Big Data Analytics on Stocks Prediction

Master's Project
Department of Computer Science

Advisor: Dr. Stefan Robila
Student: Krishnamurthy Gurum
Montclair State University
Fall 2016

DECEMBER 20, 2016

Table of Contents:

List of Figures

ABSTRACT

In Present scenario “Stock market prediction” is the act of trying to determine the future value of a company stocks or other financial instruments traded on an exchange. The successful prediction of a stock's future price could yield significant profit.

Stock market plays a pivotal role in financial aspect of nations growth, but its highly volatile. Everyday millions of dollars are spent and few investors lose and others gain. The demand to predict stocks is high, but at the same time the predicted values are highly inconsistent.

The project gives clear thought for general stakeholder's, the way to put into a specific stocks. They will be guided by this application, climate the stock is useful for number of days or number of months or for the long term from taking a gander at the historical backdrop of the stocks furthermore they can think about which stock is better on the off chance that they need to judge between two stocks.

The goal of project is, the Stakeholder’s will be helped in understanding the stock of a particular company over the years or months or days which would help an investor to decide on whether he should invest or not, and helped to compare which stock is the best to invest. This project which will help the stakeholder’s more towards the success rate..

ACKNOWLEDGMENT

This project is made possible through the help and support from everyone including parents, teachers, family and friends.

First I would like to thank Dr. Stefan Robila for not only providing me with an opportunity to work with him, but also for his support and encouragement. His mentorship and guidance contributed immensely to the implementation of this project. He encouraged me at my difficult times, during the process of this Project. My Hearty Thanks for all your support Professor.

I would also like to extend my gratitude towards

Dr. Hubert A Johnson: The Software Engineering course helped me on implementing my Project with Proper Documentation

Dr. John Jenq: The Web Development course initiated interest towards Python Programming Language.

Thanks for your support and guidance Professors throughout my studies and project at Montclair State University. Lastly, I would like to thank the faculty of the Department of Computer Science for their support during my Master's program.

1. Introduction

This chapter provides an overview of Stock, Stock exchange and how it works.

1.1 Motivation

Stock value prediction is an interesting field of research in many fields like trading, finance, statistics and computer science.

Stock value prediction helps in making an attempt to predict the future estimate of an organization stock or other budgetary instrument exchanged on a money related trade.

Stock Prices are known to be very dynamic and susceptible to quick changes because of the underlying nature of financial domain and in part because of mix of previous day parameters like closing price, P/E ratio etc. and unknown factors like politics, famine etc.

For people looking for good investments or stock trading, stock value prediction plays a pivotal role in determining where to put the money OR which stock is to be acquired or sold.

While the monetary markets are exceptionally mind boggling, deciding the best parts of a fruitful stockis not an easy task. Investors are familiar in saying - "purchase low, offer high" yet this does not give enough context to make proper investment decisions. Every investor dreams of having a prior knowledge of the direction of market, but it’s incredibly difficult to have one.

Interestingly, greater numbers of people are trying to invest their money in mutual funds and/or shares to save taxes. These are all small investments and don’t have a luxury to hire someone like Data Scientist to get the sound knowledge of the market, to make correct decisions.

On the other hand, few people depend entirely on Stock market for their bread and butter. These people typically use fundamental and/or technical analysis to understand the stock market future and make investment decisions.

With the advent of BigData and rapidly increasing capabilities of computers to handle data, programs can perform these predictions at speeds that human broker cannot imagine. This is achieved by having “Enormous data” of the past to determine patterns, trends and predict with higher probability the future of the stock market.

1.2 History

To start with, let us understand what Stock Market is all about.

In the simplest of the terms, stock is a share in the ownership of the company. It represents claim on company's assets and earnings. More the stock, higher the ownership on the respective company by the individual or the owner of the share. This stock is represented by Stock certificate. This is kept by the brokerage electronically, known as holding shares. This is done to make shares easily tradeable.

Why should company issue stock?? The reason is that at some point every company needs to raise money. This can be done by taking a loan from financial institution OR selling part of the company known as *issuing stock*. The first time company issues stock is referred to as IPO (Initial public offering).

For stocks, the market operates as "buyer – seller" trade, where buyers and sellers are matched for a fit. The entry fee into the stock market is as low as $1. The equity market is therefore more active, have many players, and hence more money being circulated.

A small error can lead to big loss or may be big win. Hence this field is worth a study.

Due to such a high risk in Stock markets, shareholders trade by giving instructions to stockbrokers, who in turn execute the orders. Stock brokers advise clients on the current sentiment in the stock market.

Stock broker's base there advises on the fundamentals of various stocks or undertakes technical analysis. None of these methods provide accurate prediction and they are just an indicator of a future trend.

1.2.1Fundamental Analysis

Fundamental analysis[1] is the traditional approach, involving study of company revenues, company expenses, annual growth rates, market position, and so on MURPHY.

It provides answers by considering the questions like –

- Is the company's revenue growing?
- Is it actually making a profit?
- Is it in a strong-enough position to beat out its competitors in the future?

- Is it able to repay its debts?
- Is management trying to "cook the books"?

The stock broker by focusing on a particular business finds out if that business stock is worth buying.

1.2.2Technical Analysis

Technical analysis[2] is based on study of patterns in the historical data. Practitioners of technical analysis, study price charts for price patterns and use price data in future predictions. This happens by feeding enormous amount of historical data into complex mathematical models. Many algorithms like Apriori, FPGrowthetc in conjunction with statistical methods decipher the pattern in the data to provide higher probabilistic predictions.

2.Background and Rationale

Under the broad spectrum of web-based technologies, there is a multitude of programming languages and libraries that can be utilized and tailored to implement this project.
Choosing one among them is a tedious task and I have used Python 3.5 here along with libraries like numpy, pandas, matplotlib, PyQt5 etc. Also, I have used Google data source API to get the earlier stock data.

An overview of these technologies is provided here.

2.1 Python

Python is general purpose interactive, object – oriented, interpreted high level programming language. It was created by Guido van Rossum and is open sourced.

It’s the most preferred language for scientific computing because the learning curve here is very small as compared to other languages. People in scientific field do a lot of things including wiring together web applications, network applications, scripting, automating data processing jobs and some analysis and modelling. All these features and much more are provided Python.

Python also helps in gluing applications written is C, C++, and Fortran, especially using excellent Cython Project.

2.2Numpy

NumPy[3] is the fundamental package for scientific computing with Python. It contains

- powerful N-dimensional array object.
- sophisticated (broadcasting) functions.
- tools for integrating C/C++ and Fortran code.
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the *BSD license*, enabling reuse with few restrictions.

2.3 Pandas

Pandas[4] is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
Some of the features are:

- Data Frame object for data manipulation with integrated indexing
- Tools for reading and writing data between in-memory data structures and different file formats
- Data alignment and integrated handling of missing data
- Reshaping and pivoting of data sets
- Label-based slicing, fancy indexing, and sub setting of large data sets
- Data structure column insertion and deletion
- Group by engine allowing split-apply-combine operations on data sets
- Data set merging and joining
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure
- Time series-functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.

2.4 Matplotlib

Matplotlib[5] is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB®* or Mathematica®†), web application servers, and six graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. We can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc, with just a few lines of code. For a sampling, see the screenshots, thumbnail gallery, and examples directory

For simple plotting the pyplot interface provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

2.5 PyQt5

Qt[6] is set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems. These include location and positioning services, multimedia, NFC and Bluetooth connectivity, a Chromium based web browser, as well as traditional UI development.

PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android.

PyQt5 may also be embedded in C++ based applications to allow users of those applications to configure or enhance the functionality of those applications.

2.6 Google Finance API

Google finance gets real time data as a feed directly from the stock market exchanges. Google provides this data in terms of REST API to the end users any time. This can be used to get current stock price as well as previous stock price.

2.7 DataSet

An Data set is a gathering of related, discrete things of related data that might be accessed to separately or in blend or oversaw all in all element. Data sets are accumulations of data. Any arrangement of things can be viewed as an data set. For instance {1,2,3} is an data set comprising of three things. On the inverse end of the scale, sets can contain a great many things, similar to the data from the US Census. Every single esteem in an data set (like 1, 2 or 3 in the above set) is known as a datum. Data doesn't need to be clearly associated with a specific end goal to call it a "set." Any gathering of things can be considered as an data set. For instance, an irregular arrangement of things like {blue,7,eyes,repair} can be viewed as a set. The way that the things are set together makes it a set. This reality comes specifically use for Data Mining, where substantial arrangements of data are filtered through to attempt and discover patterns and topics in the data. Those patterns and subjects are generally not evident to the stripped eye.

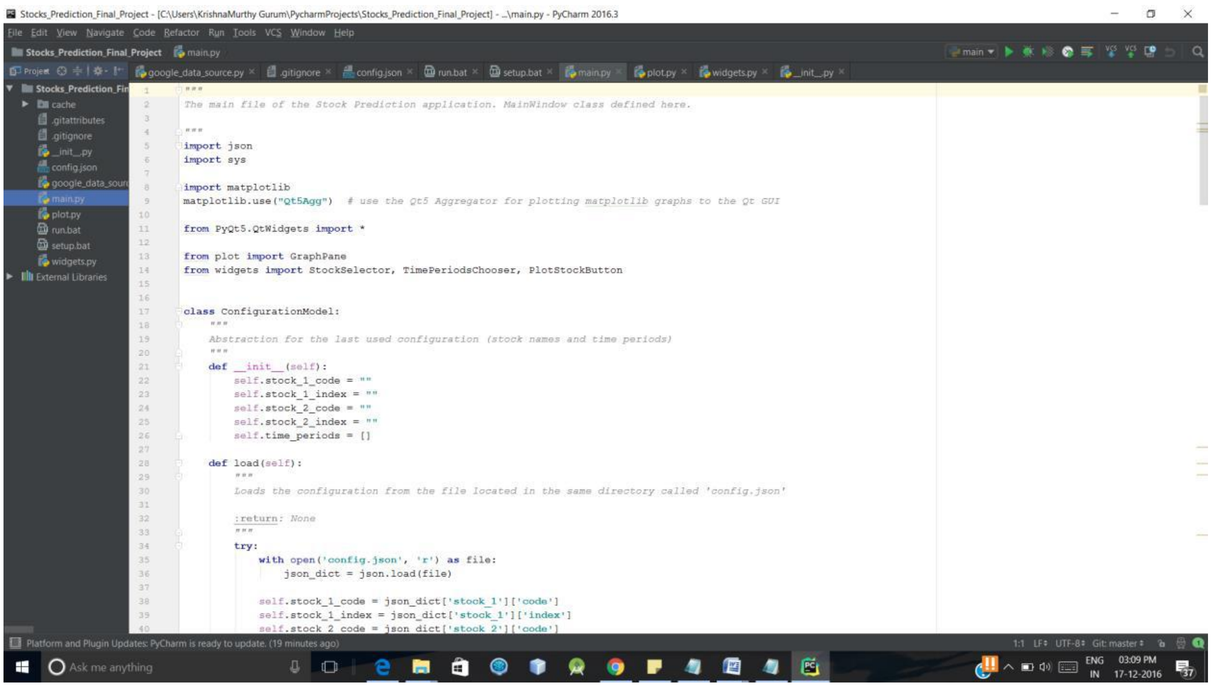
As a basic illustration, that apparently arbitrary gathering {8%,7,primary,repair}? Those are really identified with the shading blue. Eight percent of individuals have blue eyes, blue is the seventh shading in the rainbow tune, blue is an essential shading, and blue is frequently utilized on apparatus to show it needs repairing.

Certain things are basic to every single measurable data sets. For instance, the request of the data does not make a difference, which implies the course of action of the data inside the data set is not imperative. In this way the analyst has the flexibility to sort out the subjects under review in whichever arrange she thinks that it’s advantageous. Immense factual data sets are as of now accessible for some regions. For instance, the worldwide genealogical record contains family history of many individuals previously. On the off chance that a scientist needs to study designs and measurable data, she can basically make utilization of these data sets. This makes the occupation of the scientist much simpler. A specific factual data set can be utilized for various looks into. The evaluation data, for instance, contains far reaching data about the socioeconomics of a nation, which can then by used by various social researchers to study family structures, earnings, and so forth inside the nation.

A factual data set is in this way not an end in itself - it is simply the beginning stage where every one of the data is put away. How the data is gathered and translated relies on upon the analyst concentrate the data

3. Programming Environment

I have used PyCharm, it’s an Integrated Development Environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains.[2] It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django.



Stocks_Prediction_Final_Project - [C:\Users\KrishnaMurthy Gurum\PycharmProjects\Stocks_Prediction_Final_Project] - ...\main.py - PyCharm 2016.3

```python
"""
The main file of the Stock Prediction application. MainWindow class defined here.

"""
import json
import sys

import matplotlib
matplotlib.use("Qt5Agg")  # use the Qt5 Aggregator for plotting matplotlib graphs to the Qt GUI

from PyQt5.QtWidgets import *

from plot import GraphPane
from widgets import StockSelector, TimePeriodsChooser, PlotStockButton


class ConfigurationModel:
    """
    Abstraction for the last used configuration (stock names and time periods)
    """
    def __init__(self):
        self.stock_1_code = ""
        self.stock_1_index = ""
        self.stock_2_code = ""
        self.stock_2_index = ""
        self.time_periods = []

    def load(self):
        """
        Loads the configuration from the file located in the same directory called 'config.json'

        :return: None
        """
        try:
            with open('config.json', 'r') as file:
                json_dict = json.load(file)

            self.stock_1_code = json_dict['stock_1']['code']
            self.stock_1_index = json_dict['stock_1']['index']
            self.stock_2_code = json_dict['stock_2']['code']
```

Platform and Plugin Updates: PyCharm is ready to update. (19 minutes ago)

Figure 1: PyCharm 3.6 editor

PyCharm is cross-platform, with Windows, Mac OS X and Linux variants. The Community Edition is discharged under the Apache License, and there is additionally Professional Edition discharged under an exclusive permit - this has additional components.

- Coding Assistance and Analysis, with code consummation, syntax and blunder highlighting, linter incorporation, and speedy fixes

- Project and Code Navigation: specific venture sees, record structure perspectives and brisk hopping between documents, classes, techniques and utilizations
- Python Refactoring: including rename, separate technique, present variable, present consistent, pull up, push down and others
- Support for web structures: Django, web2py and Flask
- Integrated Python Debugger
- Integrated Unit Testing, with line-by-line scope
- Google App Engine Python Development
- Version Control Integration: brought together UI for Mercurial, Git, Subversion, Perforce and CVS with changelists and consolidation
- It contends fundamentally with various other Python-situated IDEs, including Eclipse's PyDev, the all the more comprehensively engaged Komodo IDE.

3.1 Scope

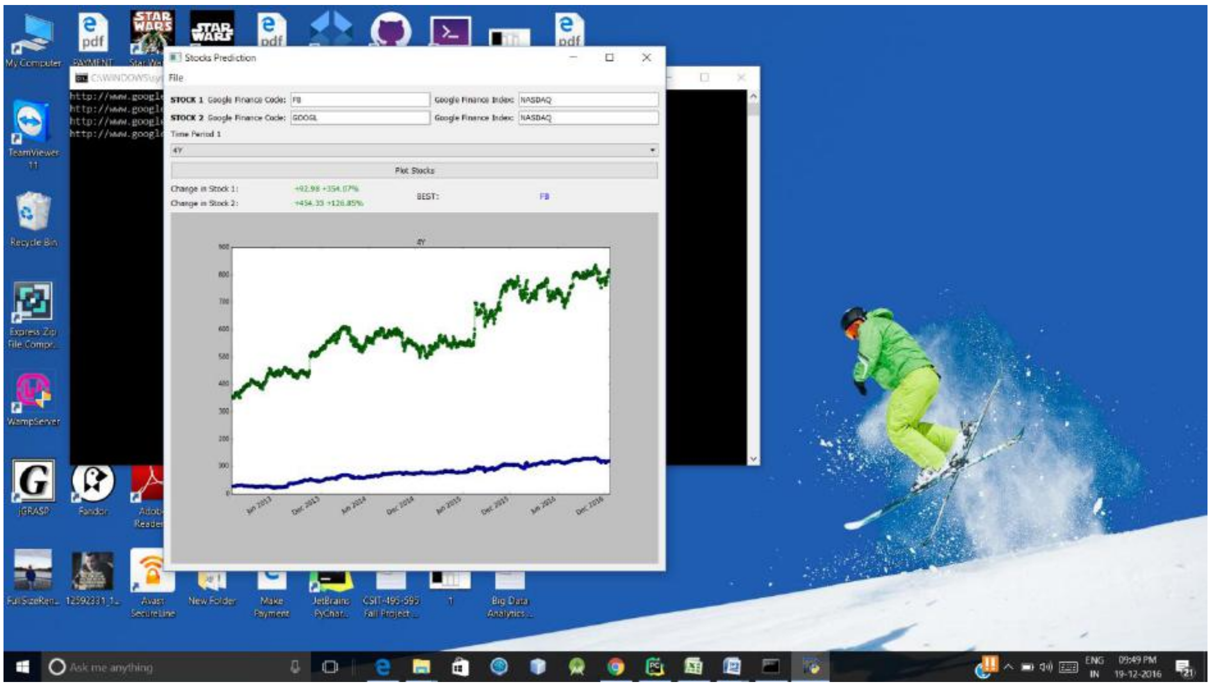
Having a clear vision of project details is necessary at the start of the project to meet all of the objectives for the said project.

In order to ensure we get the correct data, I had to choose between using Google Finance API OR Yahoo Finance API. Google finance provides stock charting capabilities which are not the strength of Yahoo Finance. Google provides more comprehensive real time stock market quotes and has more content capabilities than Yahoo. I have chosen Google Finance API in my project.

Another choice I had is to choose between, Python 2.7 and Python 3.6. Though both the versions are good and great, but Python 3.6 supports creating GUI applications, with Tkinter in the standard library and also by PyQt.

3.2 Methodology

The Project will output the Graph providing a user interface to query data from different Stocks of Google Finance (Stock Exchange Database) dataset. The system is built on a free and open-source solution stack. The project on execution, reads the dataset from the Google Finance API and takes the opening and closing rates as input and produce mean value to predict the Best Stock, and produce the graph on the basis of closing rates.



STOCK 1 Google Finance Code: FB Google Finance Index: NASDAQ
STOCK 2 Google Finance Code: GOOGL Google Finance Index: NASDAQ
Time Period 1
4Y

Change in Stock 1: +92.98 +354.07%
Change in Stock 2: +454.33 +126.85%
BEST: FB

Figure 2: Plotted Graph

3.3 Data Representation

Data need to be represented in various format before we can start analyzing it. In this section we will discuss various ways to visualize data.

3.3.1 Line Chart

The line chart[8] represents only the closing prices over a set period of time. The line is formed by connecting the closing prices over time frame. It does not provide high, low and opening prices information.

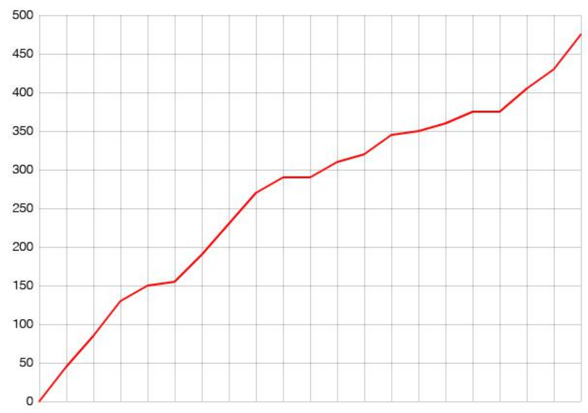


Figure 3: Line Chart

[Refernce: http://6.anychart.com/products/anychart/docs/users-guide/Line-Spline-Step-Line-Chart.html]

3.3.2Bar Chart

The bar chart[8] improves over the line chart by adding several key information to each datapoint. This provides high and low information for the trading period along with closing price.

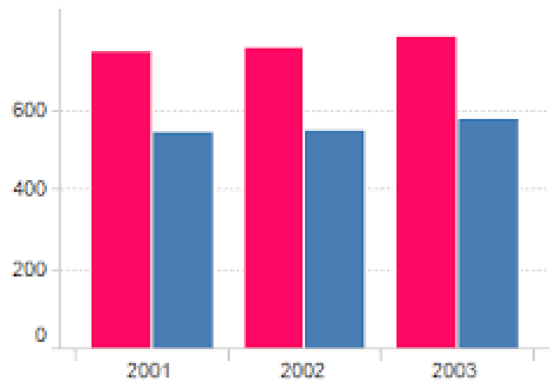


Figure 4: Bar Chart.

[Reference: http://www.8008851870.com/Help/dxpwebclient/bar_what_is_a_bar_chart.html]

3.3.3 Candlestick Chart

The candlestick chart[8] is similar to bar chart, it includes additional thin vertical line indicating the period's trading range. The difference comes in the formation of a wide bar on the vertical line, which illustrates the difference between the open and close. And, like bar charts, candlesticks also rely heavily on the use of colors to explain what has happened during the trading period.

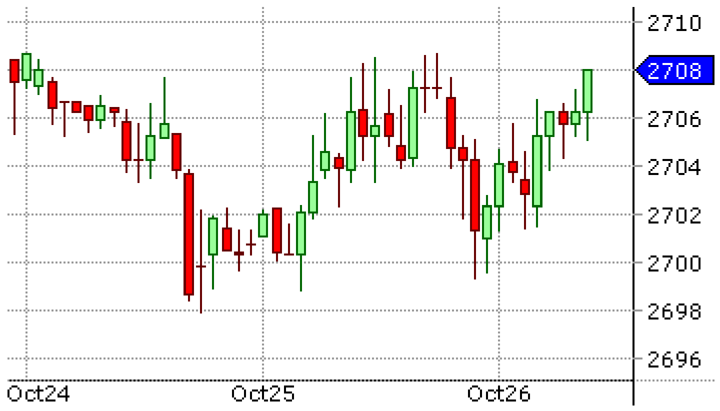


Figure 5: CandleStick Chart

[Reference: http://www.chart-formations.com/stock-charts/candlestick-charts.aspx]

3.3.4 Point and Figure Chart

The point and figure chart[8] is no more used actively but has a long history of use dating back to first technical leaders. It reflects time movements and is not concerned about the time and volume in formulation of the points. This removes the disturbances like noise or insignificant price movements from the view of the chart.

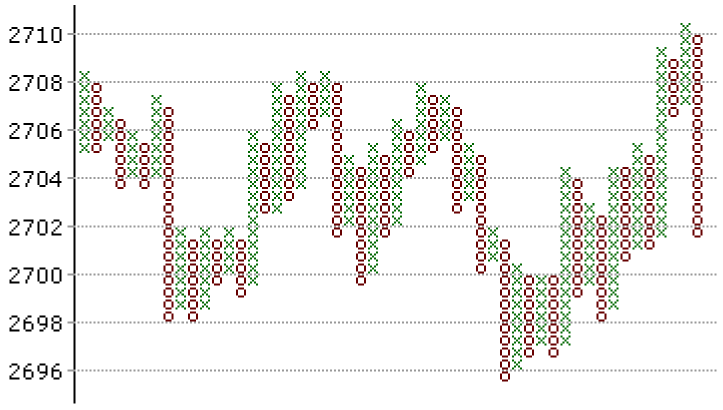


Figure 6: Point and Figure Chart.

[Reference: http://www.chart-formations.com/stock-charts/point-and-figure-charts.aspx]

In the current project, I have used Line Chart for display.

4. Future Works

Machine Learning is changing the computer science landscape. Having Machine Learning Techniques, here can provide higher degree of prediction.

Few of the techniques have been discussed here.

4.1 Artificial Neural Networks

An ANN is modelled as a representation of the humans own biological neuron and how the neural network functions in a human being. An ANN is an AI computation method that also called connectionism, parallel distributed processing or neural computation.

One reason why ANN is popular is due to its robustness, fault tolerance, ability to learn and generalize adaptability, universal function approximation and parallel data processing. This enables them to solve complex non-linear and multi input-output (IO) relationship problems

As we have multiple stages present, choosing the correct sequence of inputs like for example, having the previous day stock price in stage 1, currency value in stage 2 etc. we can get a pattern with better forecast results.

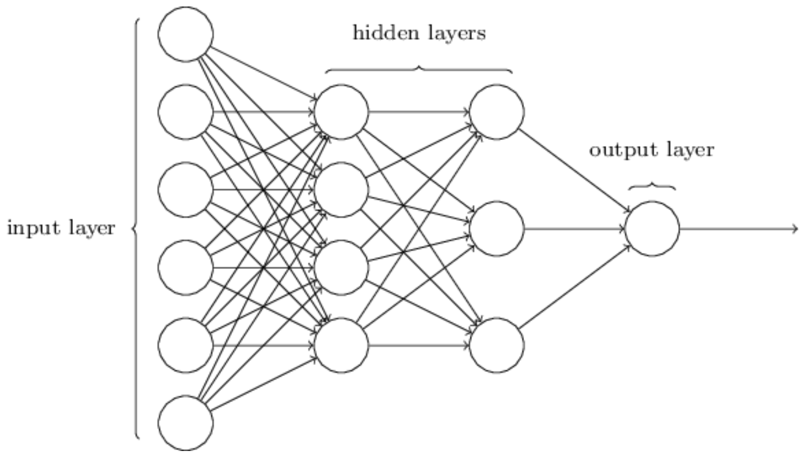


Figure 7: Neural Networks

[Refernce: http://neuralnetworksanddeeplearning.com/chap1.html]

4.2 Decision Stump

A decision stump is a machine learning model consisting of a one-level decision tree i.e. only one single split called as 1 – Rules. The resulting tree can be used to test for unseen examples.

If the given set is split into two subsets, we want the labels inside these subsets to be as homogenous as possible for further splitting. So it can also be seen as building many trees - a tree for each attribute - and then selecting the tree that produces the best split.

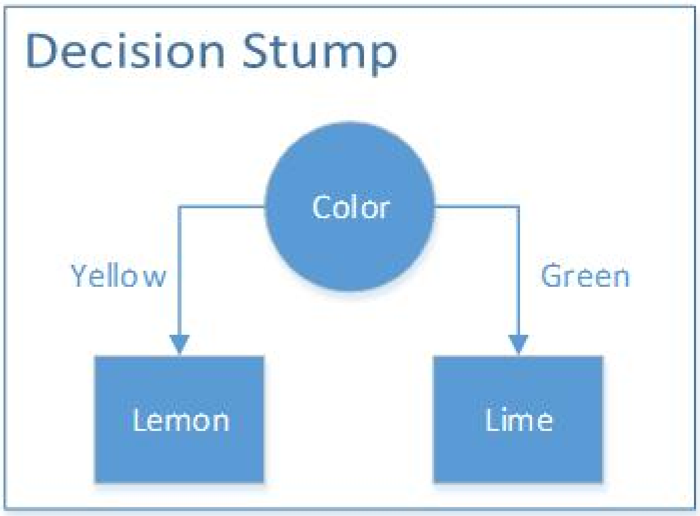


Figure 8: Decision Stump

[reference: https://software.intel.com/en-us/node/564696]

4.3 Simple Linear regression

A simple linear regression allows determining functional dependency between two sets of numbers. For example, we can use regression to find relation between stocks sold and price of the stock.

Here we have dependant and independent variable. We are trying to find the various value of dependant variable using the independent variable.

Dependent variables is also called as criterion, response variable or label. It is denoted by Y. The independent variable is also referred as covariates, predictor or features. It is denoted by X. The adjective *simple* refers to the fact that the outcome variable is related to a single predictor.

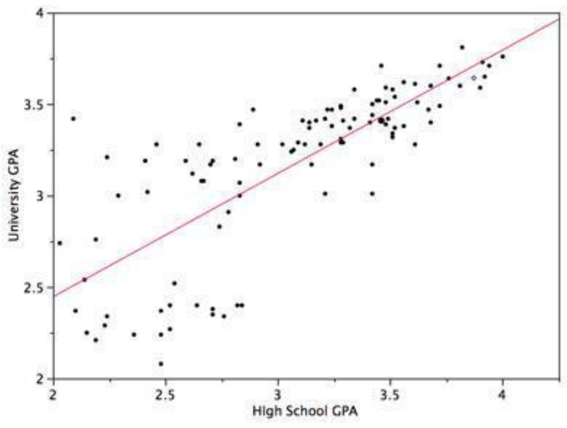


Figure 9: Simple Linear Regression

[reference: http://onlinestatbook.com/2/regression/intro.html]

4.4 Support vector machines

SVM are generalization of Nearest neighbour algorithm. The numerical points in our data forms an n – dimensional space. For example, if we have 2 input variables, we have 2 – dimensional space. We have a vector – a line in 2D space that splits the input variable space. For n- dimensional space, we use hyperplane. Once we have split the data, we can try to find equation for the line and for any new input we can find the evaluate the equation. If the value is above zero, it falls under one classification and if it is below zero, it falls under another classification. By this method, we can have many input parameters classified and future value predicted based on all the input parameters.

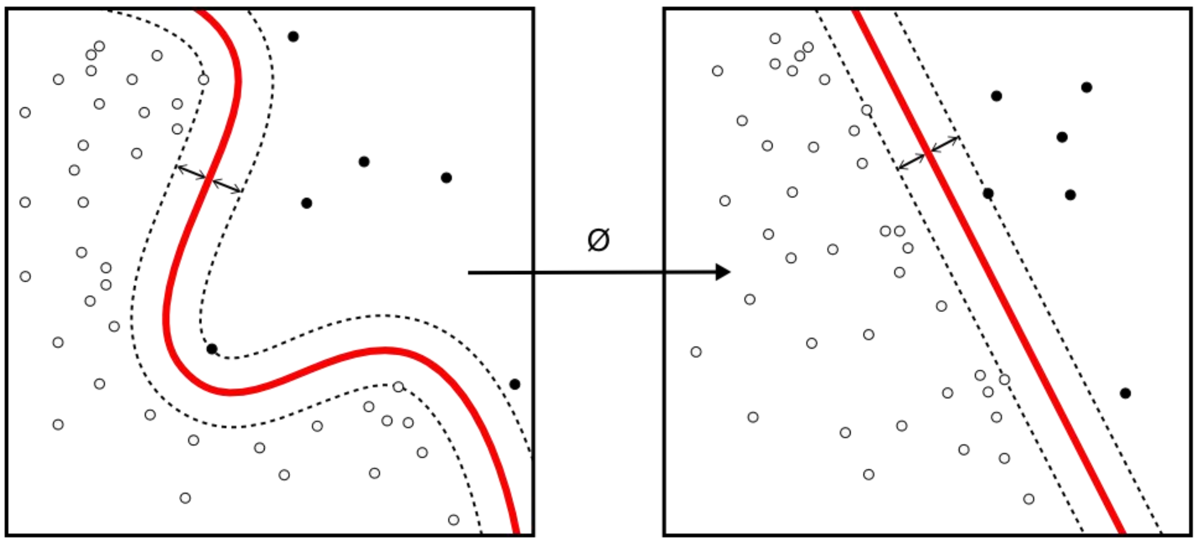
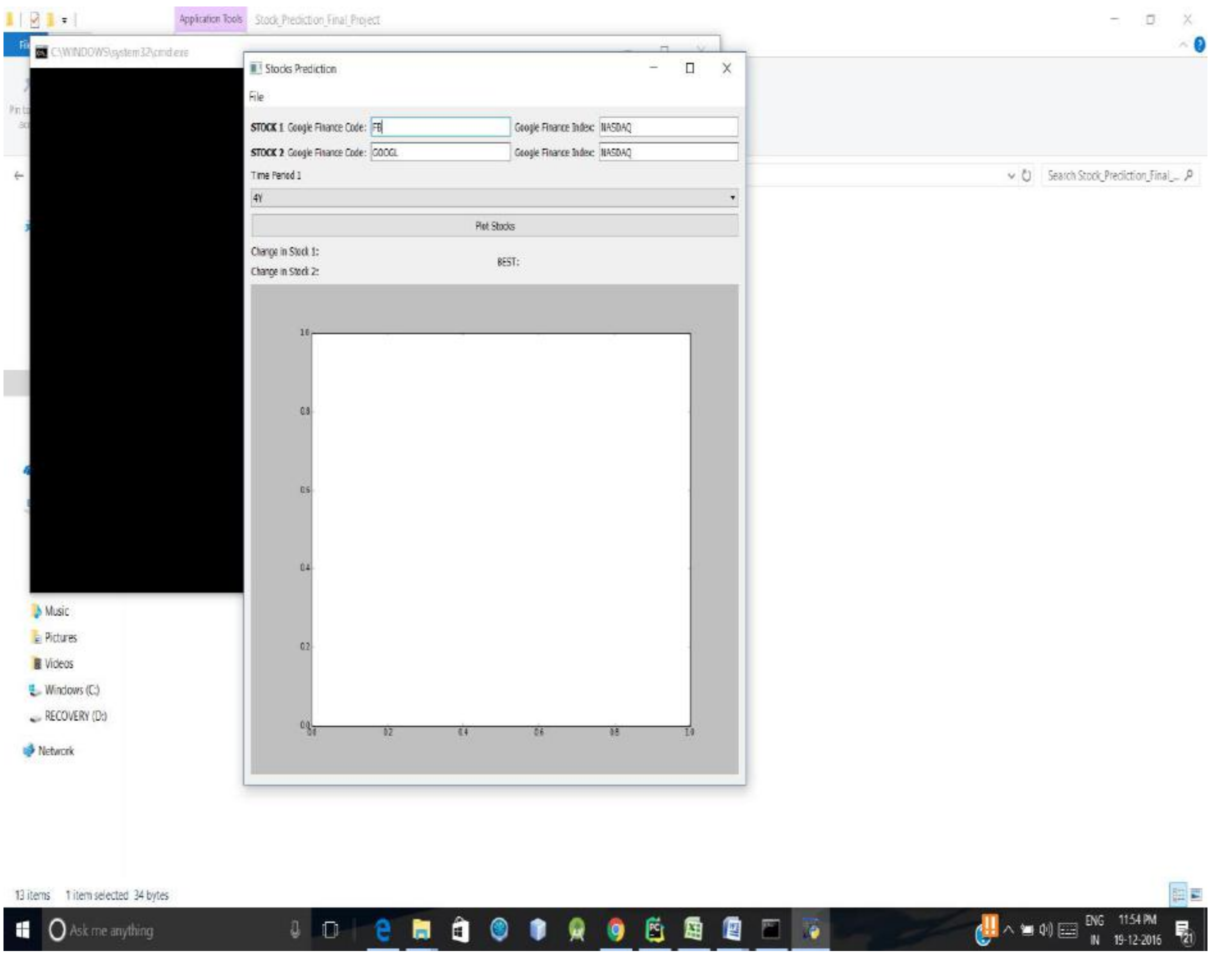


Figure 10: Support Vector Machines

[reference - https://www.npmjs.com/package/node-svm]

5. User Guide.

1. The User is interfaces a Plain Widget and needs to provide input for Google Finance Code and Google Finance Index of a unique Stock.


Stocks Prediction

STOCK 1 Google Finance Code: FB Google Finance Index: NASDAQ

STOCK 2 Google Finance Code: GOOGL Google Finance Index: NASDAQ

Time Period 1: 4Y

Change in Stock 1:

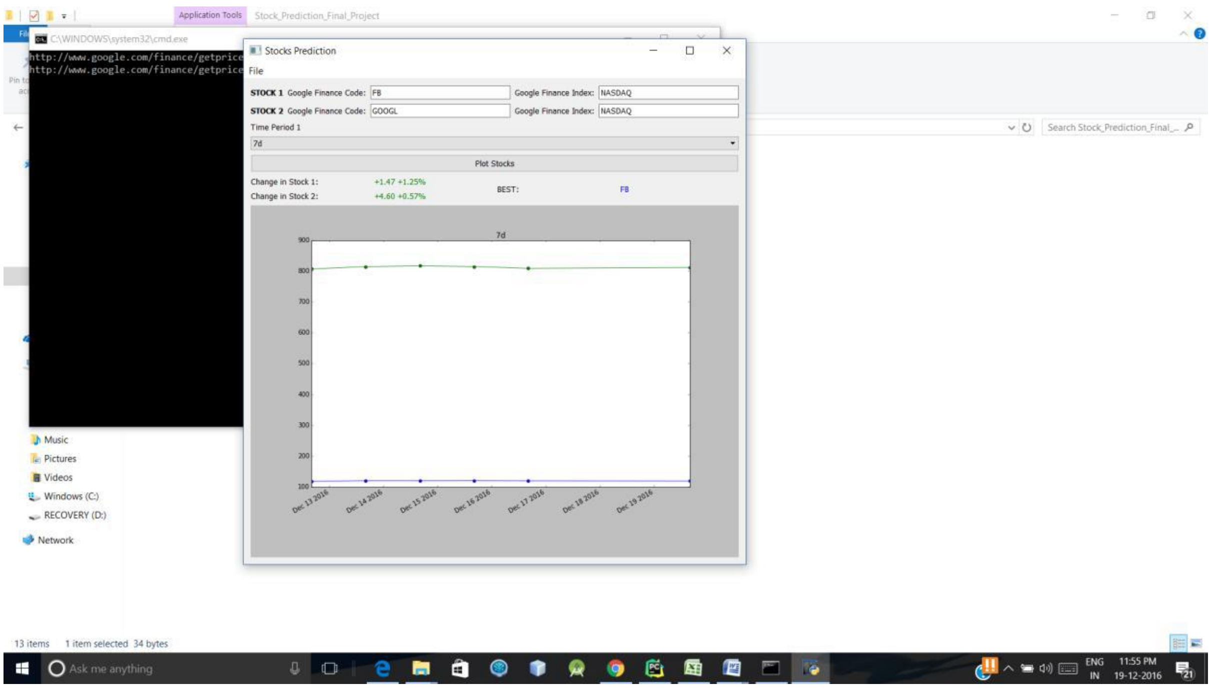
Change in Stock 2:

BEST:

Figure 11: User Guide 1

2. User can query to check which stock is best, from last 3 days to last 7 years.

- After Querying, when user clicks the button "Plot Stock" on the Widget. The Graph results are Produced, when user is Opting for number of days.



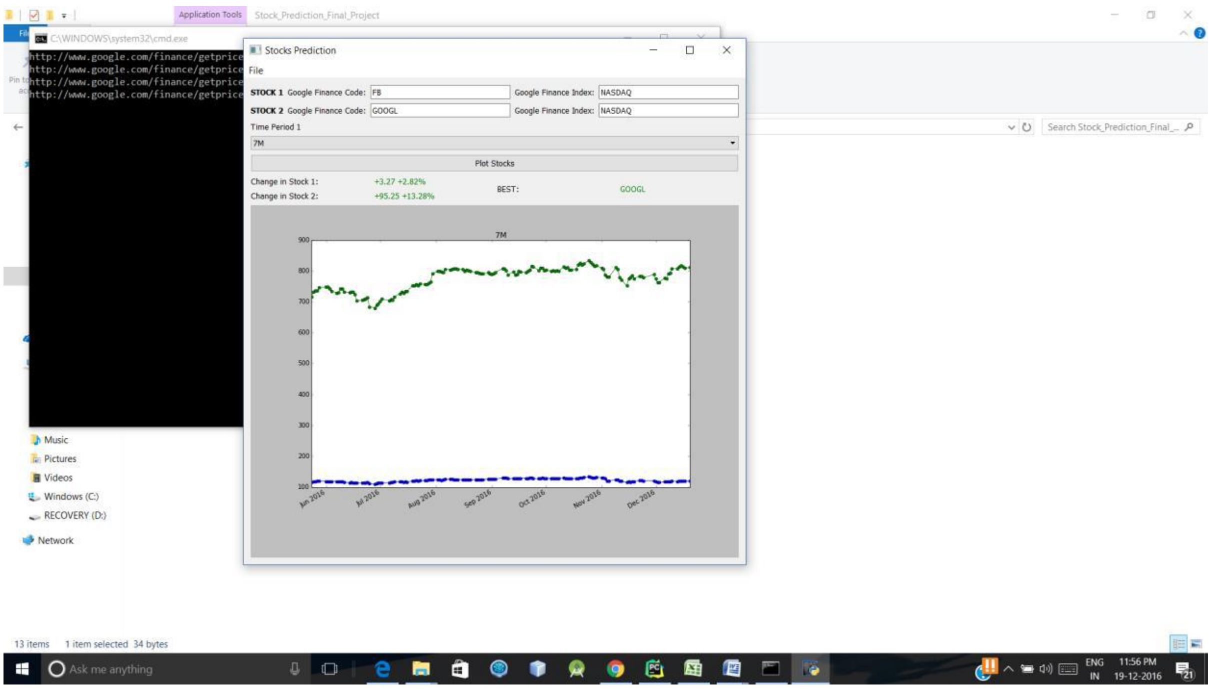
```
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
```

STOCK 1 Google Finance Code: FB Google Finance Index: NASDAQ
STOCK 2 Google Finance Code: GOOGL Google Finance Index: NASDAQ
Time Period 1: 7d

Change in Stock 1: +1.47 +1.25%
Change in Stock 2: +4.60 +0.57%
BEST: FB

Figure 12: User Guide 2

- After Querying, when user clicks the button "Plot Stock" on the Widget. The Graph results are Produced, when user is Opting for number of months.



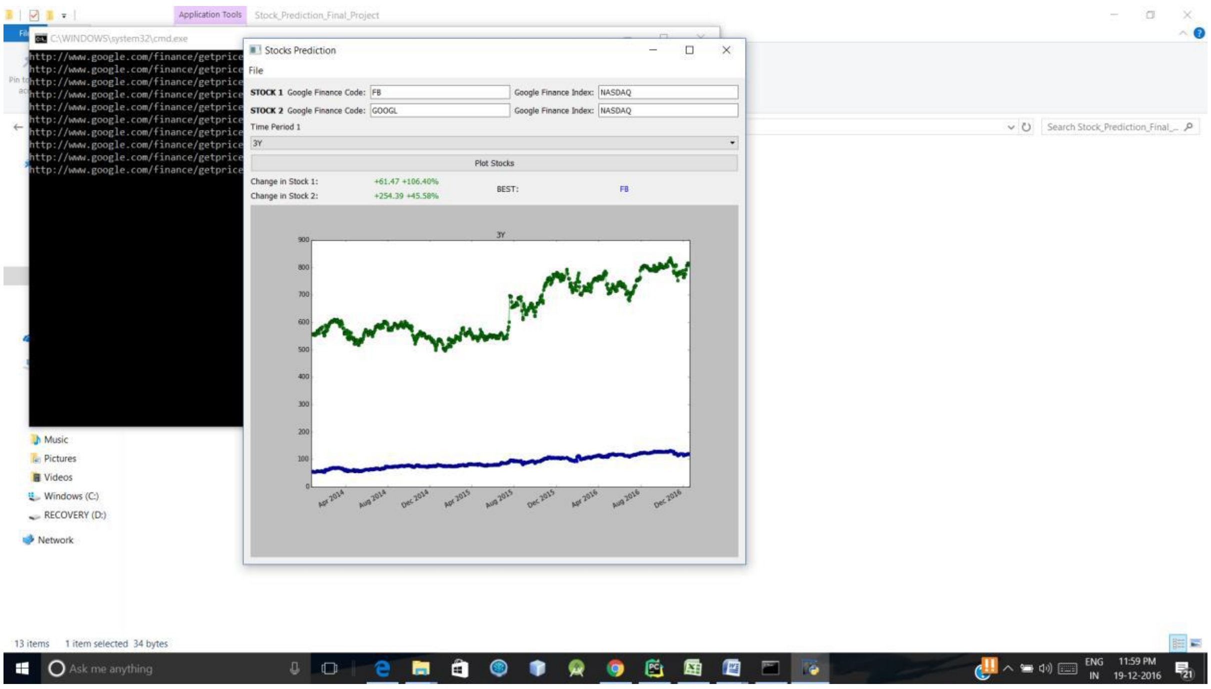
```
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
```

STOCK 1 Google Finance Code: FB Google Finance Index: NASDAQ
STOCK 2 Google Finance Code: GOOGL Google Finance Index: NASDAQ
Time Period 1: 7M

Change in Stock 1: +3.27 +2.82%
Change in Stock 2: +95.25 +13.28%
BEST: GOOGL

Figure 13: User Guide 3

- After Querying, when user clicks the button “Plot Stock” on the Widget. The Graph results are Produced, when user is Opting for number of years.



```
C:\WINDOWS\system32\cmd.exe
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
http://www.google.com/finance/getprice
```

Stocks Prediction

STOCK 1 Google Finance Code: FB Google Finance Index: NASDAQ

STOCK 2 Google Finance Code: GOOGL Google Finance Index: NASDAQ

Time Period 1: 3Y

Change in Stock 1: +61.47 +106.40%

Change in Stock 2: +254.39 +45.58%

BEST: FB

Figure 14: User Guide 4

6. Conclusion

Stock market prediction can be never be accurate. If its accurate then there is no reason for anyone to lose money. We can only get close to correct value and the prediction models can only be judged by how close it is to actual value.

The current tool focuses on providing information in the Line chart module over last few years OR last few months OR last few days. Based on this the potential investor can decide on making an investment OR taking back an investment and so on.

As suggested in chapter 4, having Machine Learning models improves greatly the accuracy of prediction.

7. References

[1] "Introduction To Fundamental Analysis". Available:
http://www.investopedia.com/university/fundamentalanalysis/

[2] "Basics Of Technical Analysis" Available:
http://www.investopedia.com/university/technical/

[3] "Numpy" Available:
http://www.numpy.org/

[4] "Python Data Analysis Library" Available:
http://pandas.pydata.org/

[5] "matplotlib" Available:
http://matplotlib.org/

[6] "PyQt5 5.7" Available:
https://pypi.python.org/pypi/PyQt5

[7] "About" Available:
https://notepad-plus-plus.org/

[8] "Technical Analysis: Chart Types" Available:
http://www.investopedia.com/university/technical/techanalysis7.asp

[9] "Decision stump" Available:
https://en.wikipedia.org/wiki/Decision_stump

[10] "Simple linear regression" Available:
https://en.wikipedia.org/wiki/Simple_linear_regression

[11] "Support vector machine" Available:
https://en.wikipedia.org/wiki/Support_vector_machine

[12] “Information about DataSets” Available:

https://explorable.com/statistical-data-sets

APPENDIX – SOURCE CODE

Main Module

Main module is trigger point of the execution of application. The main file of the Stock Prediction application. Main Window class defined here.

```
import json
import sys

import matplotlib
matplotlib.use("Qt5Agg")  # use the Qt5 Aggregator for plotting matplotlib
graphs to the Qt GUI

from PyQt5.QtWidgets import *

from plot import GraphPane
from widgets import StockSelector, TimePeriodsChooser, PlotStockButton


class ConfigurationModel:
    """
    Abstraction for the last used configuration (stock names and time periods)
    """
    def __init__(self):
        self.stock_1_code = ""
        self.stock_1_index = ""
        self.stock_2_code = ""
        self.stock_2_index = ""
        self.time_periods = []

    def load(self):
        """
        Loads the configuration from the file located in the same directory
called 'config.json'

        :return: None
        """
        try:
            with open('config.json', 'r') as file:
                json_dict = json.load(file)

            self.stock_1_code = json_dict['stock_1']['code']
            self.stock_1_index = json_dict['stock_1']['index']
            self.stock_2_code = json_dict['stock_2']['code']
            self.stock_2_index = json_dict['stock_2']['index']
            self.time_periods = json_dict['time_periods']
        except FileNotFoundError:
            print('no config file found')

    def save(self):
        """
        Saves the configuration to the file located in the same directory
called 'config.json'

        :return: None
        """
        json_dict = {
            'stock_1': {
                'code': self.stock_1_code,
                'index': self.stock_1_index
            },
            'stock_2': {
                'code': self.stock_2_code,
                'index': self.stock_2_index
            },
            'time_periods': self.time_periods
        }
        with open('config.json', 'w+') as file:
            json.dump(json_dict, file)  # serialize as json and save


class GraphPaneCollection(QHBoxLayout):
    """
    Layout to hold multiple graph panes to show graphs with different time
periods and information about those stocks.
    """
    def __init__(self, n_graphs, model):
        """
        Constructor for GraphPaneCollection.

        Initializes a GraphPaneCollection with n_graphs objects of type
GraphPane.

        :param n_graphs: The number of 'GraphPane'(s) to create.
        :param model: The ConfigurationModel to use when updating GraphPanes.
        """
        super().__init__()

        self.graphs = []
        self.model = model

        for _ in range(n_graphs):
            gp = GraphPane(model)
            self.graphs.append(gp)
            self.addLayout(gp)  # add the graph pane to the current layout


class ApplicationWindow(QMainWindow):
    """
    Class representing the main application window that all other windows and
widgets have as a parent.
    """
    def __init__(self, *args, **kwargs):
        """
        Constructor for ApplicationWindow.

        Initializes the QMainWindow base class, loads the app's
ConfigurationModel
        and then invokes init_ui to set up our application's widgets.

        :param args: positional arguments to pass to the QMainWindow
constructor
        :param kwargs: keyword arguments to pass to the QMainWindow constructor
        """
        super().__init__(*args, **kwargs)

        # set the number of graphs the application will display
        self.n_graphs = 3

        # set up the configuration model and load from file (if possible)
        self.model = ConfigurationModel()
        self.model.load()

        # create the widgets for this application.
        self.init_ui()
    def init_ui(self):
        """
        Method called during object construction to set up our application's
widgets.

        :return: None
        """

        # set window size and title
        self.setGeometry(300, 300, 800, 800)
        self.setWindowTitle('Stocks Visualising')

        # set up 'exit' action
        exit_Action = QAction('Exit', self)
        exit_Action.setShortcut('Ctrl + Q')
        exit_Action.setStatusTip('Exit from Application')
        exit_Action.triggered.connect(self.close)

        menubar = self.menuBar()
        fileMenu = menubar.addMenu('&File')
        fileMenu.addAction(exit_Action)

        # create a main widget to act as a parent for all other widgets
        mainWidget = QWidget()

        # create the main layout of the application to be a VBox layout
        layout = QVBoxLayout()

        # ==== APP LAYOUT SECTIONS HERE ====

        # Add stock choosers for stocks 1 and 2
        stock_1_chooser = StockSelector(1, self.model)
        stock_2_chooser = StockSelector(2, self.model)

        layout.addLayout(stock_1_chooser)
        layout.addLayout(stock_2_chooser)

        # add a time period chooser
        time_period_chooser = TimePeriodsChooser(self.n_graphs, self.model)
        layout.addLayout(time_period_chooser)

        # create the graph pane collection (but don't add, we want this below
the plot stock button defined below)
        graph_pane_collection = GraphPaneCollection(self.n_graphs, self.model)

        # create and add the plot stock button (which references the
graph_pane_collection)
        plot_stock_button = PlotStockButton(graph_pane_collection,
time_period_chooser, stock_1_chooser, stock_2_chooser)
        layout.addWidget(plot_stock_button)

        # add the graph pane collection, created above, to the layout
        layout.addLayout(graph_pane_collection)

        # === END APP LAYOUT SECTIONS ====

        # set the layout of the parent widget to the application layout we have
created
        mainWidget.setLayout(layout)

        # set the app's central widget to said parent widget
        self.setCentralWidget(mainWidget)
```

```
if __name__ == '__main__':
    # if we are executing this file...

    # create a QApplication
    app = QApplication(sys.argv)

    # create an Application Window
    a_Window = ApplicationWindow()

    # enable one of the following:
    a_Window.show()
    # window.showFullScreen()

    # run the app and return the exit status code of the QApplication
    sys.exit(app.exec_())
```

Google Data Source – Module

Google Data Source Module downloads the stock data from the Google Finance hidden API. This module gets the data sets from the Google Finance API and when this module is invoked by the main module, the data set is stored by the json library and data frames of the data sets are read by Pandas package when invoked by the main module.

```
import pandas as pd
import urllib.request
import datetime as dt


def get_google_data_for_stock(symbol, exchange, interval_seconds=86400,
period='1d'):
    """
    Downloads the stock data for the instrument denoted by (symbol, exchange)

    :param symbol: The Google Finance 'code' for the stock
    :param exchange: The Google Finance 'index' to which the stock belongs
    :param interval_seconds: The number of seconds in one candle/time interval
    :param period: The period of time for which we should have data.
    :return: A pandas DataFrame containing the stock data and with a
DateTimeIndex
    """
    url_root = 'http://www.google.com/finance/getprices?'
    url_root += 'q=' + symbol
    url_root += '&x=' + exchange
    url_root += '&i=' + str(interval_seconds)
    url_root += '&p=' + period
    url_root += '&f=d,o,h,l,c,v'
    url_root += '&df=cpct'

    print(url_root)

    with urllib.request.urlopen(url_root) as response:
        data = response.read().decode('ascii')

    # print(data)
    data = data.split('\n')

    # actual data starts at index = 3
    # first line contains full timestamp,
    # every other line is offset of period from timestamp
    parsd_data = []
    anchor_stamp = ''
    end = len(data)
    for i in range(7, end):
        c_data = data[i].split(',')
        if 'a' in c_data[0]:
            # first one record anchor timestamp
            anchor_stamp = c_data[0].replace('a', '')
            c_ts = int(anchor_stamp)
        else:
            try:
                coffset = int(c_data[0])
                c_ts = int(anchor_stamp) + (coffset * interval_seconds)
                parsd_data.append((dt.datetime.fromtimestamp(float(c_ts)),
float(c_data[1]), float(c_data[2]), float(c_data[3]), float(c_data[4]),
float(c_data[5])))
            except:
                pass  # for time zone offsets thrown into data
    df = pd.DataFrame(parsd_data)
    df.columns = ['ts', 'Close', 'High', 'Low', 'Open', 'Volume']
    df.index = df.ts
    del df['ts']
    return df
```

Plot – Module

Plot module contains code useful for plotting the stock graphs and caching the datasets used to generate them. This module used to plot the graph when invoked by the main module, after formulating the results of the data frames by the data sets.

```
import os.path

from PyQt5.QtWidgets import QSizePolicy, QVBoxLayout, QHBoxLayout, QLabel
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
from matplotlib.figure import Figure

from werkzeug.contrib.cache import FileSystemCache

from google_data_source import get_google_data_for_stock

# get the full path of the directory of the application
app_root = os.path.abspath(os.path.dirname(__file__))

# set up the cache directory as the subdirectory 'cache'
CACHE_DIR = os.path.join(app_root, 'cache')

# set the expiry of each object in the cache to 1 day (24 hrs) as this is what
one datapoint of the datasets represents,
# so the data won't be out of date for at least a day after
EXPIRY_SECONDS = 24 * 60 * 60  # HOURS * MINUTES * SECONDS

# create the werkzeug cache object with the cache directory and expiry
cache = FileSystemCache(CACHE_DIR, default_timeout=EXPIRY_SECONDS,
threshold=100) # threshold = 100 -> after 100 items stored, cache begins
deleting some even if not yet expired


def plot_stocks(graph_pane_collection, stock1, stock2, time_periods):
    """
    Plots new graphs for stock1 and stock2 at each of the time_periods on the
```

```
graphs in graph_pane_collection.

    :param graph_pane_collection: The graph panes to draw the new graphs on.
    :param stock1: The details of stock 1
    :param stock2: The details of stock 2
    :param time_periods: a list of time periods to plot these stocks on
    :return: Boolean, True if successful, False otherwise
    """
    data_frames_stock1 = []
    data_frames_stock2 = []

    # iterate through each time period
    for tp in time_periods:
        # determine the key in cache for stock 1
        cache_key_stock1 = "{0}:{1}.{2}".format(
            stock1.get('gf_code'),
            stock1.get('gf_index'),
            tp
        )

        # determine the key in cache for stock 2
        cache_key_stock2 = "{0}:{1}.{2}".format(
            stock2.get('gf_code'),
            stock2.get('gf_index'),
            tp
        )

        # check if there is anything in cache for stock 1 key
        if cache.get(cache_key_stock1) is not None:
            # if so, use it
            data_frames_stock1.append(cache.get(cache_key_stock1))
        else:
            # fetch new data from google
            try:
                df_1 = get_google_data_for_stock(stock1.get('gf_code'),
stock1.get('gf_index'), interval_seconds=86400, period=tp)
                data_frames_stock1.append(df_1)
                cache.set(cache_key_stock1, df_1)  # cache the new data
            except ValueError:
                # stock data invalid, error
                return False

        # check if there is anything in cache for stock 2 key
        if cache.get(cache_key_stock2) is not None:
            # if so, use it
            data_frames_stock2.append(cache.get(cache_key_stock2))
        else:
            # fetch new data from google
            try:
                df_2 = get_google_data_for_stock(stock2.get('gf_code'),
stock2.get('gf_index'), interval_seconds=86400, period=tp)
                data_frames_stock2.append(df_2)
                cache.set(cache_key_stock2, df_2)  # cache the new data
            except ValueError:
                # stock data invalid, error
                return False

    # iterate through each graph pane and draw the graph on each
    for idx, pane in enumerate(graph_pane_collection.graphs):
        # get the datafram for stock 1
        df1 = data_frames_stock1[idx]

        # extract the x and y values to a tuple
        stock_1_data = (df1.index, df1.Close)
        # get the datafram for stock 2
        df2 = data_frames_stock2[idx]

        # extract the x and y values to a tuple
        stock_2_data = (df2.index, df2.Close)

        # set the time period associated with this graph pane
        pane.time_period = time_periods[idx]

        # draw the new graph data
        pane.draw(stock_1_data, stock_2_data)

    # return true if all went successfully
    return True


class GraphCanvas(FigureCanvas):
    """
    GraphCanvas can be used as a widget for a matplotlib graph.
    """
    def __init__(self, parent=None, width=7, height=7, dpi=60):
        """
        Constructor for GraphCanvas.

        :param parent: Parent widget
        :param width: Width of matplotlib graph
        :param height: Height of matplotlib graph
        :param dpi: DPI (Dots Per Inch) of graph
        """
        figure = Figure(figsize=(width, height), dpi=dpi)
        self.axes = figure.add_subplot(111)

        FigureCanvas.__init__(self, figure)
        self.setParent(parent)

        FigureCanvas.setSizePolicy(self, QSizePolicy.Expanding,
QSizePolicy.Expanding)
        FigureCanvas.updateGeometry(self)

    def update_lines(self, line1, line2, title):
        """
        Update the graph with new lines to draw, and a new title.

        :param line1: The first series to draw.
        :param line2: The second series to draw.
        :param title: The new title of the graph
        :return: None
        """
        # clear the axes
        self.axes.cla()

        # plot stock 1
        self.axes.plot_date(line1[0], line1[1], xdate=True, ydate=False,
linestyle='solid')

        # plot stock 2
        self.axes.plot_date(line2[0], line2[1], xdate=True, ydate=False,
linestyle='solid')

        # set the new graph titile
        self.axes.set_title(title)

        # format the x axis labels so that there are no overlaps
        self.figure.autofmt_xdate()

        # draw the new graph
        self.draw()


class GraphPane(QVBoxLayout):
    """
    GraphPane is a view of a graph and details about the change in price of a 
stock since open and close,
    and which stock is best.
    """
    def __init__(self, model, time_period='1M'):
        """
        Constructor for GraphPane
        :param model: ConfigurationModel for the pane to read from.
        :param time_period: The time period to be shown on the pane's graph.
        """
        super().__init__()
        self.model = model
        self.time_period = time_period

        # create a layout for the change in price and best info
        meta_box = QHBoxLayout()

        # create a VBox layout for the differences in stocks 1 and 2
        diffs_box = QVBoxLayout()

        # create a layout, label and output label for the change in stock 1
        stock1_change = QHBoxLayout()
        stock1_change.addWidget(QLabel('Change in Stock 1:'))
        self.stock1_change_label = QLabel('')
        stock1_change.addWidget(self.stock1_change_label)
        diffs_box.addLayout(stock1_change)

        # create a layout, label and output label for the change in stock 2
        stock2_change = QHBoxLayout()
        stock2_change.addWidget(QLabel('Change in Stock 2:'))
        self.stock2_change_label = QLabel('')
        stock2_change.addWidget(self.stock2_change_label)
        diffs_box.addLayout(stock2_change)

        # add the 'differences' layout to the larger stock price metadata 
layout
        meta_box.addLayout(diffs_box)

        # create a layout for the 'best' stock with label and output label
        best = QHBoxLayout()
        best.addWidget(QLabel('BEST:'))
        self.best_label = QLabel('')
        best.addWidget(self.best_label)
        meta_box.addLayout(best)

        # add the metadata layout to the graph pane
        self.addLayout(meta_box)

        # add the actual graph canvas to the pane
        self.graph_canvas = GraphCanvas()
        self.addWidget(self.graph_canvas)

    def draw(self, line1, line2):
        """
        Updates the GraphPane with the new lines line1 and line2
        and calculates difference in open and close for each stock
        and then decided which is best and displays this information on the
relevant output labels

        :param line1: The line data for stock 1
        :param line2: The line data for stock 2
        :return: None
        """

        # draw the new lines
        self.graph_canvas.update_lines(line1, line2, self.time_period)

        # get the open and close values for stock 1
        stock1_open = line1[1][0]
        stock1_close = line1[1][-1]

        # get the open and close values for stock 2
        stock2_open = line2[1][0]
        stock2_close = line2[1][-1]

        # calculate the difference in open and close
        diff_1 = stock1_close - stock1_open
        diff_2 = stock2_close - stock2_open

        # calculate the percentage change since open for both stocks
        per_change_1 = (float(diff_1) / stock1_open) * 100
        per_change_2 = (float(diff_2) / stock2_open) * 100

        # determine which sign to use depending on if the percentage change is
positive or negative
        sign_1 = '+' if per_change_1 >= 0 else '-'
        sign_2 = '+' if per_change_2 >= 0 else '-'

        # set the stock 1 output label to show the difference and percentage
change
        self.stock1_change_label.setText("{sign}{0:.2f}
{sign}{1:.2f}%".format(abs(diff_1), abs(per_change_1), sign=sign_1))
        self.stock1_change_label.setStyleSheet('color: {0}'.format('green' if
per_change_1 >= 0 else 'red'))

        # set the stock 2 ouptut label to show the difference and percentage
change
        self.stock2_change_label.setText("{sign}{0:.2f}
{sign}{1:.2f}%".format(abs(diff_2), abs(per_change_2), sign=sign_2))
        self.stock2_change_label.setStyleSheet('color: {0}'.format('green' if
per_change_2 >= 0 else 'red'))

        # determine wich is best
        if per_change_1 == per_change_2:
            # if the same percentage change (very rare, but possible) then
neither are better

            # update the best output label to show neither
            self.best_label.setText('NEITHER')
            self.best_label.setStyleSheet('color: black')
        elif per_change_1 > per_change_2:
            # if the percentage change of stock 1 is greater than that of stock
2, stock 1 is better

            # update the best output label to show stock 1 being better
            self.best_label.setText(self.model.stock_1_code)
            self.best_label.setStyleSheet('color: blue')  # colour chosen by
matplotlib for series 1
        else:
            # finally, if the percentage change of stock 2 is greater than that
```

```
of stock 1, stock 2 is better

            # update the best output label to show stock 2 being better
            self.best_label.setText(self.model.stock_2_code)
            self.best_label.setStyleSheet('color: green')  # colour chosen by
matplotlib for series 2
```

Widgets – Module

Widgets module produce a customized widgets for this application. This module will take the inputs for the Google Finance code, Google Finance index and Time period, when it is invoked by the main module.

```
from urllib.error import URLError
from threading import Thread

from PyQt5.QtWidgets import QHBoxLayout, QLabel, QLineEdit, QVBoxLayout,
QComboBox, QPushButton, QMessageBox

from plot import plot_stocks

POSSIBLE_TIME_PERIODS = [
    '3d', '4d', '5d', '6d',
    '7d', '14d', '21d',
    '1M', '2M', '3M', '4M', '5M', '6M', '7M', '8M', '9M', '10M', '11M',
    '1Y', '2Y', '3Y', '4Y', '5Y', '6Y', '7Y'
]


class StockSelector(QHBoxLayout):
    """
    A grouping of widgets that allows the details of a stock to be input.
    """

    def __init__(self, stock_no, model):
        """
        Constructor for StockSelector.

        :param stock_no: The number of the stock this Selector should modify (1
or 2)
        :param model: The ConfigurationModel object to modify with changes.
        """
        super().__init__()

        # validate precondition that stock_no is 1 or 2
        if stock_no not in (1, 2):
            raise ValueError('stock_no must be 1 or 2')

        self.stock_no = stock_no
        self.model = model

        # Add a label with the stock number being altered
        label = QLabel('STOCK ' + str(stock_no))
        label.setStyleSheet('font-weight: bold')
        self.addWidget(label)

        # add a label for the input for the GF code
        code_label = QLabel('Google Finance Code:')
        self.addWidget(code_label)

        # create the input for GF code
        self.gf_code_input = QLineEdit()

        # load the existing value from the ConfigurationModel
        if stock_no == 1:
            code_text = model.stock_1_code
        else:
            code_text = model.stock_2_code

        self.gf_code_input.setText(code_text)

        # register handler for textChanged to update the saved model when the
input is changed
        self.gf_code_input.textChanged.connect(self.code_changed)

        # add the GF code input widget
        self.addWidget(self.gf_code_input)

        # add a label for the input for the GF index
        index_label = QLabel('Google Finance Index:')
        self.addWidget(index_label)

        # create the input for GF index
        self.gf_index_input = QLineEdit()

        # load the existing value from the ConfigurationModel
        if stock_no == 1:
            index_text = model.stock_1_index
        else:
            index_text = model.stock_2_index
        self.gf_index_input.setText(index_text)

        # register handler for textChanged to update the saved model when the
input is changed
        self.gf_index_input.textChanged.connect(self.index_changed)

        # add the GF index input widget
        self.addWidget(self.gf_index_input)

    def get_gf_code(self):
        """
        Getter for the GF code stored in the respective input box.

        :return: String - Google Finance stock code input to the LineEdit
        """
        return self.gf_code_input.text()

    def get_gf_index(self):
        """
        Getter for the GF index stored in the respective input box.

        :return: String - Google Finance stock Index input to the LineEdit
        """
        return self.gf_index_input.text()

    def code_changed(self, new_text):
        """
        Event handler for a change in the GF code input.

        :param new_text: The new text value of the input.
        :return: None
        """
        if self.stock_no == 1:
            self.model.stock_1_code = new_text
        else:
            self.model.stock_2_code = new_text

        self.model.save()  # persists the changes

    def index_changed(self, new_text):
        """
        Event handler for a change in the GF Index input.

        :param new_text: The new text value of the input.
        :return: None
        """
        if self.stock_no == 1:
            self.model.stock_1_index = new_text
        else:
            self.model.stock_2_index = new_text

        self.model.save()  # persists the changes


class TimePeriodsChooser(QHBoxLayout):
    """
    Abstraction for a number of widgets to choose Time Periods for graphs.
    """
    def __init__(self, n_time_periods, model):
        """
        Constructor for TimePeriodsChooser

        :param n_time_periods: The number of time periods to receive iinputs
for.
        :param model: The ConfigurationModel to save the time periods to.
        """
        super().__init__()
        self.n_time_periods = n_time_periods
        self.model = model

        self.time_period_inputs = []  # list to hold reference to the
QComboBox(es)

        for n in range(n_time_periods):
            layout = QVBoxLayout()

            # add label to distinguish which time period is being edited
            label = QLabel('Time Period {0}'.format(n + 1))
            layout.addWidget(label)

            # create the input QComboBox for the time period
            input_choice = QComboBox()

            # set editable to false so that user cannot type a new option in,
they are fixed to the provided options
            input_choice.setEditable(False)

            # add all possible time period choices
            for tp in POSSIBLE_TIME_PERIODS:
                input_choice.addItem(tp)

            # perform a check to see if more time periods being chosen this
time vs last time
            # if yes, load the first n where n is the size of the previous
time_periods list
            if n < len(self.model.time_periods):
                selected_tp = self.model.time_periods[n]
                idx = POSSIBLE_TIME_PERIODS.index(selected_tp)
                if idx == -1:
                    idx = 0

                input_choice.setCurrentIndex(idx)
            else:
                input_choice.setCurrentIndex(0)  # else, just select the first
option (so as there are no 'nothing selected') boxes

            # set up signal/slot handler for a new selection
            input_choice.currentIndexChanged.connect(self.handle_tps_changed)

            # add widget to the view
            layout.addWidget(input_choice)

            # add a reference to the input to the list
            self.time_period_inputs.append(input_choice)

            # add the vertical layout for this selector to the section
            self.addLayout(layout)

    def get_time_periods_list(self):
        """
        Utility method that iterates through each of the input boxes and builds
a list of time periods to use.

        :return: List of time periods selected by the user.
        """
        tp_list = []

        for combobox in self.time_period_inputs:
            tp_list.append(combobox.currentText())

        return tp_list

    def handle_tps_changed(self, index):
        """
        Event handler for one of the input boxes' selection being changed.

        :param index: The new selected index. (not used)
        :return: None
        """

        # get the new list of time periods
        new_tp_list = self.get_time_periods_list()

        # update the model's time period list with the new list
        self.model.time_periods = new_tp_list

        # persist the model
        self.model.save()


class PlotStockButton(QPushButton):
    """
    Class for the Plot Stocks button, that invokes the graph rendering flow.
    """
    def __init__(self, graph_pane_collection, time_periods_chooser,
stock_1_chooser, stock_2_chooser):
        """
        Constructor for PlotStockButton with references to key components which
must be read from.

        :param graph_pane_collection: The GraphPaneCollection containing the
output graphs to write to.
```

```
        :param time_periods_chooser: The TimePeriodChooser which contains the
time period inputs for the graphs.
        :param stock_1_chooser: The StockSelector for stock 1 inputs
        :param stock_2_chooser: The StockSelector for stock 2 inputs
        """
        super().__init__("Plot Stocks")
        self.graph_pane_collection = graph_pane_collection
        self.time_periods_chooser = time_periods_chooser
        self.stock_1_chooser = stock_1_chooser
        self.stock_2_chooser = stock_2_chooser

        # connect the button's click signal to the custom event handler
'on_click'
        self.clicked.connect(self.on_click)

    def on_click(self):
        """
        Event handler for the button being clicked. Invokes rendering of new
graphs. Handles errors with rendering
        as message box dialogs.

        :return: None
        """
        t = Thread(target=self.render_new, daemon=True)
        t.start()

    def render_new(self):
        # gather stock 1 details
        stock_1 = {
            'gf_code': self.stock_1_chooser.get_gf_code(),
            'gf_index': self.stock_1_chooser.get_gf_index()
        }

        # gather stock 2 details
        stock_2 = {
            'gf_code': self.stock_2_chooser.get_gf_code(),
            'gf_index': self.stock_2_chooser.get_gf_index()
        }

        # get time periods from TimePeriodChooser
        time_periods = self.time_periods_chooser.get_time_periods_list()

        # invoke the graph rendering
        try:
            success = plot_stocks(self.graph_pane_collection, stock_1, stock_2,
time_periods)

            # handle erroneous inputs with a message box
            if not success:
                msg = QMessageBox()
                msg.setIcon(QMessageBox.Warning)

                msg.setText("You have chosen invalid stocks.")
                msg.setWindowTitle("Invalid Stocks")
                msg.setStandardButtons(QMessageBox.Ok)

                msg.exec_()
        except URLError as error:
            # no connection, display error message
            msg = QMessageBox()
            msg.setIcon(QMessageBox.Critical)
            msg.setWindowTitle("Network Error")
            msg.setText("A network error occurred. Please ensure you are
connected to the internet.")
```

```
        msg.setInformativeText(str(error))
        msg.setStandardButtons(QMessageBox.Ok)
        msg.exec_()
```