VISUALIZING AND FORECASTING STOCKS USING DASH

A SDP PROJECT REPORT

Submitted in partial fulfilment **No index entries found.** *of the requirement for the award of the Degree of*

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

by

GOUTAM JEERALA (19BEC7047)

Under the Guidance of

DR. BEEBI NASEEBA

SCHOOL OF ELECTRONICS ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

MAY 2023

CERTIFICATE

This is to certify that the Capstone Project work titled "**VISUALIZING AND FORECASTING STOCKS USING DASH**" that is being submitted by **GOUTAM JEERALA (19BEC7047)** is in partial fulfilment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for the award of any degree or diploma and the same is certified.

Dr. Beebi Naseeba
Guide

The thesis is satisfactory/unsatisfactory

Internal Examiner **External Examiner**

Approved by

PROGRAM CHAIR
B. Tech. ECE

DEAN
School Of Electronics Engineering

ACKNOWLEDGEMENTS

We would like to express our special thanks of gratitude to our guide **Dr. Beebi Naseeba** who gave me the golden opportunity to do this wonderful project "VISUALIZING AND FORECASTING STOCKS USING DASH”. We would like to thank our subject teacher again for guiding us through the reviews whom without this project would not have been completed and our knowledge of engineering application would not have improved. Lastly, we would like to thank our **SENSE school respective DEAN** for giving us an opportunity to carry out our studies in the University.

GOUTAM JEERALA

ABSTRACT

This document explores the use of Dash, a Python framework for building analytical web applications, to visualize and forecast stocks using the Support Vector Regression (SVR) machine learning model. The ability to effectively visualize and forecast stock market trends is vital for investors and traders. Leveraging the power of Dash, combined with the SVR model, provides a comprehensive solution for analysing and predicting stock prices.

The document begins by introducing Dash and its relevance to stock visualization and forecasting. Dash's interactive components, declarative syntax, and data integration capabilities are highlighted as key features that make it an ideal framework for building stock analysis applications.

Next, the document covers the process of acquiring and preparing stock market data. It discusses methods for gathering reliable data and emphasizes the importance of data cleaning, pre-processing, and feature engineering to ensure accurate forecasting results.

The document then delves into visualizing stock data using Dash. It illustrates the creation of interactive charts, plotting historical stock prices, and visualizing technical indicators. Through code examples and demonstrations, readers gain insights into effectively displaying and exploring stock market data using Dash's intuitive visualizations.

The subsequent section focuses on utilizing the SVR machine learning model for stock price forecasting. An introduction to forecasting techniques is provided, with a particular emphasis on time series analysis and modelling. Readers learn how to implement the SVR model using Dash and understand the process of evaluating forecast accuracy.

Building upon the previous sections, the document guides readers through the development of a stock prediction dashboard using Dash. It covers dashboard layout design, integration of stock data visualization and SVR forecasting, and the addition of user interactivity and controls. Practical examples demonstrate how to create an engaging and informative dashboard for stock market analysis.

Finally, the document concludes with a summary of key takeaways and suggests future directions for enhancing the visualization and forecasting capabilities of stocks using Dash and the SVR model.

By combining the flexibility of Dash's web application framework with the predictive power of the SVR machine learning model, investors and traders can gain valuable insights into stock market trends and make informed decisions. The document provides a comprehensive guide for leveraging these tools effectively and offers a foundation for further exploration in the field of stock visualization and forecasting.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In today's dynamic and complex financial markets, the ability to effectively visualize and forecast stock market trends is crucial for investors and traders. The advent of data science and visualization tools has revolutionized the way we analyse and interpret vast amounts of stock market data. This document will explore the powerful combination of Dash, a Python framework for building analytical web applications, and the art of stock visualization and forecasting.

Dash provides a comprehensive platform for developing interactive and data-driven web applications. Its intuitive syntax, a wide range of interactive components, and seamless integration with popular data manipulation libraries make it an ideal choice for building stock analysis tools. By leveraging the flexibility and interactivity of Dash, combined with robust forecasting models, we can unlock valuable insights from historical stock data and make more informed investment decisions.

The process of visualizing and forecasting stocks using Dash involves several key steps. First, we acquire and prepare the necessary stock market data. This includes gathering reliable and up-to-date data, performing data cleaning and preprocessing, and applying feature engineering techniques to extract meaningful information from the raw data.

Once the data is ready, we can dive into the exciting world of stock visualization using Dash. We can create interactive charts and graphs to visualize historical stock prices, analyze trends, and identify patterns. Additionally, Dash allows us to incorporate technical indicators, such as moving averages or relative strength index (RSI), to provide deeper insights into stock market behaviour.
However, visualizing historical data is just the beginning. To make informed investment decisions, we need to forecast future stock prices. This is where the power of machine learning models comes into play. In this document, we focus on one such model, Support Vector Regression (SVR), which is well-suited for time series forecasting. We explore the fundamentals of SVR, its implementation using Dash, and the evaluation of forecast accuracy.

The ultimate goal is to build a comprehensive stock prediction dashboard using Dash. This interactive dashboard will combine visualizations of historical stock data, technical indicators, and real-time forecasting to provide users with a holistic view of stock market trends. Users will have the ability to explore and interact with the data, adjust parameters, and make informed decisions based on the generated insights.

By harnessing the power of Dash and the forecasting capabilities of the SVR model, we can unlock the hidden patterns and trends within stock market data. This document will guide you through visualizing and forecasting stocks using Dash, empowering you to gain a competitive edge in the world of stock market analysis and make data-driven investment choices.

1.1. OBJECTIVES

The following are the objectives of this project:

- Develop a comprehensive understanding of Dash: Gain a deep understanding of the Dash framework, its features, and its capabilities for building analytical web applications.
- Acquire and pre-process stock market data: Learn how to gather reliable and relevant stock market data, clean and preprocess the data to ensure accuracy, and apply feature engineering techniques to extract meaningful information.
- Visualize stock market data: Utilize Dash's interactive components and visualization capabilities to create visually appealing and insightful charts, graphs, and dashboards for analysing historical stock prices and identifying trends and patterns.
- Incorporate technical indicators: Explore the integration of technical indicators, such as moving averages, RSI, or MACD, into the stock visualizations to enhance the analysis and gain deeper insights into stock market behaviour.
- Implement the Support Vector Regression (SVR) model: Understand the fundamentals of SVR and its suitability for time series forecasting. Learn how to train and implement the SVR model using Dash to forecast future stock prices.
- Evaluate forecast accuracy: Gain knowledge of evaluation techniques for forecasting models, assess the accuracy and reliability of the SVR model's predictions, and understand the limitations and potential sources of error in stock market forecasting.
- Build an interactive stock prediction dashboard: Design and develop a user-friendly and interactive dashboard using Dash that integrates visualizations of historical stock data, technical indicators, and real-time forecasting. Provide users with the ability to explore and interact with the data, adjust parameters, and make informed investment decisions.
- Empower informed decision-making: Enable investors and traders to leverage the power of Dash and the SVR model to gain valuable insights from stock market data, make data-driven investment decisions, and enhance their understanding of stock market trends.

1.2. BACKGROUND AND LITERATURE SURVEY

The field of stock market analysis and forecasting has been the subject of extensive research and development due to its critical importance in financial decision-making. With the advancement of data science and machine learning techniques, there has been a significant emphasis on utilizing these approaches to visualize and forecast stock market trends accurately. This project builds upon the existing body of knowledge and research in the following areas:

Stock Market Analysis and Visualization:

Numerous studies have focused on visualizing and analyzing stock market data to uncover patterns, trends, and anomalies. Visualization techniques such as line charts, candlestick charts, and moving averages have been widely used to represent historical stock prices and understand market behavior. Researchers have explored various visualization tools and techniques to enhance the interpretation of complex financial data.

Forecasting Techniques for Stock Market Prediction:

Stock market forecasting involves predicting future price movements based on historical data. Traditional methods like time series analysis, autoregressive integrated moving averages (ARIMA), and exponential smoothing models have been widely employed. In recent years, machine learning algorithms, including support vector regression (SVR), neural networks, and random forest, have gained popularity for their ability to capture non-linear patterns and improve forecast accuracy.

Dash as a Visualization and Web Application Framework:

Dash, as a Python framework, has gained recognition as a powerful tool for creating interactive web applications and visualizations. It provides a declarative syntax, a wide range of interactive components, and seamless integration with data manipulation libraries like Pandas. Previous research has demonstrated the effectiveness of Dash in building analytical applications for various domains, including finance, by enabling real-time data visualization and interactivity.

Integration of Machine Learning with Visualization:

The integration of machine learning models with visualization techniques has been explored to enhance stock market analysis and forecasting. Researchers have applied algorithms like SVR, long short-term memory (LSTM) networks, and ensemble methods to predict stock prices. Integrating these models within interactive dashboards enables users to explore historical data, evaluate model performance, and make data-driven investment decisions.

By conducting a comprehensive literature survey, this project aims to build upon and contribute to the existing knowledge by combining the strengths of Dash's interactive visualization capabilities with the forecasting power of the SVR model. The project seeks to provide a practical implementation of these techniques in the context of visualizing and forecasting stocks, enabling users to gain valuable insights and make informed decisions in the dynamic world of stock market analysis.

1.3. ORGANIZATION OF THE REPORT

The remaining chapters of the project report are described as follows:

- Methodology
- Data Acquisition and Preparation
- Visualizing Stock market data
- Forecast Stock prices using SVR.
- Building the Stock prediction Dashboard.
- Results and analysis
- Conclusion
- References

CHAPTER 2

VISUALIZING AND FORECASTING STOCKS USING DASH

This Chapter describes Briefly introduce the topic and its significance and provide an overview of the objectives of the project.

2.1. INTRODUCTION:

The topic of "Visualizing and Forecasting Stocks Using Dash" focuses on the utilization of Dash, a Python framework, to create interactive web applications for visualizing and forecasting stock market trends. The significance of this topic lies in the critical need for investors and traders to analyze historical data, identify patterns, and make informed decisions based on accurate forecasts in the dynamic and competitive world of financial markets.

Stock market visualization plays a vital role in understanding the historical behavior of stocks, identifying trends, and recognizing patterns that can influence future price movements. Effective visualization techniques can provide valuable insights into market dynamics, enabling investors to develop strategies, identify potential risks, and seize profitable opportunities.

Moreover, accurate forecasting of stock prices is of utmost importance for investors and traders. By leveraging machine learning models such as the Support Vector Regression (SVR), it becomes possible to analyze historical patterns and predict future stock prices with enhanced accuracy. This empowers investors to make informed decisions, optimize portfolio management, and minimize risks.

The integration of Dash, a powerful visualization and web application framework, with the SVR model enables the creation of interactive dashboards that allow users to explore stock market data, visualize historical trends, and forecast future prices in real time. This combination empowers users to gain deeper insights, evaluate investment strategies, and make data-driven decisions based on robust analysis.

The significance of this topic lies in the practical application of advanced visualization techniques and machine learning algorithms to support investment decision-making. By harnessing the power of Dash and SVR, this project equips investors and traders with a comprehensive toolset to effectively analyse, visualize, and forecast stocks, enhancing their ability to navigate the complexities of the stock market and achieve their financial goals.

In this Chapter, the methodology involved describing the steps for acquiring and preprocessing stock market data, followed by discussing the visualization techniques employed using Dash, and finally detailing the implementation of the Support Vector Regression (SVR) model.

2.2. Proposed System

The proposed system for "Visualizing and Forecasting Stocks Using Dash" consists of an interactive web-based application that leverages the Dash framework to visualize historical stock data, incorporate technical indicators, and provide real-time forecasting using the SVR model. The system aims to empower users with the ability to explore and analyse stock market trends, make informed investment decisions, and evaluate the accuracy of the forecasts.

Key components of the proposed system include:

- User Interface:

The system will feature a user-friendly and intuitive interface that allows users to interact with the stock market data and visualization components. The interface will provide options for selecting stocks, choosing time ranges, adjusting parameters, and exploring different visualization options.

- Data Acquisition and Pre-processing:

The system will acquire reliable and up-to-date stock market data from relevant sources.
The acquired data will undergo pre-processing steps, including cleaning, handling missing values, and applying transformations if required.

- Visualization

The system will employ Dash's interactive components to create visually appealing and informative charts, graphs, and dashboards. Users will be able to visualize historical stock prices, trends, and technical indicators such as moving averages, RSI, or MACD. Interactive features like zooming, panning, and tooltips will enhance the user experience and facilitate data exploration.

- SVR Model Integration:

The system will implement the Support Vector Regression (SVR) model for forecasting future stock prices. The SVR model will be trained using historical stock data and relevant features. Real-time forecasts will be generated based on the trained SVR model, allowing users to access options.

- Evaluation and Performance Metrics:

The system will provide evaluation metrics to assess the accuracy and performance of the SVR model's forecasts. Metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or accuracy scores will be calculated to measure the forecast quality.

- Interactive Stock Prediction Dashboard:

The system will integrate the visualization components, SVR model forecasts, and evaluation metrics into an interactive stock prediction dashboard. Users will have the flexibility to customize the dashboard, adjust model parameters, and explore different visualizations based on their preferences.

- Deployment and Accessibility:

The system will be deployed as a web application, accessible via a web browser. Users will be able to access the system from different devices, including desktop computers, laptops, tablets, and smartphones.

The proposed system aims to provide a comprehensive platform for visualizing and forecasting stocks using Dash. It combines interactive visualization, technical analysis, and real-time forecasting capabilities to empower users with valuable insights into stock market trends and support data-driven investment decision-making.

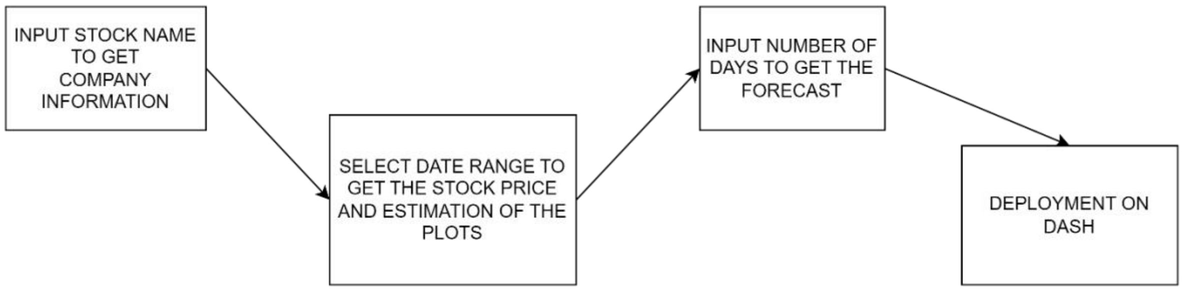

INPUT STOCK NAME TO GET COMPANY INFORMATION
SELECT DATE RANGE TO GET THE STOCK PRICE AND ESTIMATION OF THE PLOTS
INPUT NUMBER OF DAYS TO GET THE FORECAST
DEPLOYMENT ON DASH

Figure 1 Block Diagram

2.3. Working Methodology

Here is a simplified brief working methodology for visualizing and forecasting stocks using Dash and the SVR (Support Vector Regression) machine learning model. The first step is Data Acquisition, where you collect historical stock market data from reliable sources such as financial APIs or databases. This ensures that you have accurate and up-to-date information to work with.

Once you have the data, the next step is Data Preprocessing. This involves cleaning the data by handling missing values, outliers, and inconsistencies. You may also need to perform data transformations and normalization to prepare it for analysis. By ensuring the quality and integrity of the data, you can avoid any potential biases or errors in the subsequent steps.

Visualization plays a crucial role in understanding and analyzing the stock market data. That's where Dash, a Python framework for creating web applications, comes into play. In the Visualization step, you use Dash to develop interactive visualizations of the stock market data. By implementing various charts, graphs, and dashboards, you can effectively display historical stock prices, trends, and technical indicators. This enables users to explore the data visually and gain insights into market patterns.

Feature Engineering is an essential step in enhancing the predictive power of the SVR model. Here, you extract relevant features from the stock market data that can provide valuable information for forecasting. This may involve incorporating additional data sources, lagging variables, moving averages, or other technical indicators. By engineering informative features, you can improve the model's accuracy and ability to capture complex patterns in the data.

With the preprocessed data and engineered features, it's time to train the SVR model. In the Train the SVR Model step, you split the data into training and testing sets. The training set is used to train the SVR model, adjusting its hyperparameters as needed. SVR is a machine learning algorithm suitable for time series forecasting, making it a powerful tool for predicting future stock prices.

Once the SVR model is trained, you move on to the Forecasting step. Here, you utilize the trained model to generate predictions for future stock prices. By applying the model to the testing data, you can evaluate its performance in terms of accuracy and other relevant metrics. This step provides valuable insights into the potential future trends in the stock market.

Integration with Dash is a key aspect of this methodology. In the Integration with Dash step, you incorporate the SVR model's forecasts into the Dash visualization framework. This connection allows users to explore the historical data and observe the predicted future trends within the interactive charts and graphs. It provides a dynamic and comprehensive view of the stock market, empowering users to make informed decisions.

To further enhance the user experience, the User Interaction step comes into play. Within the Dash application, you provide interactive features that allow users to adjust parameters, select specific stocks or time periods, and visualize forecasted trends in real time. This interactivity enables users to personalize their analysis and gain a deeper understanding of the data.

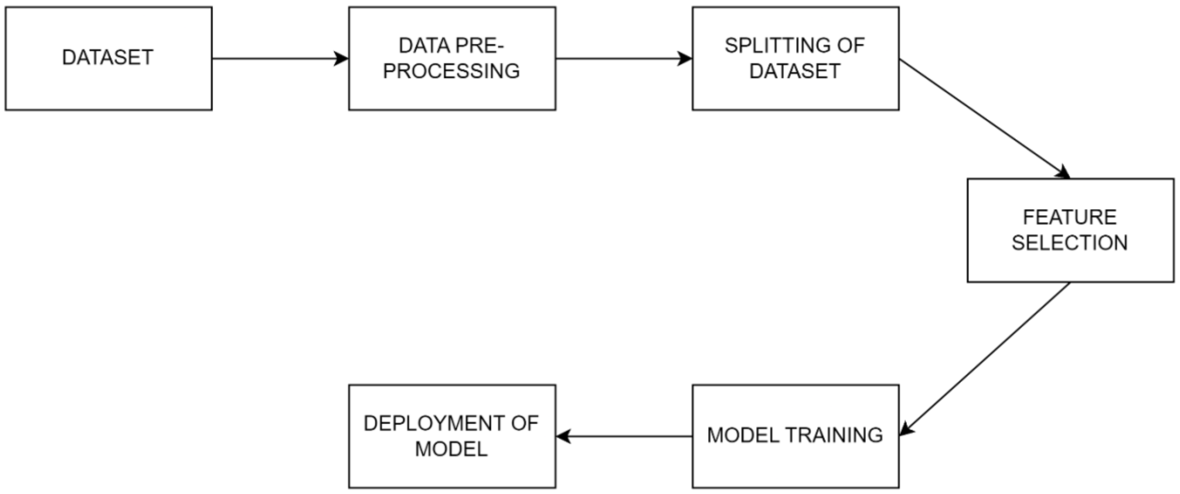


Figure 2 Methodology

The Evaluation and Analysis step involves assessing the accuracy and reliability of the SVR model's forecasts. By comparing the model's predictions to the actual stock prices, you can evaluate its performance and identify any patterns or trends. This analysis provides valuable insights for making informed investment decisions based on the model's predictions.

Finally, in the Deployment step, you make the Dash application, along with the integrated SVR model, accessible to users. This can be done by deploying the application on a web server or a cloud platform, ensuring its availability and scalability. By making the application easily accessible, you enable users to leverage the visualization capabilities of Dash and the forecasting power of the SVR model to gain valuable insights into stock market trends and make informed investment decisions.

By following this simplified working methodology, users can leverage the combined strengths of
Dash's visualization capabilities and the SVR model's forecasting power. This empowers them to gain valuable insights into stock market trends, enabling informed decision-making for investment purposes."

CHAPTER-3

3.1 STANDARDS

The various standards of this project can include the following:

Data Quality Standards: Ensure that the collected historical stock market data is accurate, reliable, and obtained from reputable sources. Validate and verify the data to minimize errors, inconsistencies, and biases.

Best Practices in Data Preprocessing: Adhere to industry best practices while handling missing values, outliers, and inconsistencies in the data. Implement appropriate techniques for data cleaning, transformation, and normalization to ensure the data is suitable for analysis.

Visualization Standards: Follow design principles and guidelines to create visually appealing and informative visualizations using Dash. Consider factors such as color schemes, chart types, labelling, and interactivity to enhance the user experience and facilitate clear understanding of the data.

Feature Engineering Guidelines: Apply sound feature engineering techniques to extract relevant features from the stock market data. Consider domain knowledge, statistical analysis, and feature selection methods to identify the most informative variables for improving the predictive power of the SVR model.

Model Training and Evaluation: Follow established practices for splitting the data into training and testing sets, ensuring proper separation of temporal data. Adjust SVR hyperparameters using appropriate optimization techniques, such as grid search or random search. Evaluate the model's performance using relevant metrics, including accuracy, mean squared error, or other domain-specific metrics.

Integration and Deployment: Adhere to best practices for integrating the SVR model's forecasts into the Dash visualization framework. Ensure smooth integration between the model and the interactive charts and graphs. Deploy the application on a web server or cloud platform following industry standards for availability, scalability, and security.

Documentation: Maintain clear and comprehensive documentation throughout the project, including data sources, preprocessing steps, feature engineering techniques, model training details, evaluation results, and deployment procedures. Documenting the project ensures reproducibility, facilitates collaboration, and enables future enhancements or troubleshooting.

Ethical Considerations: Adhere to ethical standards and legal requirements in collecting, analyzing, and utilizing stock market data. Respect data privacy and ensure compliance with relevant regulations, such as General Data Protection Regulation (GDPR) or applicable financial regulations.

Code Quality and Maintainability: Follow coding best practices, such as using modular and well-documented code, adhering to PEP 8 guidelines for Python, implementing appropriate error handling, and ensuring code readability.

Continuous Improvement and Iteration: Embrace an iterative approach to improve the project over time. Monitor and analyze the performance of the SVR model, seek feedback from users, and identify areas for enhancement or optimization. Continuously update and refine the project to incorporate new features, data sources, or emerging best practices in the field of stock market visualization and forecasting.

3.2 SYSTEM DETAILS

This section describes the software and hardware details of the system:

1. Hardware Requirements: The specific hardware requirements will depend on the size and complexity of the dataset. It is recommended to have a computer with a multi-core processor, sufficient RAM (8GB or higher), and ample storage capacity to handle the data and computational requirements effectively.

2. Software Dependencies: The project relies on several software dependencies, including:
 - Python: The project is developed using Python programming language. Make sure to have Python installed on the system along with the necessary packages. In our case, we are using the Visual Code Studio.

 - Dash: Dash is a Python framework used for creating web applications. Install Dash using the Python package manager pip. Open the command prompt or terminal and run the command:
‘`pip install dash`’.

 - Data Analysis Libraries: Install pandas for data manipulation and analysis using the command: `pip install pandas`. NumPy is another essential library for numerical operations, and you can install it with the command: `pip install numpy`. Scikit-learn is used for machine learning tasks and can be installed with the command:
`pip install scikit-learn`.

3. Data Visualization Libraries: Plotly is a powerful visualization library that integrates well with Dash. Install it using the command: `pip install plotly`. Dash Core Components are necessary for creating interactive components, and they can be installed with the command:
`pip install dash-core-components`

4. Data Acquisition: yfinance is a popular Python library that provides a simple and convenient way to access historical stock market data, as well as real-time stock data, from Yahoo Finance. It acts as a Python wrapper for the Yahoo Finance API, allowing users to fetch data for individual stocks, indices, ETFs, currencies, and more.

- Here are some key features and functionalities of yfinance:

- Historical Data Retrieval: yfinance enables users to retrieve historical price data for stocks and other financial instruments. It supports a range of options to specify the time period, frequency (daily, weekly, monthly), and data range (start date to end date) for the requested historical data.
- Real-Time Data Streaming: In addition to historical data, yfinance also provides access to real-time stock data. Users can obtain up-to-date information on stock prices, volumes, and other relevant details.
- Financial Data and Statistics: yfinance allows users to access various financial data and statistics for a specific stock or instrument. This includes information such as market capitalization, dividend yield, P/E o, outstanding shares, and more.
- Ticker Symbol Lookup: yfinance offers a ticker symbol lookup functionality to retrieve the ticker symbol for a given stock or company. This is useful when searching for specific securities or when needing to obtain the correct symbol for data retrieval.
- Multiple Stock Data Retrieval: Users can retrieve data for multiple stocks or financial instruments in a single call, which makes it efficient for bulk data retrieval and analysis.
- Customizable Data Columns: yfinance allows users to specify the specific columns or attributes they want to retrieve for a given stock or instrument. This includes information such as open price, high price, low price, close price, volume, and more.
- Support for Indices and ETFs: In addition to individual stocks, yfinance supports fetching data for stock indices, ETFs (Exchange-Traded Funds), currencies, and other financial instruments.
- Integration with Pandas: yfinance seamlessly integrates with the popular data manipulation and analysis library, Pandas. The retrieved data can be easily transformed into a Pandas DataFrame, enabling further analysis, visualization, and manipulation.
- Install it using the command: **`pip install yfinance`**

5. Data Preprocessing: Data preprocessing tasks involve cleaning the data by handling missing values, outliers, and inconsistencies. Use libraries like pandas to perform data cleaning, imputation, and normalization.

```
# load the data

    df = yf.download(stock, period='60d')
    df.reset_index(inplace=True)
    df['Day'] = df.index

    days = list()
    for i in range(len(df.Day)):
        days.append([i])
```

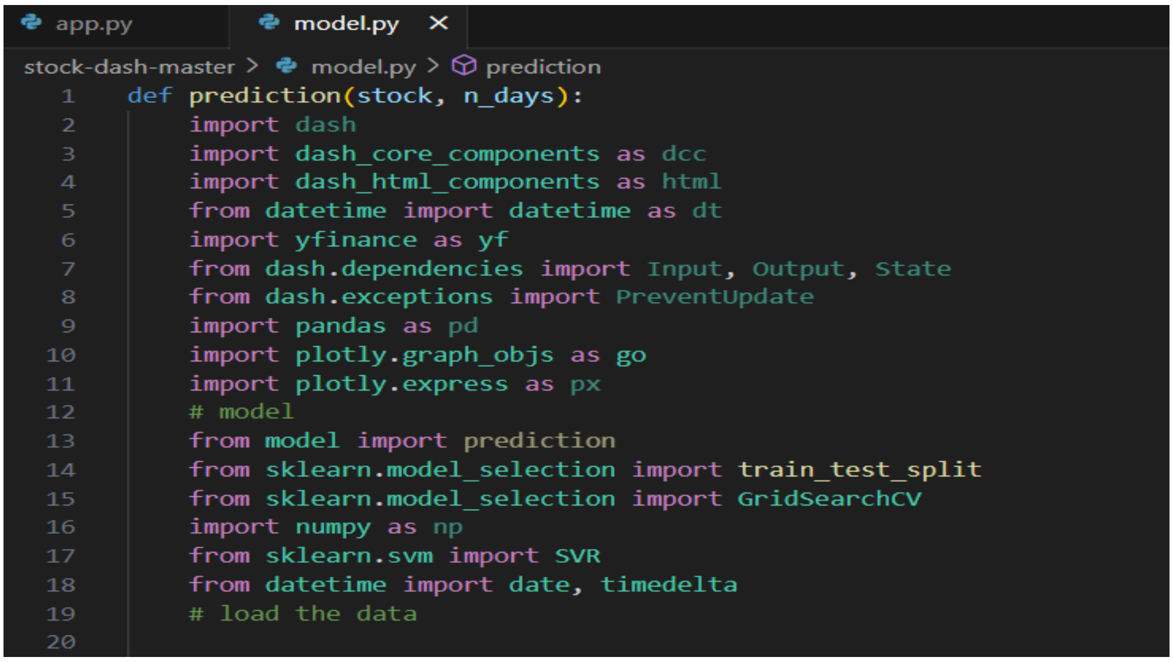
6. Feature Engineering: Apply feature engineering techniques to extract relevant features from the stock market data. This can include lagging variables, moving averages, technical indicators (such as RSI, MACD, or Bollinger Bands), or other domain-specific features. Pandas provides a range of functions and methods to manipulate the data and derive new features.

7. Model Training: Split the preprocessed data into training and testing sets using scikit-learn's train_test_split() function. Instantiate an SVR model from scikit-learn’s SVR class. Train the model using the training data by calling the fit() method and passing the feature variables and target variables. Adjust the hyperparameters of the SVR model, such as the kernel type, regularization parameter (C), and kernel coefficient (gamma), using techniques like grid search or random search.

8. Evaluation Metrics: Evaluate the performance of the SVR model using suitable metrics. Common metrics for regression tasks include mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), or R-squared (coefficient of determination). Calculate these metrics by comparing the predicted values from the model with the actual target values in the testing dataset.

9. Deployment: Deploy the Dash application along with the integrated SVR model on a web server or cloud platform. Ensure the deployment follows best practices for availability, scalability, and security.

10. Maintenance: Maintain comprehensive documentation covering the project's architecture, data sources, preprocessing steps, feature engineering techniques, model training details, evaluation results, and deployment instructions. Regularly update and maintain the system to address any issues, incorporate new features, and keep it up to date with changes in libraries or dependencies.



```python
def prediction(stock, n_days):
    import dash
    import dash_core_components as dcc
    import dash_html_components as html
    from datetime import datetime as dt
    import yfinance as yf
    from dash.dependencies import Input, Output, State
    from dash.exceptions import PreventUpdate
    import pandas as pd
    import plotly.graph_objs as go
    import plotly.express as px
    # model
    from model import prediction
    from sklearn.model_selection import train_test_split
    from sklearn.model_selection import GridSearchCV
    import numpy as np
    from sklearn.svm import SVR
    from datetime import date, timedelta
    # load the data

```

Figure 3 Libraries installation

3.3 SOFTWARE DETAILS

- **Python**: The project is primarily developed using the Python programming language. Python provides a wide range of libraries and frameworks for data analysis, visualization, and web development.

- **Plotly**: Plotly is a powerful graphing library that enables interactive and dynamic visualizations. It offers a wide range of chart types and customization options, making it ideal for creating interactive stock market visualizations within Dash applications. All graph objects are dictionary- and list-like objects used to generate and/or modify every feature of a Plotly plot. The plotly.tools module contains many helpful functions facilitating and enhancing the Plotly experience. Functions for subplot generation, embedding Plotly plots in IPython notebooks, saving and retrieving your credentials are defined in this module. A plot is represented by Figure object which represents Figure class defined in plotly.graph_objs module. Its constructor needs following parameters −

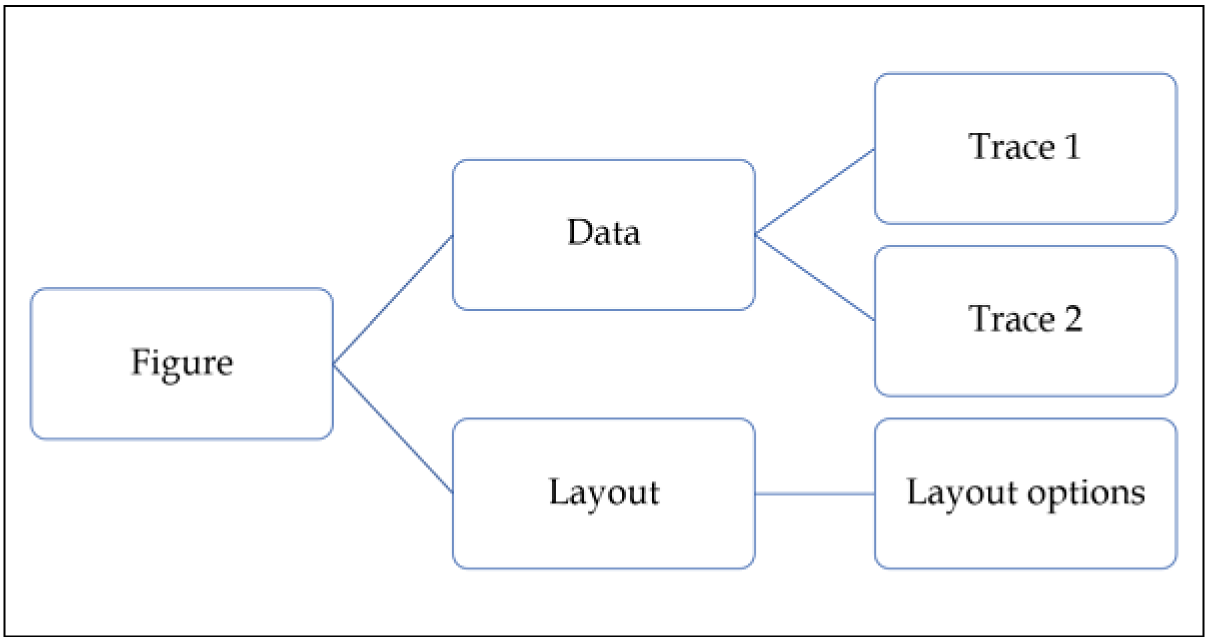
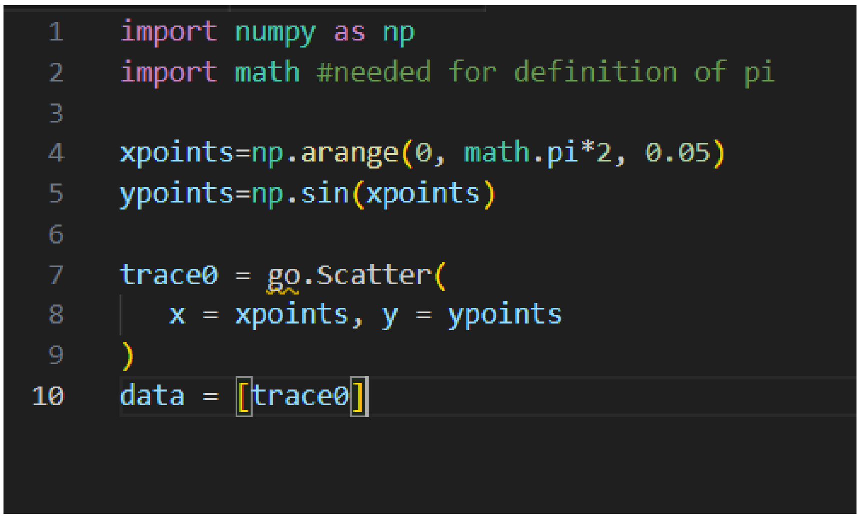


Figure 4 Plotly Architecture



```python
import numpy as np
import math #needed for definition of pi

xpoints=np.arange(0, math.pi*2, 0.05)
ypoints=np.sin(xpoints)

trace0 = go.Scatter(
    x = xpoints, y = ypoints
)
data = [trace0]
```

Scikit-learn: Scikit-learn is a popular machine-learning library in Python. It provides a range of algorithms and tools for data modeling, including the SVR (Support Vector Regression) algorithm used for stock market forecasting in this project. Scikit-learn facilitates model training, evaluation, and prediction tasks.

The module structure model defines the organization of the system's source code and related external systems, in terms of the modules into which the individual source files are collected and the dependencies among these modules. Layers are identified for scikit-learn with each layer consisting of one or more module(s). These layers are:

Domain layer: consisting of all main functionality modules: data transformations , data loader, model selection, supervised learning , and unsupervised learning .

Utility layer: consisting of modules that support basic functionality that can be used in domain layer, such as testing, validation, preprocessing, sparse tool, and external configuration. Platform layer, which contains modules of the required packages, such as python, NumPy, and SciPy. Build tool layer, which contains build modules to build the library.

Each module consists of files to download, install, testing, or setting the required library. Dependency of one layer to the other layer(s) is demonstrated by a dashed arrow which points to the destination of the required layer. As an example, the utility layer uses all libraries available from python, NumPy, SciPy, and Pandas by importing them in the module. In addition, there are explicit intermodular dependencies for all modules in the domain layer for python because all files under each module requires python.

SVR was initially proposed by Drucker et al., which is a supervised learning technique, based on the concept of Vapnik's support vectors. SVR aims at reducing the error by determining the hyperplane and minimising the range between the predicted and the observed values. Minimising the value of w in the Equation given below is similar to the value defined to maximise the margin

$$\min ||w||^2 + C\sum_{i}^{n}(\xi_i^+ + \xi_i^-)$$

where the summation part represents an empirical error. Hence, to minimise this error, we use the following equation.

$$f(x) = \sum_{i}^{n}(\alpha_i^* + a_i)K(x, x_i) + B$$

where the alpha term represents the Lagrange multiplier and its value is greater than equal to 1. ***K*** represents the kernel function and ***B*** represents the bias term. In this study, we have used the Polynomial kernel given by:

$$K(x, x_i) = \gamma(x * x_i + 1)^d$$

Where ***d*** is the polynomial degree and ***γ*** is the polynomial constant.

SVR performs better performance prediction than other algorithms like Linear Regression, KNN and Elastic Net, due to the improved optimisation strategies for a broad set of variables. Moreover, it is also flexible in dealing with geometry, transmission, data generalisation and additional functionality of kernel.

This additional functionality enhances the model capacity for predictions by considering the quality of features. SVR is used in many applications such as image processing, remote sensing, and blockchain. It has superb generalisation competence along with high accuracy. Also, the computational complexity is independent of the input feature data set.

The training samples influence the SVR model's fitting performance since the SVR algorithm is sensitive to the interference in the training data. Besides, SVR is useful in resolving high dimensional features regression problem, and well-function if the feature metrics is larger than the size of the sample.

In this study, we have extracted four features, namely anchor ratio, transmission range, node density and the number of iterations from modified CS algorithm simulation.Feature scaling is essential for SVR because, when one function has greater magnitudes than others, the other features will dominate while measuring the distance. To avoid this, we have used various standardisation approaches. Based on this, we have proposed three methods, as shown **in Fig.2**

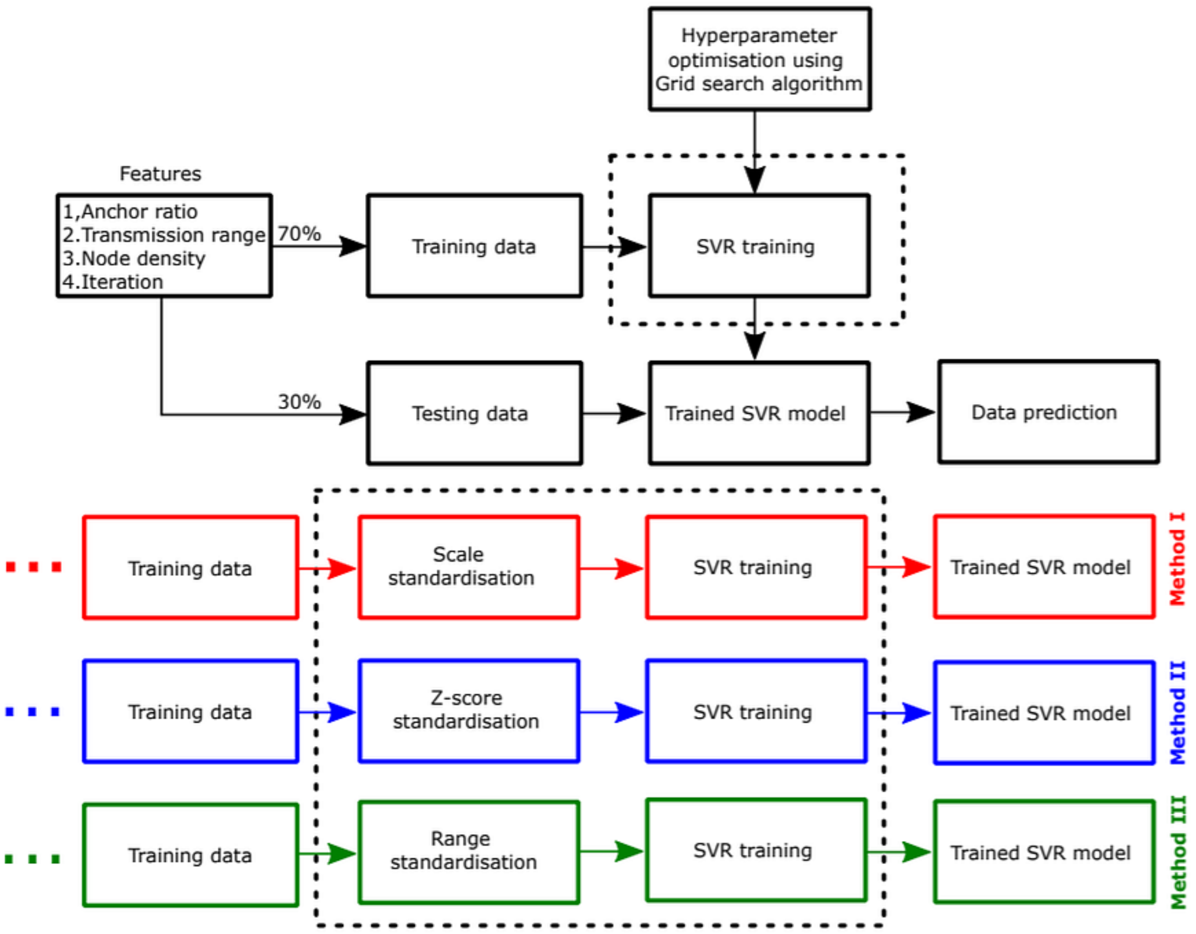


Figure 5 SVR Architecture

The method I is S-SVR (Scaling SVR). In this method, we first standardised the features using the Equation given below;

$$x_s = \frac{x}{\sigma}$$

Where **x** is the feature vector, **xs** is the standardised data, and **σ** is the standard deviation of the feature vector. The method II is Z-SVR (Z-score SVR). In this method, we have standardised the features using the Equation given below;

$$x_s = \frac{x - \overline{x}}{\sigma}$$

Where x bar is the mean of the feature vector. The method III is the R-SVR (Range SVR). In this method, we have standardised the features using the Equation given below;

$$x_s = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Afterward, we trained and tested the SVR models in a 70:30 ratio, as shown in Fig. 2. In this study, the dimension of the features vector is 107 × 1. Hence, we have used 75 data for training and the remaining 32 for testing.

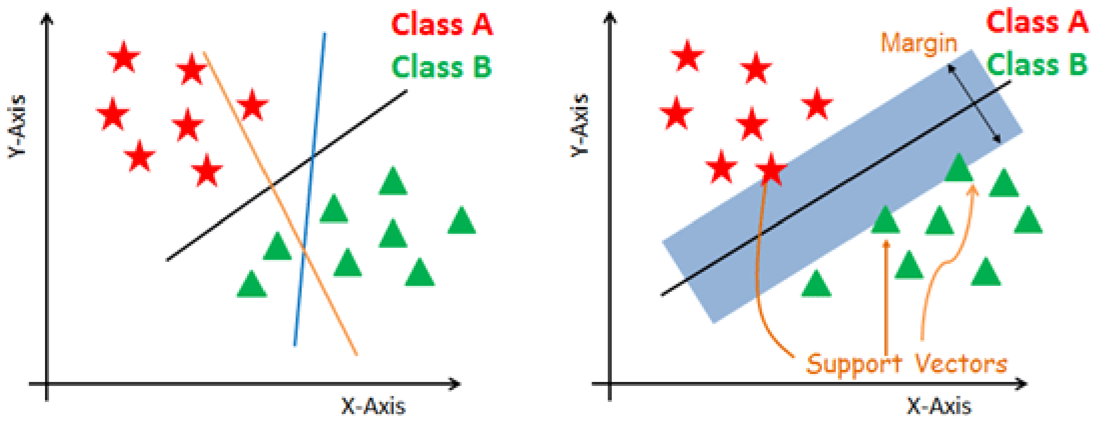
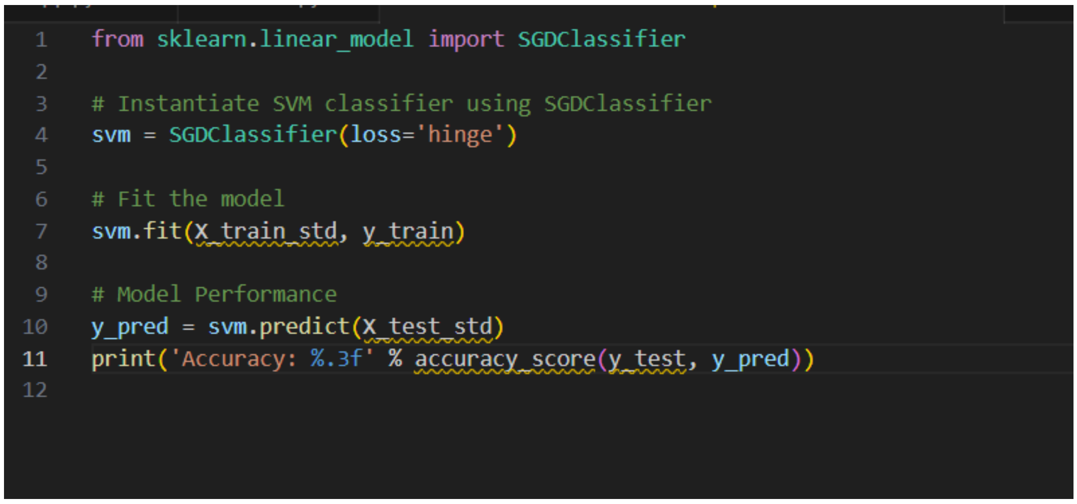


Figure 6 Support Vector Regression (i)



```python
from sklearn.linear_model import SGDClassifier

# Instantiate SVM classifier using SGDClassifier
svm = SGDClassifier(loss='hinge')

# Fit the model
svm.fit(X_train_std, y_train)

# Model Performance
y_pred = svm.predict(X_test_std)
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
```

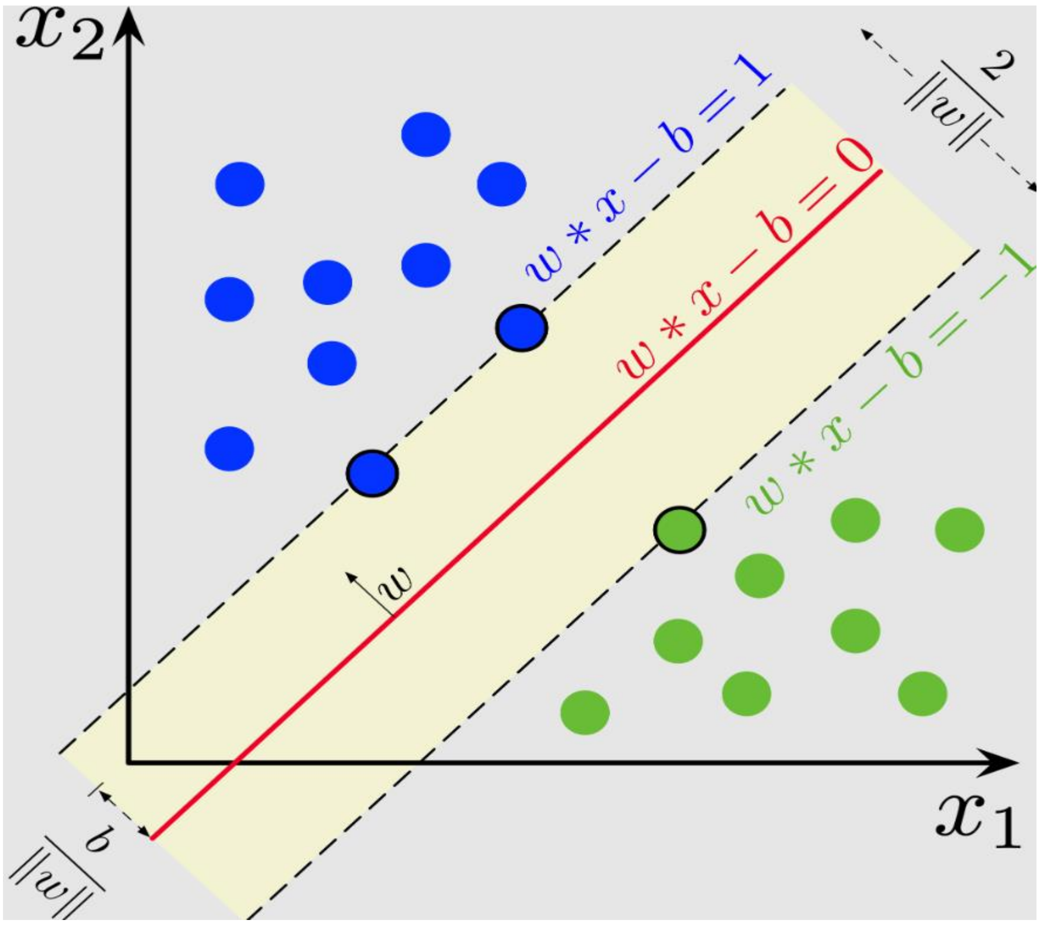


Figure 6 Support Vector Regression (ii)

Financial APIs: To acquire historical stock market data, the project may utilize financial APIs such as Alpha Vantage, Yahoo Finance, or Quandl. These APIs allow access to historical price data, company fundamentals, and other financial indicators.

```
$ pip install yfinance --upgrade --no-cache-dir
```

We will have to import it from the *stock_info* module, so we do:

```
from yahoo_fin.stock_info import get_data
```

It takes the arguments:

- **ticker**: case insensitive ticker of the desired stock/bond
- **start_date**: date you want the data to start from (mm/dd/yyyy)
- **end_date**: date you want the data to end (mm/dd/yyyy)
- **index_as_date**: {True, False}. Default is true. If true then the dates of the records are set as the index, else they are returned as a separate column.
- **interval**: {“1d”, “1wk”, “1mo”}. Refers to the interval to sample the data: “1d”= daily, “1wk”= weekly, “1mo”=monthly.

```
get_data(ticker, start_date = None, end_date = None, index_as_date =
True, interval = “1d”)
```

```
def get_stock_price_fig(df):

    fig = px.line(df,
                  x="Date",
                  y=["Close", "Open"],
                  title="Closing and Openning Price vs Date")

    return fig
```

```
@app.callback([
    Output("description", "children"),
    #Output("logo", "src"),
    Output("ticker", "children"),
    Output("stock", "n_clicks"),
    Output("indicators", "n_clicks"),
    Output("forecast", "n_clicks")
], [Input("submit", "n_clicks")],
   [State("dropdown_tickers", "value")]
)

@app.callback([
    Output("graphs-content", "children"),
], [
    Input("stock", "n_clicks"),
    Input('my-date-picker-range', 'start_date'),
    Input('my-date-picker-range', 'end_date')
], [State("dropdown_tickers", "value")])
def stock_price(n, start_date, end_date, val):
    if n == None:
        return [""]
        #raise PreventUpdate
    if val == None:
        raise PreventUpdate
    else:
        if start_date != None:
            df = yf.download(val, str(start_date), str(end_date))
        else:
            df = yf.download(val)

    df.reset_index(inplace=True)
    fig = get_stock_price_fig(df)
    return [dcc.Graph(figure=fig)]
```

Web Server or Cloud Platform: The Dash application, along with the integrated SVR model, can be deployed on a web server or cloud platform. The deployment and release lifecycle begins when your git push to Dash Enterprise. It creates a new image based off of the changes that you pushed and runs the image as containers. The files you include in your project folder determine how Dash Enterprise builds, deploys, and releases your apps and workspaces.

A few key files are required for your Dash app to successfully deploy on Dash Enterprise. In addition to **app.py**, you'll need a requirements.txt file to describe your app's dependencies and a Procfile to declare what commands and processes should be run to run the Dash app and any other background processes.

```
|-- app.py
|-- Procfile
|-- requirements.txt
```

app.py

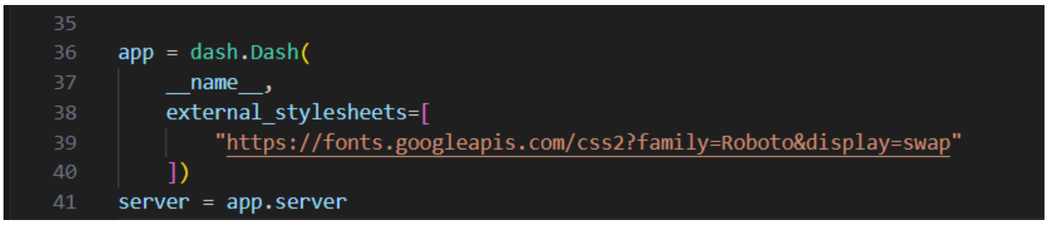
By convention, this is usually called app.py or index.py. This file is called by the command you specify in your Procfile. It contains your Python code and must be placed in your project's root directory. This file must also contain a line that defines the server variable so that it can be exposed for the Procfile:

```
import dash

app = dash.Dash(__name__)
```

```
server = app.server
...
```

A basic requirements.txt file looks like this:

```
dash-design-kit==1.6.7
dash==2.4.1
gunicorn==20.0.4
pandas==1.1.4
```



```
app = dash.Dash(
    __name__,
    external_stylesheets=[
        "https://fonts.googleapis.com/css2?family=Roboto&display=swap"
    ])
server = app.server
```

A simple Dash app Procfile looks like this:

```
web: gunicorn app:server --workers 4
```

A Dash app running a background task queue might have a Procfile similar to this:

```
web: gunicorn app:server --workers 4
worker: celery -A app:celery_instance worker
```

Note: Process type names are arbitrary except for web—a special process. In the example above, worker could be anything, but we recommend using descriptive names as they appear in logs.

A Dash app periodically generating reports with the Snapshot Engine might have a Procfile like this:

```
web: gunicorn index:server --workers 4
worker: celery -A index:celery_instance worker --concurrency=2
scheduler: celery -A index:celery_instance beat
```

Integrated Development Environment (IDE): An IDE such as Visual Studio Code is often used for development, providing features like code editing, debugging, and project management to streamline the development process.

Python Version:

To change the Python version, specify the version you want to use in project.toml. Note that changing the Python version is not supported in internet-restricted Dash Enterprise instances.

A `project.toml` file to use Python 3.9 looks like this:

```
[build]
  [[build.env]]
    name = 'BP_CPYTHON_VERSION'
    value = '3.9.*'
```

CREATING A WEBSITE LAYOUT:

The basic layout of the application will be built using `Dash` in this task.

The layout is composed of a tree of "components" such as `html.Div` and `dcc.Graph`.The Dash HTML Components module (dash.html) has a component for every HTML tag.

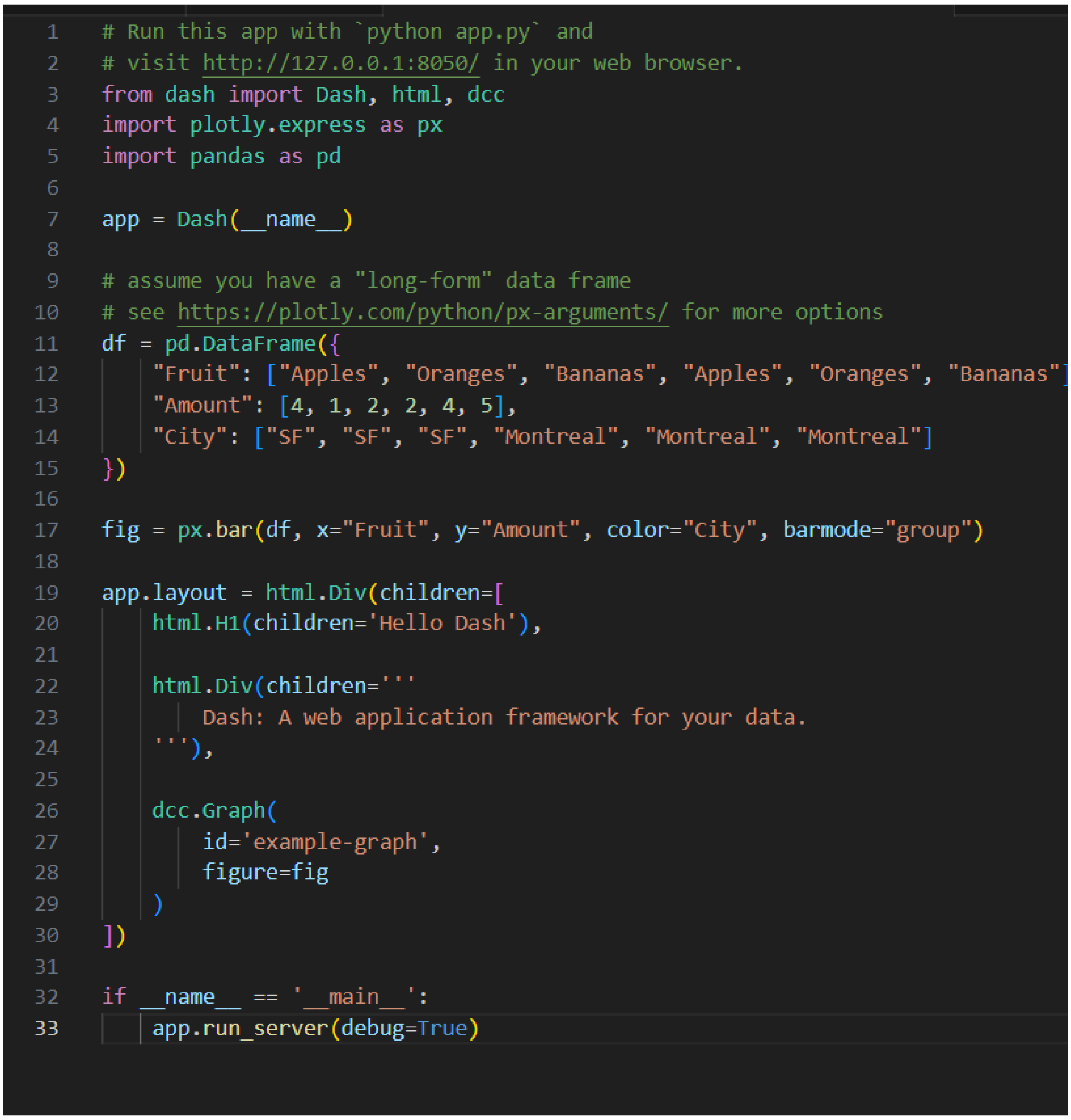
The `html.H1(children='Hello Dash')` component generates a `<h1>Hello Dash</h1> HTML` element in your app. Not all components are pure HTML.

The Dash Core Components module `(dash.dcc)` contains higher-level components that are interactive and are generated with JavaScript, HTML, and CSS through the React.js library.

Each component is described entirely through keyword attributes. Dash is *declarative*: you will primarily describe your app through these attributes.

create a file named `app.py` , copy the code below into it, and then run it with python `app.py`.

```
$ python app.py
...Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```



```python
# Run this app with `python app.py` and
# visit http://127.0.0.1:8050/ in your web browser.
from dash import Dash, html, dcc
import plotly.express as px
import pandas as pd

app = Dash(__name__)

# assume you have a "long-form" data frame
# see https://plotly.com/python/px-arguments/ for more options
df = pd.DataFrame({
    "Fruit": ["Apples", "Oranges", "Bananas", "Apples", "Oranges", "Bananas"
    "Amount": [4, 1, 2, 2, 4, 5],
    "City": ["SF", "SF", "SF", "Montreal", "Montreal", "Montreal"]
})

fig = px.bar(df, x="Fruit", y="Amount", color="City", barmode="group")

app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),

    html.Div(children='''
        Dash: A web application framework for your data.
    '''),

    dcc.Graph(
        id='example-graph',
        figure=fig
    )
])

if __name__ == '__main__':
    app.run_server(debug=True)
```

Dash is a web app framework that provides pure Python abstraction around HTML, CSS, and JavaScript.

Instead of writing HTML or using an HTML templating engine, you compose your layout using Python with the Dash HTML Components module (dash.html).

```python
from dash import html
```

Simple HTML Structure:

```
from dash import html

html.Div([
    html.H1('Hello Dash'),
    html.Div([
        html.P('Dash converts Python classes into HTML'),
        html.P("This conversion happens behind the scenes by Dash's JavaScript
front-end")
    ])
])
```

which gets converted (behind the scenes) into the following HTML in your web app:

```
<div>
    <h1>Hello Dash</h1>
    <div>
        <p>Dash converts Python classes into HTML</p>
        <p>This conversion happens behind the scenes by Dash's JavaScript
front-end</p>
    </div>
</div>
```

Import relevant libraries as shown below:

`import dash`

`import dash_core_components as dcc`

`import dash_html_components as html`

`from datetime import datetime as dt`

Create a Dash instance and store it in an `app` variable. Also, store the application's server property in a `server` variable as it will be frequently used.

`app = dash.Dash(__name__)`

`server = app.server`

Make the web layout using Dash HTML Components and Dash Core Components, then store it in the app's layout component i.e. the `app.layout`. Your code should look something like this :

`app.layout = html.Div([item1, item2])`

We need two divisions (.Div) for the entire layout.
The first one is for our inputs like stock code, date range selector, number of days of forecast and buttons. These components are the ones which the user will be interacting with. You can follow the code given below:

```
app.layout = html.Div(
    [
        html.Div(
            [
                # Navigation
                html.P("Welcome to the Stock Dash App!", className="start"),
                html.Div([
                    html.P("Input stock code: "),
                    html.Div([
                        dcc.Input(id="dropdown_tickers", type="text"),
                        html.Button("Submit", id='submit'),
                    ],
                             className="form")
                ],
                         className="input-place"),
                html.Div([
                    dcc.DatePickerRange(id='my-date-picker-range',
                                        min_date_allowed=dt(1995, 8, 5),
                                        max_date_allowed=dt.now(),
                                        initial_visible_month=dt.now(),
                                        end_date=dt.now().date()),
                ],
                         className="date"),
                html.Div([
                    html.Button(
                        "Stock Price", className="stock-btn", id="stock"),
                    html.Button("Indicators",
                                className="indicators-btn",
                                id="indicators"),
                    dcc.Input(id="n_days",
                              type="text",
                              placeholder="number of days"),
                    html.Button(
                        "Forecast", className="forecast-btn", id="forecast")
                ],
                         className="buttons"),
                # here
            ],
            className="nav"),
```

The second division will be for the data plots and company's basic information (name, logo, brief intro) only.

```
        # content
        html.Div(
            [
                html.Div(
                    [  # header
                        #html.Img(id="logo"),
                        html.P(id="ticker")
                    ],
                    className="header"),
                html.Div(id="description", className="decription_ticker"),
                html.Div([], id="graphs-content"),
                html.Div([], id="main-content"),
                html.Div([], id="forecast-content")
            ],
            className="content"),
    ],
    className="container")
```

Write the following at the end of the main file for running the app in development mode

```
if __name__ == '__main__':
    app.run_server(debug=True)
```

STYLING THE APPLICATIONS WEB PAGE

Using CSS, we will style our webpage to make it look more neat and user friendly

REQUIREMENTS

Give a class name to the parent division (the one which contains both our main divisions) you created in Task 1. Name it something appropriate like `container`.

Style the `container` division by giving the `display` property `flex` value. This will ensure that both the divisions are laid out in the same horizon.

```
.container {
  display: flex;
}
```

Similarly, style the first division (the one containing our inputs). Again, give the display property flex value and the flex-direction property column value. Give it a suitable width too, ideally less than 40%. You can keep the items aligned at the center.

```css
* {
  margin: 0;
  padding: 0;
  box-sizing: 0;
  font-family: "Roboto", sans-serif;
}
body {
  overflow-x: hidden;
}
.container {
  display: flex;
}
.nav {
  /*position: fixed;*/
  width: 25vw;
  /* height: 100%; */
  align-items: center;
  display: flex;
  justify-content: flex-start;
  flex-direction: column;
  background-color: rgb(5, 107, 107);
}
.content {
  width: 65vw;
  padding: 1rem 5rem;
}
.start {
  margin-top: 2rem;
  margin-bottom: 5rem;
  text-align: center;
  font-size: larger;
  color: rgb(166, 243, 248);
}
.input-place p {
  color: antiquewhite;
  font-size: larger;
}
.form {
  display: flex;
  margin-top: 1rem;
  margin-bottom: 2rem;
  height: 1.5rem;
}
.form button {
  width: 5rem;
  background-color: yellow;
  color: black;
  border: none;
}
.buttons {
  margin: 2rem;
  width: 100%;
  display: flex;
  flex-wrap: wrap;
  position: relative;
  justify-content: space-around;
}
#forecast {
  margin-top: 2rem;
  height: 2.2rem;
}
#n_days {
  height: 2rem;
  margin-top: 2rem;
}
.buttons button {
  width: 7.5rem;
  height: 2.5rem;
  border: none;
  background-color: rgb(166, 255, 0);
  color: rgb(0, 0, 0);
}
.header {
  display: flex;
  justify-content: flex-start;
  align-items: center;
  margin-bottom: 2rem;
}
.header p {
  font-size: 4rem;
  margin-left: 3rem;
  line-height: 80%;
}
#logo {
  max-width: 15rem;
  height: auto;
}
```

Machine Learning model.Py

Make the model in the `model.py` file.

Use the support vector regression (SVR) module from the `sklearn` library. Fetch the stock prices for the last 60 days. Split the dataset into 9:1 ratio for training and testing respectively.

Use the rbf kernel in `GridSearchCV` for tuning your hyperparameters.

Then, train the SVR model with the training dataset.

Test your model's performance by using metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) on the testing dataset.

After the model is built, make sure another callback function (as seen in Task 2) for the same is made in the main file i.e. `app.py` (where the model is to be imported).

1. **Alpha Vantage Stock API**. Before you start, however, you will first need an API key, which you can obtain for free here. After that, you can assign that key to the api_key variable. In this tutorial, we will retrieve 20 years of historical data for the American Airlines stock. As an optional reading, you may refer to this stock API starter guide for the best practices of working with historical market data.
2. Use the data from **this page**. You will need to copy the *Stocks* folder in the zip file to your project home folder.

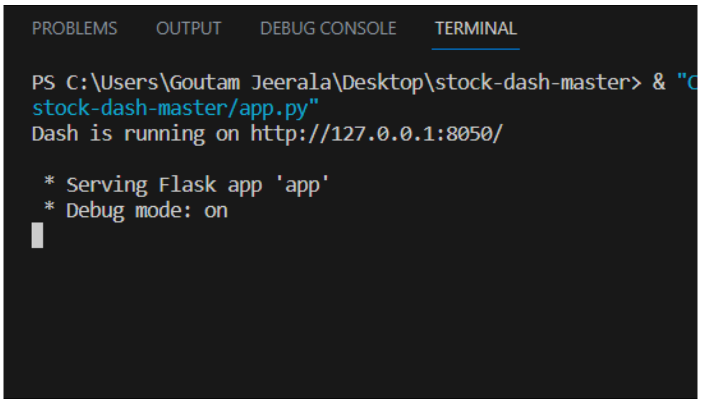
Stock prices come in several different flavours. They are,

- Open: Opening stock price of the day
- Close: Closing stock price of the day
- High: Highest stock price of the data
- Low: Lowest stock price of the day

CHAPTER - 4

RESULTS AND DISCUSSIONS

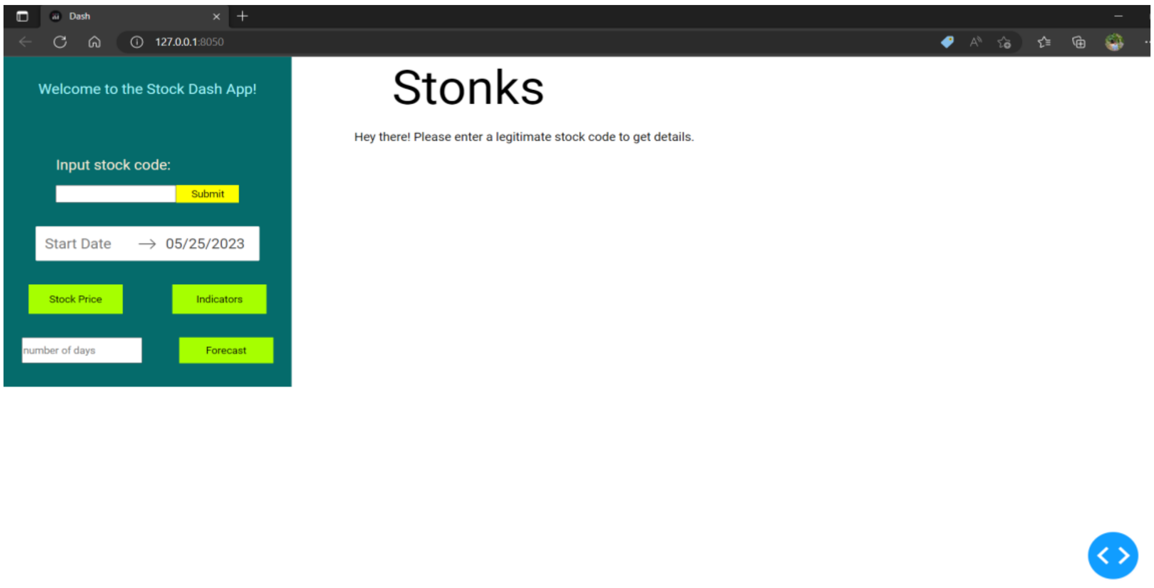
Finally, our web app is deployed and can be accessed by anyone in the world. When we run the app.py in the IDE the DASH runs and launches the website



```
PS C:\Users\Goutam Jeerala\Desktop\stock-dash-master> & "C
stock-dash-master/app.py"
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app 'app'
 * Debug mode: on
```

Figure 8 app debug

This is the webpage gets launched :



Stonks

Hey there! Please enter a legitimate stock code to get details.

Figure 9 Webpage

Steps:

Here I am using the TESLA stock name code for the reference.

- Enter the Stock Code **TSLA**

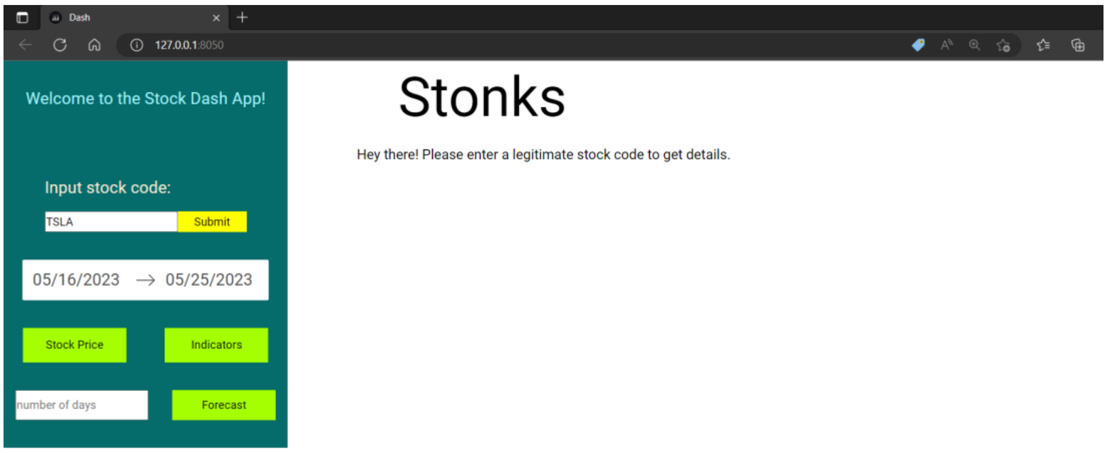


Figure 10 Webpage filled

- Select the Start date and End date and click on stock price and submit

- Now you will be able to see the opening price and closing price from the picked dates

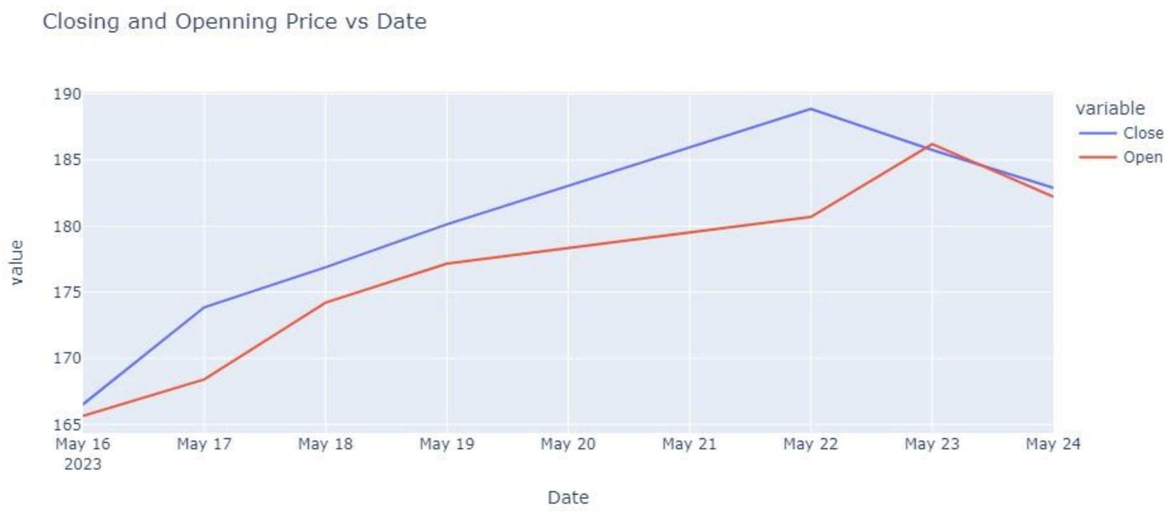


Figure 11 TSLA (i)

- Here you will be able to see the Average stock price with respective to dates we selected

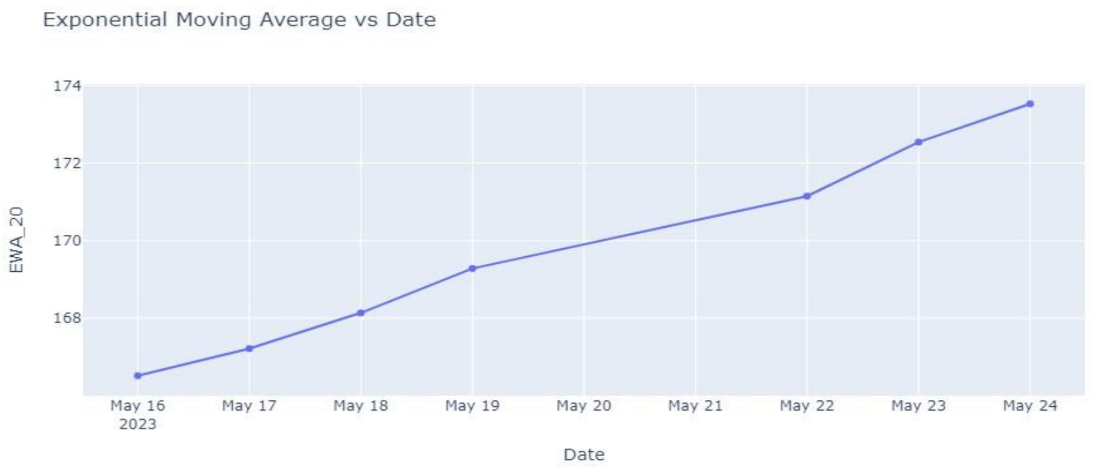


Figure 12 TSLA (ii)

- Here we are selecting the eight days, So the algorithm calculates the estimaetd stock price for the next eight days for our reference.

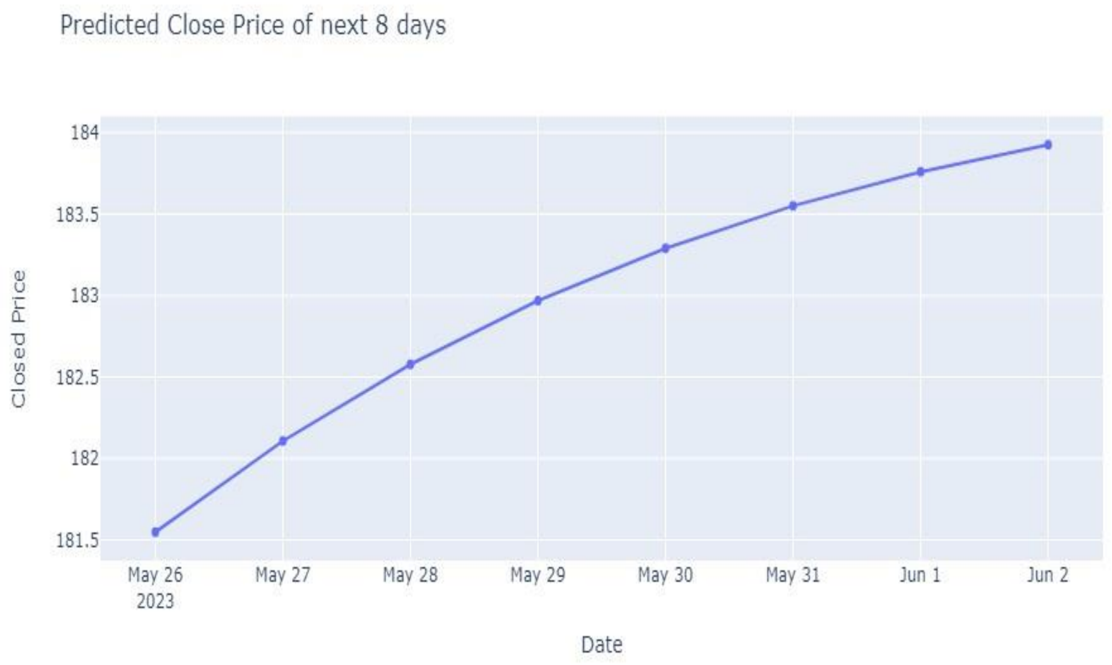
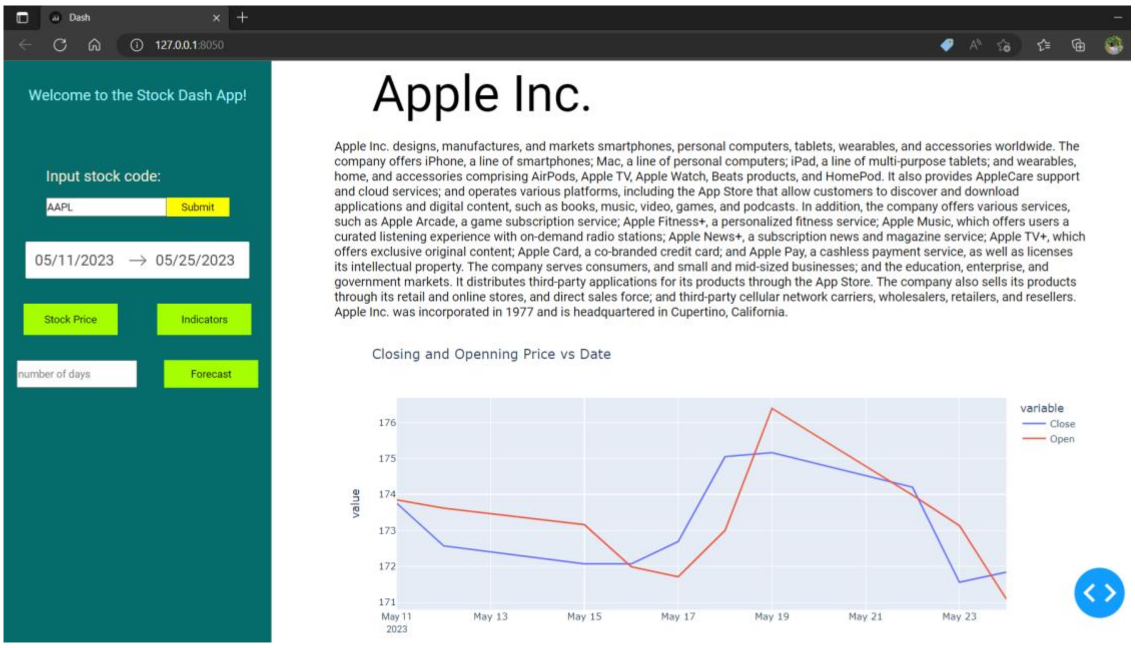


Figure 13 TSLA (iii)

- **Another Stock : APPLE (AAPL) STOCK**



Apple Inc.

Apple Inc. designs, manufactures, and markets smartphones, personal computers, tablets, wearables, and accessories worldwide. The company offers iPhone, a line of smartphones; Mac, a line of personal computers; iPad, a line of multi-purpose tablets; and wearables, home, and accessories comprising AirPods, Apple TV, Apple Watch, Beats products, and HomePod. It also provides AppleCare support and cloud services; and operates various platforms, including the App Store that allow customers to discover and download applications and digital content, such as books, music, video, games, and podcasts. In addition, the company offers various services, such as Apple Arcade, a game subscription service; Apple Fitness+, a personalized fitness service; Apple Music, which offers users a curated listening experience with on-demand radio stations; Apple News+, a subscription news and magazine service; Apple TV+, which offers exclusive original content; Apple Card, a co-branded credit card; and Apple Pay, a cashless payment service, as well as licenses its intellectual property. The company serves consumers, and small and mid-sized businesses; and the education, enterprise, and government markets. It distributes third-party applications for its products through the App Store. The company also sells its products through its retail and online stores, and direct sales force; and third-party cellular network carriers, wholesalers, retailers, and resellers. Apple Inc. was incorporated in 1977 and is headquartered in Cupertino, California.

Figure 14 AAPL (i)

Detailed source view,

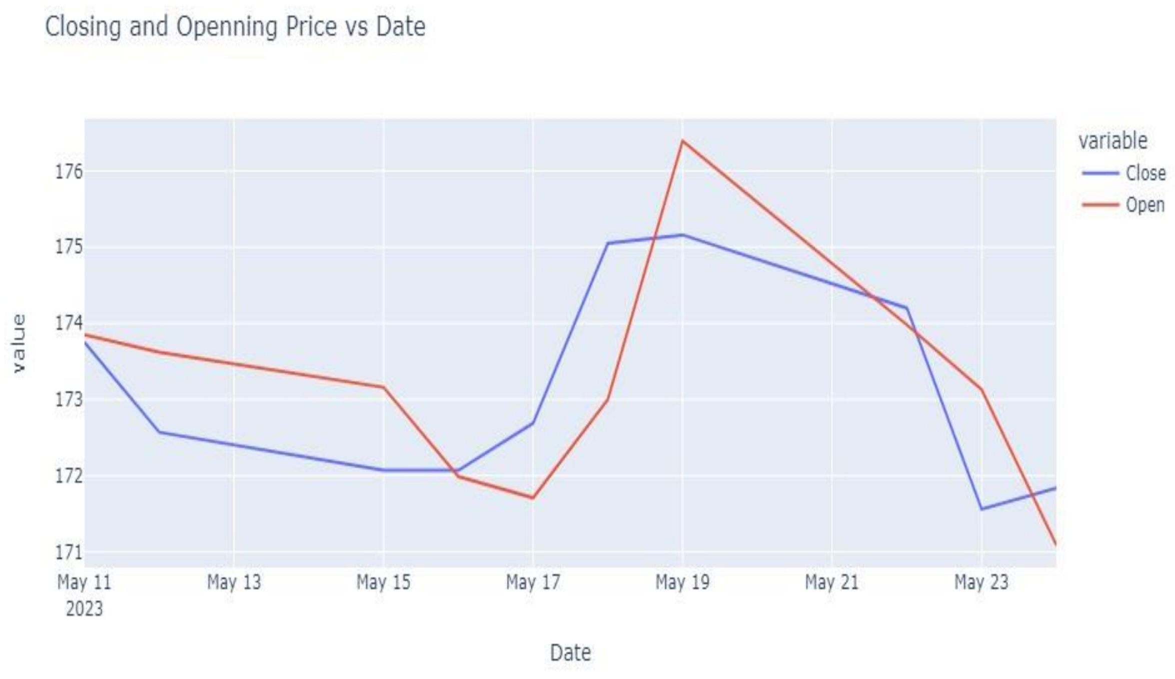


Figure 15 AAPL (ii)

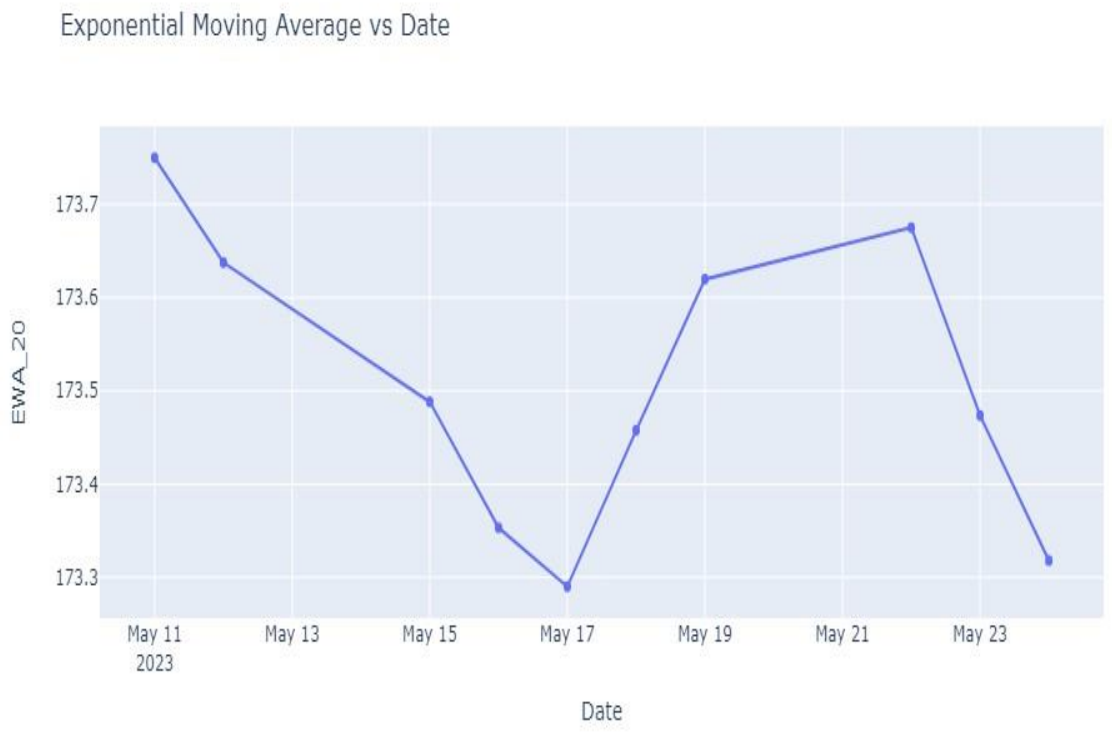


Figure 16 AAPL (iii)

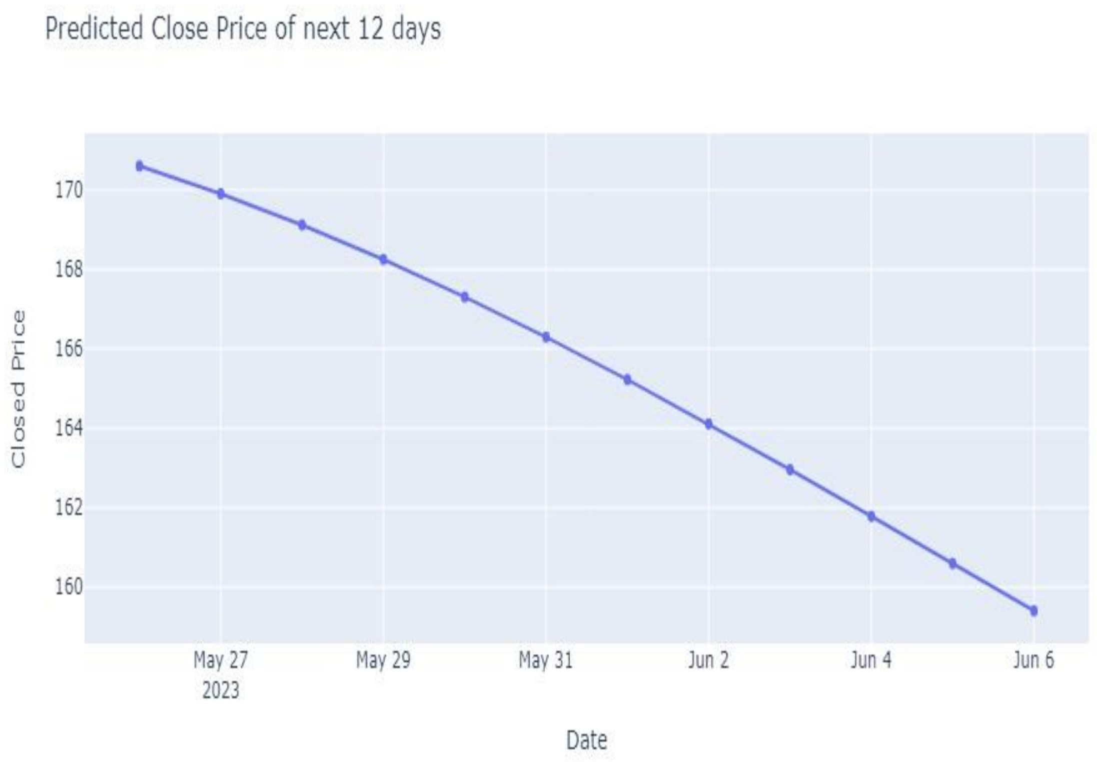
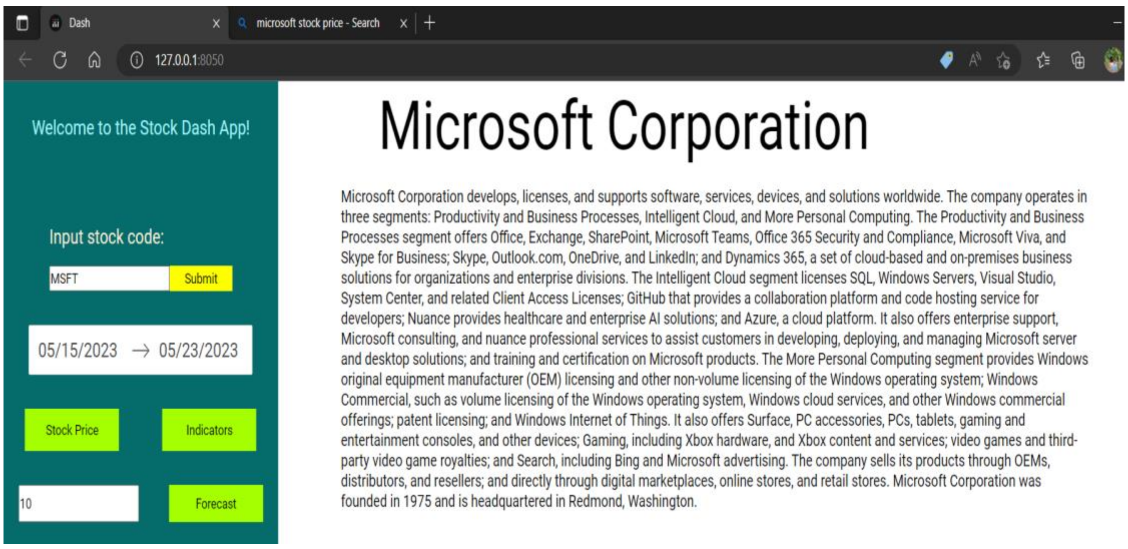


Figure 17 AAPL (iv)

- **Another Stock – Microsoft Stock (MSFT)**



Microsoft Corporation

Microsoft Corporation develops, licenses, and supports software, services, devices, and solutions worldwide. The company operates in three segments: Productivity and Business Processes, Intelligent Cloud, and More Personal Computing. The Productivity and Business Processes segment offers Office, Exchange, SharePoint, Microsoft Teams, Office 365 Security and Compliance, Microsoft Viva, and Skype for Business; Skype, Outlook.com, OneDrive, and LinkedIn; and Dynamics 365, a set of cloud-based and on-premises business solutions for organizations and enterprise divisions. The Intelligent Cloud segment licenses SQL, Windows Servers, Visual Studio, System Center, and related Client Access Licenses; GitHub that provides a collaboration platform and code hosting service for developers; Nuance provides healthcare and enterprise AI solutions; and Azure, a cloud platform. It also offers enterprise support, Microsoft consulting, and nuance professional services to assist customers in developing, deploying, and managing Microsoft server and desktop solutions; and training and certification on Microsoft products. The More Personal Computing segment provides Windows original equipment manufacturer (OEM) licensing and other non-volume licensing of the Windows operating system; Windows Commercial, such as volume licensing of the Windows operating system, Windows cloud services, and other Windows commercial offerings; patent licensing; and Windows Internet of Things. It also offers Surface, PC accessories, PCs, tablets, gaming and entertainment consoles, and other devices; Gaming, including Xbox hardware, and Xbox content and services; video games and third-party video game royalties; and Search, including Bing and Microsoft advertising. The company sells its products through OEMs, distributors, and resellers; and directly through digital marketplaces, online stores, and retail stores. Microsoft Corporation was founded in 1975 and is headquartered in Redmond, Washington.

Figure 18 MSFT (i)

Detailed source view,

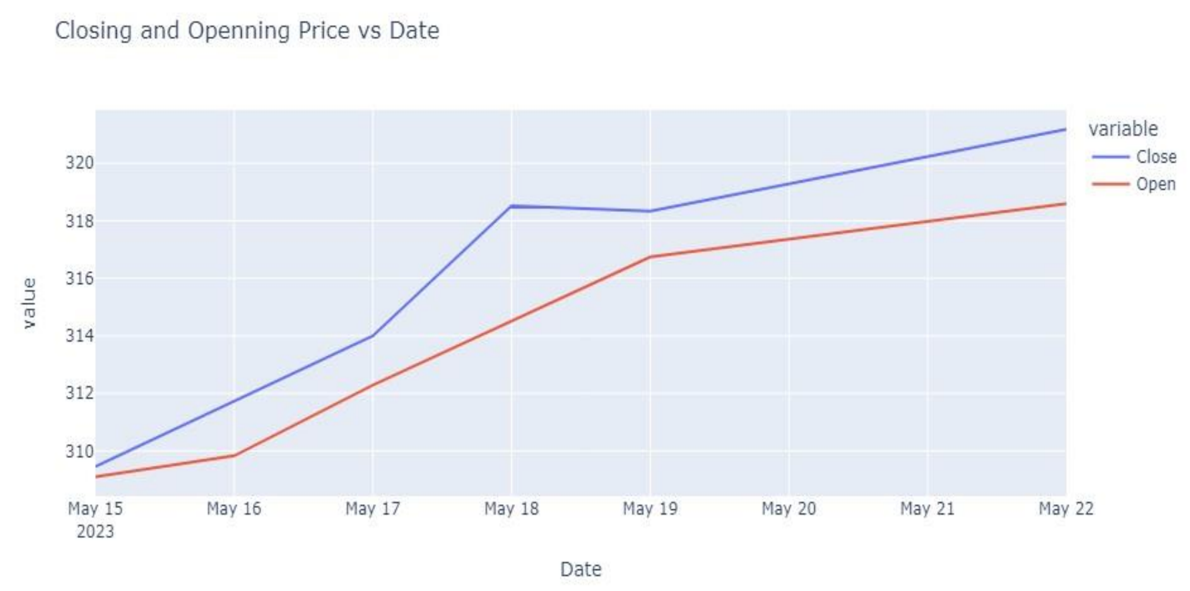


Figure 19 MSFT (ii)

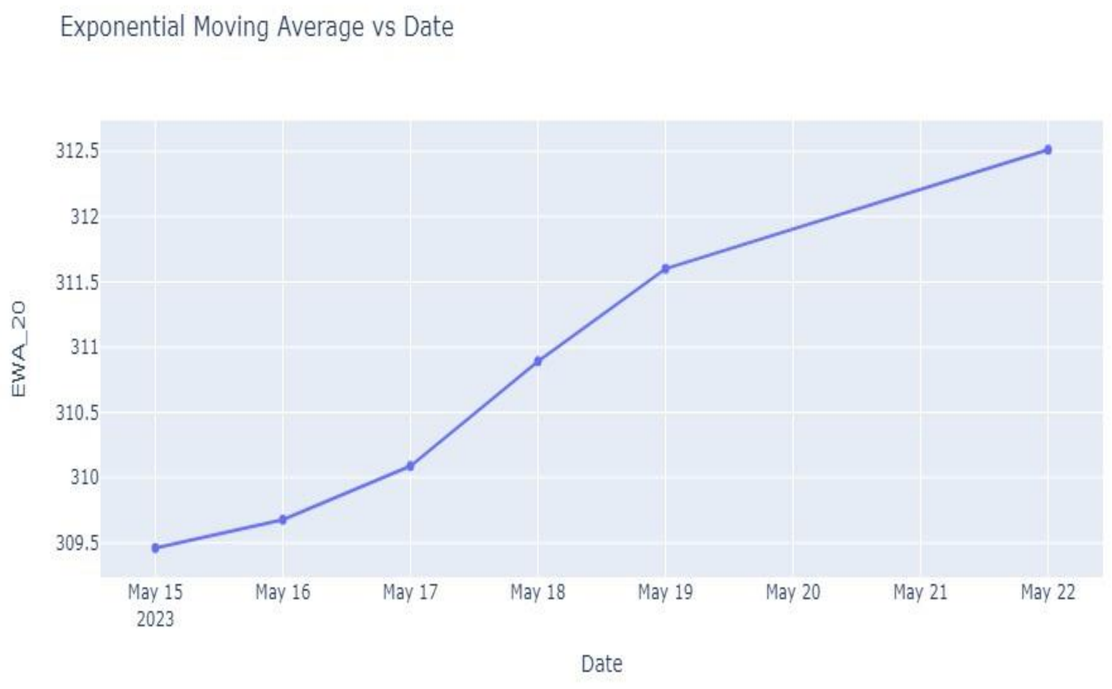


Figure 20 MSFT (iii)

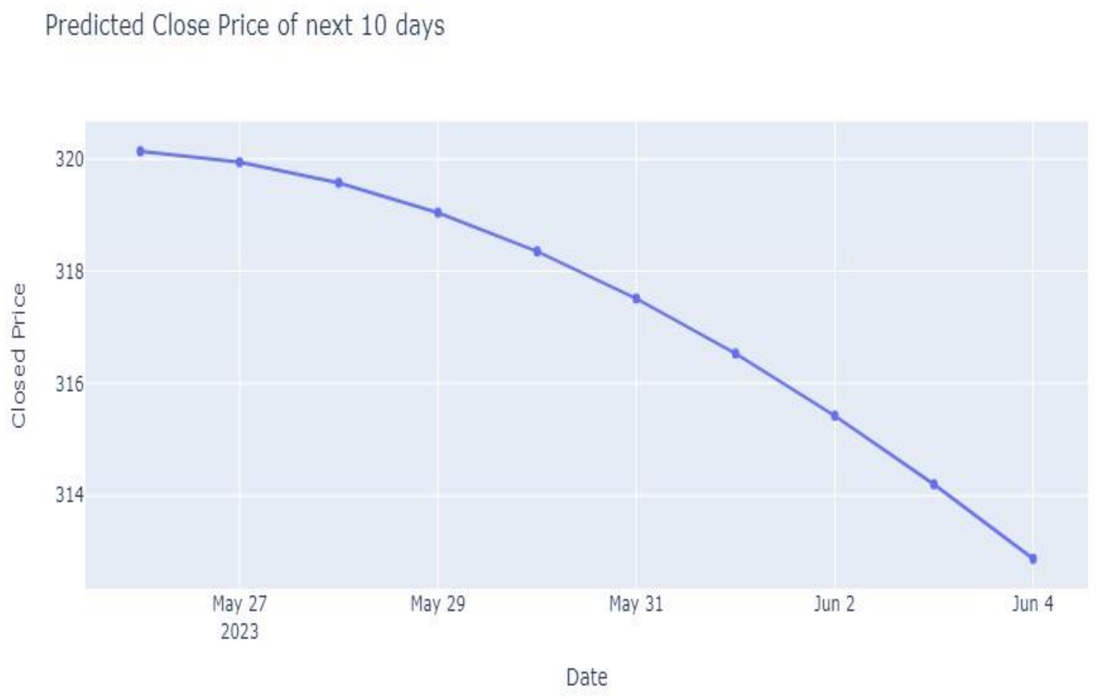


Figure 21 MSFT (iv)

CHAPTER 5

CONCLUSION AND FUTURE WORK

In conclusion, visualizing and forecasting stocks using Dash with the integration of the SVR ML model offers several advantages for analysing and predicting stock market trends. By leveraging the capabilities of Dash, users can interactively explore historical stock data, customize visualizations, and gain insights into past market behaviour. Integrating the SVR ML model allows for the generation of forecasts and predictions for future stock prices, providing valuable information for investment decisions.

The future scope of "Visualizing and Forecasting Stocks Using Dash Using SVR ML Model" is promising. Here are some potential areas of further development and improvement:

1. Advanced Visualization Techniques: Explore more advanced visualization techniques to present stock market data in a visually appealing and informative way. This could include

heatmaps, network graphs, or sentiment analysis visualizations to incorporate additional dimensions of analysis.

2. Integration of Multiple ML Models: Instead of relying solely on the SVR model, consider integrating multiple ML models for stock forecasting. Ensemble methods or combining different algorithms could potentially improve the accuracy and robustness of predictions.

3. Sentiment Analysis and News Integration: Incorporate sentiment analysis and news data to assess the impact of news events, social media sentiment, or macroeconomic indicators on stock prices. This could provide valuable insights into the relationship between news sentiment and market behaviour.

4. Real-Time Data Streaming: Enhance the application to handle real-time streaming data, enabling users to visualize and forecast stocks with up-to-the-minute information. This would require integrating real-time data sources and implementing efficient data processing and visualization techniques.

5. Interactive Portfolio Management: Extend the application to allow users to manage their stock portfolios, track performance, and simulate different investment strategies. This could include features such as portfolio optimization, risk analysis, and performance evaluation.

6. Integration with Trading Platforms: Integrate the Dash application with trading platforms or brokerage APIs, enabling users to execute trades directly from the application based on the visualized and forecasted information.

Overall, the combination of visualizing and forecasting stocks using Dash with the SVR ML model opens up exciting possibilities for improving stock market analysis, decision-making, and trading strategies.

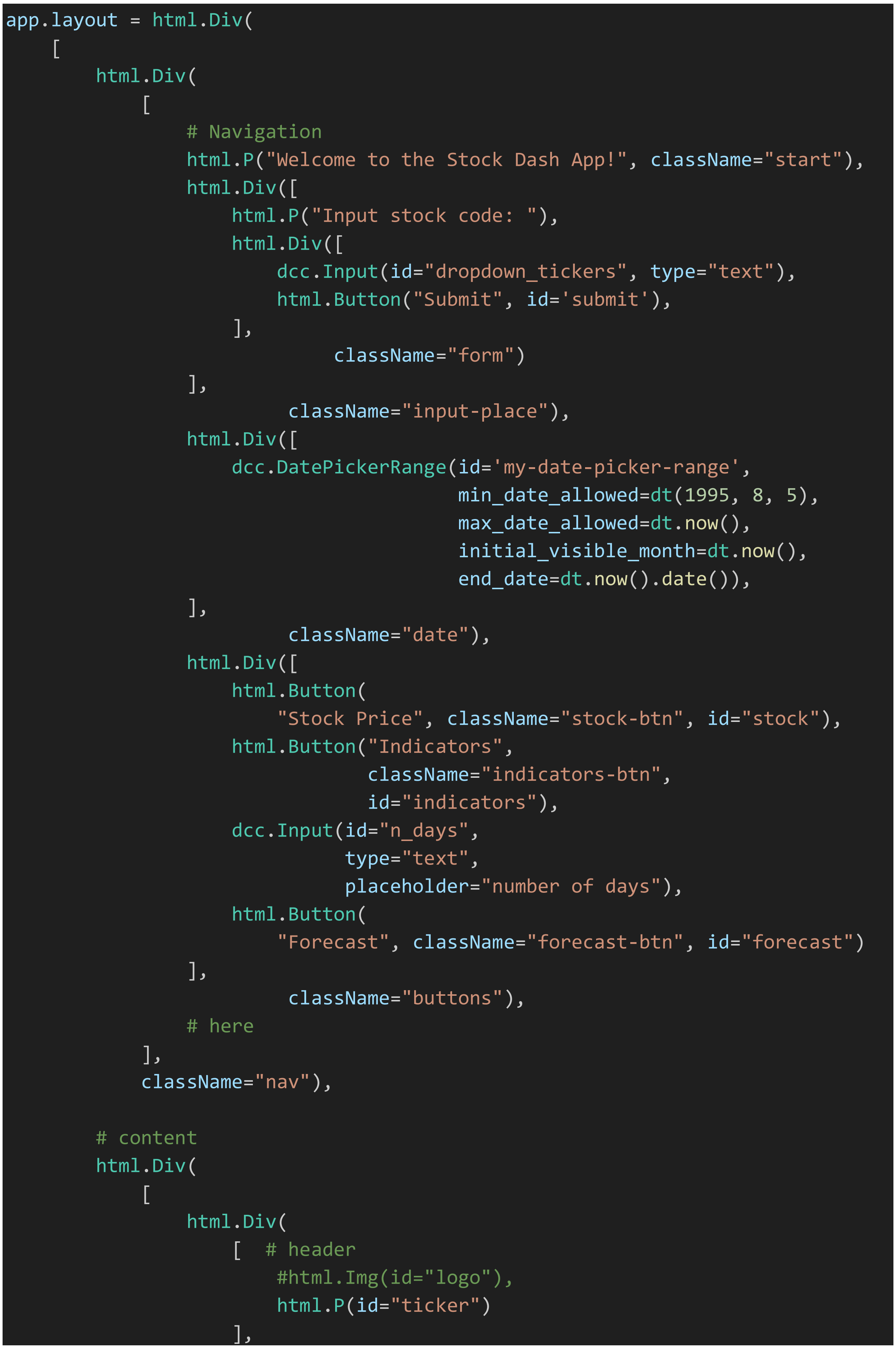
As technology advances and more data becomes available, the future scope of this domain will continue to expand, enabling users to make more informed and data-driven investment choices.

CHAPTER 6

APPENDIX

App.py Code

```
import dash
from dash import dcc
from dash import html
from datetime import datetime as dt
import yfinance as yf
from dash.dependencies import Input, Output, State
from dash.exceptions import PreventUpdate
import pandas as pd
import plotly.graph_objs as go
import plotly.express as px
# model
from model import prediction
from sklearn.svm import SVR


def get_stock_price_fig(df):

    fig = px.line(df,
                  x="Date",
                  y=["Close", "Open"],
                  title="Closing and Openning Price vs Date")

    return fig


def get_more(df):
    df['EWA_20'] = df['Close'].ewm(span=20, adjust=False).mean()
    fig = px.scatter(df,
                     x="Date",
                     y="EWA_20",
                     title="Exponential Moving Average vs Date")
    fig.update_traces(mode='lines+markers')
    return fig


app = dash.Dash(
    __name__,
    external_stylesheets=[
        "https://fonts.googleapis.com/css2?family=Roboto&display=swap"
    ])
server = app.server
# html layout of site
```



```
app.layout = html.Div(
    [
        html.Div(
            [
                # Navigation
                html.P("Welcome to the Stock Dash App!", className="start"),
                html.Div([
                    html.P("Input stock code: "),
                    html.Div([
                        dcc.Input(id="dropdown_tickers", type="text"),
                        html.Button("Submit", id='submit'),
                    ],
                             className="form")
                ],
                         className="input-place"),
                html.Div([
                    dcc.DatePickerRange(id='my-date-picker-range',
                                        min_date_allowed=dt(1995, 8, 5),
                                        max_date_allowed=dt.now(),
                                        initial_visible_month=dt.now(),
                                        end_date=dt.now().date()),
                ],
                         className="date"),
                html.Div([
                    html.Button(
                        "Stock Price", className="stock-btn", id="stock"),
                    html.Button("Indicators",
                                className="indicators-btn",
                                id="indicators"),
                    dcc.Input(id="n_days",
                              type="text",
                              placeholder="number of days"),
                    html.Button(
                        "Forecast", className="forecast-btn", id="forecast")
                ],
                         className="buttons"),
                # here
            ],
            className="nav"),

        # content
        html.Div(
            [
                html.Div(
                    [  # header
                        #html.Img(id="logo"),
                        html.P(id="ticker")
                    ],
                    className="header"),
                html.Div(id="description", className="decription_ticker"),
                html.Div([], id="graphs-content"),
                html.Div([], id="main-content"),
                html.Div([], id="forecast-content")
            ],
            className="content"),
    ],
    className="container")


# callback for company info
@app.callback([
    Output("description", "children"),
    #Output("logo", "src"),
    Output("ticker", "children"),
    Output("stock", "n_clicks"),
    Output("indicators", "n_clicks"),
    Output("forecast", "n_clicks")
], [Input("submit", "n_clicks")],
   [State("dropdown_tickers", "value")]
)
def update_data(n, val):  # inpur parameter(s)
    if n == None:
          return "Hey there! Please enter a legitimate stock code to get
details.",  "Stonks", None, None, None
        # raise PreventUpdate
    else:
        if val == None:
            raise PreventUpdate
        else:
            ticker = yf.Ticker(val)
            inf = ticker.info
            df = pd.DataFrame().from_dict(inf, orient="index").T
            df[[ 'shortName', 'longBusinessSummary']]
                                                                              return
df['longBusinessSummary'].values[0],   df['shortName'].values[0], None, None,
None


# callback for stocks graphs
@app.callback([
    Output("graphs-content", "children"),
], [
    Input("stock", "n_clicks"),
    Input('my-date-picker-range', 'start_date'),
    Input('my-date-picker-range', 'end_date')
], [State("dropdown_tickers", "value")])
def stock_price(n, start_date, end_date, val):
```

```python
    if n == None:
        return [""]
        #raise PreventUpdate
    if val == None:
        raise PreventUpdate
    else:
        if start_date != None:
            df = yf.download(val, str(start_date), str(end_date))
        else:
            df = yf.download(val)

    df.reset_index(inplace=True)
    fig = get_stock_price_fig(df)
    return [dcc.Graph(figure=fig)]


# callback for indicators
@app.callback([Output("main-content", "children")], [
    Input("indicators", "n_clicks"),
    Input('my-date-picker-range', 'start_date'),
    Input('my-date-picker-range', 'end_date')
], [State("dropdown_tickers", "value")])
def indicators(n, start_date, end_date, val):
    if n == None:
        return [""]
    if val == None:
        return [""]

    if start_date == None:
        df_more = yf.download(val)
    else:
        df_more = yf.download(val, str(start_date), str(end_date))

    df_more.reset_index(inplace=True)
    fig = get_more(df_more)
    return [dcc.Graph(figure=fig)]


# callback for forecast
@app.callback([Output("forecast-content", "children")],
              [Input("forecast", "n_clicks")],
              [State("n_days", "value"),
               State("dropdown_tickers", "value")])
def forecast(n, n_days, val):
    if n == None:
        return [""]
    if val == None:
        raise PreventUpdate
    fig = prediction(val, int(n_days) + 1)
    return [dcc.Graph(figure=fig)]


if __name__ == '__main__':
    app.run_server(debug=True)
```

Model.py Code

```
def prediction(stock, n_days):
    import dash
    import dash_core_components as dcc
    import dash_html_components as html
    from datetime import datetime as dt
    import yfinance as yf
    from dash.dependencies import Input, Output, State
    from dash.exceptions import PreventUpdate
    import pandas as pd
    import plotly.graph_objs as go
    import plotly.express as px



    # model
    from model import prediction
    from sklearn.model_selection import train_test_split
    from sklearn.model_selection import GridSearchCV
    import numpy as np
    from sklearn.svm import SVR
    from datetime import date, timedelta
    # load the data

    df = yf.download(stock, period='60d')
    df.reset_index(inplace=True)
    df['Day'] = df.index

    days = list()
    for i in range(len(df.Day)):
        days.append([i])

    # Splitting the dataset
```

```
X = days
Y = df[['Close']]

x_train, x_test, y_train, y_test = train_test_split(X,
                                                    Y,
                                                    test_size=0.1,
                                                    shuffle=False)


gsc = GridSearchCV(
    estimator=SVR(kernel='rbf'),
    param_grid={
        'C': [0.001, 0.01, 0.1, 1, 100, 1000],
        'epsilon': [
            0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10,
            50, 100, 150, 1000
        ],
        'gamma': [0.0001, 0.001, 0.005, 0.1, 1, 3, 5, 8, 40, 100, 1000]
    },

    cv=5,
    scoring='neg_mean_absolute_error',
    verbose=0,
    n_jobs=-1)

y_train = y_train.values.ravel()
y_train
grid_result = gsc.fit(x_train, y_train)

best_params = grid_result.best_params_
best_svr = SVR(kernel='rbf',
               C=best_params["C"],
               epsilon=best_params["epsilon"],
               gamma=best_params["gamma"],
               max_iter=-1)

# Support Vector Regression Model

# RBF model
#rbf_svr = SVR(kernel='rbf', C=1000.0, gamma=4.0)
rbf_svr = best_svr

rbf_svr.fit(x_train, y_train)

output_days = list()
for i in range(1, n_days):
    output_days.append([i + x_test[-1][0]])
```

```python
    dates = []
    current = date.today()
    for i in range(n_days):
        current += timedelta(days=1)
        dates.append(current)

    # plot Results
    # fig = go.Figure()
    # fig.add_trace(
    #     go.Scatter(x=np.array(x_test).flatten(),
    #                y=y_test.values.flatten(),
    #                mode='markers',
    #                name='data'))


    # fig.add_trace(
    #     go.Scatter(x=np.array(x_test).flatten(),
    #                y=rbf_svr.predict(x_test),
    #                mode='lines+markers',
    #                name='test'))
    fig = go.Figure()
    fig.add_trace(
        go.Scatter(
            x=dates,  # np.array(ten_days).flatten(),
            y=rbf_svr.predict(output_days),
            mode='lines+markers',
            name='data'))
    fig.update_layout(
        title="Predicted Close Price of next " + str(n_days - 1) + " days",
        xaxis_title="Date",
        yaxis_title="Closed Price",
        # legend_title="Legend Title",
    )

    return fig
```

CHAPTER 7
REFERENCES

[1]. Data Visualization and Stock Market and Prediction, Ashutosh Sharma, Sanket Modak, Eashwaran Sridhar, International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056, Volume: 06 Issue: 09 | Sep 2019

[2]. Visualizing and Forecasting stock using dash framework Prof. Mangesh Manake, Shital Pawar, Onkar Nakate, Viraj Bangar, Swapnil kale
[3]. Ashutosh Sharma, Sanket Modak, Eashwaran Sridhar "Data Visualization and Stock Market and Prediction" International Research Journal of Engineering and Technology (IRJET),(2019)

[4]. Ashish Sharma, Dinesh Buriya, Upendra Singh "Survey of Stock Market Prediction Using Machine Learning Approach" International Conference on Electronics, Communication and Aerospace Technology ICECA 2017

[5]. K. Hiba Sadia, Aditya Sharma, Adarrsh Paul, SarmisthaPadhi, Saurav Sanyal "Stock Market Prediction Using Machine Learning Algorithms" International Journal of Engineering and Advanced Technology (IJEAT),2020

[6]. V Kranthi Sai Reddy "Stock Market Prediction Using Machine Learning" International Research Journal of Engineering and Technology (IRJET) 2018.

[7]. Xi Zhang1 , Siyu Qu , Jieyun Huang , Binxing Fang , Philip Yu "Stock Market Prediction via Multi-Source Multiple Instance Learning" IEEE International Conference 2020

[8]. Minal Khan, Vijay Kumar Trivedi, Dr. Bhupesh Gour "A Review Paper on Stock Prediction Using Machine Learning Algorithms" The International journal of analytical and experimental modal analysis, Volume XII, Issue XII, December/2020, ISSN NO:0886-9367.

[9]. Sachin Sampat Patil, Prof. Kailash Patidar, Asst. Prof. Megha Jain "A Survey on Stock Market Prediction Using SVM" International Journal of Current Trends in Engineering & Technology Volume: 02, Issue: 01 (JAN- FAB 2019).

[10]. Ryo Akita, Akira Yoshihara, Takashi Matsubara, Kuniaki Uehara "Deep learning for stock Prediction Using Numerical and Textual Information" IEEE Xplore 2021.

[11]. Mustain Billah, Sajjad Waheed, Abu Hanifa "Stock Market Prediction Using an Improved Training Algorithm of Neural Network" 2nd International Conference on Electrical, Computer & Telecommunication Engineering (ICECTE) 8-10 December 2019

[12]. Tanapon Tantisripreecha, Nuanwan Soonthornphisaj "Stock Market Movement Prediction using LDA- Online Learning Model" IEEE Xplore 2020. 11. Aditya Menon, Shivali Singh, Hardik Parekh "A Review of Stock Market Prediction Using N