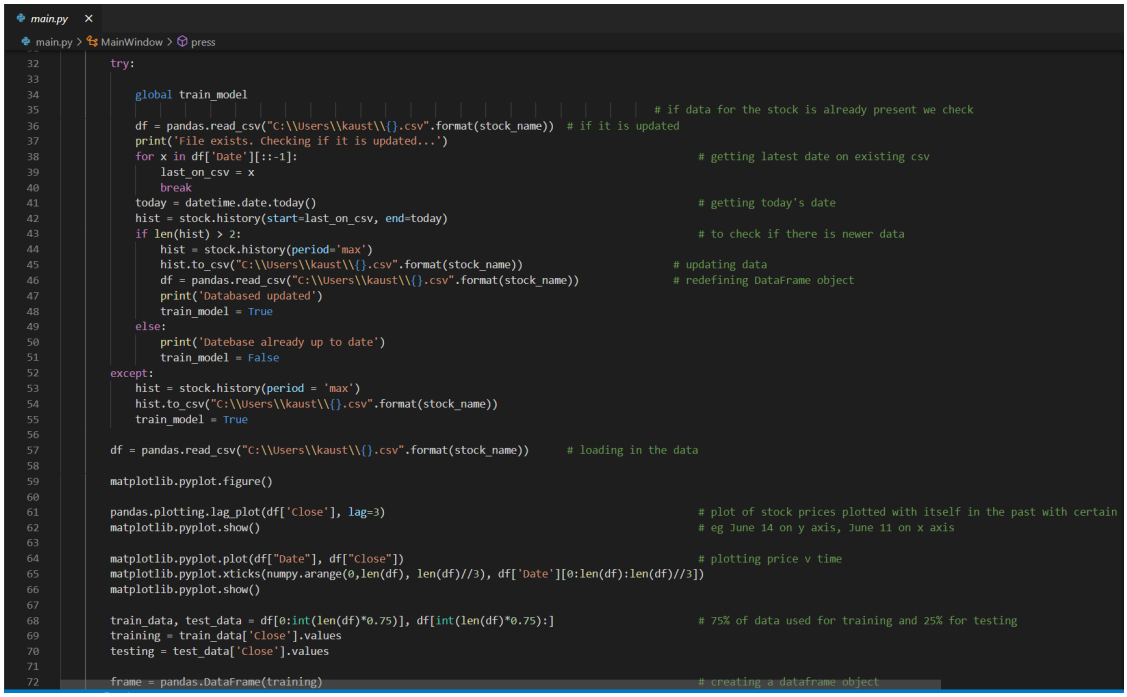
The Flow of Data:



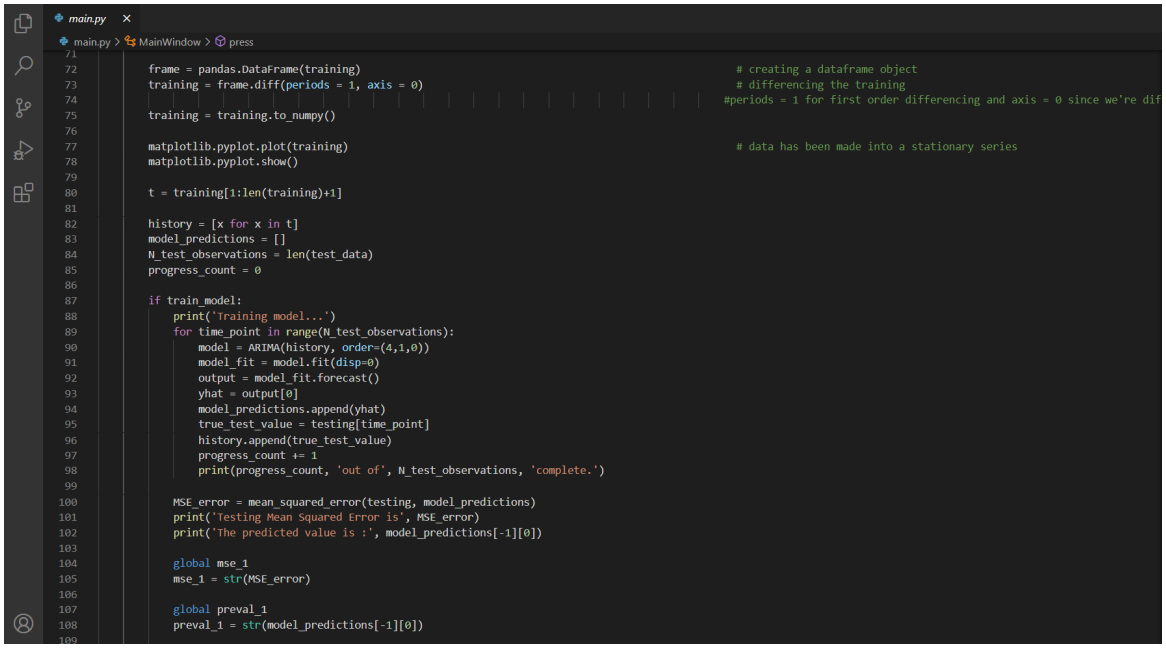
Our Project

We used the ARIMA Model to try and predict the Future Price of a company's stock using historical data that was available on Yahoo Finance.

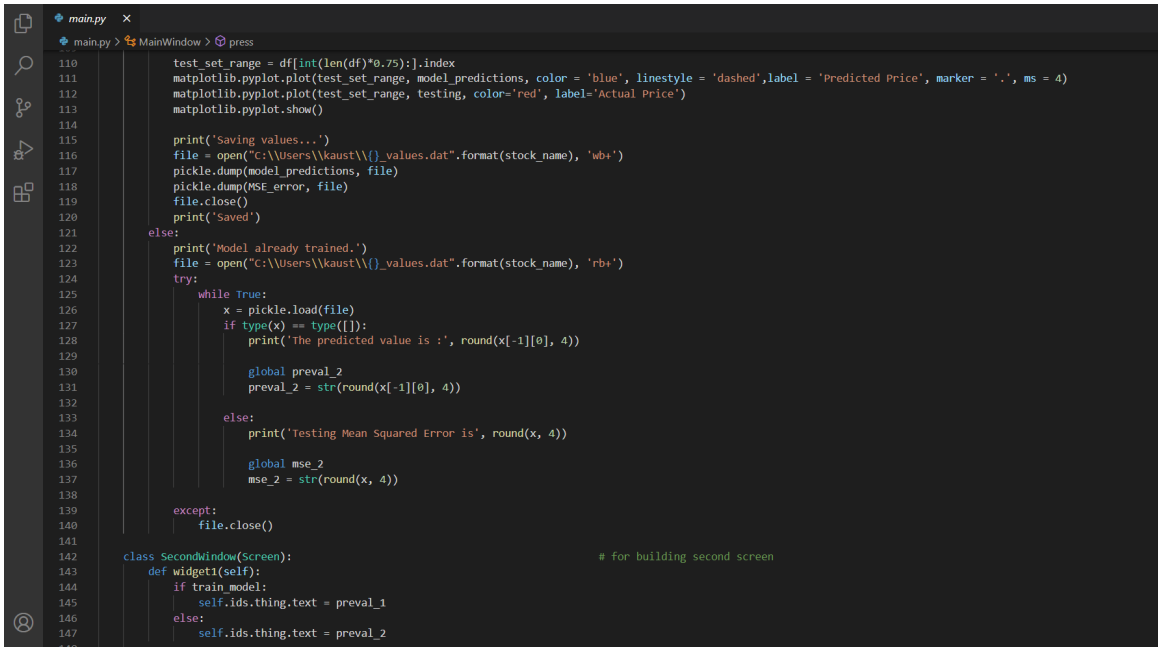
The Code:



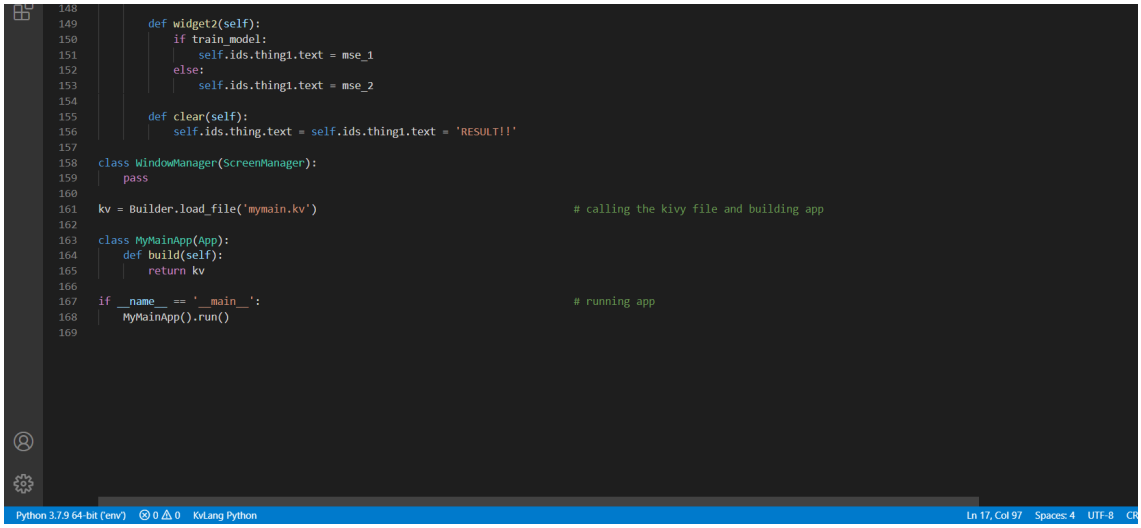
```
main.py
main.py > MainWindow > press
32      try:
33
34          global train_model
35                                                                  # if data for the stock is already present we check
36          df = pandas.read_csv("C:\\Users\\kaust\\{}.csv".format(stock_name))  # if it is updated
37          print('File exists. Checking if it is updated...')
38          for x in df['Date'][::-1]:                                          # getting latest date on existing csv
39              last_on_csv = x
40              break
41          today = datetime.date.today()                                       # getting today's date
42          hist = stock.history(start=last_on_csv, end=today)
43          if len(hist) > 2:                                                   # to check if there is newer data
44              hist = stock.history(period='max')
45              hist.to_csv("C:\\Users\\kaust\\{}.csv".format(stock_name))      # updating data
46              df = pandas.read_csv("C:\\Users\\kaust\\{}.csv".format(stock_name))  # redefining DataFrame object
47              print('Databased updated')
48              train_model = True
49          else:
50              print('Datebase already up to date')
51              train_model = False
52      except:
53          hist = stock.history(period = 'max')
54          hist.to_csv("C:\\Users\\kaust\\{}.csv".format(stock_name))
55          train_model = True
56
57      df = pandas.read_csv("C:\\Users\\kaust\\{}.csv".format(stock_name))      # loading in the data
58
59      matplotlib.pyplot.figure()
60
61      pandas.plotting.lag_plot(df['Close'], lag=3)                            # plot of stock prices plotted with itself in the past with certain
62      matplotlib.pyplot.show()                                                # eg June 14 on y axis, June 11 on x axis
63
64      matplotlib.pyplot.plot(df["Date"], df["Close"])                         # plotting price v time
65      matplotlib.pyplot.xticks(numpy.arange(0,len(df), len(df)//3), df['Date'][0:len(df):len(df)//3])
66      matplotlib.pyplot.show()
67
68      train_data, test_data = df[0:int(len(df)*0.75)], df[int(len(df)*0.75):]  # 75% of data used for training and 25% for testing
69      training = train_data['Close'].values
70      testing = test_data['Close'].values
71
72      frame = pandas.DataFrame(training)                                      # creating a dataframe object
```



```
main.py
main.py > MainWindow > press

        frame = pandas.DataFrame(training)                                  # creating a dataframe object
        training = frame.diff(periods = 1, axis = 0)                        # differencing the training
                                                                           #periods = 1 for first order differencing and axis = 0 since we're dif
        training = training.to_numpy()

        matplotlib.pyplot.plot(training)                                    # data has been made into a stationary series
        matplotlib.pyplot.show()

        t = training[1:len(training)+1]

        history = [x for x in t]
        model_predictions = []
        N_test_observations = len(test_data)
        progress_count = 0

        if train_model:
            print('Training model...')
            for time_point in range(N_test_observations):
                model = ARIMA(history, order=(4,1,0))
                model_fit = model.fit(disp=0)
                output = model_fit.forecast()
                yhat = output[0]
                model_predictions.append(yhat)
                true_test_value = testing[time_point]
                history.append(true_test_value)
                progress_count += 1
                print(progress_count, 'out of', N_test_observations, 'complete.')

            MSE_error = mean_squared_error(testing, model_predictions)
            print('Testing Mean Squared Error is', MSE_error)
            print('The predicted value is :', model_predictions[-1][0])

            global mse_1
            mse_1 = str(MSE_error)

            global preval_1
            preval_1 = str(model_predictions[-1][0])
```

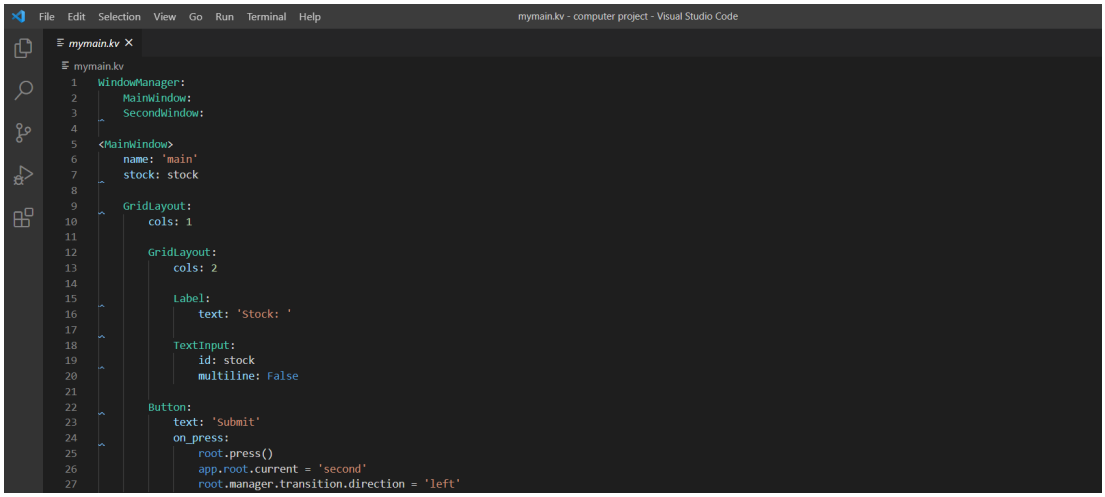


```
main.py
main.py > MainWindow > press

            test_set_range = df[int(len(df)*0.75):].index
            matplotlib.pyplot.plot(test_set_range, model_predictions, color = 'blue', linestyle = 'dashed',label = 'Predicted Price', marker = '.', ms = 4)
            matplotlib.pyplot.plot(test_set_range, testing, color='red', label='Actual Price')
            matplotlib.pyplot.show()

            print('Saving values...')
            file = open("C:\\Users\\kaust\\{}_values.dat".format(stock_name), 'wb+')
            pickle.dump(model_predictions, file)
            pickle.dump(MSE_error, file)
            file.close()
            print('Saved')
        else:
            print('Model already trained.')
            file = open("C:\\Users\\kaust\\{}_values.dat".format(stock_name), 'rb+')
            try:
                while True:
                    x = pickle.load(file)
                    if type(x) == type([]):
                        print('The predicted value is :', round(x[-1][0], 4))

                        global preval_2
                        preval_2 = str(round(x[-1][0], 4))

                    else:
                        print('Testing Mean Squared Error is', round(x, 4))

                        global mse_2
                        mse_2 = str(round(x, 4))

            except:
                file.close()

    class SecondWindow(Screen):                                     # for building second screen
        def widget1(self):
            if train_model:
                self.ids.thing.text = preval_1
            else:
                self.ids.thing.text = preval_2
```



```
148
149         def widget2(self):
150             if train_model:
151                 self.ids.thing1.text = mse_1
152             else:
153                 self.ids.thing1.text = mse_2
154
155         def clear(self):
156             self.ids.thing.text = self.ids.thing1.text = 'RESULT!!'
157
158 class WindowManager(ScreenManager):
159     pass
160
161 kv = Builder.load_file('mymain.kv')                    # calling the kivy file and building app
162
163 class MyMainApp(App):
164     def build(self):
165         return kv
166
167 if __name__ == '__main__':                             # running app
168     MyMainApp().run()
169
```

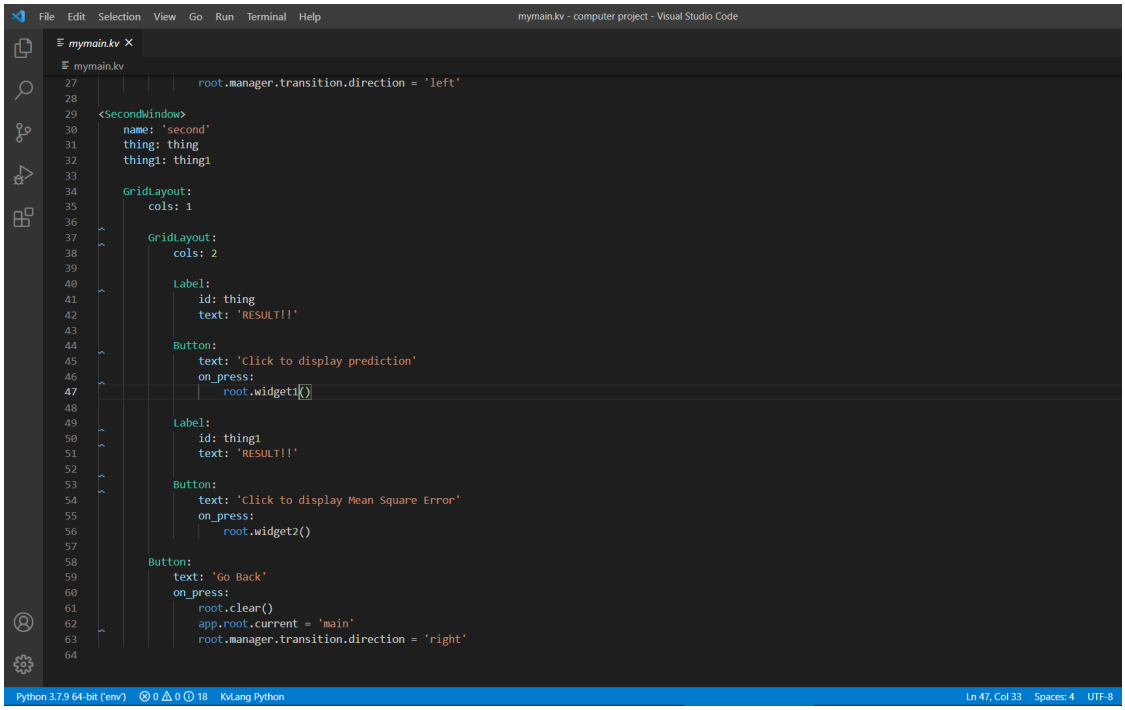
Python 3.7.9 64-bit ('env') ⊗ 0 ⚠ 0 KvLang Python Ln 17, Col 97 Spaces: 4 UTF-8 CR



File Edit Selection View Go Run Terminal Help mymain.kv - computer project - Visual Studio Code

mymain.kv

```
1  WindowManager:
2      MainWindow:
3      SecondWindow:
4
5  <MainWindow>
6      name: 'main'
7      stock: stock
8
9      GridLayout:
10         cols: 1
11
12         GridLayout:
13             cols: 2
14
15             Label:
16                 text: 'Stock: '
17
18             TextInput:
19                 id: stock
20                 multiline: False
21
22         Button:
23             text: 'Submit'
24             on_press:
25                 root.press()
26                 app.root.current = 'second'
27                 root.manager.transition.direction = 'left'
```

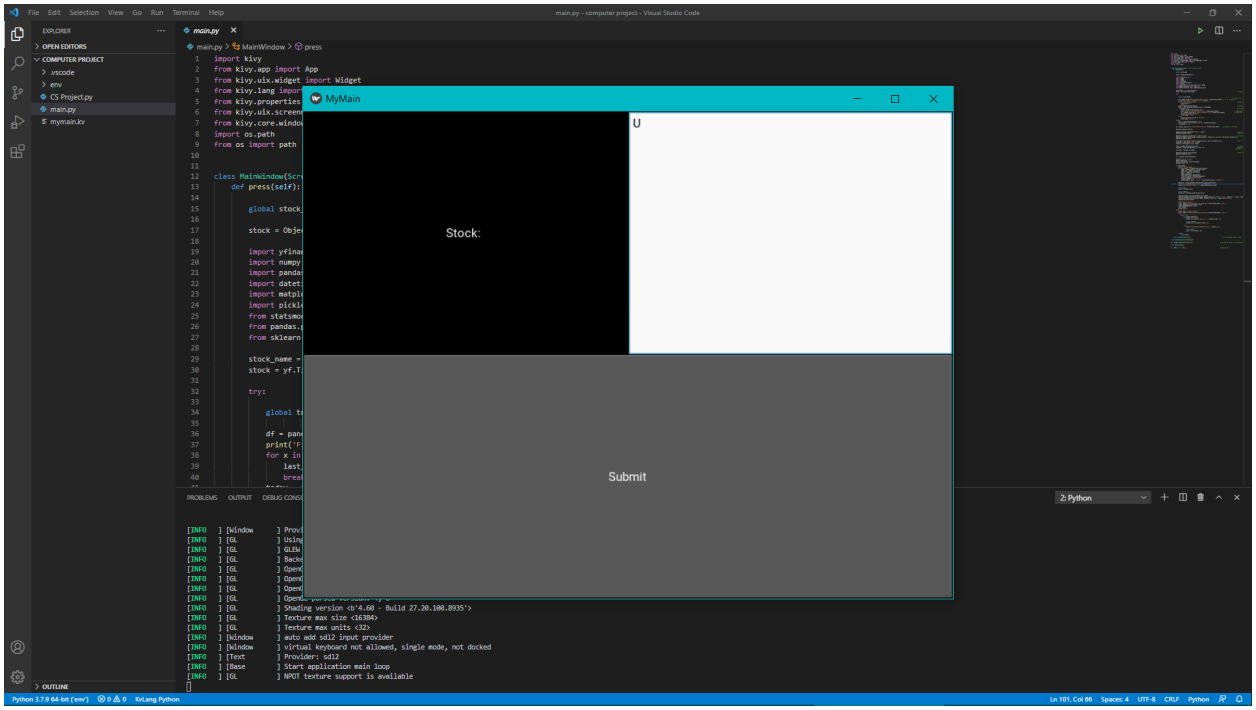


The Output:

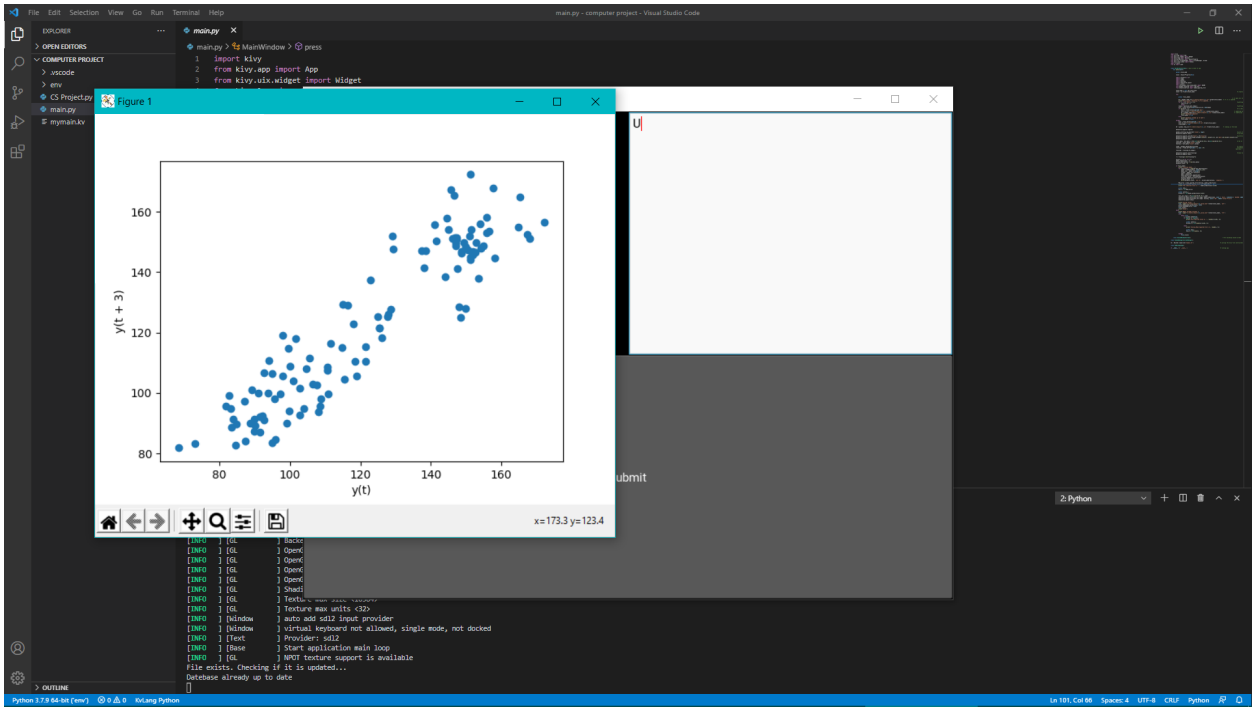
We have taken five different Stocks with different case-scenarios and hae attached them below.

Case 1 - Unity:

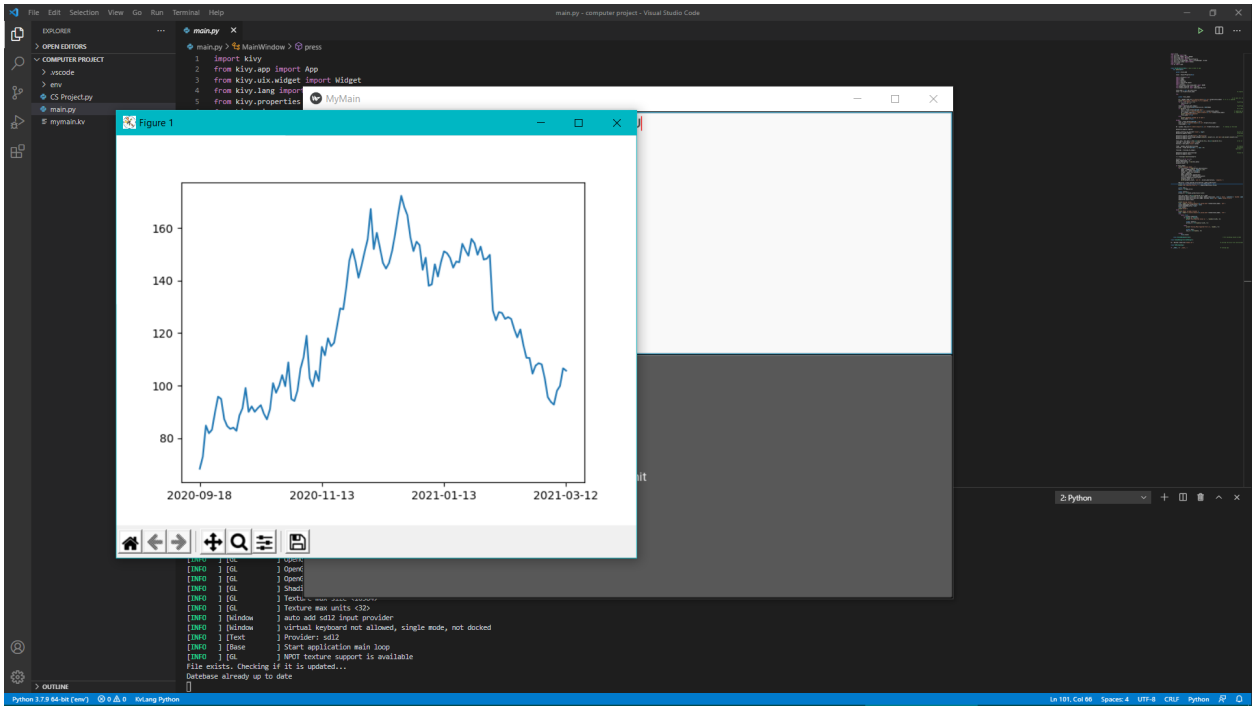
NYSE Stock Name - U



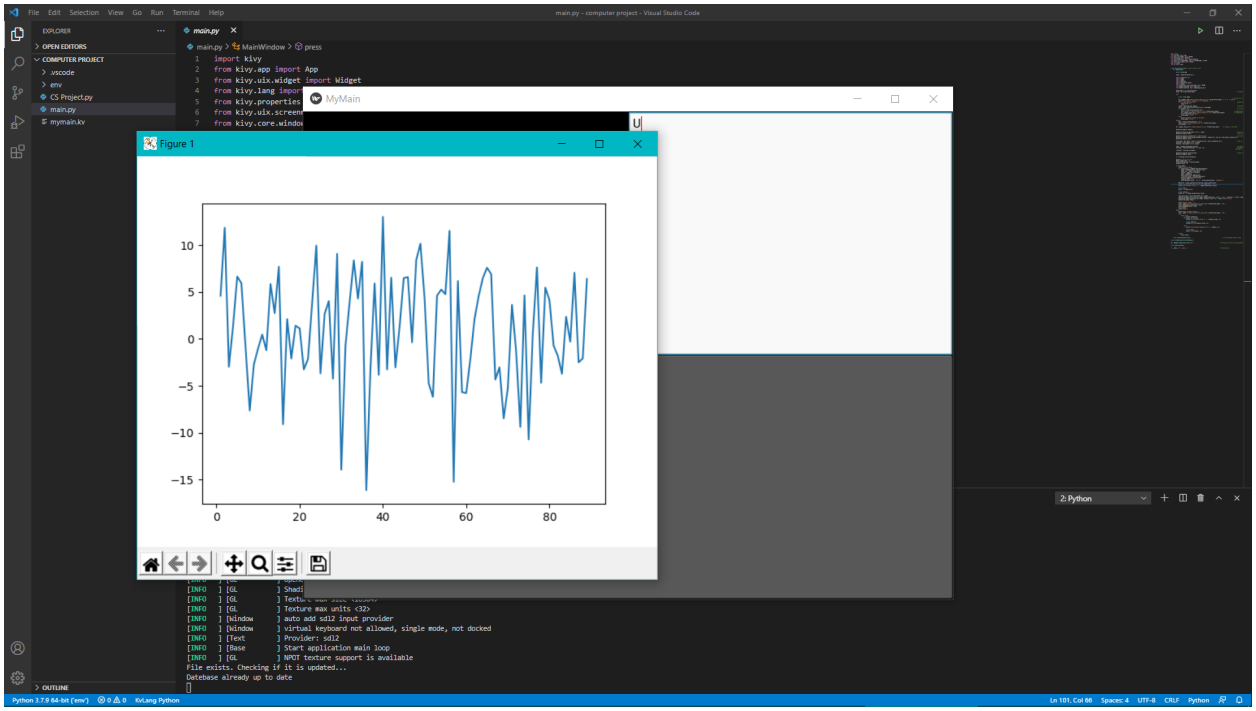
Start/Input Window



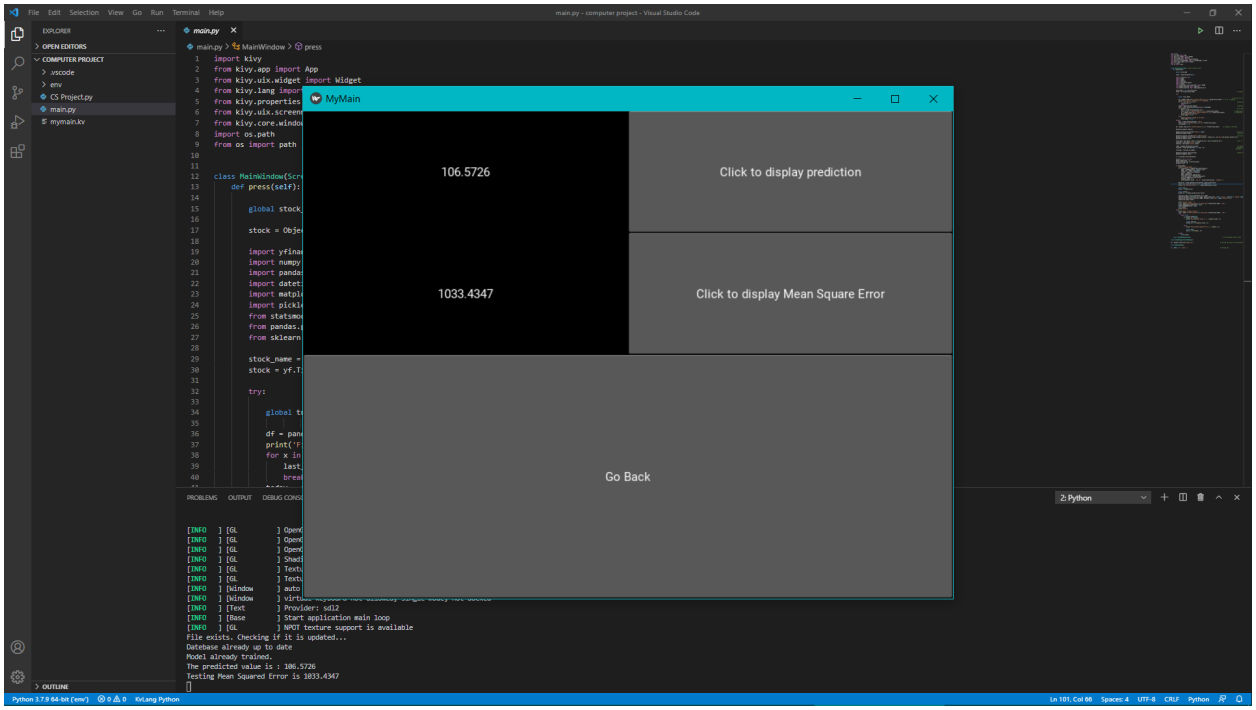
Auto-Correlation Graph



Price vs Time Graph



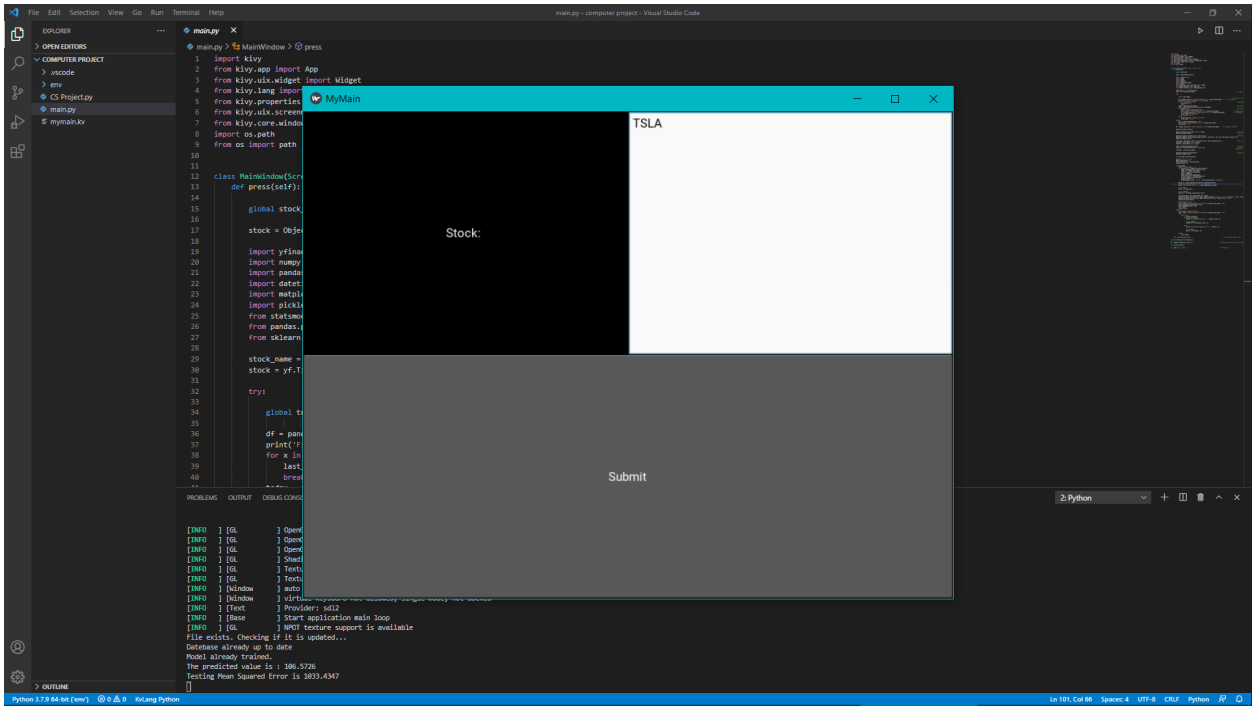
Stationary Curve Graph



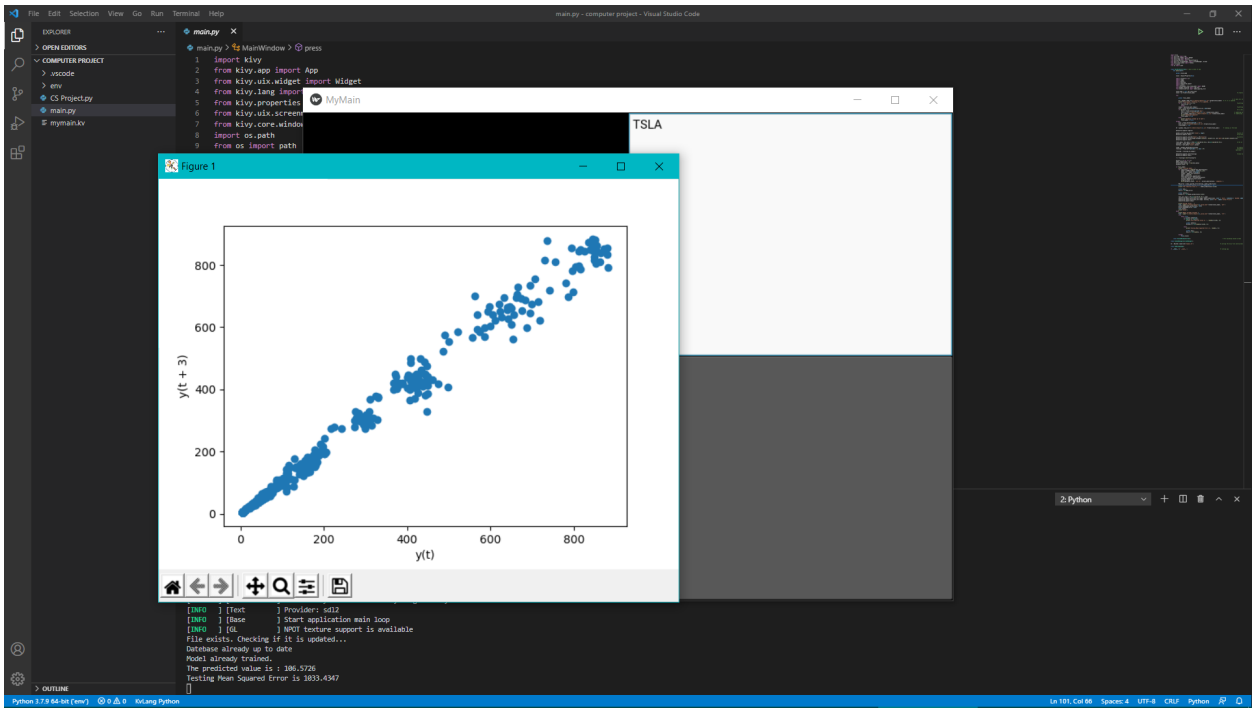
Result Window

Case 2 - Tesla:

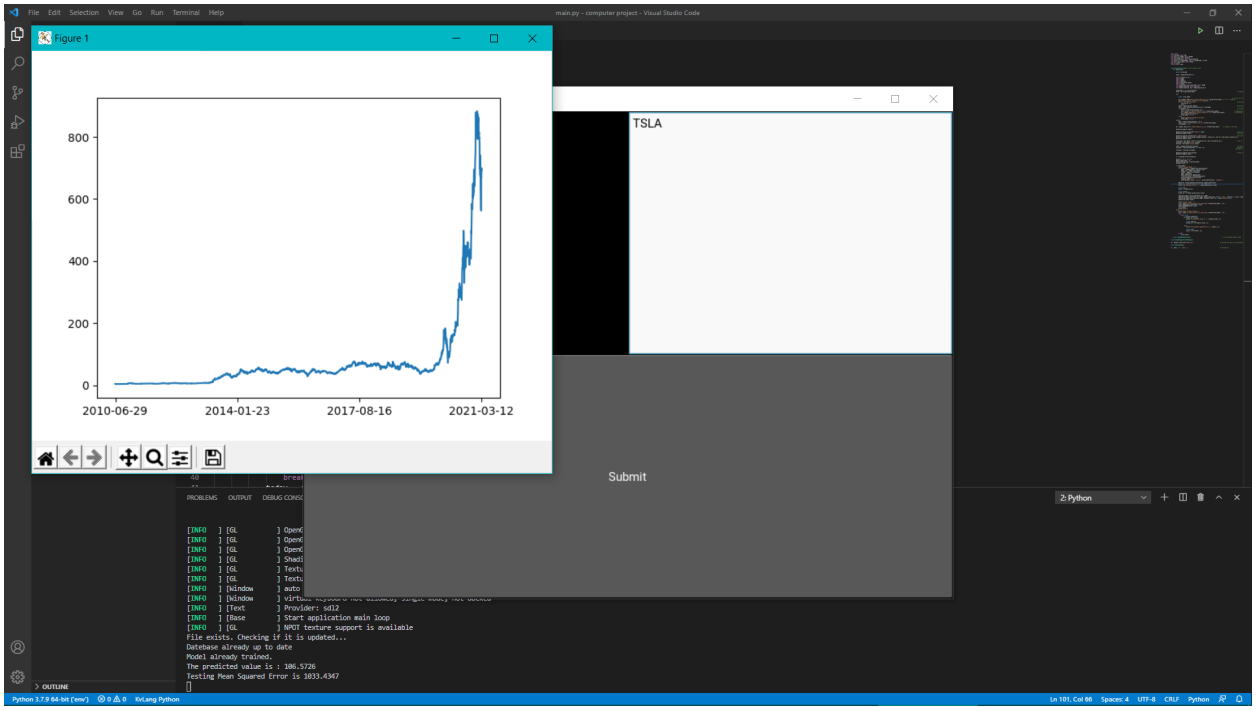
NYSE Stock Name - TSLA



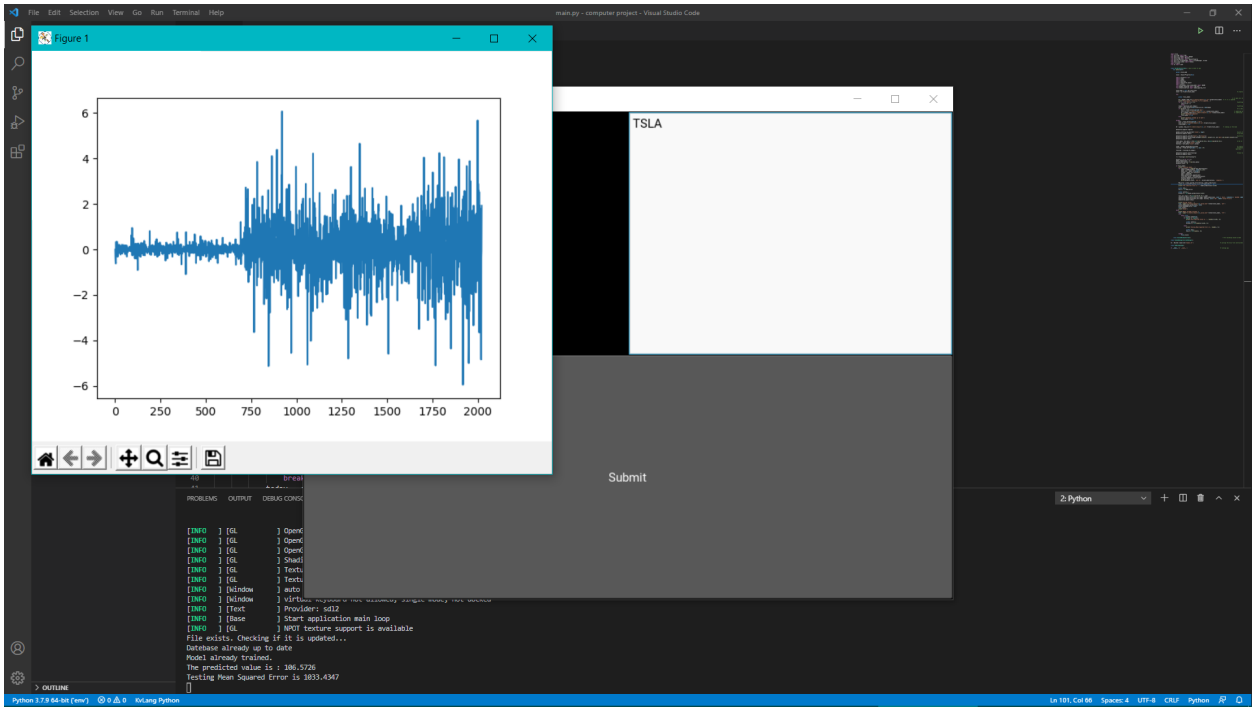
Start/Input Window



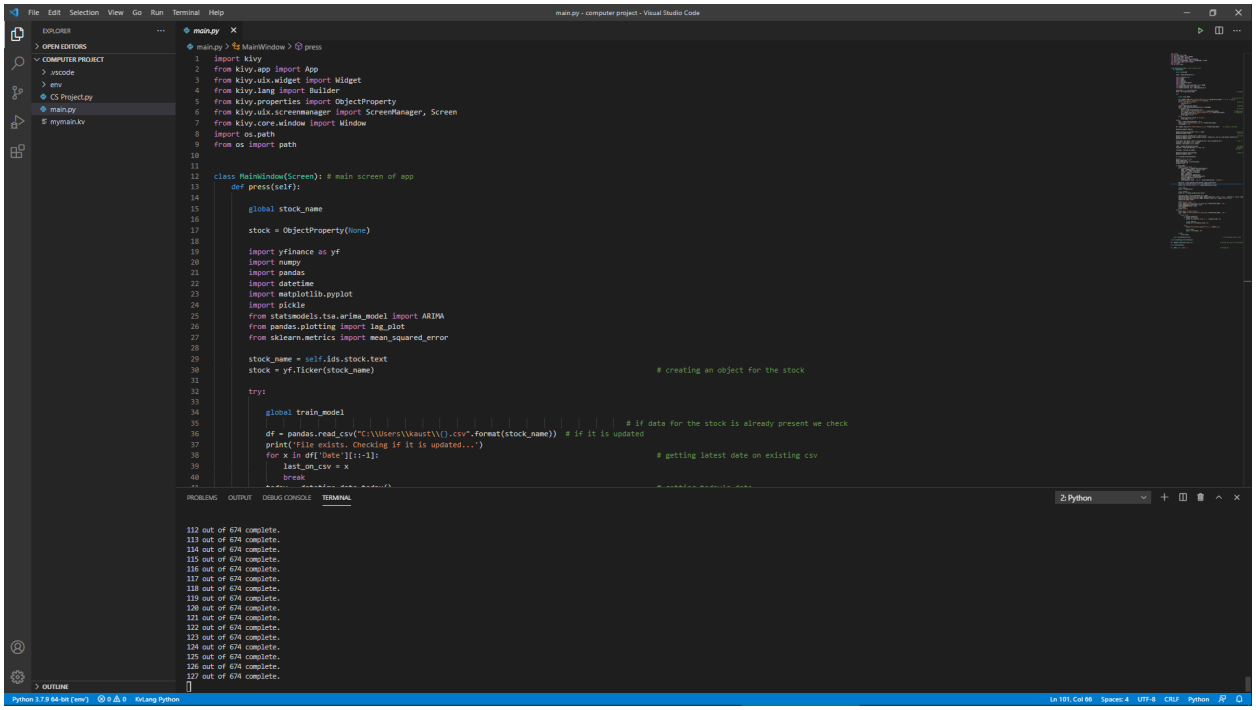
Auto-Correlation Graph



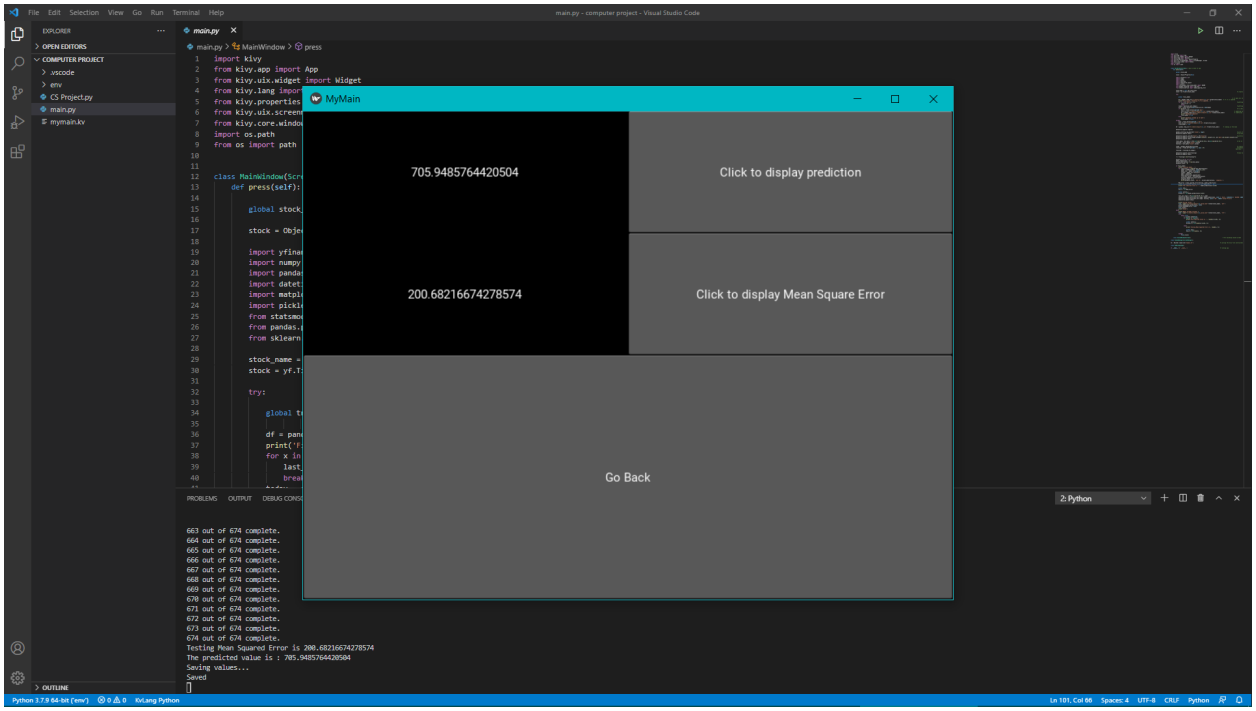
Price vs Time Graph



Stationary Curve Graph



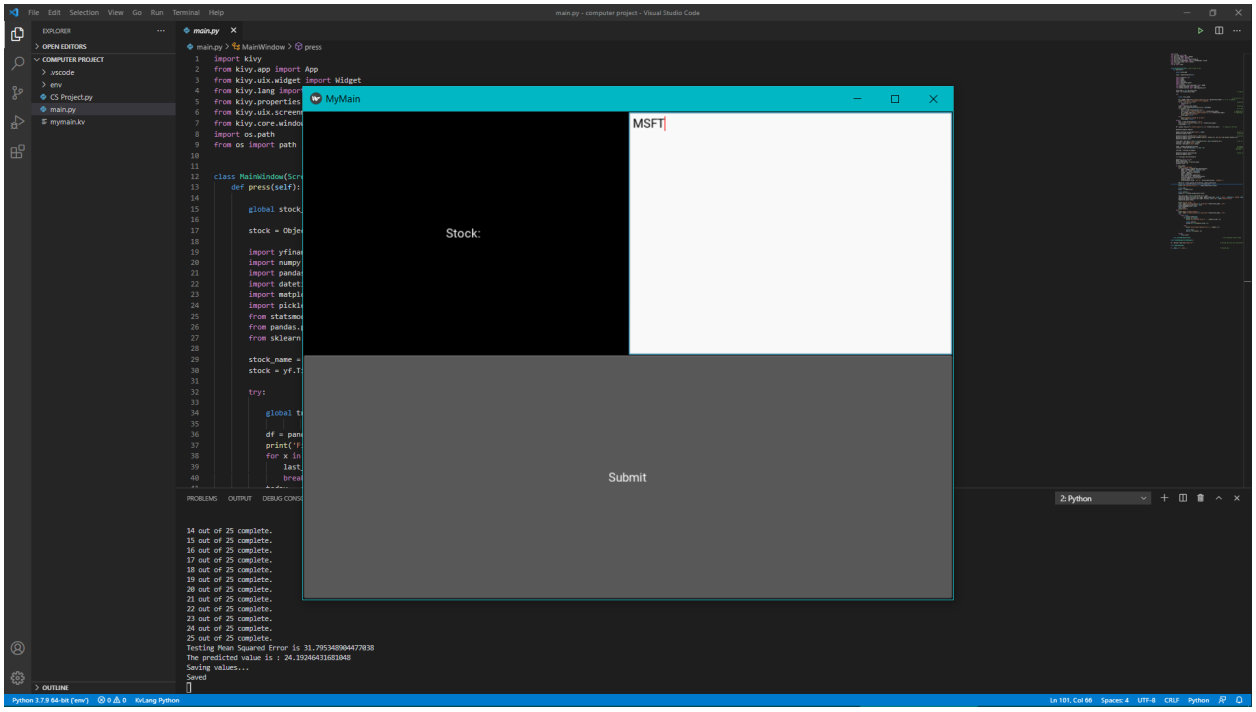
Data Training



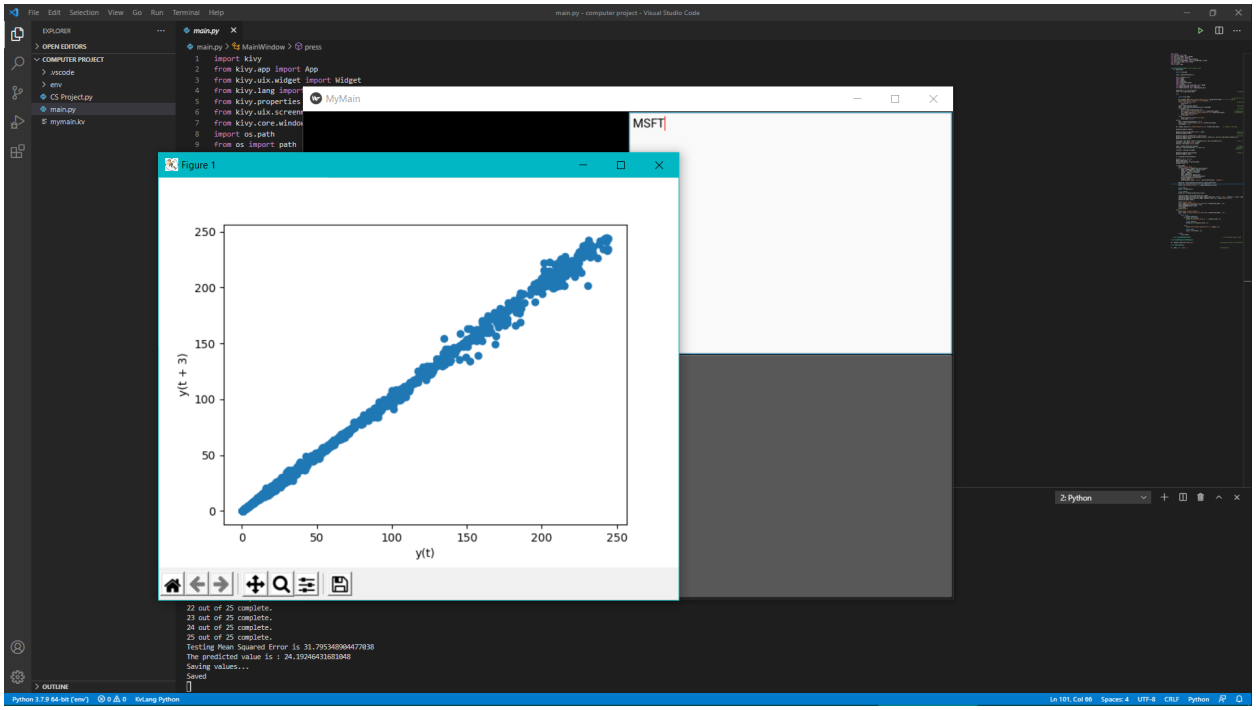
Result Window

Case 3 - Microsoft:

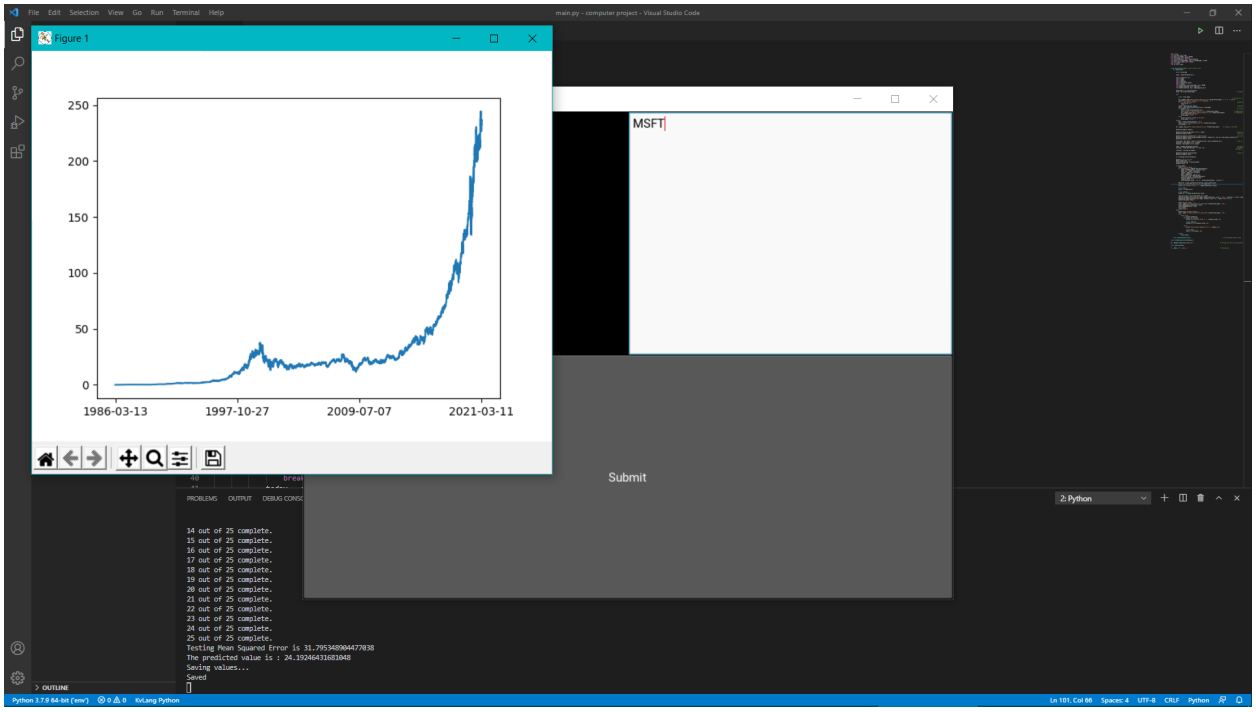
NYSE Stock Name - MSFT



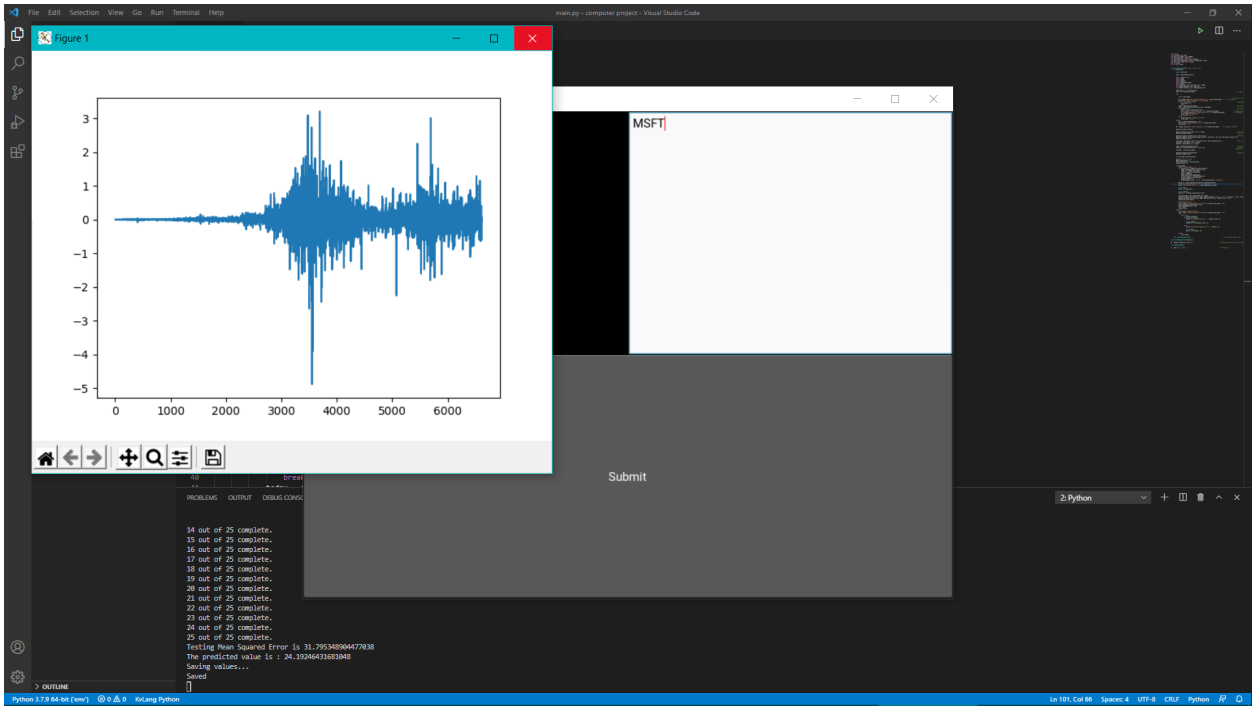
Start/Input Window



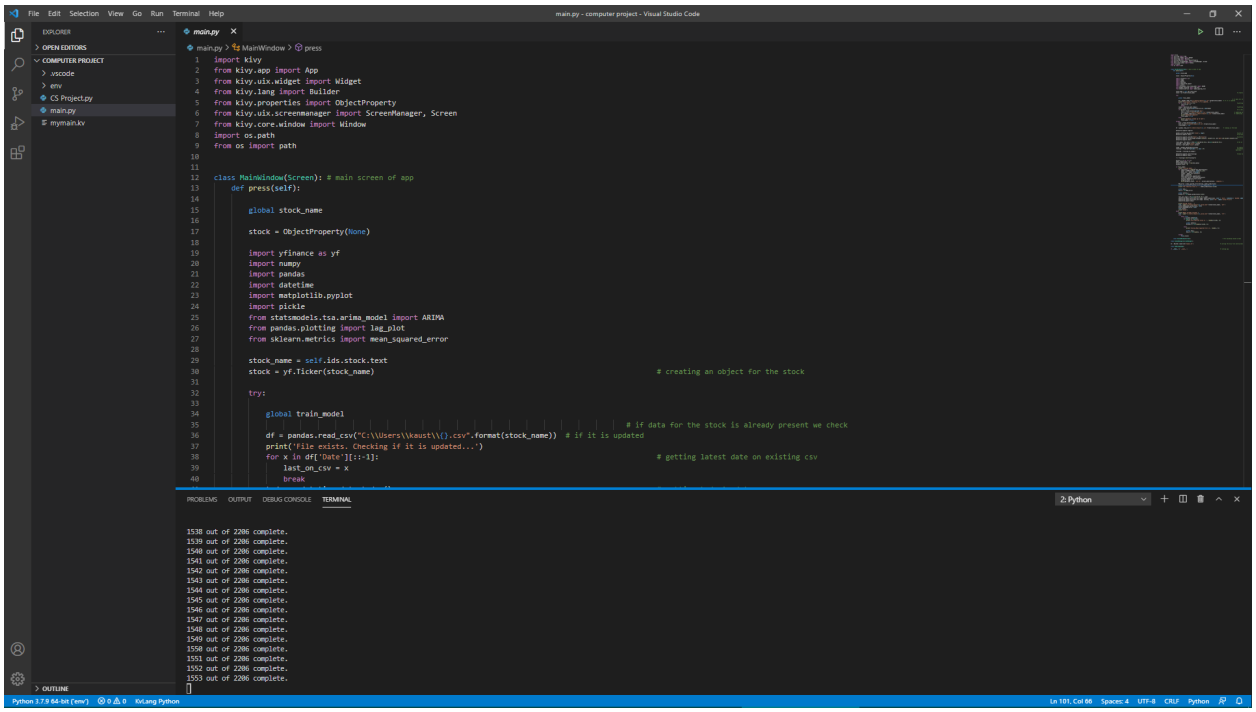
Auto-Correlation Graph



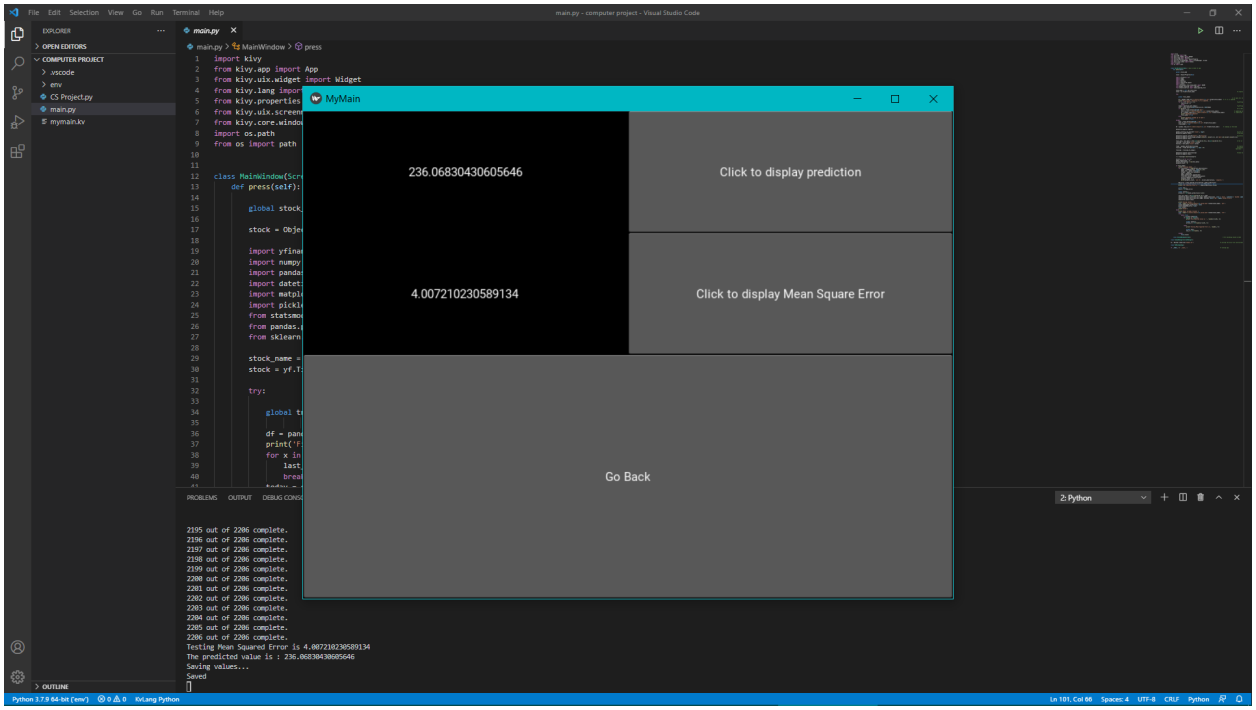
Price vs Time Graph



Stationary Curve Graph



Data Training



Result Window