Chapter 1

1.1 Introduction

In the modern financial market, the most crucial problem is to find an essential approach to outline and visualize the predictions in stock-markets to be made by individuals to attain maximum profit by investments. The stock market is a transformative, non-straight dynamical and complex system. Long term investment is one of the major investment decisions. Though, evaluating shares and calculating elementary values for companies for long term investment is difficult. Stock price forecasting is a popular and important topic in financial and academic studies. Stock investments provide one of the highest returns in the market. Even though they are volatile in nature, one can visualize share prices and other statistical factors which help the keen investors to carefully decide on which company they want to spend their earnings on.

In this project we have created a single page web application using the Dash library (of Python), we have made dynamic plots of the financial data of a specific company by using the tabular data provided by yfinance python library. On top of it, we have used machine learning algorithm to predict the upcoming stock prices.

Exchanging the stocks on money markets is one of the significant speculation exercises. Already, scientists developed different stock examination systems that could empower them to envision the bearings of stock esteem development. Predicting and foreseeing of significant future cost, in perspective of the present cash related information and news, is of colossal use to the financial pros. financial masters need to know whether some stock will get higher or lower over a particular time-period. To obtain the accurate output, the approach used to implement is machine learning along with supervised learning algorithms. Results are tested using different types of supervised learning algorithms with a different set of features.

Stock investment decisions require time, knowledge and awareness including historical data, the stock market contains a huge amount of data that varies over time. Stock prices are influenced by various factors ranging from the performance of the company itself to the conditions of the economy in general. Thus, to manage investment portfolios, stock market data has to be analysed regularly to identify potential relationships between various stocks, hence, to adjust investment based on related stocks trends. A picture is worth a thousand words presenting data in visual form can assist humans in exploring deep insight of vast amounts of complex raw data, especially when

people have limited knowledge of the data. Visual representation is one of the most efficient ways to assist investors to have a clear overview of movements of the stock market, as well as providing a deeper understanding of each individual stock.

Predicting this stock value offers enormous profit opportunities which are a huge motivation for research in this area. Even a fraction of a second's knowledge of a stock's worth can result in large earnings. Similarly, in the repeated context, a probabilistically- correct prediction might be highly profitable. This attractiveness of finding a solution has prompted researchers, in both industry and academics to find a way past the problems like volatility, seasonality and dependence on time, economics and the rest of the market. However, the platform's prices and liquidity are highly unpredictable, which is where technology comes into aid.

1.2 Visualizing and forecasting stocks using Dash.

We will be creating a single-page web application using Dash (a python framework) and some machine learning models which will show company information (logo, registered name, and description) and stock plots based on the stock code given by the user. Also, the ML model will enable the user to get predicted stock prices for the dates inputted by the user. Visualizing and forecasting using Dash is a powerful and popular topic in the field of data science and web development. Dash is a Python framework that allows you to build interactive web applications for data visualization, analytics, and forecasting. It is built on top of Flask, Plotly, and React.js, making it an excellent choice for creating interactive and dynamic dashboards.

Dash provides a comprehensive set of tools and components that enable you to create interactive visualizations with ease. You can incorporate a wide range of charts, graphs, maps, tables, and other interactive elements into your dashboards. These visualizations can be customized and updated in real-time based on user input or data changes. One of the key strengths of Dash is its ability to handle real-time data updates and live streaming. It allows you to connect to various data sources such as databases, APIs, or real-time streams and update your visualizations automatically as new data becomes available. This feature is particularly useful for applications that require up-to-date insights and forecasting. Forecasting, another essential aspect of this topic, involves using historical data to predict future trends or outcomes.

Dash provides a robust platform for integrating forecasting algorithms and models into your web applications. You can leverage popular forecasting libraries in Python, such as Prophet, ARIMA, or LSTM networks, and visualize the forecasted results in real-time. The interactivity and flexibility offered by Dash enable users to interact with the data and explore different scenarios. Users can select specific time periods, regions, or variables of interest, and the visualizations dynamically update to reflect the changes. This allows for better data exploration, analysis, and decision-making. Furthermore, Dash provides various features for enhancing the user experience of your applications. You can incorporate user authentication and access controls to ensure secure access to sensitive data. Additionally, you can add interactive filters, dropdown menus, sliders, or input fields to allow users to customize the visualizations and forecasts based on their preferences.

1.2.1. DATA VISUALIZATION

- **What is Data visualization?**

  Data visualization refers to the techniques involved in graphically representing data, using visual elements like charts and graphs to spot trends, patterns, and outliers, for quick insights, and to help in real-time decision-making. It's increasingly important in today's world to understand the overwhelming volume of data being generated by businesses every single day.

- **Why is data visualization important?**

  Because of the way the human brain processes information, using charts or graphs to visualize large amounts of complex data is easier than poring over spreadsheets or reports. Data visualization is a quick, easy way to convey concepts in a universal manner – and you can experiment with different scenarios by making slight adjustments.

- Data visualization can also:
- Identify areas that need attention or improvement.
- Clarify which factors influence customer behaviour.
- Help you understand which products to place were.
- Predict sales volumes.

1.2.2. Necessity of Data Visualization

According to the World Economic Forum, the world produces 2.6 quintillion bytes of data every day, and 90% of all data has been created in the last two years. With so much data, it’s become increasingly difficult to manage and make sense of it all. It would be impossible for any single person to wade through data line-by-line and see distinct patterns and make observations. Data proliferation can be managed as part of the data science process, which includes data visualization. We need data visualization because a visual summary of information makes it

easier to identify patterns and trends than looking through thousands of rows on a spreadsheet. It's the way the human brain works. Since the purpose of data analysis is to gain insights, data is much more valuable when it is visualized. Even if a data analyst can pull insights from data without visualization, it will be more difficult to communicate the meaning without visualization. The charts and graphs make communicating data findings easier even if we identify the patterns without them. The numerous types of data visualizations for example Line charts, Box plots, Area charts, Scatter plots, Bar charts. Population pyramids, Pie charts, Heat maps, Bar charts (actual vs. expected), Tree maps, Histograms, Bubble charts, Choloropleth, Network diagrams etc.

1.2.3. Sub-fields of Data Visualization

Data visualization is the presentation of quantitative information in a graphical form. In other words, data visualizations turn large and small datasets into visuals that are easier for the human brain to understand and process. Data visualizations are surprisingly common in our everyday life, but they often appear in the form of well-known charts and graphs. In terms of business intelligence (BI), these visualizations help users make better data-based decisions. Data visualization transforms raw data into information. The data visualization has three significant sub-fields.

1.2.3.1. Scientific Visualization

Scientific visualization is the representation of data graphically to gain understanding and insight into the data. This allows the researcher to gain insight into the system that studied information in ways previously impossible. It is also referred to as visual data analysis. The purpose is to convey the scientific data accurately, reveal underlying structures in data and encourage the exploration of the data. Scientific visualization is an interdisciplinary research and application field in science, focusing on the visualization of three-dimensional phenomena, such as architecture, meteorology, medicine or biological systems. Its purpose is to graphically illustrate scientific data, enabling scientists to understand, explain, and collect patterns from the data.

1.2.3.2. Visual Analytics

Visual Analytics can be perceived as an integrated approach that combines visualization, human factors, and data analysis. Visual analytics is a new field that has

evolved with the development of scientific visualization and information visualization, with an emphasis on analytical reasoning through an interactive visual interface. Visual analytics methods allow decision makers to combine their human flexibility, creativity, and background knowledge with the enormous storage and processing capacities of today's computers to gain insight into complex problems. Visual analytics in the context of visualization relates to the areas of information visualization and computer graphics, and with respect to data analysis, it benefits largely from methodologies of information retrieval, data management & knowledge representation as well as data mining.

1.2.3.3. Information Visualization

Information visualization refers to the use of computer-supported, interactive visual representations of numerical and non-numerical abstract data sets in order to amplify human cognition. Information visualization, the art of representing data in a way that it is easy to understand and to manipulate, can help us make sense of information and thus make it useful in our lives. Information visualization is the communication of abstract data through interactive visual interfaces. Graphics such as histograms, trend graphs, flow charts, and tree diagrams all belong to information visualization and the design of these graphics transforms abstract concepts into visual information.

1.2.4. Goal of Data Visualization

The visual representation of data is more scientific than artistic in our modern world. The main goal of data visualization is effectively, efficiently, elegantly, accurately as well as meaningfully communicating information. It fulfils its objectives only if it encodes the given input in such a manner that our eyes can recognize and our brain can comprehend.

1.2.5. Data Visualization using Dash.

Dash ships with a Graph component that renders charts with plotly.js. Plotly.js is a great fit for Dash: it's declarative, open source, fast, and supports a complete range of scientific, financial, and business charts. Plotly.js is built on top of D3.js (for publication-quality, vectorized image export) and WebGL (for high performance visualization).

Dash's Graph element shares the same syntax as the open source plotly.py library, so you can easily switch between the two. Dash's Graph component hooks into the plotly.js event system, allowing Dash app authors to write applications that respond to hovering, clicking, or selecting points on a Plotly graph.

1.2.6. Data Visualization in the Stock Market.

Data visualization helps traders when making decisions quickly and enables them to easily synthesize large amounts of complex information.

- **How Data Visualization Can Improve Decision Making?**

- **Illustrating patterns:** Visuals are exceptional at presenting patterns. The human mind is wired to recognize patterns, but it's hard to make sense of those patterns when they appear as individual data points, such as in stock prices that fluctuate over a period of months. However, if those data points are plotted onto a linear graph, it's much easier to spot the highs and lows, and make a guess at how that pattern could unfold in the future.
- **Highlighting multiple variables:** Data visuals also allow you to highlight and control for multiple variables. Depending on your goals, this could help you factor multiple variables into your decision all at once, or help you drill down so you exclusively focus on one variable's effects on your investments. Either way, you can use visuals to make more informed decisions.
- **Reducing complex subjects:** One of the biggest advantages of data visualization is its ability to make complex subjects easier to understand, which is imperative in this age of big data. There are too many independent data points for any one person to track, but an automated platform can easily create a digestible version that appeals to your visual senses. Granted, not all financial topics are reducible to a simple conclusion, but visualization can make things more approachable, at the very least.
- **Adding customizing ability:** Most data visualization platforms have multiple controls that help a user customize their graphs and get exactly the visual they need to make a given decision. For example, you might be able to add or subtract specific variables, extend or contract the date range, or overlay competing subjects to get a clearer picture of what's going on.
- **Easing communication.** If you're investing with a partner, you should know data visuals also make it easier to communicate complex topics. Running a point-by-point analysis about each independent variable in your data set isn't going to go over well with a nontechnical audience; but a graph can easily highlight your most important takeaways.

1.3 Background

Data visualization and forecasting are crucial components of data-driven decision-making. As the volume and complexity of data continue to increase, organizations seek effective ways to interpret and communicate insights. Dash, a Python framework, emerged as a popular solution for creating interactive web applications that facilitate data visualization, analytics, and forecasting.

Dash was developed by Plotly, a company known for its interactive plotting library, and is built on Flask, a micro web framework, and React.js, a JavaScript library for building user interfaces. This combination of technologies provides a powerful toolset for creating interactive dashboards with rich visualizations and real-time updates.

Data visualization plays a vital role in conveying complex information in a clear and intuitive manner. Visualizations enable users to identify patterns, trends, and outliers within data, making it easier to understand and interpret. By presenting data in visually appealing charts, graphs, and maps, Dash empowers users to explore and analyse information interactively.

Forecasting, on the other hand, involves predicting future trends or outcomes based on historical data. It is an essential aspect of many industries, such as finance, sales, supply chain management, and weather forecasting. Accurate forecasts help organizations make informed decisions, optimize resource allocation, and plan.

Traditionally, forecasting has been performed using statistical models or machine learning algorithms. However, visualizing forecasted results and making them accessible to stakeholders can be challenging. Dash addresses this challenge by providing a platform to seamlessly integrate forecasting algorithms and models with interactive visualizations. This allows users to not only view forecasted data but also interact with it in real-time.

The flexibility and interactivity of Dash make it a powerful tool for data visualization and forecasting. It supports a wide range of chart types, including line plots, scatter plots, bar charts, heatmaps, and more. Users can easily customize the appearance of visualizations, add interactive elements like sliders and dropdown menus, and incorporate real-time data updates.

Furthermore, Dash simplifies the process of connecting to data sources, whether it's a database, API, or real-time stream. This makes it easier to access and analyse the most up-to-date information. By integrating real-time data updates into the dashboards, users can stay informed about the latest trends and make data-driven decisions promptly.

1.4 Statement of the Problem

- In today's data-driven world, organizations face the challenge of effectively visualizing and forecasting large volumes of complex data. While there are various tools and frameworks available, there is a need for a comprehensive solution that combines interactive data visualization, real-time updates, and forecasting capabilities. The problem at hand is the absence of a unified platform that seamlessly integrates these functionalities and provides a user-friendly interface for creating interactive dashboards.
- Traditional data visualization tools often lack the ability to handle real-time data updates and interactive elements. They may require manual data refreshing or lack the flexibility to incorporate forecasting algorithms directly into the visualizations. This results in static and outdated dashboards that fail to capture the dynamic nature of the data.
- Additionally, the process of forecasting often involves complex statistical or machine learning models. However, the challenge lies in presenting the forecasted results in an intuitive and interactive manner. Without a unified platform, users may struggle to visualize and explore the forecasted data, hindering their ability to make informed decisions.
- Another problem is the difficulty in connecting to different data sources and ensuring data integrity. Organizations often have data stored in various databases, APIs, or streaming platforms, making it challenging to access and integrate these disparate sources. Lack of seamless integration leads to data silos and inefficiencies in updating and presenting real-time information.
- Moreover, existing solutions may require extensive coding or lack a user-friendly interface, limiting the accessibility to non-technical users. This restricts the potential user base and slows down the decision-making process as users have to rely on data analysts or developers to create and update visualizations.
- Furthermore, security and access control are crucial concerns when dealing with sensitive data. Organizations need to ensure that only authorized users can access specific data or functionalities within the dashboards. The absence of robust authentication and access control mechanisms exposes organizations to potential data breaches and compromises data confidentiality.

- Organizations require a solution that can be easily deployed and scaled to handle growing data volumes and user demands. Scalability becomes a concern when dealing with large datasets or when multiple users simultaneously access the dashboards. The lack of a scalable solution hampers performance, leading to slow loading times and limited user concurrency.

    i. Inadequate data visualization: Traditional static charts and graphs fail to effectively convey complex data, making it challenging for users to gain insights and make informed decisions.
    ii. Limited interactivity: Existing visualization tools often lack interactive features, preventing users from exploring data from different angles or customizing visualizations to suit their needs.
    iii. Lack of real-time updates: Many visualization tools do not support real-time data updates, hindering the ability to monitor and analyse dynamic data streams or make timely decisions based on the latest information.
    iv. Separation of forecasting and visualization: Forecasting models and algorithms are typically disconnected from the visualization process, making it difficult to seamlessly integrate and communicate forecasted results to stakeholders.
    v. Complexity of forecasting implementation: Implementing forecasting algorithms and models requires significant effort and expertise, especially when it comes to visualizing the forecasted data effectively.
    vi. Limited accessibility: Dashboards and visualizations are often limited to technical users, making it challenging for non-technical stakeholders to interact with and understand the data.
    vii. Data security concerns: Maintaining data security and privacy is a critical concern, and existing visualization tools may not provide adequate safeguards for sensitive data.
    viii. Lack of customization options: Users may have specific requirements for visualizations, such as adding filters, adjusting parameters, or incorporating additional data sources, which may not be easily achievable with existing tools.
    ix. Inefficient data exploration: The absence of interactive features hinders users' ability to explore and analyse data in real-time, limiting the depth of insights that can be derived.

x. Inability to handle large datasets: Some visualization tools struggle with handling large datasets, leading to performance issues and limiting the scalability of visualizations.

xi. Complex integration with other technologies: Integrating visualization tools with existing technologies or data infrastructure can be cumbersome and time-consuming, impeding the adoption of data-driven decision-making.

xii. Limited forecasting capabilities: Existing visualization tools may lack built-in forecasting functionalities, requiring users to employ separate tools or scripts to generate forecasts.

xiii. Difficulty in communicating insights: Traditional static visualizations may not effectively communicate insights to stakeholders, leading to misunderstandings or misinterpretations of the data.

xiv. Lack of collaboration features: Collaborative features, such as shared dashboards or annotation capabilities, are often absent in existing visualization tools, hindering effective teamwork and knowledge sharing.

xv. Insufficient support for domain-specific visualizations: Certain industries or domains may require specialized visualization types that are not readily available in existing tools, limiting their applicability.

xvi. Absence of storytelling capabilities: Storytelling through data is a powerful way to convey insights and engage stakeholders, but many visualization tools lack features to create compelling narratives.

xvii. Limited support for predictive analytics: Existing visualization tools may not adequately support predictive analytics, making it challenging to integrate forecasting and visualize predicted outcomes.

xviii. Lack of integration with machine learning: Integrating machine learning models with visualization tools can be complex, preventing users from leveraging advanced predictive capabilities.

xix. Inability to handle real-time streaming data: Real-time streaming data sources, such as social media feeds or IoT sensor data, may not be easily visualized in existing tools, limiting their usefulness in dynamic environments.

xx. Difficulty in tracking and analysing historical trends: Existing visualization tools may not provide effective mechanisms for tracking and analysing historical trends, impeding the ability to identify patterns and make accurate forecasts.

xxi. Limited support for geospatial visualizations: Visualization tools may not offer robust geospatial capabilities, hindering the ability to analyse and visualize data in a geographical context.

xxii. Inefficient utilization of computational resources: Some visualization tools may not efficiently utilize computational resources, leading to slow performance or unresponsive dashboards.

xxiii. Lack of support for multi-dimensional data: Existing tools may struggle to handle multi-dimensional datasets.

The problem lies in the absence of a comprehensive platform that seamlessly integrates interactive data visualization, real-time updates, forecasting capabilities, and user-friendly interface. The challenges include the limitations of traditional visualization tools, difficulties in incorporating forecasting algorithms, connecting to multiple data sources, lack of accessibility for non-technical users, security concerns, and scalability issues. Addressing these challenges is crucial to empower organizations with a unified solution that enables them to visualize, analyse, and forecast their data effectively, leading to better decision-making and business outcomes.

1.5 Tools Used

i. Python

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. It is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc), it has a simple syntax similar to the English language. syntax allows developers to write programs with fewer lines than some other programming languages. It runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick. Python can be treated in a procedural way, an object-oriented way or a functional way.

ii. Details of Packages used.

1. Dash: Dash is an open-source framework which helps to create interactive, responsive and dynamic websites by using the Python or R programming languages. We used Dash core components to create tools to add inputs, sliders, drop down, graph, and other components to the web application to allow users to interact with the data. Dash's Hypertext Markup Language (HTML) components allows for composing the application layout using Python structures, rather than writing in HTML or using an HTML templating engine. Dash applications are made up of 2 building blocks : Layout and
Callbacks.

    a. Layout describes the look and feel of the app, it defines the elements such as graphs, drop downs etc and the placement, size, colour etc of these elements. Dash contains Dash HTML components using which we can create and style HTML content such as headings, paragraph, images etc using python. Elements such as graphs, drop downs, sliders are created using Dash Core components.
    b. Callbacks are used to bring interactivity to the dash applications. These are the functions using which, for example, we can define the activity that would happen on clicking a button or a drop down.

2. Plotly: Plotly is an open-source library that provides a list of chart types as well as tools with callbacks to make a dashboard. Plotly Python graphing library allows for creating interactive and dynamic graphs. Dash is used to build the interface which is populated by the graphs that are created from and using Plotly.

3. Pandas: Pandas is a data analytics library in Python and commonly used for processing tabular data. Here we used Pandas to read the main data file, pre-process the unclean data, extract the necessary information and plot it using Plotly on Dash.

4. Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier.

5. NumPy library is used for multi-dimensional array operations.

6. yfinance is a library that allows us to fetch financial data of a company (since its listing in the stock market) from its stock code directly.

7. gunicorn and lxml libraries will be used for the application's deployment i.e. to host the app on a target server.

8. sklearn and scikit-learn are tools used in the development of Machine Learning (ML) models.
9. dash_core_components: It is used for making beautiful charts, other visual components such as selectors, drop-downs, dates, and other visualizations.

10. dash_html_components: It includes various HTML tags for designing the dashboards. Dash HTML Components are similar to HTML syntaxes, so try relating the two syntax together.

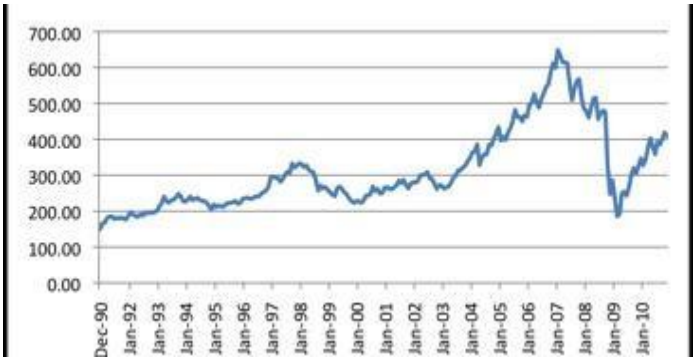
11. dash-bootstrap-components is a library of Bootstrap components for Plotly Dash, that makes it easier to build consistently styled apps with complex, responsive layouts.

iii Visual Studio Code

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs, such as Visual Studio IDE. It’s available on platforms like Windows, macOS and Linux. It comes

with built-in support for JavaScript, TypeScript and Node. VS Code uses experiments to try out new features or progressively roll them out. Our experimentation framework calls out to a Microsoft-owned service and is therefore disabled when telemetry is disabled. Code editing in Visual Studio Code Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

iv. Time-series graph



A time series chart, also called a times series graph or time series plot, is a data visualization tool that illustrates data points at successive intervals of time. Each point on the chart corresponds to both a time and a quantity that is being measured.

Generally, the horizontal axis of the chart or graph is used to plot increments of time and the vertical axis pinpoints values of the variable that is being measured. When the values are connected in chronological order by a straight line that creates a series of peaks and valleys, a time series chart may also be referred to as a fever chart.

A time series chart can be thought of as a series of snapshots that have been taken at regular intervals. The apples-to-apples comparison provided by this type of chart makes it an ideal tool for executive dashboards that help end users quickly identify a trend, spot an outlier in a cyclical pattern or analyse how a key metric is changing over time.

Chapter 2 - System Analysis & Requirements Specifications

2.1 System Requirements

Hardware Requirements

- Processor: Minimum 1 GHz; Recommended 2GHz or more.
- Ethernet connection (LAN) OR a wireless adapter (Wi-Fi).
- Hard Drive: Minimum 32 GB; Recommended 100 GB.
- Memory (RAM): Minimum 1 GB; Recommended 4 GB or above.

Software Requirements

- Operating System: Windows/ Mac/ Linux
- Python 3.5 and above in Visual Studio code
- Operating System: windows 7 and above or Linux based OS or MAC OS

2.2 Functional requirements

Functional requirements describe what the software should do (the functions). Think about the core operations.

Because the "functions" are established before development, functional requirements should be written in the future tense. In developing the software for Stock Price Prediction, some of the functional requirements could include:

- The software shall accept the tw_spydata_raw.csv dataset as input.
- The software should do pre-processing (like verifying for missing data values) on input for model training.
- The software shall use SVM ARCHITECTURE as main component of the software.
- It processes the given input data by producing the most possible outcomes of a CLOSING STOCK PRICE.

Notice that each requirement is directly related to what we expect the software to do. They represent some of the core functions.

2.3 Non-Functional requirements

Product properties

- Usability: It defines the user interface of the software in terms of simplicity of understanding the user interface of stock prediction software, for any kind of stock trader and other stakeholders in stock market.
- Efficiency: maintaining the possible highest accuracy in the closing stock prices in shortest time with available data.

Performance: It is a quality attribute of the stock prediction software that describes the responsiveness to various user interactions with it.

2.4 SYSTEM ARCHITECTURE

1) Preprocessing of data

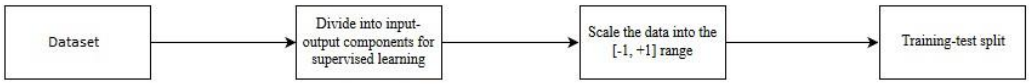


Fig.1: Pre-processing of data

2) Overall Architecture

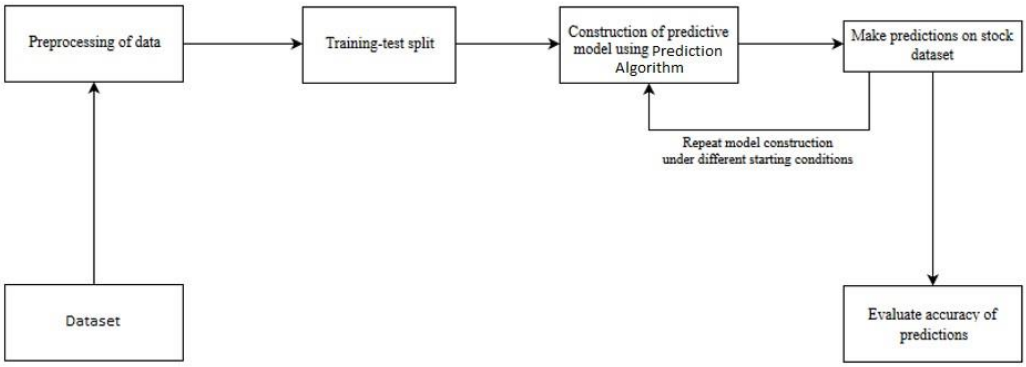


Fig.2: Overall Architecture

2.5. Methodology

The prediction methods can be roughly divided into two categories, statistical methods, and artificial intelligence methods. The methodology involves data fetching, visualization, technical analysis, and forecasting techniques to create an interactive and informative dashboard for visualizing and forecasting stock prices using the Dash framework and relevant libraries.

- Importing Dependencies: The required libraries and modules are imported to enable various functionalities, including Dash for creating the web application, yfinance for fetching stock data, datetime for handling dates, pandas for data manipulation, plotly for interactive visualizations, ta for calculating technical indicators, and the model module for stock price forecasting.
- Data Visualization Functions: Several functions are defined to create different types of visualizations for stock data. These functions use Plotly to generate candlestick charts, moving average charts, and relative strength index (RSI) charts based on the provided stock data.
- Dash Application Setup: The Dash application is initialized, and the server object is created. The HTML layout of the application is defined using the Dash HTML and CSS

components. The layout includes navigation elements, input fields for stock selection and date range, buttons to trigger different functionalities, and placeholders to display the visualizations.

- Callbacks: Callback functions are defined to handle user interactions and update the content of the Dash application dynamically. These callbacks include updating the company information based on the selected stock ticker, displaying stock price graphs based on the selected date range, generating moving average and RSI graphs based on the selected date range, and forecasting future stock prices based on the selected stock ticker and number of days.
- Company Information: The update data callback fetches information about the selected stock ticker using the yfinance library. The company logo, name, and other details are displayed in the application's header section.
- Stock Price Graph: The stock_price callback fetches the historical stock data for the selected ticker within the specified date range. It then generates a candlestick chart using the get_stock_price_fig function and displays it in the application.
- Indicator Graphs: The indicators, indicators, and indicators callbacks fetch the stock data and generate moving average charts and RSI charts using the get more functions, respectively. These charts represent different technical indicators and provide insights into the stock's price trends and momentum.
- Forecasting: The forecast callback utilizes a prediction function from the external "model" module. It takes the selected ticker and the number of days for forecasting as input and generates a forecast graph using the returned predictions. The graph is displayed in the application's forecast section.
- Application Execution: Finally, the Dash application is executed by running the app.run_server () function with debug mode enabled.

The main objective of this project is to develop a web-based application using a machine learning model and predicting the price of a given stock. The challenge of this project is to accurately predict the future closing price of a given stock across a given period in the future.

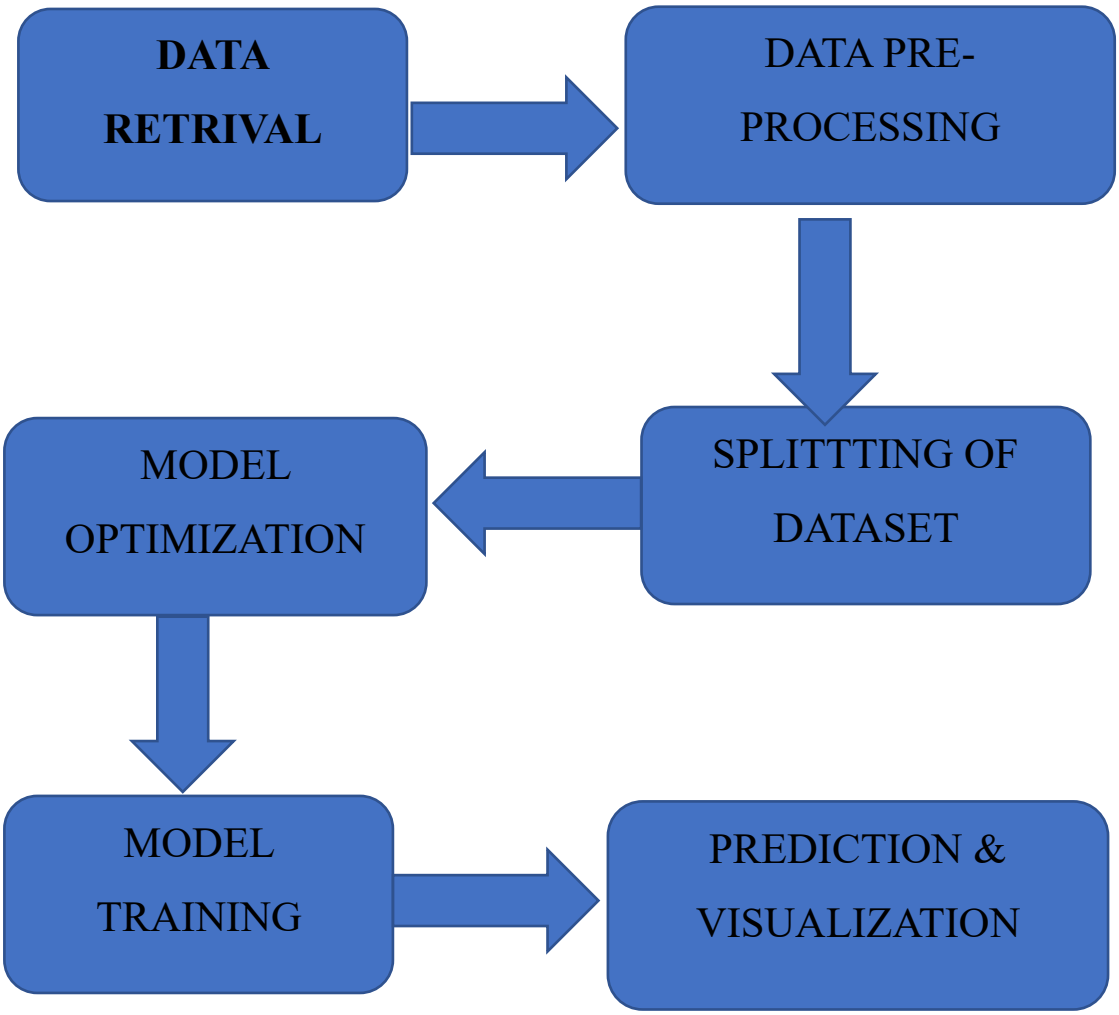


Fig.3. Methodology used for prediction.

Chapter 3 - System Design

3.1 Structure Chart

A structure chart (SC) in software engineering and organizational theory is a chart which shows the breakdown of a system to its lowest manageable levels. They are used in structured programming to arrange program modules into a tree. Each module is represented by a box, which contains the module's name.

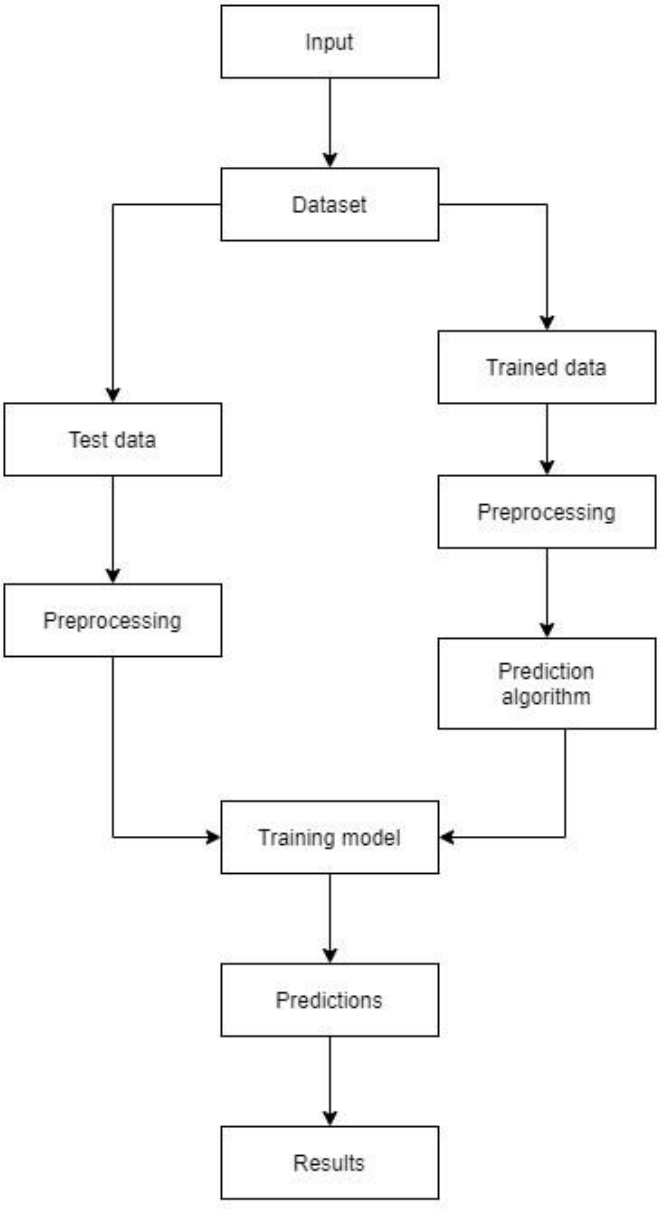


Fig.4: Training and prediction

3.2 UML Diagrams

A UML diagram is a partial graphical representation (view) of a model of a system under design, implementation, or already in existence. UML diagram contains graphical elements (symbols) - UML nodes connected with edges (also known as paths or flows) - that represent elements in the UML model of the designed system. The UML model of the system might also contain other documentation such as use cases written as templated texts.

The kind of the diagram is defined by the primary graphical symbols shown on the diagram. For example, a diagram where the primary symbols in the contents area are classes is class diagram. A diagram which shows use cases and actors is use case diagram. A sequence diagram shows sequence of message exchanges between lifelines.

UML specification does not preclude mixing of different kinds of diagrams, e.g., to combine structural and behavioural elements to show a state machine nested inside a use case. Consequently, the boundaries between the various kinds of diagrams are not strictly enforced. At the same time, some UML Tools do restrict set of available graphical elements which could be used when working on specific type of diagram.

UML specification defines two major kinds of UML diagram: structure diagrams and behaviour diagrams.

Structure diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts.

Behaviour diagrams show the dynamic behaviour of the objects in a system, which can be described as a series of changes to the system over time.

3.2.1 Use Case Diagram

In the Unified Modelling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems.
- Goals that your system or application helps those entities (known as actors) achieve.
- The scope of your system.

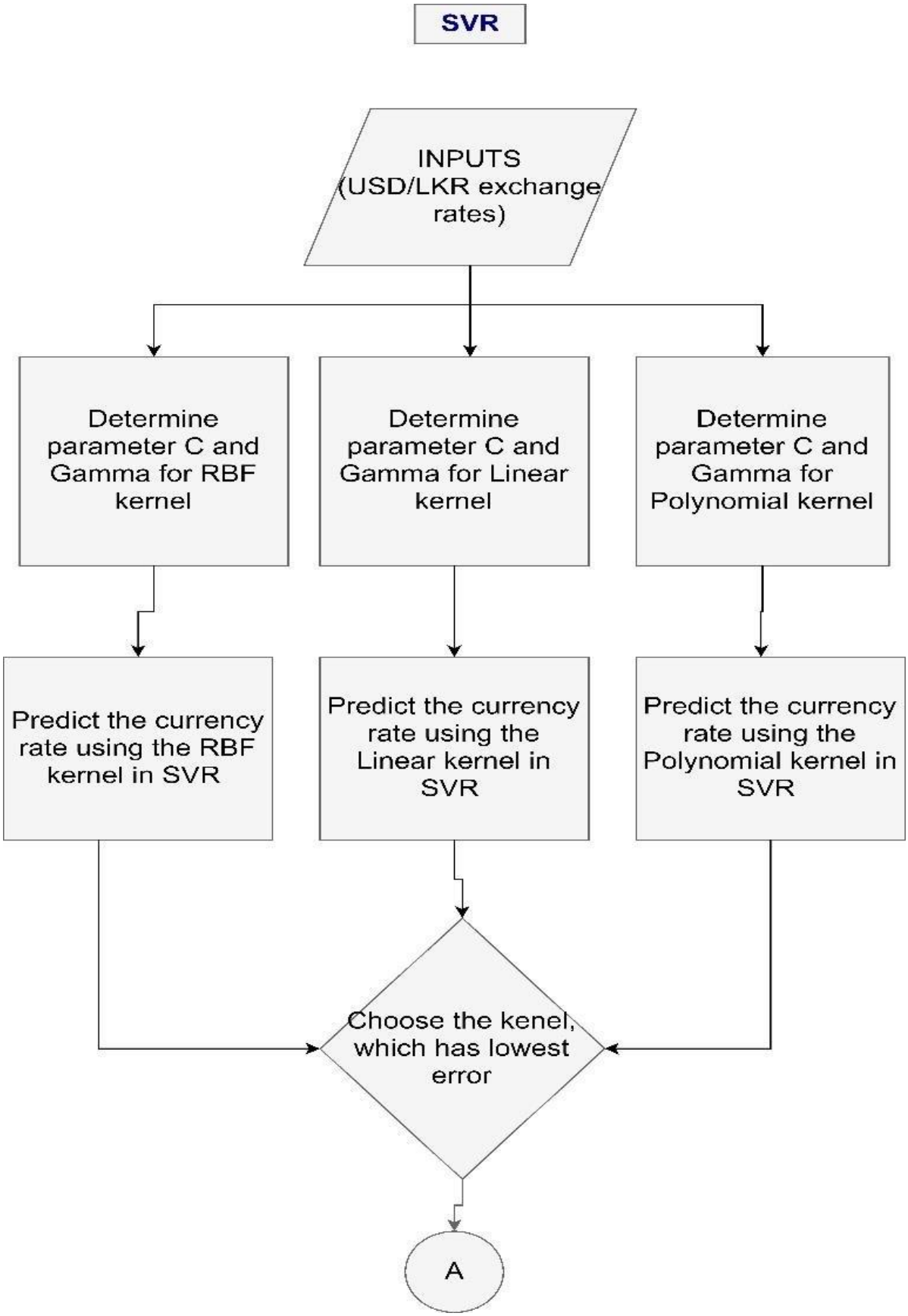


Fig.5. Use Case Diagram

3.2.2 Sequence Diagram

A sequence diagram is a type of interaction diagram because it describes how and in what order a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

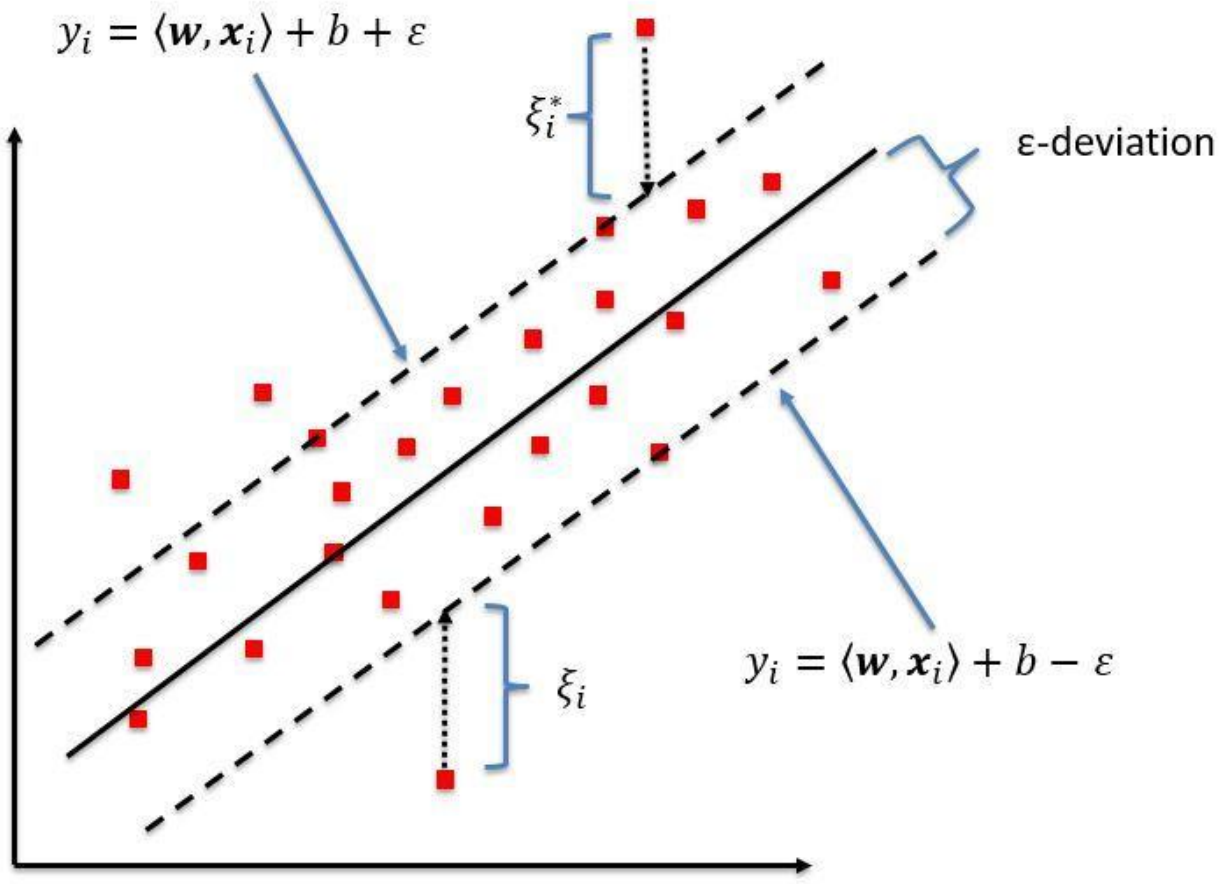


Fig.6: Execution based on model selection

3.2.3 Activity Diagram

An activity diagram is a behavioral diagram i.e. it depicts the behavior of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.

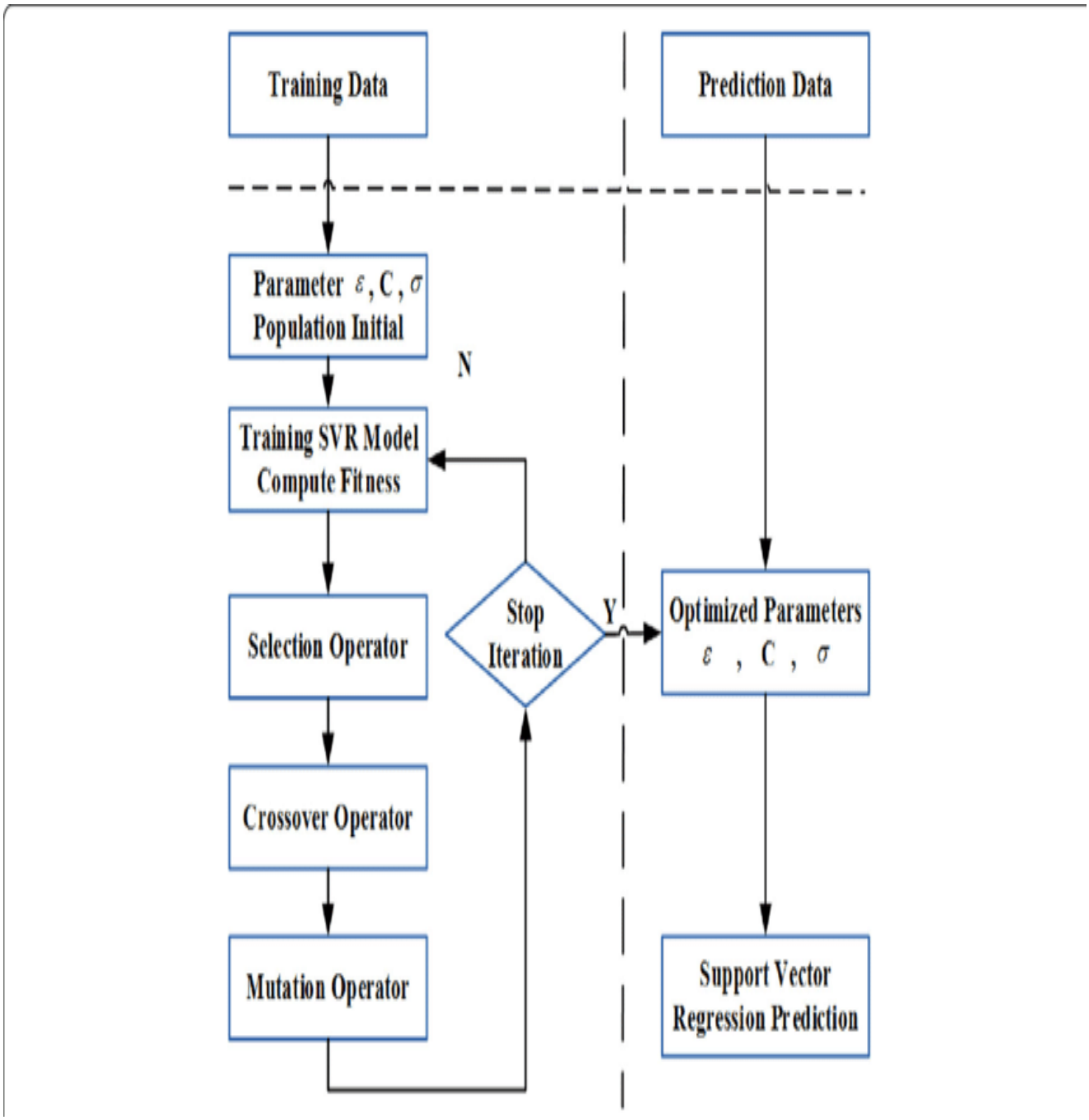


Fig.7: Execution based on algorithm selection.

3.2.4 Collaboration Diagram

Collaboration diagrams are used to show how objects interact to perform the behaviour of a particular use case, or a part of a use case. Along with sequence diagrams, collaboration is used by designers to define and clarify the roles of the objects that perform a particular flow of events of a use case. They are the primary source of information used to determining class responsibilities and interfaces.

The collaborations are used when it is essential to depict the relationship between the objects. Both the sequence and collaboration diagrams represent the same information, but the way of portraying it quite different. The collaboration diagrams are best suited for analyzing use cases.

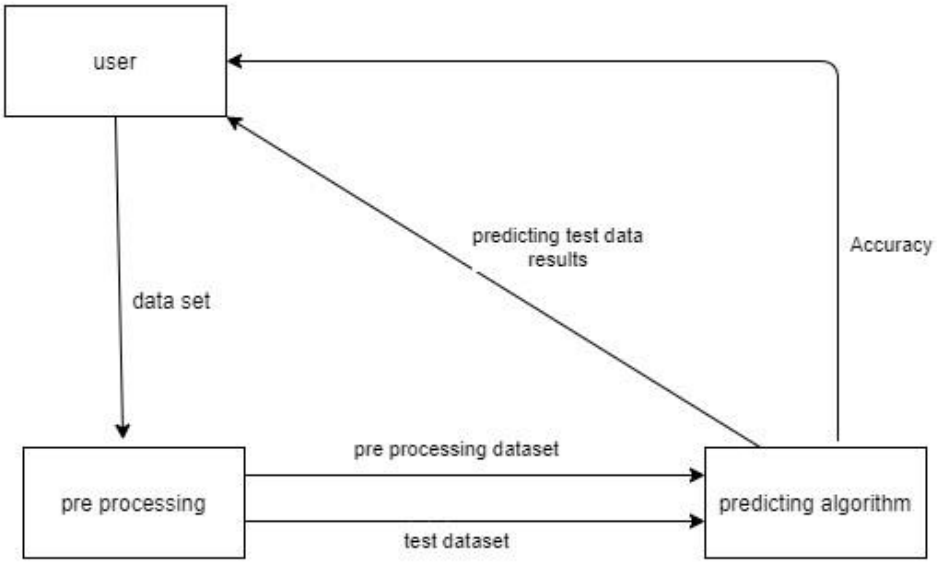


Fig.8: Data transfer between modules

3.2.5 Flow Chart

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows.

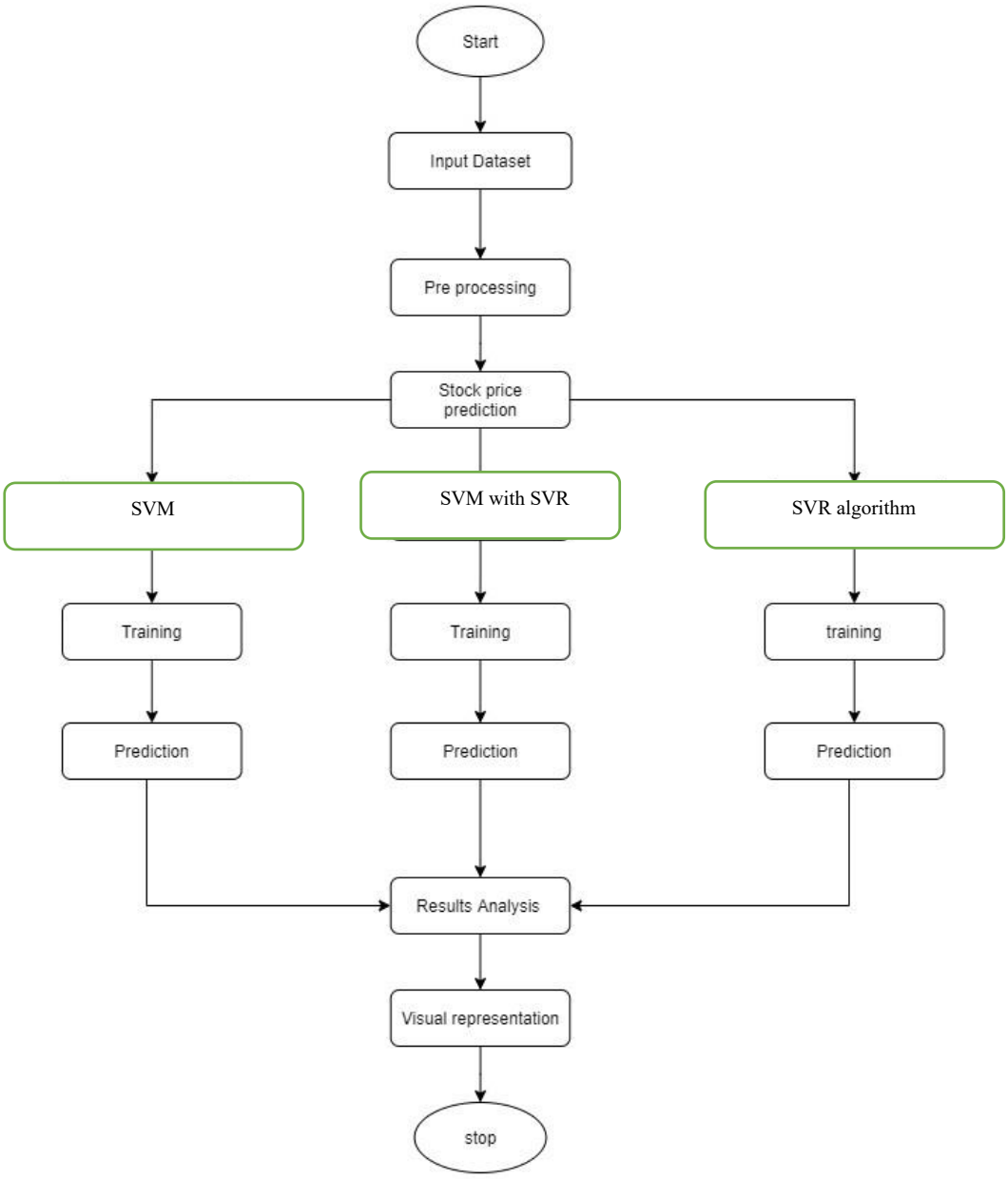


Fig.9: Flow of execution

3.2.6 Component Diagram

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system, but it describes the components used to make those functionalities.

Component diagrams are used in modelling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.

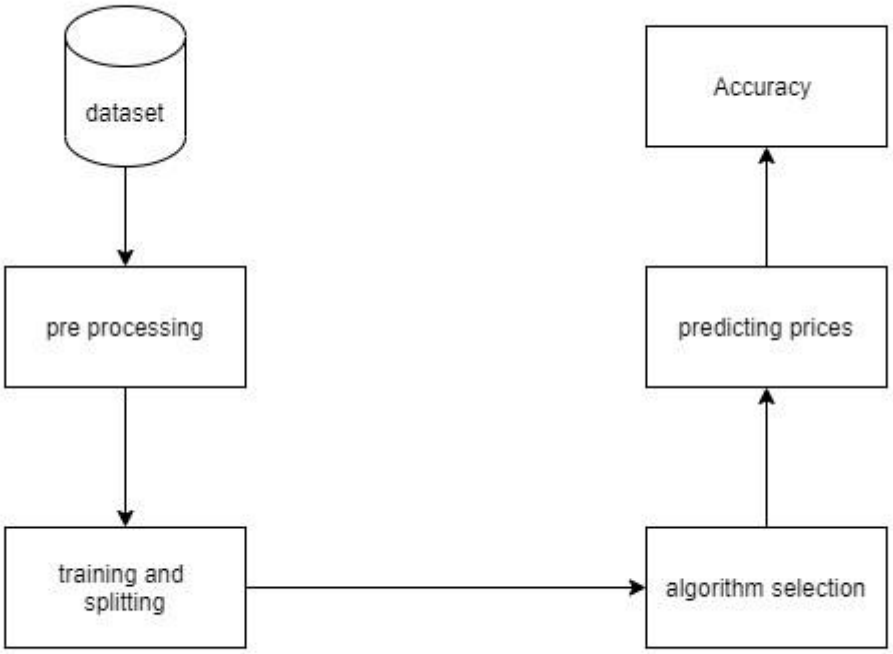


Fig.10: Components present in the system.

Chapter 4- Implementation

To implement the code for stock visualization and forecasting using Dash, follow the steps below:

1. Install Required Libraries:

- Open your command-line interface or terminal.
- Use the command `pip install dash yfinance plotly ta scikit-learn` to install the necessary libraries.

2. Create a New Python File:

- Open your preferred code editor.
- Create a new Python file and save it with an appropriate name, such as "stock_visualization.py".

3. Import Dependencies:

- In the Python file, import the required libraries and modules for the code implementation. This includes `dash`, `dash_core_components`, `dash_html_components`, `datetime`, `yfinance`, and others. Ensure that all necessary dependencies are imported.

4. Define Supporting Functions:

- Define the supporting functions required for generating different types of visualizations and conducting forecasting. These functions include `get_stock_price_fig()`, `get_more()`, `get_moree()`, and `get_moreee()`. Each function is responsible for creating a specific type of visualization, such as candlestick charts, moving average charts, and RSI charts.

5. Set Up the Dash Application:

- Initialize the Dash application by creating an instance of `dash.Dash()`. Assign it to a variable, such as `app`.
- Create the `server` variable and assign it the value of `app.server`. This is required for deployment purposes.
- Define the HTML layout of the Dash application using `dash_html_components`. Structure the layout components as desired, including navigation elements, input fields, buttons, and graph placeholders. Apply appropriate CSS classes and styling using the `className` parameter.

6. Define Callback Functions:

- Create callback functions using the `@app.callback` decorator to handle user interactions and update the application dynamically based on the input provided.

- Each callback function should have an appropriate set of input parameters, such as button clicks, date picker values, and dropdown selections. They should also have output parameters that define the components to be updated in the application based on the user interactions.

- Inside each callback function, write the necessary logic to retrieve data, generate visualizations, and update the application accordingly. Utilize the supporting functions defined earlier to create the required visualizations.

7. Run the Application:

- Add a `if __name__ == '__main__':` block at the end of the code.

- Inside this block, use the `app.run_server(debug=True)` statement to start the Dash application server in debug mode.

- Save the Python file.

8. Execute the Application:

- Open your command-line interface or terminal.

- Navigate to the directory where the Python file is saved.

- Run the Python script by executing the command `python stock_visualization.py`.

- The application will start, and the command-line interface will display the local server address **(e.g., http://127.0.0.1:8050/).**

9. Interact with the Application:

- Open a web browser and enter the server address provided by the command-line interface.

- The Dash application will load in the browser, and you can interact with the various components, such as input fields, buttons, and date pickers.

- Enter a valid stock ticker symbol in the input field and click the "Submit" button to retrieve company information.

- Select a date range using the date picker to specify the desired time period for visualizing the stock data.

- Click the different buttons to display the corresponding visualizations, such as stock price charts, moving average charts, and RSI charts.

- Enter the number of days for forecasting in the input field and click the "Forecast" button to generate a forecast graph.

10. Explore the Application:

- Explore the different visualizations and forecast results generated by the application.
- Analyze the stock price patterns, moving average trends, and RSI values to gain insights into the stock's performance.
- Examine the forecast graph to understand the projected future prices of the stock based on the selected forecasting method.

11. Terminate the Application:

- To stop the application, go back to the command-line interface or terminal.
- Press Ctrl+C to terminate the application's server.

By following these steps, you can successfully implement and run the code for stock visualization and forecasting using Dash. The Dash application provides an interactive and dynamic environment to explore and analyze stock data, enabling you to gain insights into price trends and make informed decisions. Remember to customize the code as per your requirements and add any additional features or functionality as needed.

PROJECT STAGES

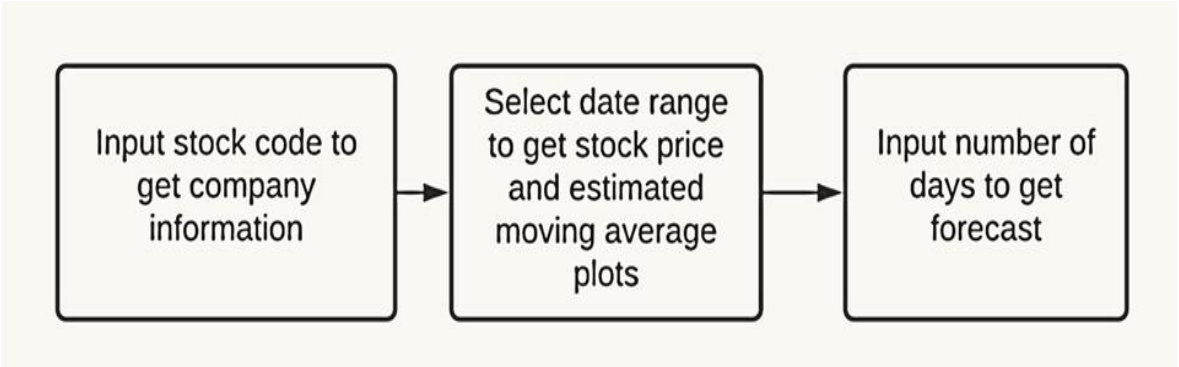


Fig.11: Project Stages

APPROACH USED

- Install all the required python libraries and framework

- Create a basic website layout. The basic layout of the application will be built using Dash. Create the main website's structure using mainly Dash HTML Components and Dash Core Components.

- Enhance the site's UI by styling using CSS. Using CSS we will style our web page to make it look more neat and user friendly.

- Generate plots of data(company's information) using the plotly library of Python. By fetching the data using yfinance python library.We are going to use the yfinance python library to get company information (name, logo and description) and stock price history. Dash's callback functions will be used to trigger updates based on change in inputs.

- Implement a machine learning model(Support Vector Regression) to predict the stock price for the dates requested by the user. Using the support vector regression (SVR) module from the sklearn library. Fetch the stock prices for the last 45 days. Split the data-set into a 9:1 ratio for training and testing respectively. Use the rbf kernel in GridSearchCV for tuning your hyper parameters. 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.

- Use the following command in the terminal of the working directory to run your Dash app's server locally - $python filename.py

Chapter 5- Project Management

PROJECT MANAGEMENT

Project management of Visualization and forecasting of Stock using Dash project involves organizing and overseeing the various tasks and resources to ensure the successful execution and completion of the project.

Project management for stock price prediction project involves effectively planning, organizing, and executing various tasks to ensure the successful completion of the project. Clearly define the objectives and goals of the project. Identify the specific outcomes you aim to achieve through the data preprocessing and SVR model. Develop a detailed project plan that outlines the tasks, timelines, and resources required to complete the project. Define milestones and deliverables and allocate resources accordingly. Consider factors like data collection, preprocessing, analysis techniques, model selection, evaluation, and reporting. Assemble a team with the necessary skills and expertise for the project.

Identify team members responsible for data collection, preprocessing, sentiment analysis, visualization, and reporting. Assign roles and responsibilities and establish clear communication channels within the team. Determine the data sources and methods for collecting Twitter data. Define the search queries, hashtags, or user profiles to target. Ensure data privacy and compliance with relevant regulations. Establish procedures for data preprocessing, including text cleaning, tokenization, and feature extraction. Research and select appropriate sentiment analysis techniques for the project. Consider both traditional machine learning approaches (e.g., SVM, Naive Bayes) and modern deep learning methods (e.g., recurrent neural networks, transformers). Evaluate the pros and cons of each technique based on project requirements. Develop and train the sentiment analysis models using suitable datasets. Fine-tune the models based on labelled data specific to your project. Optimize hyperparameters, assess model performance, and ensure the models align with project objectives. Document the process for reproducibility. Integrate the developed models into the sentiment analysis pipeline. Perform extensive testing and validation to ensure the accuracy and reliability of sentiment predictions. Monitor the system's performance, evaluate any limitations or biases, and iteratively refine the models as needed. Analyze the sentiment analysis results to extract meaningful insights. Generate visualizations such as sentiment distribution charts, word clouds, or sentiment trends over time. Interpret the findings to gain insights into public opinion, brand

reputation, or other relevant aspects related to the project objectives. Regularly assess the project's progress and performance against the defined objectives and milestones. Conduct periodic reviews, gather feedback from stakeholders, and make necessary adjustments to ensure the project stays on track. Document lessons learned for future improvements. Prepare comprehensive reports and presentations summarizing the project methodology, findings, and insights. Clearly communicate the results to stakeholders, clients, or project sponsors. Tailor the presentation to the intended audience, emphasizing the impact and relevance of the sentiment analysis results. Document the project workflow, code, and any relevant procedures, including data sources and preprocessing steps. Maintain proper documentation for future reference, replication, or extension of the project. Establish a system for ongoing maintenance and support as needed. Conduct a thorough project review to evaluate the overall success and lessons learned. Archive project artifacts and finalize documentation. Celebrate achievements, recognize team members' contributions, and transition any ongoing responsibilities or maintenance to the appropriate stakeholders. Celebrate the successful completion of the project with the team. Perform a final review to ensure all project deliverables are met. Archive project documentation and code for future reference.

Throughout the project management process, it's crucial to maintain open communication with the team, stakeholders, and clients. Regularly track progress, address challenges proactively, and adapt the project plan if needed. Flexibility, collaboration, and effective project management techniques will contribute to the successful completion of a Twitter sentiment analysis project.

We should remember that effective project management involves regular communication, collaboration among team members, and adaptability to handle unforeseen challenges or changes throughout the project lifecycle.

5.1 Process Planning and Scheduling:

Project Planning and Scheduling‘, though separate, are two sides of the same coin in project management. Fundamentally, _Project planning‘ is all about choosing and designing effective policies and methodologies to attain project objectives. While Project scheduling ‘is a procedure of assigning tasks to get them completed by allocating appropriate resources within an estimated budget and time-frame.

The basis of project planning is the entire project. Unlikely, project scheduling focuses only on the project-related tasks, the project start/end dates and project dependencies. Thus, a _project plan ‘is a comprehensive document that contains the project aims, scope, costing, risks, and schedule. And a project schedule includes the estimated dates and sequential project tasks to be executed.

Project Planning: The project planning phase refers to:

Developing a project to make it ready for investment.

Determines the jobs/tasks required to attain project objectives.

Stages of Project Planning

The project planning stages are enlisted below:

Identifying the key project sponsors and stakeholders, to determine the basis of project scope, budget, and timeframe for project execution.

Upon enlisting the stake-holder requirements, prioritizing/setting project objectives.

Identifying the project deliverables required to attain the project objectives.

Creating the project schedule.

Identifying the project risks, if any, and develop suitable mitigation plans.

Communicating and presenting the project plan to stakeholders.

Benefits of Project Planning

- Route-Map: The project plan offers a roadway that gives direction to the project from start to end.

- Documentation of Customer Requirements: A well-articulated project plan enables the record of the requirements of the customers in a documented form. This provides a precise direction instead of relying on assumptions, which could be incorrect and may lead to project errors.
- Task Autonomy: Planning enables one to assign tasks to specific team members and gives autonomy. The team feels a sense of responsibility and ownership of the success or failure of a project. Consequently, it urges them to work better or encourages them to bring inconsistent results.
- Resource Estimation: Planning is vital as in a way, it enables us to estimate resources, costing and time. It gives a judgment of any delays if several members are working on various projects at a time.
- Mitigation Plan: The project plan gives a way to forecast risks, if any, and plan for mitigation strategies accordingly.
- Identification of Employee Capabilities: The planning phase enables to identify employees with certain skill sets or expertise. And as the tasks get assigned, team members get trained on a lacking skill sets or either upgraded on the ones they possess.
- Strengths and Short-Comings of Previous Projects: Project plans also help to analyze and improve or learn from the previous project records and facilitate decision-making.

Project Scheduling

The project scheduling phase refers to:

- Estimation of human resource and material requisite at every stage of the project; and approximate calculative time to complete each of these tasks.
- Indicates the start and end date of each project task and logical connectivity among various project tasks/activities.

Stages of Project Scheduling

The project scheduling stages are outlined below:

- Based on the project scope, design and develop the TBS (Task-Breakdown Structure).
- Identify the project-related tasks.
- Identify the human resources and material requisite.
- Evaluate the approximate time required for each and every task.
- Allocation of resources
- Analyze the detailed schedule.
- Monitor and govern the schedule.

Benefits of Project Scheduling

- Reduces Lead Time: The project schedule gives an outline of the tasks that are to be completed on a priority basis or simultaneously with other tasks. This keeps the team members notified about it and prevents any delays or postponing of tasks, thus reducing the lead time.
- Cost Reductions: It enables to monitor of the resources by preventing the overlapping of tasks. It also leads to the effective utilization of resources and returns the unconsumed resources in time, thus cutting costs.
- Facilitates Productivity: Upon evaluating logical connectivity between the tasks, resources that are not optimally utilized can be assigned on extra tasks, thus enhancing productivity.
- Foresee problems in Advance: A precise project schedule enables one to foresee any problems in advance pertaining to either, under or over-utilization, of resources and ensures optimum consumption of the same.
- Sets a Goal: A project schedule allows us to set goals, short-term or long-term, providing a direction and vision while executing the project. It also makes everyone in a team aware of the guidelines and methods to attain these goals. Without a schedule, the project would be vaguely defined. Thus, making it cumbersome to manage and organize the tasks so as to run it successfully.
- Current Progress Updates and Alerts: The project schedule is a sketch that gives way to the project. A project might go through certain challenges, however, if there is no route map, how would a project move in the right direction? In such a case, a project schedule helps in assessing how off-track a project has been and possible ways to bring it in the correct direction.

5.1.1 Project Development Approach

In Handwritten digits recognition, the choice of project development approach depends on factors such as project complexity, team size, timeline, and available resources. Here are a few common development approaches that can be applied to Handwritten digits recognition:

Waterfall Approach:

The Waterfall approach follows a linear and sequential development process. Each phase (requirements gathering, design, development, testing, deployment) is completed before moving on to the next. This approach works well when project requirements are well-defined and unlikely to change significantly during the development process. However, it may not be suitable for projects where requirements evolve rapidly or require frequent iterations.

Agile Approach:

Agile methodologies, such as Scrum or Kanban, emphasize flexibility, collaboration, and iterative development. The project is divided into smaller iterations or sprints, usually lasting from 1 to 4 weeks. Each iteration involves planning, development, testing, and review. Regular communication and feedback from stakeholders drive the development process. This approach is beneficial when requirements are not fully defined, allowing for adaptive planning and quick iterations to incorporate changes and feedback.

Prototype Approach:

The prototype approach involves developing an initial version or prototype of the sentiment analysis system. The prototype focuses on the core functionality and features. Users or stakeholders can provide feedback based on the prototype, guiding further development. This approach allows for early validation of concepts and user requirements, reducing the risk of developing a system that does not meet expectations.

However, careful planning is necessary to ensure the prototype can evolve into a fully functional system without significant rework.

Incremental Approach:

The incremental approach involves dividing the project into increments or modules, each delivering a specific set of features or functionalities. Each increment builds upon the previous increment, adding new functionality or improving existing features. This approach allows for the delivery of working components in stages, providing value early in the development process. It can be particularly useful when there are time constraints or when specific features need to be prioritized.

Hybrid Approach:

A hybrid approach combines elements of different development approaches based on project requirements and constraints. It allows flexibility in adapting to changing needs and preferences.

For example, a project might start with a prototype or Agile approach to quickly gather feedback and refine requirements, followed by a more structured development process like Waterfall or incremental for subsequent stages.

It's important to note that the choice of development approach should align with the project's unique requirements, team capabilities, and available resources. Regular communication, collaboration, and adaptability are key to successful project development in Twitter sentiment analysis.

5.1.2 Project Plan

A project plan—sometimes called a work plan—is a blueprint of the goals, objectives, and tasks your team needs to accomplish for a specific project. Our project plan should include information about our project schedule, scope, due dates, and deliverables for all phases of the project lifecycle. But not all project planning processes are created equal—which leads some teams to underutilize them or skip over them completely. To write an effective project plan, we need to be methodical (follow a series of steps), specific, and clear when it comes to our ideas and execution strategy.

Project planning is the second stage in the project management process, following project initiation and preceding project execution. During the project planning stage, the project manager creates a project plan, which maps out project requirements. The project planning phase typically includes setting project goals, designating project resources, and mapping out the project schedule. Project plans set the stage for the entire project. Without one, we're missing a critical step in the overall project management process. When we launch into a project without defined goals or objectives, it can lead to disorganized work, frustration, and even scope creep. A clear, written project management plan provides a baseline direction to all stakeholders, while also keeping everyone accountable. It confirms that we have the resources you need for the project before it actually begins.

A project plan also allows us, as the person in charge of leading execution, to forecast any potential challenges you could run into while the project is still in the planning stages. That way, you can ensure the project will be achievable—or course-correct if necessary. According to a study conducted by the Project Management Institute, there is a strong correlation between project planning and project success—the better your plan, the better your outcome. So, conquering the planning phase also makes for better project efficiency and results.

The basic outline of any project plan can be summarized in these five steps:

- Define your project's stakeholders, scope, quality baseline, deliverables, milestones, success criteria and requirements. Create a project charter, work breakdown structure (WBS) and a statement of work (SOW).
- Identify risks and assign deliverables to your team members, who will perform the tasks required and monitor the risks associated with them.
- Organize your project team (customers, stakeholders, teams, ad hoc members, and so on), and define their roles and responsibilities.
- List the necessary project resources, such as personnel, equipment, salaries, and materials, then estimate their cost.
- Develop change management procedures and forms.
- Create a communication plan, schedule, budget and other guiding documents for the project.

Milestones

A project milestone is a project planning tool that‘s used to mark a point in a project schedule. Project milestones can note the start and finish of a project, mark the completion of a major phase of work or anything that‘s worth highlighting in a project, such as the production of project deliverables. Milestones help project teams coordinate their efforts by helping everybody understand the objectives of the project and the action steps that must be taken to achieve them.

Project milestones help project teams focus on major progress points in a project, which helps project managers with project planning and scheduling. Just as tasks break a larger project into manageable parts, milestones break down project phases to help project managers plan, schedule and execute them.

Project milestones provide a way to more accurately estimate the time it‘ll take to complete your project by marking important dates and events, making them essential for precise project planning and scheduling. Because of their versatility, milestones are an important element of project documents such as the project schedule, project charter and project plan. They’re also used in scheduling methodologies, such as the critical path method (CPM), or project management tools like Gantt charts, which can determine major scheduling periods. With project milestones, you can better calculate the slack in your project by segmenting the project timeline into intervals, or smaller time frames to control and track progress.

Project management software, like Project Manager, makes it easy to build a schedule with project milestones. Use our online Gantt charts to quickly build a project schedule with phases, subtasks, milestones and dependencies.

Deliverables

Projects produce deliverables, which are simply the results of project activities. Project deliverables can be big or small, and their number varies depending on the project. They‘re agreed upon by the project management team and stakeholders during the project planning phase. Put another way, there are inputs and outputs in any type of project. Inputs are what you put into the project, such as data, resources, etc., and the outcomes are the deliverables. Again, those deliverables vary greatly. For example, a project deliverable can be either a product or service or it can be the documentation that‘s part of the project closure.

Project management tools such as Gantt charts, kanban boards and project calendars can help us track the progress of our team on the completion of project deliverables. In addition to these project management tools, Project Manager has one-click reporting that captures data on project variance, time, cost and more. These reports can be shared as PDF attachments or printed depending on the stakeholder ‘s preference.

A deliverable is quantifiable. It‘s something that was created over time, with resources and effort. A project milestone, while encompassing deliverables, is a marker in time to indicate the transition from one thing to another.

Dependencies

When creating a project schedule, it is crucial to identify and account for dependencies between tasks. In project management, dependency is defined as a relationship between two tasks in which the completion of one task is dependent on the completion of the other task.

Different types of dependencies can be identified in project schedules, and each type has its own set of implications for project planning and execution.

Dependencies in project management are connections between tasks on a project timeline. These dependencies determine sequencing and resource delegation in every project.

Types of Dependencies in Project Management:

- Logical (Casual) Dependencies

Logical dependencies are the usual project dependencies that project managers usually encounter. These dependencies reflect the sequence of tasks and their logical relationship to each other.

- Resource Dependencies

Resource dependencies occur when a project task cannot be started or completed until the required resources become available. These dependencies are restricted by cost, time, or the overall scope of the project (constraints).

- Preferential Dependencies

In the case of preferential dependencies, some tasks are given a preferred status over others— when project managers prioritize project tasks by using a must-do, should-do, and nice-to-do approach.

- Sequential Dependencies

Sequential dependencies are project dependencies that must be completed in a specific order. These types of dependencies can often be found in project processes where tasks are divided into smaller work packages with multiple steps. Once project managers complete a project task, they can go to the following project deliverable.

- External Dependencies

External dependencies are project dependencies that come from outside of the project. They can be caused by factors such as natural disasters, changes in government policy, and supplier failure.

E.g., if a supplier doesn't deliver the correct materials on time, it will delay project timeline.

Project dependencies can be a cause of project delays and should be taken into account when planning projects.

Risk Management

Risk management is the process of identifying, assessing and controlling threats to an organization's capital and earnings. These risks stem from a variety of sources, including financial uncertainties, legal liabilities, technology issues, strategic management errors, accidents and natural disasters. A successful risk management program helps an organization consider the full range of risks it faces. Risk management also examines the relationship between risks and the cascading impact they could have on an organization's strategic goals.

Risk management involves identifying, analyzing, and accepting or mitigating uncertainty in investment decisions. Put simply, it is the process of monitoring and dealing with the financial risks associated with investing. Risk management essentially occurs when an investor or fund manager analyzes and attempts to quantify the potential for losses in an investment, such as a moral hazard, and then takes the appropriate action (or inaction) to meet their objectives and risk tolerance.Risk management structures are tailored to do more than just point out existing risks. A good risk management structure should also calculate the uncertainties and predict their influence on a business. Consequently, the result is a choice between accepting risks or rejecting them. Acceptance or rejection of risks is dependent on the tolerance levels that a business has already defined for itself.

If a business sets up risk management as a disciplined and continuous process for the purpose of identifying and resolving risks, then the risk management structures can be used to support other risk mitigation systems. They include planning, organization, cost control, and budgeting. In such a case, the business will not usually experience many surprises, because the focus is on proactive risk management.

Response to risks usually takes one of the following forms:

- Avoidance: A business strives to eliminate a particular risk by getting rid of its cause.
- Mitigation: Decreasing the projected financial value associated with a risk by lowering the possibility of the occurrence of the risk.
- Acceptance: In some cases, a business may be forced to accept a risk. This option is possible if a business entity develops contingencies to mitigate the impact of the risk, should it occur.

Estimation

To provide an estimation for a project, I would need more specific details about the nature of the project, such as its scope, requirements, deliverables, and any constraints or dependencies involved. Additionally, factors such as team size, expertise, and the complexity of the project would also affect the estimation.

Without specific information about the project, I can provide you with a general framework for estimating a project:

Define the project scope: Clearly outline the objectives, deliverables, and boundaries of the project. This will help identify the work that needs to be done.

Break down the project into tasks: Divide the project into smaller, manageable tasks. This breakdown will allow for a more accurate estimation of effort and resources required.

Estimate effort for each task: Assess the amount of time and resources needed to complete each task. This estimation can be done by considering factors like complexity, dependencies, and past experience.

Determine resource availability: Identify the resources required for the project, including human resources, equipment, and materials. Consider the availability and capacity of each resource.

Create a project schedule: Develop a timeline that outlines the sequence of tasks and their estimated durations. Account for dependencies and any potential risks or constraints.

Identify risks and contingencies: Evaluate potential risks that may impact the project timeline or budget. Plan for contingencies and allocate buffer time or resources to mitigate these risks.

Review and refine: Once the initial estimation is complete, review and refine the estimates based on expert judgment, historical data, or feedback from stakeholders.

Chapter 6- Input Design

Our Web App looks as shown below

```
(base) C:\newpython\Mini Project>python app.py
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
C:\newpython\Mini Project\model.py:3: UserWarning:


The dash_core_components package is deprecated. Please replace
`import dash_core_components as dcc` with `from dash import dcc`

C:\newpython\Mini Project\model.py:4: UserWarning:


The dash_html_components package is deprecated. Please replace
`import dash_html_components as html` with `from dash import html`

[**********************100%**********************]  1 of 1 completed
```

Fig.12. Loading the Server

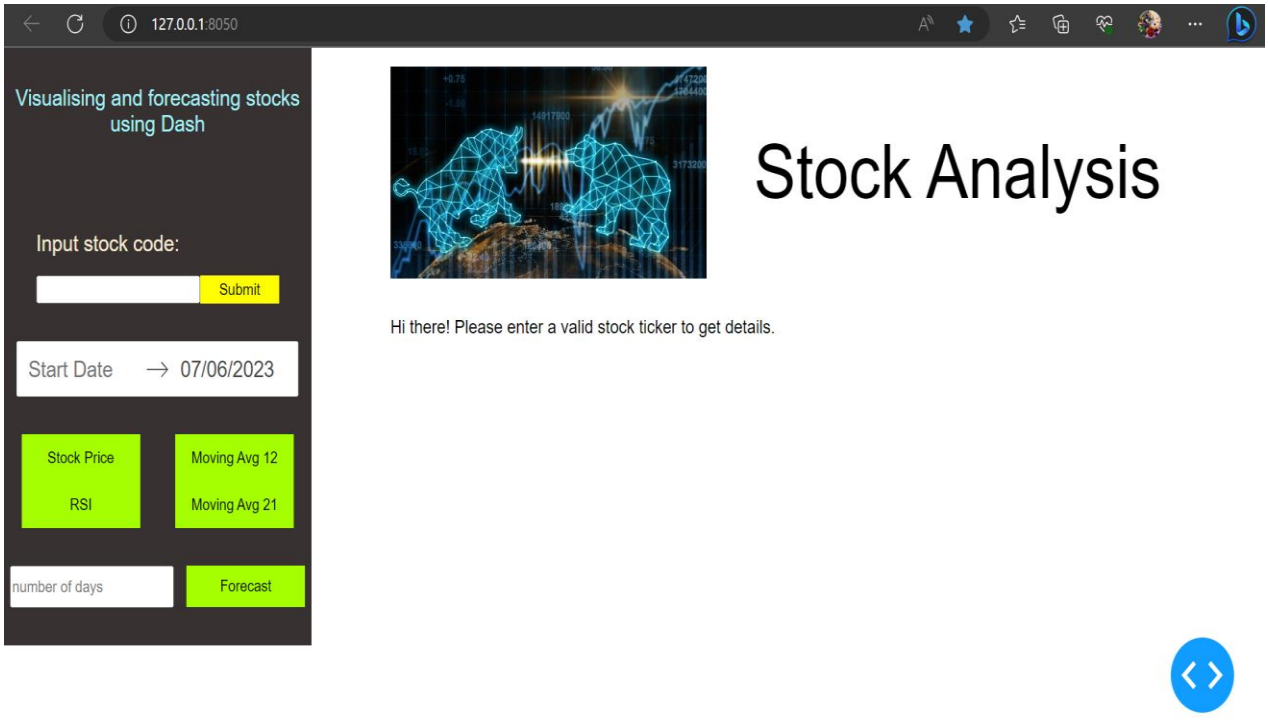


Fig.13. Basic web app look

Chapter 7- Output Design

. Input a stock code to get company's information as shown in Fig.14

Fig.14. Chart View

Fig.15. RSI indicator

Using the date range selector provide the start and end date to get the stock prices (closing and opening prices) of the company whose stock code you have provided earlier as shown in Fig.15

. You can choose the indicator button to get the indicators by providing the company stock code and date range as shown in Fig.16

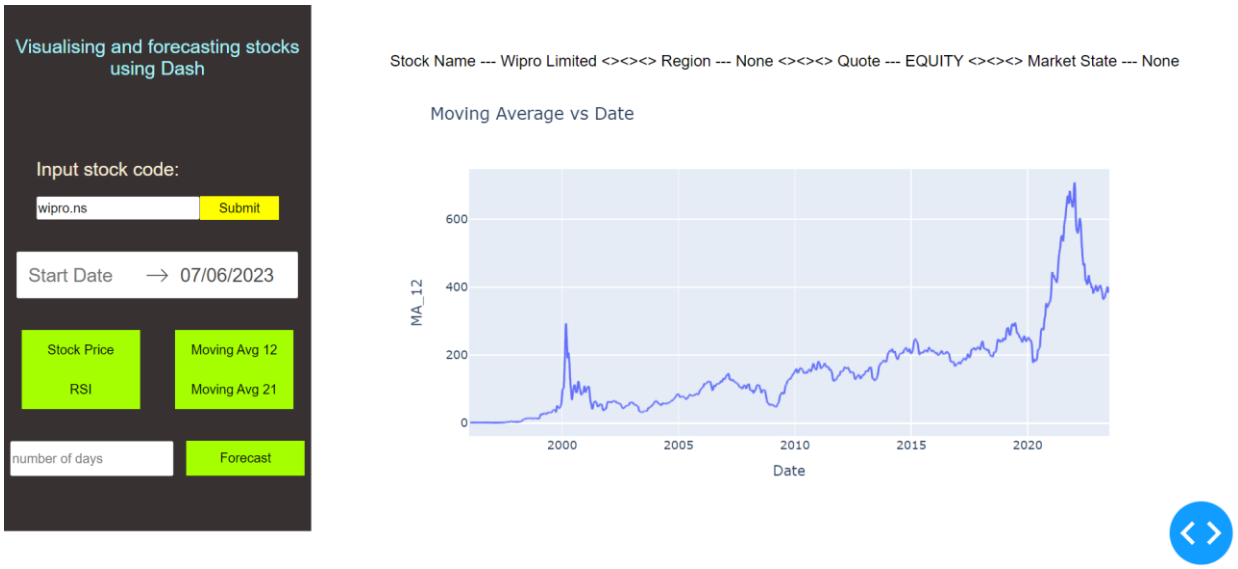


Fig.16. Moving Average Indicator

Fig.17. Prediction

. You can predict the company stock prices (closed price) by providing the number of days and choosing the forecast button as shown in Fig.17 you will get a time series plot between Date and Closed Price.

Chapter 8 – System Testing & Maintenance

System testing and maintenance are crucial aspects of ensuring the reliability, functionality, and performance of a software system. Let's take a closer look at each of these activities:

System Testing: System testing involves the evaluation of the entire software system to verify that it meets the specified requirements and functions as expected. The primary goal is to identify defects or issues that may exist within the system. Here are some commonly performed system testing techniques:

a. Functional Testing: This involves testing the system's functions and features against the functional requirements to ensure they work as intended.

b. Performance Testing: It assesses the system's performance under different conditions, such as heavy user loads or high data volumes, to determine if it meets the performance criteria.

c. Security Testing: It focuses on identifying vulnerabilities and weaknesses in the system's security measures to protect against potential threats or breaches.

d. Usability Testing: This examines the user-friendliness and ease of use of the system, ensuring it meets the needs of its intended users.

e. Compatibility Testing: It checks the system's compatibility with different hardware, software, operating systems, and browsers to ensure it functions correctly across various environments.

f. Regression Testing: It involves retesting previously tested functionalities to ensure that changes or fixes do not introduce new defects.

System Maintenance: System maintenance involves the activities performed to keep a software system operational, up-to-date, and in line with user requirements. It typically consists of the following tasks:

a. Bug Fixes: Addressing and resolving reported issues or defects discovered during testing or reported by users.

b. Updates and Enhancements: Incorporating new features, improvements, or functionality to meet evolving user needs or industry standards.

c. Performance Optimization: Analyzing and optimizing the system's performance to ensure efficient resource utilization and responsiveness.

d. Security Updates: Applying security patches, addressing vulnerabilities, and keeping the system protected against emerging threats.

e. Database Maintenance: Regularly optimizing and managing the database to ensure data integrity, performance, and reliability.

f. User Support: Providing assistance and support to system users, including troubleshooting, answering queries, and addressing user issues.

g. Monitoring and Logging: Implementing monitoring tools to track system performance, identify potential issues, and generate log files for analysis.

h. Backup and Recovery: Establishing regular backup procedures and recovery plans to protect against data loss and ensure business continuity.

By performing systematic testing and implementing proactive maintenance practices, organizations can ensure that their software systems remain robust, reliable, and effectively serve their intended purpose.

Chapter 9 – Summary and Future Scope

Developing predictive price models for the stock market is challenging, but it is an important task when building profitable financial market transaction strategies. Computationally intensive methods, using past prices, are developed to facilitate better management of market risk for investors and speculators. Of the machine learning techniques available, this study uses SVR and measures its performance on various Brazilian, American, and Chinese stocks with different characteristics, for example, small cap or blue chip. The predictive variables are calculated using TA indicators on asset prices. The results show the magnitude of the mean squared errors for the three common kernels in the literature, using specific algorithm training strategies with different price frequencies of days and minutes. The results are contrasted with those of a random walk-based model.

This study shows that using a fixed training set on daily prices, it is possible to obtain smaller prediction errors in the test set than in the training set when using a linear kernel. Moreover, this kernel was more adequate for price predictions than the radial and polynomial kernels in the case of daily prices and fixed training models and outperformed the random model for some stocks classified as blue chips and small caps in the three studied countries. However, increasing the price frequency to minutes reduced the model's predictive power using a fixed training period. In particular, SVR obtained inferior predictive results relative to a random walk model for almost all stocks studied in up-to-the-minute prices, using fixed training, regardless of the adopted kernel function.

The periodically updated models provided important evidence. In these cases, the use of linear and radial kernels resulted in smaller errors that the random walk model for almost all daily stock prices. The only exception was a stock with a high missing data rate. Constant model updating was also beneficial in the up-to-the-minute price frequency, and SVR models with linear and radial kernels achieved better results than the random walk model when this strategy was used. To emphasize the stability of the predictions over the long run, we processed a 2-years up-to-the-minutes prices period for the selected Brazilian stocks, confirming better results with a constantly updated model. The analyses presented in this study suggest that periodically updating the SVR model reduces the mean square error compared to using a rigid model without periodic updating. This result contrasts with that of Hsu who did not achieve better performance when using a sliding window on the training data.

An important contribution of this study is a comparison of price prediction results of the presented SVR models with those of the random walk model, according to which markets are unpredictable in the long term. In this respect, the results presented here show that some SVR models, with periodic or fixed updates, may achieve better than random predictive performance, especially with the use of the linear kernel. Another result which prompts further investigation is the indication of a strong relationship between SVR price prediction and volatility, considering a moving training window.

Importantly, despite the evidence of asset price predictability presented here, this article does not propose transactional strategies applicable to the stock market. The results therefore do not directly refute the EMH. Given that the focus of the analysis is not the identification of purchasing or sales strategies that allow for extraordinary gains, the study does not address issues such as transaction costs or portfolio risk levels.

As the focus of the study is the analysis of asset price prediction errors, it is possible to build risk management models using SVR-based estimates. Exposure limits may be obtained by evaluating model errors. This study therefore provides a basis for the construction of systems that, while not directly evaluating the EMH, make possible the study of market efficiency and risk analysis. This study obtained results using SVR that were better than those of a null mean return random model.

- In recent years, it has been noted that most people are investing in the stock market in order to make fast money. At the same time, an investor stands a good risk of losing all his or her money. Data visualization helps traders when making decisions quickly and enables them to easily synthesize large amounts of complex information.

- Stock forecasting and visualization using dash is a machine learning project. We have built a single page web application using the plotly dash python framework and using some machine models to get company's information and plot graphs for visualization based on it. Stock market forecasting is done using a machine learning model (SVR).

Dash provides a fast and quick way of creating beautiful and interactive dashboards without extensive knowledge of web development. It combines web development and analytics and provides a useful mechanism for serving dashboards for web response.

References/Bibliography

- Dash HTML Components | Dash for Python Documentation | Plotly
- Dash Core Components | Dash for Python Documentation | Plotly
- CSS Tutorial
- A Complete Guide to Flexbox | CSS-Tricks
- Part 3. Basic Callbacks | Dash for Python Documentation | Plotly
- Build A Stock Prediction Program
- Python LSTM (Long Short-Term Memory Network) for Stock Predictions | Data Camp
- An introduction to Grid Search. This article will explain in simple… | by Krishni | DataDrivenInvestor
- Plotly Express
- Hands-On Guide To Using YFinance API In Python
- https://www.crio.do/projects/category/python-projects/
- Part 2. Layout | Dash for Python Documentation | Plotly

Appendix

Appendix A: Data Visualization and Forecasting 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
import ta
# model
from model import prediction
from sklearn.svm import SVR

# ... (rest of the code)
```

Appendix B: Supporting Functions

```
def get_stock_price_fig(df):
    # Function to generate a candlestick chart for stock price visualization
    # ...
def get_more(df):
```

```
    # Function to generate a moving average chart
    # ...
def get_moree(df):
    # Function to generate another moving average chart
    # ...
def get_moreee(df):
    # Function to generate an RSI chart
    # ...
```

Appendix C: Dash Application Layout

```
app = dash.Dash(__name__)
server = app.server
# html layout of site
app.layout = html.Div(
    [
        # ... (rest of the layout code)
    ],
    className="container")
```

Appendix D: Callback Functions

```
# callback for company info
@app.callback(
```

```
    [
        Output("description", "children"),
        Output("logo", "src"),
        Output("ticker", "children"),
    ],
    [Input("submit", "n_clicks")],
    [State("dropdown_tickers", "value")],
)
def update_data(n, val):
    # Callback function to update company information
    # ...


# 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):
```

```
    # Callback function to display stock price graph
    # ...


# 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):
    # Callback function to display moving average chart
    # ...


# callback for indicators
@app.callback(
    [Output("main-content1", "children")],
    [
        Input("indicatorss", "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):
    # Callback function to display another moving average chart
    # ...


# callback for indicators
@app.callback(
    [Output("main-content2", "children")],
    [
        Input("indicatorsss", "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):
    # Callback function to display RSI chart
    # ...


# 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):
    # Callback function to display forecast graph
    # ...


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

The above appendices provide the additional code snippets that complement the original code. They include the main code for data visualization and forecasting, supporting functions for generating visualizations, the Dash application layout, and the callback functions that handle user interactions and update the content of the application.