Wrocław University of Science and Technology
Faculty of Information and Telecommunication Technology

Field: ISTANG

TEAM PROJECT

System Supporting Prediction of Stock Market Courses on the Basis of Analysis of Textual and Numerical Data Streams.

RAHUL VIJAYVARIYA - 245784
ARJUN SAHAJPAL - 252482
SHUBHAM KISHORE - 253558
ALAN AUSTINLAL - 257317

Supervisor
dr hab. inż. Maciej Piasecki

Keywords: LSTM, XG-Boost, RNN, Twitter, FinViz, Yahoo Finance.

WROCŁAW (2022)

TABLE OF CONTENT

DESIGN DOCUMENTATION

USER DOCUMENTATION

LIST OF FIGURES

DESIGN DOCUMENTATION

1.List of acronyms and designations

Acronyms	Designations
RNN	Recurrent neural network
XGBoost	Extreme gradient boosting
RSI	Relative strength index
MACD	moving average convergence/divergence
LSTM	Long short term memory
SMA	simple moving average
NFLX	Netflix inc
GOOG	Google LLC
MSFT	Microsoft corporation
TSLA	Tesla
ABNB	Airbnb
APPL	Apple Inc
BABA	Alibaba Group Holding Limited
RELI	Reliance Global Group Inc
WMT	Walmart Inc
AMZN	Amazon Inc
NVDA	NVIDIA Corporation
IBM	International Business Machines Corporation
INFY	Infosys Limited
WIT	Wipro Limited
META	META PLATFORMS
TSM	Taiwan Semiconductor Mfg. Co. Ltd

2. Purpose and scope of the project

2.1 PROJECT INTRODUCTION

An attempt to forecast the future value of a company stock or other financial instrument traded on a financial exchange is referred to as a stock market prediction. An correct future price forecast for a company could generate a large profit. The markets on stock exchanges are not efficient. Investors often act in a herd, but not all information is shared at once, and various investors require varied lengths of time to process it before taking action.

Despite the fact that stock prices are quite volatile, it is very challenging for one person to make an accurate prediction. There is a slim probability to correctly predict stock prices using machine learning and deep learning approaches, In order to assist customers with their trading decisions, we felt it would be difficult to forecast the feelings of news and tweets using machine learning and deep learning approaches..

Because the ideal time to invest or hold depends on a range of circumstances, such as interest rates, current affairs, or the introduction of new products, it is challenging to anticipate stock prices with any degree of accuracy. The feelings expressed in tweets and news are two more factors that have an impact on stock values in different ways.

2.2 PURPOSE OF THE PROJECT:

Studying and enhancing artificial intelligence stock price prediction systems is the goal of this research. The stock market has always been the focus of attention for the general public, academic interest, economists, policymakers, stock market traders, and market makers in the modern world where we have the resources and such advanced technology of artificial intelligence, which has made significant progress in recent years. Our objective is to create a web application that uses artificial intelligence to forecast stock values.

While some newspapers stress the stock prices' irregular and unpredictable nature, others provide beneficial business suggestions. How to anticipate the stock market has been a topic of discussion recently, which caught our attention and motivated us to choose this area of research as the subject of our Major Project..

2.3 TECHNICAL OBJECTIVE

The project's technical goal will be carried out in Python. The program must be able to access data from the Yahoo Finance API regarding a list of stocks. To forecast the tone of the news, the system will pull tweets from the Twitter API and financial news from the FinViz NLTK library.
LSTM, XGboost, and RNN are examples of machine learning and deep learning techniques used to predict stock values.
The web application will be developed using the open-source Python app framework stream-lit.

3. Glossary

1	Data mining	Large data sets are sorted through in data mining in order to find patterns and relationships that may be used in data analysis to assist solve business challenges.
2	Moving Average	A statistical process that people decide when to buy and sell by reducing or eliminating data fluctuations.
3	Relative Strength Index	A technical indicator used in the research of financial markets is the relative strength index. Based on the closing prices of a recent trading session, it aims to chart the present and past strength or weakness of a stock or market.
4	Technical Analysis	Technical analysis is a process used in finance to analyze and predict price movements by looking at historical market data, especially price and volume.
5	Trend	The general drift, tendency or bent of a set of statistical data as related to time.
6	Volume	The shares which are traded for a given market or tradable within a specified time period.
7	50 – day moving average	The 50-day moving average is a common technical indicator that can help investors analyze past prices. It's the average closing price of a security over the past 50 days.
8	100-day moving average	The 100-day moving average is a common technical indicator that can help investors analyze past prices. It's the average closing price of a security over the past 100 days.
9	200-day moving average	The 200-day moving average is a common technical indicator that can help investors analyze past prices. It's the average closing price of a security over the past 200 days.
10	MovingAverage Convergence Divergence(MACD)	The MACD shows how the prices of two different moving averages are changing over time. This helps you predict whether a security's price is about to rise (positive divergence) or fall (negative divergence).
11	Stock ticker	The stock market is a place where people can buy and sell different types of securities. The prices of these securities change all the time, and this information is displayed on a ticker.

12	Sentimental analysis	is an approach to natural language processing(NLP) that is used to identify the emotional tone behind a body of text.
13	Neural Network	A computer software with artificial intelligence capable of learning by making mistakes throughout the training phase.
14	Regression	A mathematical way of stating the statistical linear relationship between one independent and one dependent variable
15	Long short- term memory(LSTM)	is an artifical neural network that is used in the fields of artifical intelligence
16	RNN	A recurrent neural network is a type of artifical neural network which uses sequential data or time series data.
17	Xgboost	(Xtreme Gradient Boosting) is an open source software library which provides a gradient boosting frameowrk. It aims to provide a 'Scalable Portable and Distributed Gradient Boodting Library'
18	N_estimators	are the number of trees that you need to build before taking the maximum voting or average of predictions
19	Optimiser	Optimizers are algorithms or techniques that are used to alter the weights and learning rates of your neural network in order to lessen losses.
20	Activation function	The output of a node in an artificial neural network is determined by its activation function when given an input or set of inputs. A digital network of activation functions can be "ON" or "OFF" depending on the input to a standard integrated circuit.

4. State of knowledge in the project field

This chapter includes the comparisons and how the application is optimized compared to other existing solutions, moreover how it is beneficial over other solutions and stands out. We have a product with minimum maintenance and more reliability as AI is the future so this solution is a small attempt to take and have a peek at AI's power and what problems can be solved with the technologies in the future with less human efforts.

4.1 Existing solutions available on internet:

4.1.1 inteliCharts Predictive Stock Market Analytics: It is a quantitative modeling tool for predicting financial time series. Since each stock, index, or other financial instrument is unique, the system is fundamentally able to adapt as it understands the patterns and geometrical connections created by previous time series data points.

4.1.2 Markettrak: Its stock market forecast system that mainly consists of two major parts: an extensive database and a forecast model. The forecast model reads the database and then makes a prediction of where the market is headed. From this prediction, it determines a trading position for the Dow Diamonds or the SP500 Spiders. The database and forecast are updated daily at the close of trading.

4.1.3 tipranks.com:gives active day traders, short- and long-term investors access to cutting-edge price prediction technologies. For predicting and analysis of the stock market, they provide web-based tools. The stock-forecasting system makes predictions about stock prices, creates "Buy-Hold-Sell" trading signals, determines the most thriving trade to invest in, and evaluates the precision of forecasts.

4.1.4 yahoo finance: It provides financial news, data and commentary including stock quotes, press releases, financial reports, and original content. It also offers some online tools for personal finance management.

4.1.5 investing.com: Is one of the top three international financial websites in the world, investing.com is a financial platform and news website. It provides market quotations, details on stocks, futures, options, analysis, and commodities, as well as a calendar of economic events.

4.1.6 Awario: Awario is a web-based social media monitoring platform with several features, including sentiment analysis. Forums, blogs, websites, and social media platforms (such as Twitter, Facebook, and Reddit) provide the data that Awario examines. It has the ability to instantly evaluate the emotion of mentions. The keywords we wish to track may be entered in Awario to set it up to collect data online. They may be related to a particular brand, item, rivalry, sector, or any other phenomenon we wish to look up online.

4.2 Existing research available on internet:

4.2.1 Stock Price Prediction Using Support Vector Machine Approach by Naliniprava Tripathy from Indian Institute of Management

In this research they predict the direction of S&P BSE TECK price movement in the Indian stock market. The study finds that the average prediction performance of SVM, Models show a 60.2% increase after the 2008 financial crisis. To find the most efficient combination, different feature set combinations are used.

4.2.2 Stock Price Prediction using ARIMA Model by ARAVIND GANESAN, ADARSH KANNAN from Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya

In this research they have predicted using Auto ARIMA model and implemented with various packages in python. Historical stock data of ICICI Bank and Reliance Industries have been collected from Yahoo Finance

Where they are using yahoo finance for getting stock information and using arima for preprocessing the all-time data in dataset which predicts the probable analysis and good accuracy, where they made it simple and less featured as it needed to be added new features

Comparison Table

Implemented Method v/s ARIMA

1. LSTM is more accurate when using datasets with longer sequences 2. LSTM typically achieves error rates that are between 84 and 87 percent lower. 3. LSTM ability to caption long team dependencies and maintain a memory of past events
1. XGBoost is that it is a highly scalable algorithm 2. XGBoost is a tree-based algorithm, which can handle non-linear relationships in the data
1. can be trained quickly and efficiently on a small amount of data 2. process sequences of data and maintain a memory of past events

Implemented Method V/s SVM

1. LSTM is a highly flexible algorithm, which can be used to model a wide variety of data types and prediction tasks 2. process sequences of data and maintain a memory of past events
1. XGBoost is that it is a highly scalable algorithm 2. XGBoost is a tree-based algorithm, which can handle non-linear relationships in the data
1. can be trained quickly and efficiently on a small amount of data 2. process sequences of data and maintain a memory of past events 3. SVM can be more computationally intensive for large data

5. Preliminary assumptions

In this chapter, we will be taking a look into the preliminary assumptions of our system and also in the following chapter, we will also take a look into the technologies that we have used in our project.

5.1 High level List of requirements:

Requirement ID	Requirement Description	Significance level	Group
Access	User can access the system	Must	UI
View.list of stocks	User can view and select the stock from list of stocks available	Must	UI
Display.real time data	User can view the real time data regarding the stock such as real time stock price,opening price, volume etc.	Must	UI
Get.real time data	The system can fetch real time data.	Must	System
Download.stock data	The user can download entire market information related to stock.	Optional	UI
Get.stock data	The system can generate current market information about a stock.	Must	System
Download.stock info	The user can download the historical data related to a stock.	Optional	UI
Get.stock info	The system can generate historical data related to a stock.	Must	System
Display.RSI	The user can view relative strength index for the selected stock in a plotted chart..	Optional	UI
Get.RSI	The system can generate relative strength index chart for the selected stock.	Must	System
Display.MACD	The user can view moving average convergence and divergence for the selected stock in a plotted chart.	Optional	UI
Get.MACD	The system can generate moving average convergence and divergence for the selected stock in a plotted chart.	Must	System
Display.50D SMA	The user can view 50 days simple moving average for the selected stock in a plotted chart.	Optional	UI

Get. 50D SMA	The system can generate 50 days simple moving average for the selected stock in a plotted chart.	Must	System
Display.100D SMA	The user can view 100 days simple moving average for the selected stock in a plotted chart.	Optional	UI
Get.100D SMA	The system can generate 100 days simple moving average for the selected stock in a plotted chart.	Must	System
Display.200D SMA	The user can view 200 days simple moving average for the selected stock in a plotted chart.	Optional	UI
Get. 200D SMA	The system can generate 200 days simple moving average for the selected stock in a plotted chart.	Must	System
Display.trend	The user can view stock price trend for the selected stock in a plotted chart.	Must	UI
Get.trend	The system can generate stock price trend for the selected stock in a plotted chart.	Must	System
Display.two side view	The user can view the two side view of price for the selected stock in a plotted chart.	Must	UI
Get.two side view	The system can generate the two side view of price for the selected stock in a plotted chart.	Must	System
Display.tweets sentiment	The user can view public opinion about a stock from twitter from selected list of stocks.	Must	UI
Get. tweets sentiment	The system can generate sentiment analysis based on tweets about a stock from twitter from selected list of stocks.	Must	System
Display.daily news sentiment	The user can view the sentiment analysis generated based on news.	Must	UI
Get.daily new sentiment	The system can generate sentiment analysis based on news from selected list of stocks.	Must	System
Display.daily new affecting price	The user can view how the news regarding the stock is affecting its price in a plotted chart	Must	UI

Get.daily new affecting price	The system can generate a sentiment analysis chart based on available news.	Must	System
Display.LSTM prediction	The user can view the LSTM prediction for the selected stock in a plotted chart.	Optional	UI
Get.LSTM prediction	The system can generate LSTM prediction for the selected stock in a plotted chart from the data aquired.	Must	System
Display.Xgboost prediction	The user can view the Xgboost prediction for the selected stock in a plotted chart.	Optional	UI
Get.Xgboost prediction	The system can generate Xgboost prediction for the selected stock in a plotted chart from the data aquired	Must	System
Display.RNN prediction	The user can view the RNN prediction for the selected stock in a plotted chart.	Optional	UI
Get.RNN prediction	The system can generate RNN prediction for the selected stock in a plotted chart from the data aquired	Must	System

5.2 TECHNOLOGIES USED

In this following subchapter, a brief description of the technologies used in our project and also why we will be using these technologies in our project is explained.

5.2.1 YAHOO FINANCE API

A Python package owned by Yahoo,The Yahoo Finance API is a set of programming instructions that allow developers to access financial data from Yahoo Finance. This API can be used to retrieve stock data, currency data, news, and other information from Yahoo Finance. Developers can use this API to build financial apps, websites, and other tools that can help people track and manage their financial portfolios and make more informed financial decisions.For this project we are using yahoo finance api as numerical data source for real time data,techinical analysis and for prediction of stock prices.

5.2.2 Visual studio Code

Microsoft created the source code editor Visual Studio Code for Windows, Linux, and macOS. Debugging support, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring are all features of this tool. Users may alter the editor's theme, keyboard shortcuts, and preferences thanks to its customization features. Developers frequently utilize the free and open-source Visual Studio Code to create a wide range of applications. Visual Studio Code 2019 was utilized for this project..

5.2.3 STREAMLIT

With the help of the open-source Python module Streamlit, programmers may build interactive web apps for data science and machine learning. You can use Streamlit to build apps that let users interact with data visualizations like graphs and charts. Streamlit offers developers a straightforward, user-friendly interface that makes it simple and quick to create complicated apps. Hot reloading is also supported, enabling you to view changes you make to your code in real time. For developing and deploying machine learning models and data-driven applications, data scientists and machine learning engineers frequently employ Streamlit..

5.2.4 TENSORFLOW

Google created the open-source machine learning framework known as TensorFlow. It offers a variety of tools and libraries for developing and deploying machine learning applications, as well as for implementing and training machine learning models. TensorFlow has support for a number of methods and architectures, including deep learning and neural networks, and it enables developers to construct and manipulate tensors, which are multidimensional arrays used in machine learning. Many different businesses, including robotics, banking, and healthcare, use TensorFlow extensively. In this project, LSTM and RNN machine learning models are implemented and trained using Tensor flow.

5.2.5 NLTK

Working with human language data is made easier by the Python module known as the Natural Language Toolkit (NLTK). With the help of NLTK, you can perform basic text processing tasks like tokenization, stemming, and part-of-speech tagging as well as more difficult ones like syntactic parsing and text categorization. A sizable set of test data, including corpora and wordnet, is also included in NLTK and can be used to create and assess natural language processing models. NLTK is a well-liked option for creating programs that can comprehend and produce human language, and it is frequently used by researchers and developers in the field of natural language processing. In order to implement sentimental analysis for this project, we are using NLTK.

5.2.6 Pandas

A well-liked Python package for working with tabular data is called Pandas. It is widely used in the field of data science and offers a number of tools and data structures for quickly editing and analyzing data. Large datasets may be quickly loaded, processed, and analyzed using Pandas, which also offers techniques for dealing with time series data and missing data. Pandas offers a robust and adaptable data analysis platform by supporting data visualization and integrating with other well-known Python libraries like NumPy and Matplotlib. We are utilizing Pandas for data analysis on this project.

5.2.7 Scikit-learn

A well-known Python library for machine learning is called Scikit-learn. It is built on top of other well-known Python libraries like NumPy and pandas and offers a variety of techniques and tools for creating and training machine learning models. Scikit-learn is an easy-to-use algorithm library that supports a number of techniques, including as classification, regression, clustering, and dimensionality reduction. Scikit-learn is a well-liked option for developing and deploying machine learning models in a number of applications, and it is widely used in the field of machine learning..

5.2.7 TWEEPY

A Python library for using the Twitter API is called Tweepy. It offers capabilities for logging into the API, sending and receiving tweets, and other frequently performed actions, enabling developers to simply interact with the Twitter network. Tweepy offers a user-friendly interface for interacting with the Twitter API and supports real-time tweet streaming. Developers that want to create programs that can communicate with Twitter frequently use Tweepy. As a textual data source for sentimental analysis based on tweets from Twitter, we are employing Tweepy for this project.

5.2.8 Matplotlib & Seaborn

A well-liked Python package for producing static, animated, and interactive graphics in 2D and 3D is called Matplotlib. It offers a large selection of plots and charts, including line plots, scatter plots, histograms, and heatmaps, and you can alter the way your plots look by choosing from a number of choices and styles. The subject of data visualization is dominated by the use of Matplotlib, which is frequently combined with other libraries like pandas and seaborn to produce detailed, insightful data representations.

A Python module called Seaborn is used to build statistical visualizations. It offers a high-level interface for producing stunning, instructive, and simple-to-understand statistical visuals and is developed on top of Matplotlib. Support for displaying a variety of data formats is provided by Seaborn., and it is a popular choice for creating attractive and informative visualizations in Python. For this project we are using Matplotlib & Seaborn for data visualizations.

6. Requirement specification and analysis

In this chapter, we will be defining the user stories and the functional and non-functional requirements of our system. We then give the textual description of user stories and we define the use-case diagram and the activity diagram of our system.

6.1 user stories

This section is an informal, general explanation of the software features written from the perspective of the end user.

EPICS:

- As a User, I want to access the application and get predictions,sentimental analysis,technical analysis and real time data so that I can take decision on the stock which I am going invest in.

- As an Administrator of the software, I want to access the backend of stock prediction application so that I can provide updates

FEATURES:

- As a user, I want to view the list of stocks which is available so that I can select the stock from list and view the predictions and analysis.

- As a user, I want to download the current market information about a particular stock from the list of stocks available. So that I can get detailed information about the company and the foundation of stock.

- As a user, I want to view the real time stock data such as real time stock price, volume of the stock, opening price of a selected stock from the list of stocks available so that I can get an overview of how the stock is performing in real time and compare with historical data.

- As a user, I want to view the historical price of a selected stock from the list of stocks available so that I can get overview of stock.

- As a user, I want to view the trend of a selected stock from the list of stocks available, in a plotted chart so that I can see the trend of the stock whether the prices are falling or rising.

- As a user, I want to view the day moving average of a selected stock from the list of stocks available, in a plotted chart so that I can see how the stock performed in last 50 days.

- As a user, I want to view the Relative strength index of a selected stock from the list of stocks available, in a plotted chart so that i can measure the change in the speed and magnitude of a security's recent price.

- As a user, I want to view the MACD (Moving Average Convergence Divergence) of a selected stock so that i can determine the current trend direction (bullish or bearish).

- As a user, I want to see the prediction of the stock prices so that I can take decision on the stock which I am going to invest in.

- As a user, I want to see the public opinion from twitter about a selected stock from the list of stocks available so that I can get overview whether public response on twitter are positive, negative, or neutral.
- As a user, I want to see how the news affect the stock prices about a selected stock from the list of stocks available so that I can get clearer picture of the current perception of a stock and whether news about it is good, negative, or neutral.

TECHNICAL STORIES:

- As an Administrator of the software, I want to have full control of the application so I can provide updates and give quality analysis

6.2 Functional requirements

Requirement No.1	View stocks	The system should let the user view the list of stocks displayed.
Requirement No.2	Select stocks	The system should let the user select one of the stocks from the lists of stocks displayed.
Requirement No.3	Get historical stock prices	The system will acquire the historical stock prices and stock from a list of stocks listed on Yahoo! Finance when clicked on that particular stock.
Requirement No.4	Plot graph	The program will output the stock prices as a graph of prices versus date.
Requirement No.5	Download market info	The system should allow the user to download the current market information about a particular stock when clicked on it.
Requirement No.6	View trend	The system should let the user view the trend of a stock whether the stock is rising or falling.
Requirement No.7	View 50-day chart	The system should let the user view the 50-day moving average of a selected stock in the form of a chart.
Requirement No.8	View 100-day chart	The system should let the user view the 100-day moving average of a selected stock in the form of a chart.
Requirement No.9	View 200-day chart	The system should let the user view the 200-day moving average of a selected stock in the form of a chart.
Requirement No.10	View Relative strength index	The system should let the user view the Relative strength index of a selected stock in the form of a chart.
Requirement No.11	Moving Average Convergence Divergence	The system should let the user view the Moving Average Convergence Divergence of a selected stock in the form of a chart.
Requirement No.12	View prediction	The system should let the user see the prediction made by the program.
Requirement No.13	Get economic news sentiments	The system should let the user see the economic news sentiments of a stock to get an sentiments of a stock whether the news it is positive, negative or neutral.
Requirement No.14	Get public tweets	The system should let the user see the public tweets sentiments of a stock to get an overview of a tweets regarding stocks whether the tweet is positive, negative or neutral.

6.3 Non-Functional requirements

Requirement No.1	Performance	Current data transfer should allow for quick and easy navigation of the site and features of the website and it's features
Requirement No.2	Functionality	The website is going to predict the stock prices based on numerical data and plot it in the form of a graph to help the users with their trading decisions. The website is going to give out sentimental analysis based on tweets and business news and plot it in the form of charts and graphs to help the users with their trading decisions.
Requirement No.3	Portability and compatibility	The website is completely portable and the recommendations completely trustworthy as the data is dynamically updated. Application should be accessed via every browser, Such as: - Chrome - Firefox - Edge It should be accessed with desktop and mobile versions of browser. Application should be compatible with every OS and every environment. It should be cross-platform, cross-browsing and mobile-responsive.
Requirement No.4	Reliability	Prediction will be made using real-time data to make the prediction.
Requirement No.5	Usability	The website is very simple and clean so that the user can view everything easily and access mostly everything on the click of a few buttons. The website will make the stock prediction as an overlay result on the price versus time graph.
Requirement No.6	Security	All the libraries used are certified and standard.
Requirement No.7	Learnability	User should understand main features of the program in less than 5 minutes. Every button should have a non-ambiguous title.
Requirement No.8	Efficiency	User should access the page they need in 3 steps or less.
Requirement No.9	Errors	There should not be hidden details to avoid user errors. User should make less than 3 errors during the application usage.

6.4 System use case diagram

The system use case diagram represents the functional requirements discussed in subchapter In the diagram shown in Figure 6.4.1 . The dependencies and what the user can do and view within the application is presented.

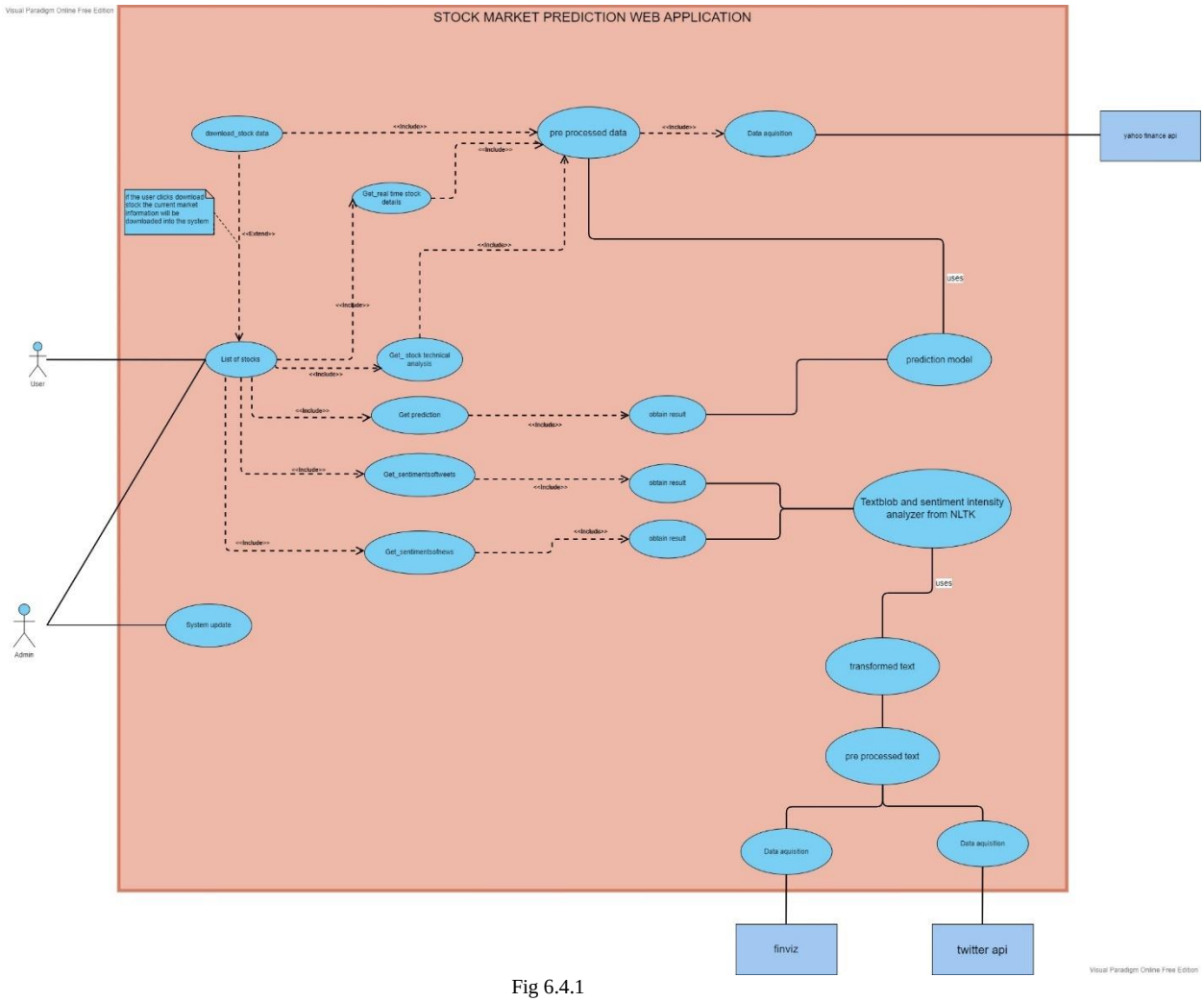


Fig 6.4.1

Textual description of major components present in fig 6.4.1

- Actors – User,Admin,finviz,twitter api and yahoo finance api
- List of stocks- where user can select the stock which needs to analyzed.
- Download stock data- where user can download entire market information related to a stock from the list of stocks available
- Get real time stock details- where user can see real time data about a stock from the list of stocks available such as real time price, opening price etc.
- Get stock technical analysis- where user can view different types of technical analysis chart related to a stock from the list of stocks available
- Get prediction - where user can view the predicted price trend chart of a stock from the list of stocks available.

- Get sentiments of tweets- where user can see the public opinion from twitter about a selected stock from the list of stocks available
- Get sentiments of news- where user can see the current perception of a stock from the list of stocks available whether news about it is good, negative, or neutral and how it affects the stock price

6.4.1 Textual use case specifications

ACTOR'S GOALS: To download market info
PARTICIPATING ACTOR'S : USER
PRE-CONDITIONS: The user does not have the downloaded market info of a stock
POST – CONDITIONS- The user has the downloaded market info of the selected stock
SUCCESS SCENARIO:
1) Select the stock for which the user wants to download market info
2) Select the download market info
3) Check the info that is downloaded

ACTOR'S GOALS: To check the technical analysis
PARTICIPATING ACTOR'S : USER
PRE-CONDITIONS: The user does not have the technical analysis of a stock
POST – CONDITIONS- The user has the technical analysis of the selected stock
SUCCESS SCENARIO:
1) Select the stock for which the user wants to see the technical analysis
2) Check the technical analysis of a stock

ACTOR'S GOALS: To check the sentiments analysis of news
PARTICIPATING ACTOR'S : USER
PRE-CONDITIONS: The user does not have the sentiments analysis of news.
POST – CONDITIONS- The user has the sentiments analysis of the news
SUCCESS SCENARIO:
1) Select the stock for which the user wants to see the sentiments analysis of the news
2) Check the sentiments analysis of the news

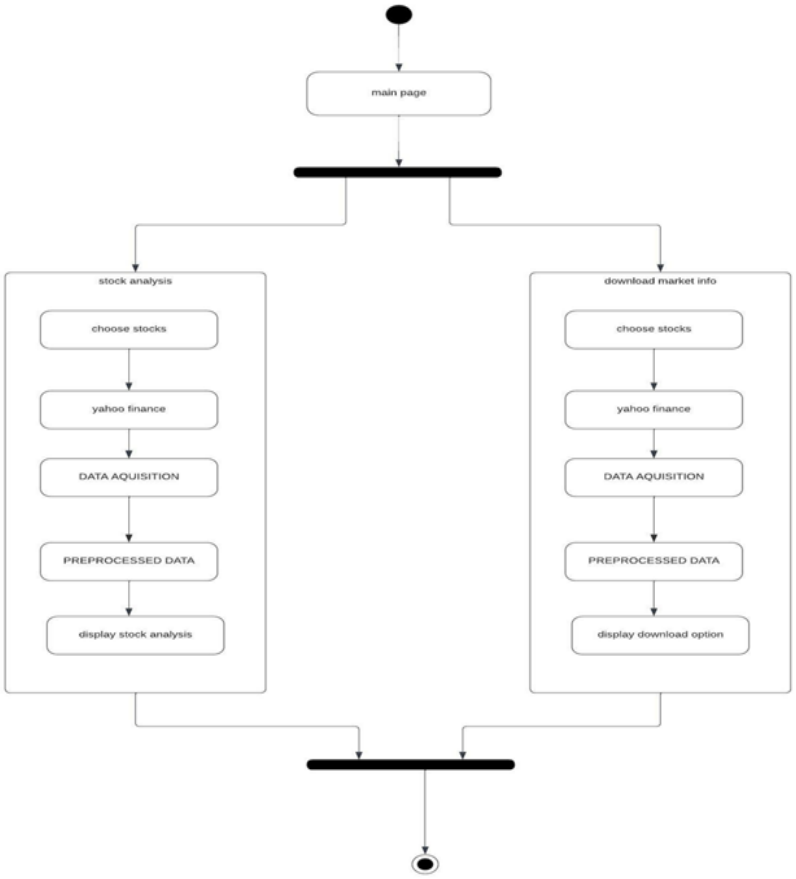
ACTOR'S GOALS: To check the sentiments analysis of tweets
PARTICIPATING ACTOR'S: USER
PRE-CONDITIONS: The user does not have the sentiment analysis of tweets.
POST – CONDITIONS- The user has the sentiments analysis of the tweets
SUCCESS SCENARIO:
1) Select the stock for which the user wants to see the sentiments analysis of the tweets
2) Check the sentiments analysis of the tweets

ACTOR'S GOALS: To check the stock prediction of a stock
PARTICIPATING ACTOR'S: USER
PRE-CONDITIONS: The user does not have the stock prediction of a stock.
POST – CONDITIONS- The user has the stock prediction of a stock
SUCCESS SCENARIO:
1) Select the stock for which the user wants to see the stock prediction
2) Check the stock prediction of a stock

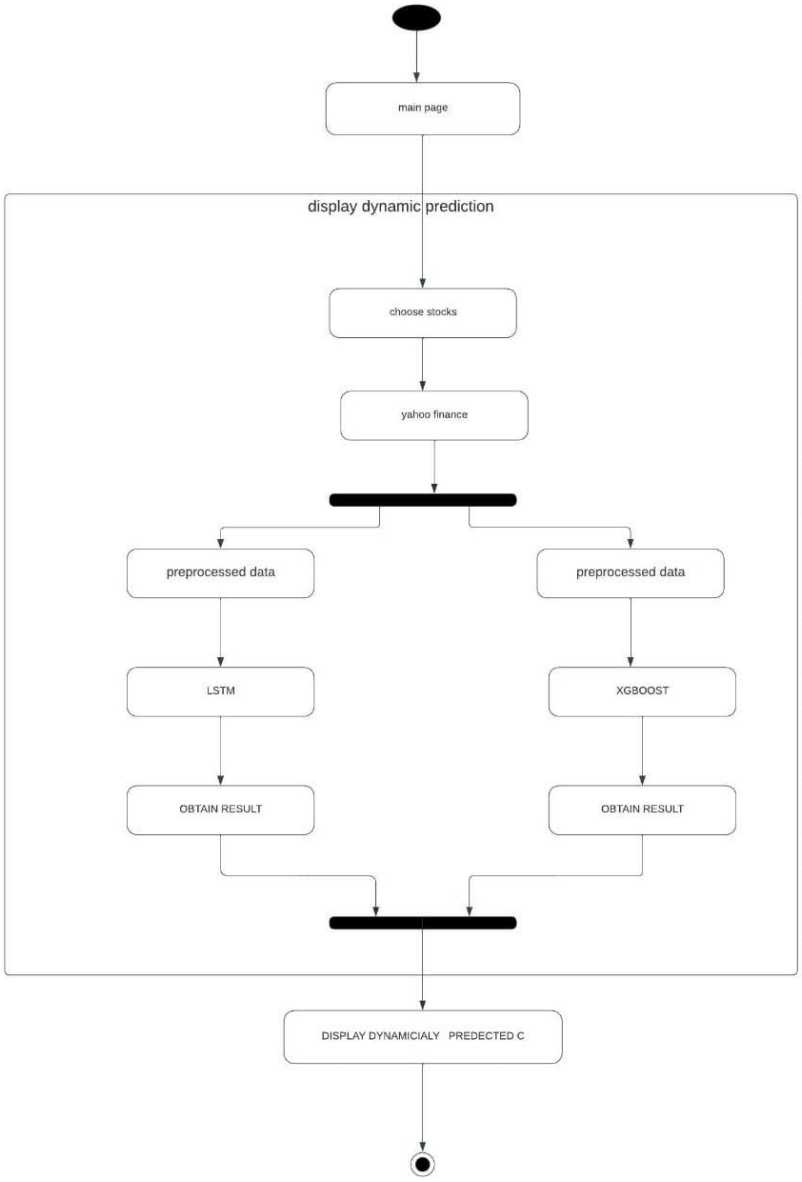
6.5 Activity diagrams for critical use cases

Activity diagrams are used in this section to describe the process in critical use cases.

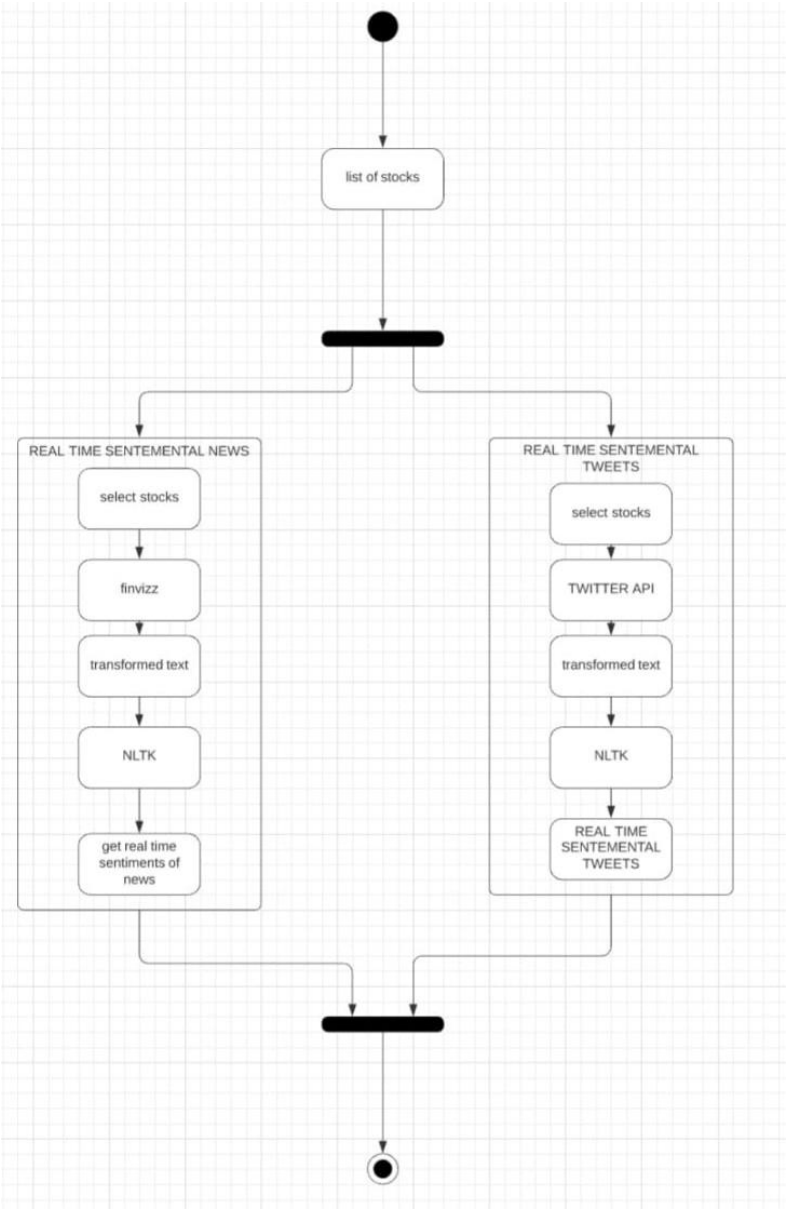
6.5.1 Activity diagram for use case - Download stock data and Get stock technical



6.5.2 Activity diagram for use case - Get prediction



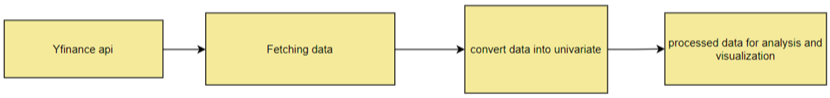
6.5.3 Activity diagram for use case – Get sentements of news and sentiments of tweets



7. Software product design

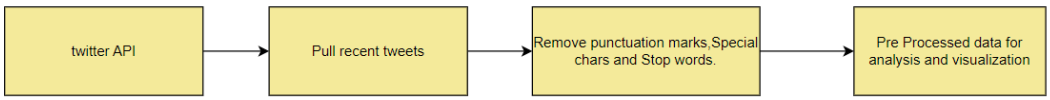
In this project, we are developing a system for predicting stock prices that uses recurrent neural networks (RNN), long short-term memory (LSTM) algorithms, and XGBoost for prediction. Additionally, We are also incorporating sentiment analysis based on tweets and financial news in our system.

7.1 Preprocessing of numerical data



Preprocessing of numerical data: This is a diagram that shows the data how numerical data is processed. We fetch the data from Yahoo finance using the yfinance API and once the data is fetched, the data is passed on into the training model and technical analysis is performed on the data that is fetched. As shown in the diagram below, data is fetched from Yfinance API and after fetching the data, the data is being sent for analysis and visualization.

7.2Preprocessing of textual data sentimental analysis based on tweets

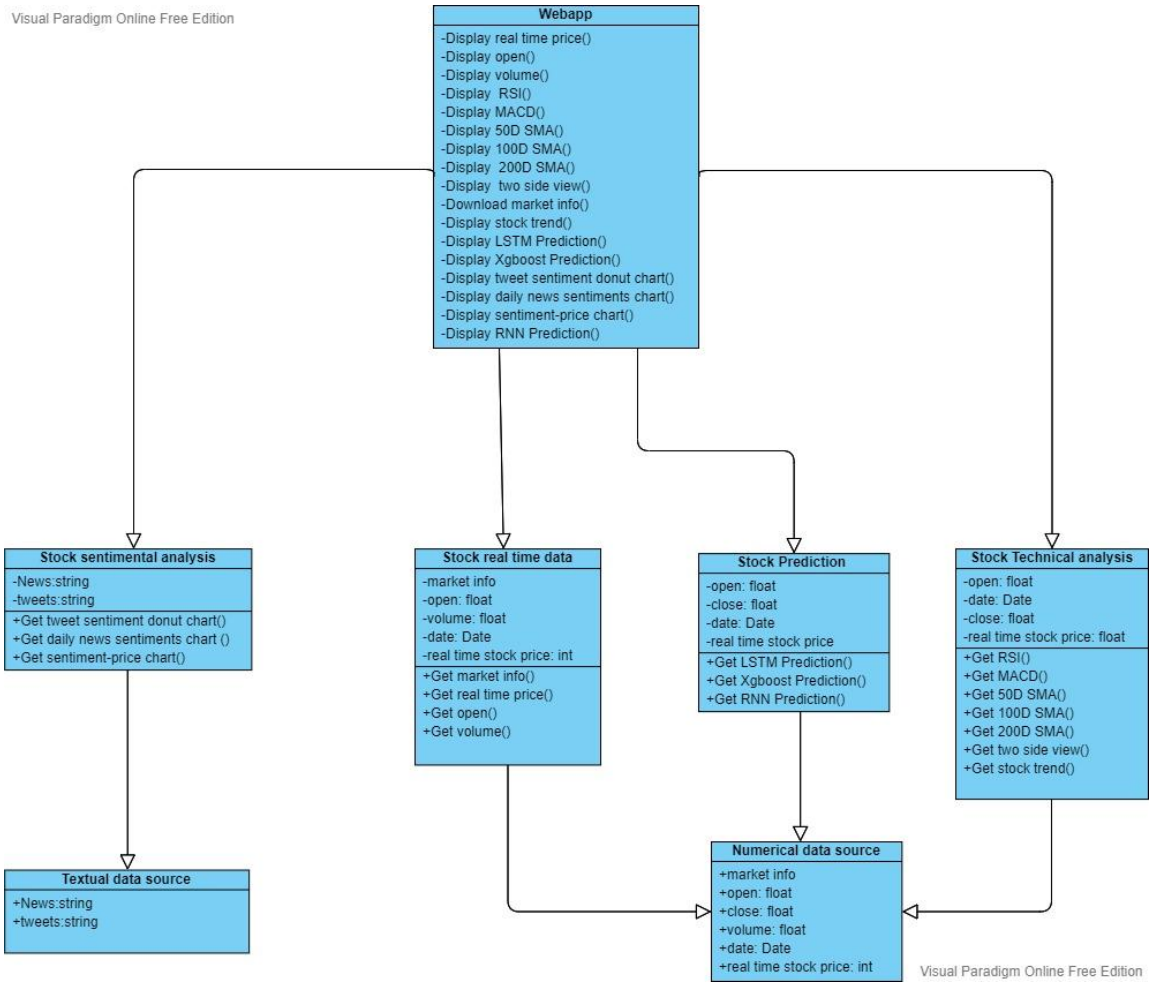


Preprocessing of textual data sentimental analysis based on tweets
This diagram describes the pre-processing of textual data. Tweets have been fetched from twitter using tweepy which is the twitter API. Once the tweets have been fetched, It removes the punctuation marks, special chars, and stop words. And once the tweets have been processed, the tweets are sent for analysis and visualization.

7.3 Preprocessing of textual data for sentimental analysis based on news

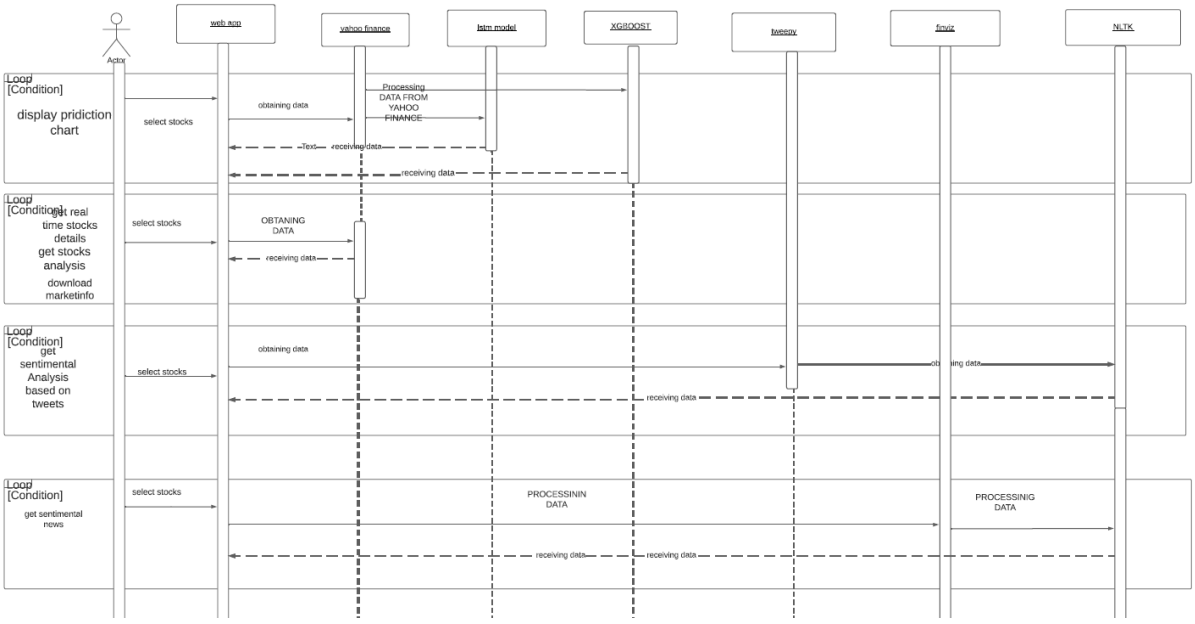
Preprocessing of textual data for sentimental analysis based on news:This diagram describes the pre-processing of news that is fetched from finviz. News is being fetched from finviz using the finviz API. Once the news has been fetched. And once the news has been processed, the scrapped news is sent for analysis and visualization.

7.4 Class diagram



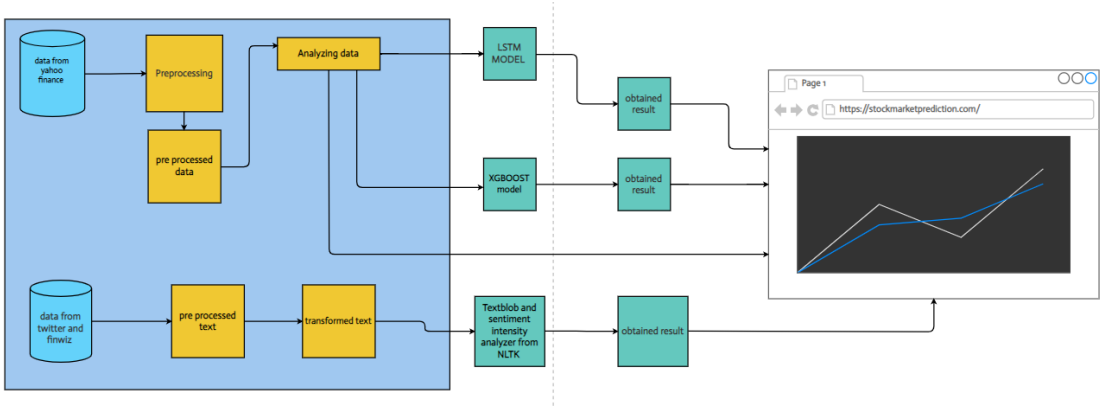
A class diagram in software engineering is a kind of static structure diagram that reveals the classes, properties, operations, and relationships between objects in a system to describe the system's structure. As shown in the class diagram below, we have created seven classes in our project. The main class in our diagram is the webapp which collects data from four different classes. The next class is the stock sentiment analysis class that fetches textual data from the textual data source class. From the numerical data source, data is fetched and real time data, stock prediction, stock technical analysis are displayed on the webapp.

7.5 Sequence diagram



A sequence diagram is a diagram created using the Unified Modeling Language (UML) that shows the flow of messages sent and received by objects during an interaction. A group of objects that are represented by lifelines and the messages they exchange over the course of an interaction makes up a sequence diagram.

7.6 System architecture

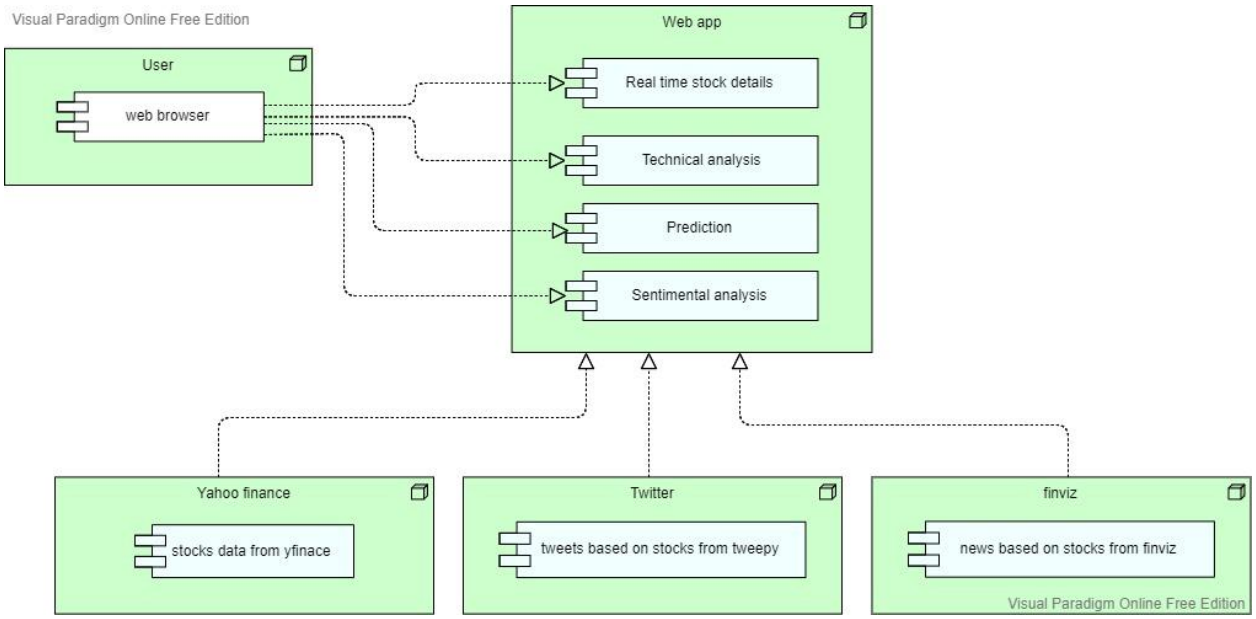


As seen in the figure above Yahoo Finance, Twitter, and finviz are the key three data sources used by the system. The system uses the yfinance api to retrieve real-time data from Yahoo Finance for technical analysis of a stock. The system retrieves real-time data from Yahoo Finance via the yfinance api, preprocesses it, and then passes it to the lstm and xgboost models to generate predictions. The system retrieves textual data from Finviz, preprocesses and transforms it, and then sends it to Textblob and Sentiment Intensity Analyzer which is part of nltk, which produces the sentiment analysis based on news obtained.The system retrieves textual data from twitter, using tweepy api and preprocesses and transforms it, and then sends it to Textblob and Sentiment Intensity Analyzer which is part of nltk, which produces the sentiment analysis based on tweets obtained.All of these collected data are sent to a web application so that users may see the outcomes.

7.7 Deployment diagram

Deployment diagrams in UML represent a system's physical architecture. The relationships between the system's hardware and software components as well as the physical distribution of the processing are displayed in deployment diagrams.

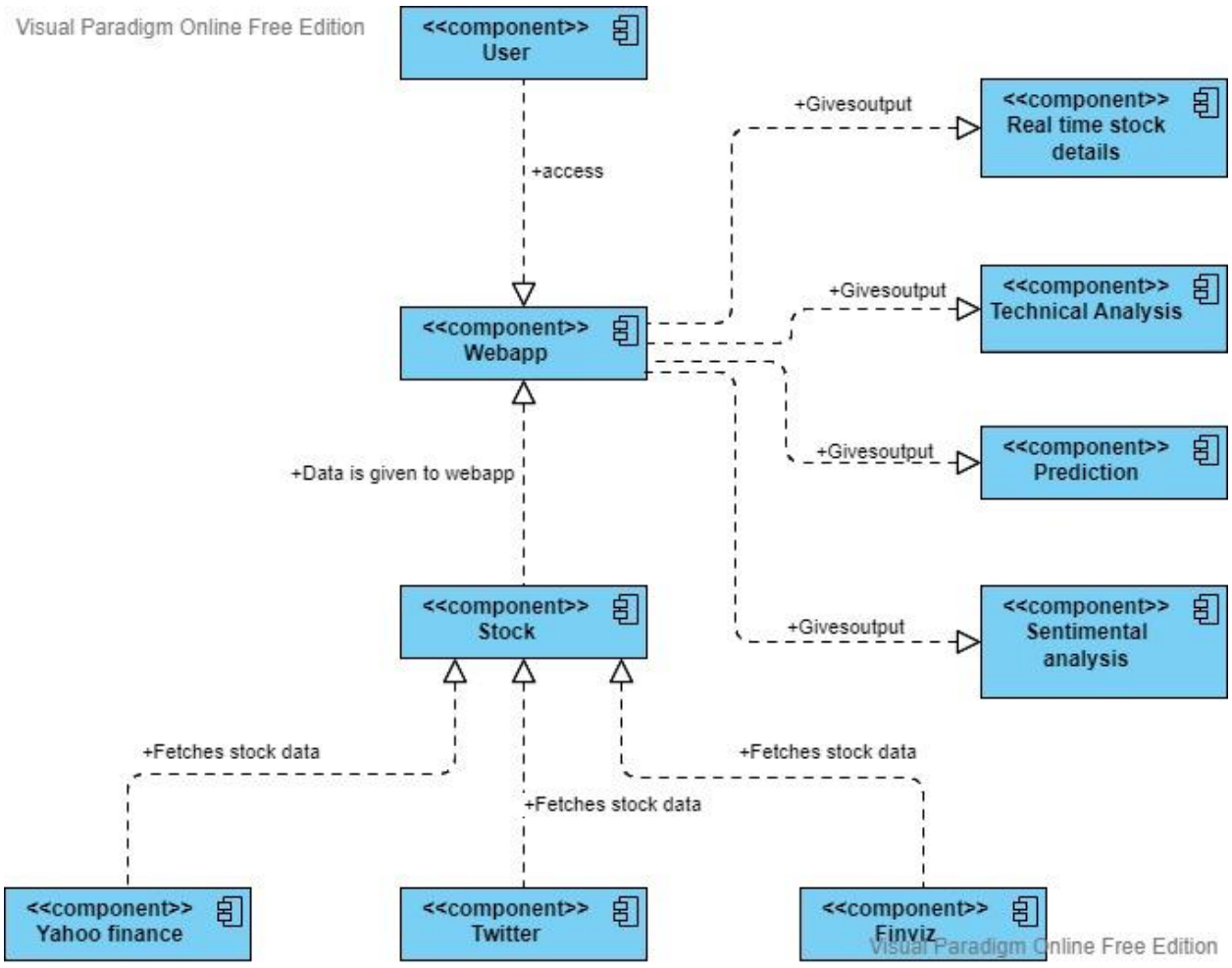
As you can see in the diagram below, data is fetched from Yahoo finance, finviz, and twitter and the fetched data is passed to the training models. And on processing the data the results are shown on the webapp in the form of analysis, visualization, prediction and the stock price of the selected stock.



7.8 Component diagram

A component diagram, often called a UML component diagram, shows how the physical parts of a system are wired up and organized. To model implementation specifics and ensure that all necessary functionalities of the system are covered by planned development, component diagrams are frequently created.

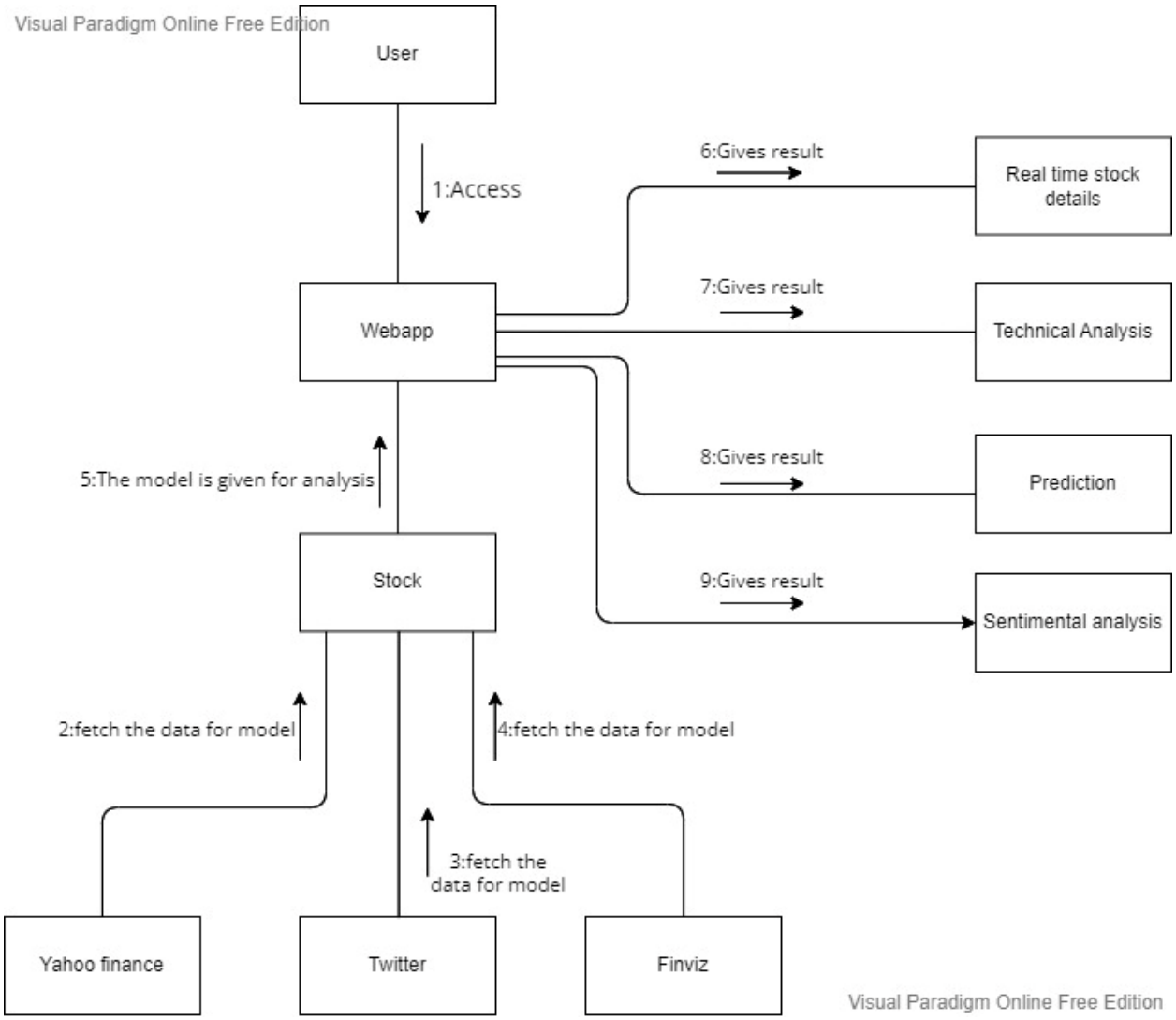
As you see in the diagram below when the user try to access the webapp, the data is fetched from three different components that are Yahoo finance, Twitter ,and finviz. Then the fetched data is passed on to the training model and the webapp shows the output to the user in three different components as stock prices, analysis and visualization and prediction of the selected stock.



7.9 Comunications diagram

In a similar way to a sequence diagram, a communication diagram depicts the interactions between elements at run-time. However, whereas sequence diagrams are better at depicting processing across time, communication diagrams are used to visualize interactions between objects.

As you can see in the diagram below, data is fetched from Yahoo finance, finviz, and twitter and the fetched data is passed to the training models. And on processing the data the results are shown on the webapp in the form of analysis, visualization, prediction and the stock price of the selected stock.



8. I mplementation:

This chapter covers the architecture of machine learning models we have used and In this chapter, an explanation of the implementation of the solution will be observed. How the application is built from scratch whether there is the frontend or the backend part of the application and all the processes of machine learning algorithms. In the chapters below we shall be going through each and everything in detail.

8.1 Architecture of machine learning models:

This subchapter covers the architecture of LSTM,RNN and Xgboost models

8.1.1 LSTM model:

To learn from time series data, recurrent neural networks (RNNs) of the Long Short-Term Memory (LSTM) type are particularly well adapted. It was developed to address the disappearing gradients problem, a common issue with vanilla RNNs that prevents these networks from learning long-term associations. The multiple gates in LSTM networks, which control the information flow across the network, enable the ability to selectively recall or forget knowledge from the past. They are thus well suited for tasks like speech recognition and language translation where a word's meaning may depend on the words that come before or after it. Deep learning architectures that can comprehend even more complex patterns in the data can be made by stacking LSTM networks. These designs are often trained using a sort of backpropagation via time.

The architecture of an LSTM network for stock prediction typically consists of the following components:

Input layer, which may also handle past prices, transaction volumes, and other significant attributes, is the stock data.

LSTM layer: The central layer of the LSTM network is in charge of processing input data and determining long-term dependencies between the inputs. The LSTM layer, which is frequently composed of several LSTM cells coupled in a certain way, makes up the LSTM network.

Output layer: This layer creates stock forecasts based on the data gathered by the LSTM layer. One neuron may be present in the output layer for regression tasks (which involve predicting a continuous value), however many neurons may be present for classification tasks.

Activation function: Each neuron's output from the output layer is subject to this function, which establishes a range for the output (e.g. 0 to 1 for binary classification tasks). Popular LSTM network activation functions include sigmoid, tanh, and ReLU.

Loss function: During training, the network weights are adjusted using this function to compute the difference between true and predicted values. Mean squared error (MSE) for regression tasks and cross-entropy loss for classification tasks are two common loss functions for LSTM networks.

In general, the architecture of an LSTM network for stock prediction is built to discover and capture the long-term dependencies in the data and produce precise forecasts based on this discovered knowledge.

8.1.2 XGboost model:

Making use of decision trees as its base model, XGBoost is a well liked and effective gradient boosting toolkit for machine learning. It offers a quick and precise method for training decision tree-based models on huge datasets

The core XGBoost method is built on decision trees, a type of supervised learning model that provides predictions based on the values of the input characteristics. Gradient boosting, a method for training many weak decision tree models progressively and combining their predictions to get a single, more accurate forecast, is used to train these decision trees in XGBoost. A weak decision tree model is first trained on the data to accomplish this, and the residual error—the discrepancy between the actual target values and the anticipated values—is then calculated using the model's predictions. To provide a more accurate forecast, the second decision tree model is merged with the first model and trained using the residual error. This process is done iteratively, with each consecutive tree trained on the residual error of the one that came before it. This technique enables XGBoost to build highly precise models with a massive number of parameters.

XGBoost includes a number of features in addition to its gradient boosting method that make it simple to use and enhance the models created. Parallel computing is supported, the stochastic gradient descent process is successfully implemented, and tools are offered for monitoring and optimizing the performance of the trained models.

The fundamental design of XGBoost involves a number of parts cooperating to provide predictions. These elements consist of:

Booster: which is the main element of XGBoost, is in responsible of creating the decision tree models and combining their projections to get the final prediction. Tree and Linear are the two booster types that XGBoost supports. The Tree booster use the gradient descent method to train decision trees, whereas the Linear booster creates linear regression models.

Objective function: This mathematical function describes the loss that XGBoost seeks to minimize when training the decision tree models. The objective function takes into account the precise sort of prediction problem being done, as well as any additional limits or regularization terms that are desired (such as regression or classification).

The Evaluation metric: Based on a holdout dataset, this statistic is used to assess how well the trained model performed. The performance of various models is evaluated using the evaluation measure, which also helps determine whether the training process has reached convergence.

The Base Learner: The decision tree model that XGBoost trained has the following appearance. The basis learner is trained on a subset of the data, and its predictions are combined with those of other base learners to get the final prediction. To control the model's complexity, the basic learner's parameters—such as the maximum tree depth and the bare minimum of samples per leaf—can be altered.

Overall, the architecture of XGBoost is designed to train decision tree models in a way that is efficient and scalable, and that is able to make accurate predictions on a wide range of problems.

8.1.3 RNN model:

Recurrent neural networks (RNNs), a subclass of artificial neural networks, are particularly well adapted to processing sequential input, including time series data or natural language. Unlike traditional feedforward neural networks, which take a fixed sized input and produce a fixed sized output, RNNs take a succession of inputs and produce a sequence of outputs. This gives them the ability to comprehend the relationships between sequence elements and use those relationships to predict the connections between incoming sequence sections.

The fundamental building blocks of an RNN are a number of repeating modules, or cells, each of which processes a single input element. Using the hidden state of the previous cell and the current input element as inputs, each cell generates an output and a new hidden state. The knowledge that an RNN has learned from the input sequence up to that point is stored in a fixed-sized vector called the hidden state. The information in the hidden state is kept up to date while the RNN analyzes each element of the input sequence, allowing it to learn long-term dependencies in the data.

Backpropagation through time (BPTT), RNNs can be trained using a backpropagation technique that unrolls the network in time and performs the standard backpropagation algorithm to each element of the input sequence. The network may eventually learn to base its predictions on the relationships between the sequence's components as a result. RNNs have been used in many different applications, such as speech recognition, language translation, and time series forecasting. Deep learning systems can be made by stacking them. An RNN's architecture for stock prediction typically includes the following elements:

Input layer: This layer processes the stock data, which could include include historical prices, trade volumes, and other important characteristics.

RNN layer: Processing input data and locating long-term relationships between the data are the responsibilities of the central layer of the RNN. The RNN layer is often made up of a large number of RNN cells connected in a certain arrangement.

Output layer: This layer generates stock projections using information learned from the RNN layer. While classification tasks may employ several neurons, regression activities (which involve predicting a continuous value) may only have a single neuron in the output layer (predicting a discrete class label).

Activation function: This function is applied to the output of each neuron from the output layer, which restricts the output to a specific range (e.g. 0 to 1 for binary classification tasks). RNNs frequently employ the sigmoid, tanh, and ReLU activation functions.

Loss function: This function calculates the difference between the true values and the expected values when updating the network weights during training. Two frequently used loss functions for RNNs are mean squared error (MSE) for regression tasks and cross-entropy loss for classification tasks.

Overall, the architecture of an RNN for stock prediction is designed to learn and capture the long-term dependencies in the data, and generate accurate predictions based on this learned information

8.2 Frontend documentation:

The front end documentation of the application is written as a client part for the sake of simplicity of reading the code for future developers and anyone can grasp the application very easily. Let’s go through each component of the application and try to understand the skeleton of the application then to summing up and observe the application as a whole.

```
You, 3 days ago | 1 author (You)
class StockApp:
    def __init__(self):
        self.path = StockDatapipeline.get_current_dir()
        self.settings = StockDatapipeline.load_settings(self.path)
        self.css = os.path.join(self.path, self.settings['css_file'])
        self.local_css(self.css)
        self.stock_list = self.settings['stock_list']      You, last week • Stock Marke

    @staticmethod
    def local_css(file_name):
        with open(file_name) as f:
            st.markdown(f"<style>{f.read()}</style>", unsafe_allow_html=True)
```

Fig 8.2.1 – source code for StockApp class initialization

In the Fig 8.2.1 The above code defines the class StockApp which has a local_css method that is used to apply custom CSS styles to the application, and an __init__ method that is run when an instance of the class is created.

```
def layout(self):
    stock = st.sidebar.selectbox(      You, last week • App 2.0 Final Version
        label='Select Stocks', options=self.stock_list)

    curr_price = StockDatapipeline(
        stock_ticker=stock).get_realtime_stock_price()

    curr_open = StockDatapipeline(
        stock_ticker=stock).get_current_stock_open_price()

    curr_vol = StockDatapipeline(stock_ticker=stock).get_stock_volume()

    b1, b2, b3 = st.columns(3)
    b1.metric("Stock Price", str(curr_price)+' USD')
    b2.metric("Open", str(curr_open)+' USD')
    b3.metric("Volume", str(curr_vol)+' USD')

    with st.container():
        selected = option_menu("Technical Analysis", ["", "RSI", 'MACD', '50D SMA', '100D SMA', '200D SMA'],
                               menu_icon="cast", orientation="horizontal", default_index=0)

        if selected == "RSI":
            st.pyplot(fig=StockDatapipeline(stock_ticker=stock).plot_rsi())
        if selected == "MACD":
            st.pyplot(fig=StockDatapipeline(
                stock_ticker=stock).plot_macd())
        if selected == "50D SMA":
            st.pyplot(fig=StockDatapipeline(
                stock_ticker=stock).plot_50_days_sma())
        if selected == "100D SMA":
            st.pyplot(fig=StockDatapipeline(
                stock_ticker=stock).plot_100_days_sma())
        if selected == "200D SMA":
            st.pyplot(fig=StockDatapipeline(
                stock_ticker=stock).plot_200_days_sma())
```

Fig 8.2.2 – source code for method layout part 1

The above code starts by creating a drop down menu using st.sidebar.selectbox that allows the user to select a stock from a list of options. Then, it uses the StockDatapipeline class to retrieve the current price, open price, and volume of the selected stock. These values are displayed using the st.columns and st.metric methods to create a table like layout. Next, the method creates an option menu using the option_menu function (from the streamlit_option_menu module) that allows the user to select a technical analysis to view. Depending on the selection, the method uses the StockDatapipeline class to plot the selected analysis using st.pyplot.

```
63          with st.sidebar:
64              col1, col3 = st.columns(2)
65              with col1:
66                  if st.button(label='Download Stock Data'):
67                      StockDatapipeline(
68                          stock_ticker=stock).download_stock_ticker_data()
69                      st.write('## Stock Data Downloaded')
70
71              with col3:
72                  if st.button(label="Download Stock Info",):
73                      StockDatapipeline(
74                          stock_ticker=stock).download_market_info()
75                      st.write('## Market Info Downloaded')
76
77          with st.container():
78              st.markdown('### Stock Price Chart')
79              st.pyplot(fig=StockDatapipeline(
80                  stock_ticker=stock).plot_trend())
81
82          c1, c2 = st.columns((5, 5))
83          with c1:
84              st.markdown('### Two Side View')
85              st.pyplot(fig=StockDatapipeline(
86                  stock_ticker=stock).plot_two_side_view())
87
88          with c2:
89              st.markdown('### Stocks Tweets Sentiment Overview')
90              st.pyplot(fig=StockNews(
91                  stock_ticker=stock).plot_tweet_sentiment_donut_chart())
92
93          if st.container():
94              st.markdown('### Daily Financial News Sentiments')
95              st.pyplot(fig=StockNews(
96                  stock_ticker=stock).plot_daily_sentiment_barchart())
97          else:       You, 3 days ago • App 2.0
98              st.error(f'cannot plot chart')
```

Fig 8.2.3 – source code for method layout part 2

The code shown above stats with creating two buttons in the sidebar that allow the user to download stock data and market information. When these buttons are clicked, the StockDatapipeline class is used to download the data and a message is displayed indicating that the download was successful.

Next, the class creates several plots using the StockDatapipeline and StockNews classes. These plots include a stock price chart, a two-side view chart, and a chart showing sentiment in tweets and news articles. The plots are displayed using the st.pyplot method.

Finally, the method includes an if statement that checks whether a chart can be plotted or not. If not, an error message is displayed using the st.error method.

```
with st.container():
    st.markdown('### LSTM Predictions')
    st.pyplot(fig=LSTM_Model(
        stock_ticker=stock).plot_lstm_prediction())

with st.container():
    st.markdown('### XGBoost Predictions')
    st.pyplot(fig=XGBoostModel(
        stock_ticker=stock).plot_xgboost_prediction())

with st.container():
    st.markdown('### RNN Predictions')
    st.pyplot(fig=RNN_Model(stock_ticker=stock).plot_rnn_prediction())
```

Fig 8.2.4 - source code for method layout part 3

The StockApp class creates a plot using the LSTM, XGBoost and RNN classes. These containers are used to generate predictions using their respective machine learning models, and the resulting predictions are plotted using the st.pyplot method. The plots are labelled with the name of the model that was used to generate the predictions.

8.3 Backend documentation

The Backend documentation of the application is written as a client part for the sake of simplicity of reading the code for future developers and anyone can grasp the application very easily. Let's go through each classes present in the application.

StockDataAnalyzer

```python
@staticmethod
def get_stock_data_from_ticker(stock_ticker):
    start = "2017-01-01"
    end = datetime.today().strftime('%Y-%m-%d')
    data = yf.download(stock_ticker, start, end)
    data.interpolate(
        method="linear", limit_direction="backward", inplace=True)
    data.reset_index(inplace=True)
    data.rename(columns={'index': 'Date'}, inplace=True)
    return data
```

Fig 8.3.1- Source code for General methods for retrieving and storing data

In the Fig 8.2.1 above the Code shows how we created a function that interacts with Yahoo's Yfinance API, which contains a feature from yahoo finance api. When you use the function yf.download(stock_ticker), you may download stock data into a Pandas dataframe. Sometimes there are missing values in the data, therefore I've used a pandas method to interpolate to handle them. By using linear interpolation in a backwards manner, we can replace missing values with close numbers.

```python
def get_realtime_stock_price(self) -> BeautifulSoup:
    """sending get request and retrieving html table using requests and beautiful soup,
    finding price class using beautiful soup and passing html address for price
    """
    self.logger.info(
        f"Getting Realtime Stock Price for {self.stock_ticker}")
    try:
        url = f"{self.settings['yfinance_url']}/{self.stock_ticker}"
        page = requests.get(url)
        soup = BeautifulSoup(page.text, "lxml")
        price = soup.find(
            self.settings["html_address"], class_=self.settings["html_class"]
        ).text
        self.logger.info(
            f"Retrived Realtime Price Successfully {self.stock_ticker}: {price}"
        )
        return float(price)
    except Exception as e:
        self.logger.error(f"Cannot Get Realtime Stock Price for: {e}")
```

Fig 8.3.2 – Source code for real time stock price

In the Fig 8.3.2 above the Code shows The get real time stock price method shown in the above image allows you to web scrape any stock's real-time stock price. It is implemented using requests and BeautifulSoup, which allow you to enable connection with the website and read html data, respectively. It lets user see the current and real time stock price.

```
def download_market_info(self) -> None:
    """It let's you download market info in csv file"""
    self.logger.info(f"Downloading Market info for {self.stock_ticker}")
    try:
        sec = yf.Ticker(self.stock_ticker)

        data = sec.history()
        # print(data.head())

        my_max = data["Close"].idxmax()
        my_min = data["Close"].idxmin()
        list_data = [
            data,
            my_max,
            my_min,
            sec.info,
            sec.isin,
            sec.major_holders,
            sec.institutional_holders,
            sec.dividends,
            sec.splits,
            sec.actions,
            sec.calendar,
            sec.recommendations,
            sec.quarterly_earnings,
            sec.earnings,
            sec.financials,
            sec.quarterly_financials,
            sec.balance_sheet,
            sec.quarterly_balance_sheet,
            sec.cashflow,
            sec.quarterly_cashflow,
            sec.sustainability,
            sec.options,
        ]       You, 4 days ago • Minor Changes
```

```
                sec.options,
            ]
            with open(f"{(self.stock_ticker).upper()}_Market_Info.csv", "w") as f:
                writer = csv.writer(f, delimiter="\t")
                writer.writerows(zip(list_data))
            self.logger.info("Market Info Download Successfully")
        except Exception as e:
            self.logger.error(f"Cannot Download Market Data: {e}")
```

Fig 8.3.3 - Source code for download market info

In the Fig 8.3.3 above the Code shows Users can download market data information about stocks using the way that is shown above. With this method, we are able to retrieve a comprehensive number of items, including the top stockholders and investors as well as quarterly income and revenue, dividends, stock split.

```
def download_stock_ticker_data(self) -> None:
    """It let's you download stock ticker data in Excel File"""
    try:
        self.logger.info(
            f"Downloading Stock Ticker Data {self.stock_ticker}")
        data = self.data
        data["Date"] = pd.to_datetime(data["Date"]).dt.date
        writer = pd.ExcelWriter(
            f"{(self.stock_ticker).lower()}_stock_data.xlsx")
        data.to_excel(writer, sheet_name="stock_data",
                      index=False, na_rep="NaN")
        writer.sheets["stock_data"].set_column(0, 5, 15)
        writer.save()
        self.logger.info(f"Stock Data Downloaded Successfully")
    except Exception as e:
        self.logger.error(f"Cannot Download stock ticker data: {e}")
```

Fig 8.3.4 - Source code for download stock data

Based on previously implemented retrieve stock ticker data, the above Fig 8.3.4 demonstrates method which allows users to download stock data into an Excel file.

```python
def get_current_stock_open_price(self) -> BeautifulSoup:
    """sending get request and retrieving html table using requests and beautiful soup,
    finding price class using beautiful soup and passing html address for open price
    """
    self.logger.info(f"Getting Current Open Price for {self.stock_ticker}")
    try:
        url = f"{self.settings['yfinance_url']}/{self.stock_ticker}"
        page = requests.get(url)
        soup = BeautifulSoup(page.text, "lxml")
        price = soup.find(
            self.settings["open_address"], class_=self.settings["open_class"]
        ).text
        self.logger.info(
            f"Retrived Open Price Successfully {self.stock_ticker}: {price}"
        )
        return price
    except Exception as e:
        self.logger.error(f"Cannot Get Open Price for: {e}")
```

Fig 8.3.5- Source code for opening price

In the Fig 8.3.5 above the Code shows The function is implemented similarly to get realtime stock price,so that it will retrieve the day open price for any stock, which is crucial information in stock trading.

```python
def get_stock_volume(self) -> BeautifulSoup:
    """sending get request and retrieving html table using requests and beautiful soup,
    finding price class using beautiful soup and passing html address for open price
    """
    self.logger.info(f"Getting Stock Volume for {self.stock_ticker}")
    try:
        url = f"{self.settings['yfinance_url']}/{self.stock_ticker}"
        df = pd.read_html(url)
        df = df[0]
        volume = df[1].iloc[6]
        return volume
    except Exception as e:
        self.logger.error(f"Cannot Get Stock Volume: {e}")
```

Fig 8.3.6- Source code for volume

In the Fig 8.3.6 above the Code shows The method get stock volume Despite the fact that we are web scraping this using the pandas.html read method, the html table that we are web scraping it is not

similar to previously implemented get stock realtime and get open price method. Stock volume is also an important KPI for stock trading, it lets users see how much people are buying and selling particular stock.

```
def __plot_trend(self, df, x, y, title=f"", xlabel='Date', ylabel='Stock price', dpi=200):
    fig = plt.figure(figsize=(10, 2), dpi=dpi)
    plt.plot(x, y, color='tab:Red')
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)
    plt.legend([self.stock_ticker])
    plt.ion()
    return fig

def plot_trend(self):
    df = self.data
    return self.__plot_trend(df, x=df['Date'], y=df['Close'],
                             title=f'Trend and Seasonality {self.stock_ticker}')
```

Fig 8.3.7 - Source code for treand seasonality chart

The code above defines a function **__plot_trend** that creates a line plot of the given data. It takes the following arguments:

- **df**: a dataframe containing the data to plot
- **x**: the data to use as the x-axis values
- **y**: the data to use as the y-axis values
- **title**: the title of the plot
- **xlabel**: the x-axis label
- **ylabel**: the y-axis label
- **dpi**: the resolution of the plot

The function creates a new figure using **matplotlib.pyplot** and sets the figure size and resolution using the **dpi** argument. It then plots the data using the **x** and **y** arguments and sets the title, x-axis label, and y-axis label using the corresponding arguments. It sets the legend to show the value of **self.stock_ticker** and enables interactive mode using **plt.ion()**. Finally, the function returns the **fig** object.

The **plot_trend** function simply calls **__plot_trend** with the data from **self.data** and the appropriate x-axis and y-axis values. It returns the resulting figure object trend and seasonality

```
def plot_two_side_view(self):
    df = self.data
    x = df['Date'].values
    y1 = df['Close'].values
    fig, ax = plt.subplots(1, 1, figsize=(10, 9.5), dpi=120)
    plt.fill_between(x, y1=y1, y2=-y1, alpha=0.5,
                     linewidth=2, color='seagreen')
    #plt.ylim(-800, 800)
    plt.title(f'{self.stock_ticker} (Two Side View)', fontsize=16)
    plt.hlines(y=0, xmin=np.min(df['Date']),
               xmax=np.max(df['Date']), linewidth=.5)
    return fig      You, 3 days ago • update 1.00 …
```

Fig 8.3.8 – source code for Time Series Analysis

In the Fig 8.3.8 above the Code shows that As it is evident that the data we are retrieving is in daily time intervals and may contain recurring patterns annually, we built the above-described solution to alleviate this problem and allow users to compare side by side patterns year wise.

```
def __get_ema(self):
    df = self.data
    df['SMA_50'] = df['Close'].rolling(50).mean().shift()
    df['SMA_100'] = df['Close'].rolling(100).mean().shift()
    df['SMA_200'] = df['Close'].rolling(200).mean().shift()
    return df
```

Fig 8.3.9 - source code for calculating simple moving average

In the Fig 8.3.9 above the Code shows a method in which A pandas dataframe with all moving averages generated is produced by applying the pandas rolling method and taking the average of time intervals to calculate the moving average over three different time intervals: 50 days, 100 days, and 200 days.

```python
@staticmethod
def __plot_sma(df, stock_ticker, sma, days):
    df.set_index(df['Date'], inplace=True)
    fig = plt.figure(figsize=(7, 2))
    plt.plot(df['Close'], 'k-', label='Original')
    plt.plot(df[f'{sma}'], 'r-', label='Running average')
    plt.ylabel('Price')
    plt.xlabel('Date')
    plt.grid(linestyle=':')
    plt.fill_between(df.index, 0, df[f'{sma}'], color='r', alpha=0.1)
    plt.legend(loc='upper left')
    plt.title(f'{stock_ticker} {days} Days Simple Moving Average')
    # plt.show()
    return fig
```

```python
def plot_100_days_sma(self):
    df = self.__get_ema()
    return self.__plot_sma(df=df, stock_ticker=self.stock_ticker,
                           sma='SMA_100', days=100)

def plot_200_days_sma(self):
    df = self.__get_ema()
    return self.__plot_sma(df=df, stock_ticker=self.stock_ticker,
                           sma='SMA_200', days=200)

def plot_50_days_sma(self):
    df = self.__get_ema()
    return self.__plot_sma(df=df, stock_ticker=self.stock_ticker,
                           sma='SMA_50', days=50)
```

Fig 8.3.10 - source code for plotting simple moving average,50,100,200 days moving average

In the Fig 8.3.10 above the Code shows a method in which the moving average was calculated using the previous method; the moving average will now be visualised using the newly implemented method. The newly implemented method shown above uses a dataframe, column, stock ticker, and days as parameters to visualise each and every moving average that was previously calculated. And above the Code also shows how the functions plot_100_days_sma, plot_200_days_sma, and plot_50_days_sma functions call __plot_sma with the appropriate data and arguments to plot the SMA of the given number of days. They return the resulting figure object.

```
@staticmethod
def __relative_strength_idx(df, n):
    close = df['Close']
    delta = close.diff()
    delta = delta[1:]
    pricesUp = delta.copy()
    pricesDown = delta.copy()
    pricesUp[pricesUp < 0] = 0
    pricesDown[pricesDown > 0] = 0
    rollUp = pricesUp.rolling(n).mean()
    rollDown = pricesDown.abs().rolling(n).mean()
    rs = rollUp / rollDown
    rsi = 100.0 - (100.0 / (1.0 + rs))
    return rsi
```

Fig 8.3.11 - source code for RSI

In the Fig 8.3.11 above the code is trying to find the relative strength index of a stock.
The code starts by finding the close price for each day in the data set and then subtracting it from today's close price. This gives us a list of prices that went up or down over time. Next, we take these lists and calculate their rolling averages using mean(). Finally, we divide this number by 100 to get an RSI value between 0 and 100 where 0 is no change in price and 100 is complete change in price. The code first finds all days with positive changes in Close Price (pricesUp) which are less than zero (pricesDown).Then it calculates the rolling average of these values (rollUp) as well as calculating its absolute value (rollDown).It then divides rollUp/rollDown to get an RSI value between 0-100 where 0 is no change in price and 100 is complete change in price. The code calculates the close of a stock and then divides it by its delta to get the prices up and down. The prices up and down are then averaged to get the rolling mean, which is then divided by 100 to get the relative strength index.

```
def plot_rsi(self):
    df = self.data
    df['RSI'] = self.__relative_strength_idx(df=df, n=14)
    fig = plt.figure(figsize=(7, 2))
    plt.plot(df['Date'], df['RSI'], label="RSI")
    plt.ylabel('RSI')
    plt.xlabel('Date')
    plt.grid(linestyle=':')
    plt.legend(loc='upper left')
    plt.title(f'{self.stock_ticker} Relative Strength Index')
    return fig
```

Fig 8.3.12 - source code for plotting RSI

In the Fig 8.3.12 above the code above creates a line chart for the previously generated relative strength index. This index is very useful for traders to determine if a company stock is overbought or oversold using matplotlib.

```
@staticmethod
def __get_macd_and_ema(df):
    EMA_12 = pd.Series(df['Close'].ewm(span=12, min_periods=12).mean())
    EMA_26 = pd.Series(df['Close'].ewm(span=26, min_periods=26).mean())
    df['MACD'] = pd.Series(EMA_12 - EMA_26)
    df['MACD_signal'] = pd.Series(
        df.MACD.ewm(span=9, min_periods=9).mean())
    return df       You, 7 days ago • New Class and Functions
```

Fig 8.3.13 - source code for MACD

In the Fig 8.2.13 above the code calculates the moving average of these prices using an exponential moving average (EMA) and plots them on a chart. The code then calculates two different indicators: MACD and MACD_signal. The first is calculated using EMA_12 - EMA_26 which is plotted on a chart as well. The second indicator uses df['MACD'] which is calculated by taking the difference between EMA_12 and EMA_26 and plotting that value over time on a chart as well.
The code is used to calculate the moving average of the MACD.
The code above returns a pandas dataframe with two columns, 'MACD' and 'MACD_signal'

```python
def plot_macd(self):
    df = self.__get_macd_and_ema(self.data)
    fig = plt.figure(figsize=(7, 2))
    plt.plot(df['Date'], df['MACD'], label='MACD')
    plt.plot(df['Date'], df['MACD_signal'], label='MACD Signal')
    plt.xlabel('Date')
    plt.grid(linestyle=':')
    plt.legend(loc='upper left')
    plt.title(f'{self.stock_ticker} MACD and MACD Signal')
    return fig
```

Fig 8.3.14 - source code for plotting MACD

Using the previously implemented get macd and ema method, which returns a macd and macd signal, we are able to draw a line chart for the macd and macd signal using above shown code in Fig 8.3.14

StockNewsAnalyzer.py

```python
You, 2 days ago | 1 author (You)
class StockTweets:
    def __init__(self, stock_ticker):
        self.stock_ticker = stock_ticker
        load_dotenv(find_dotenv())        You, 2 weeks ago • Improved Sentiment
        self.path = StockDatapipeline.get_current_dir()
        self.settings = StockDatapipeline.load_settings(self.path)
        self.logger = StockDatapipeline.app_logger(self.settings["app_name"])
        self.bearer_token = os.environ["bearer_token"]
        self.api_key = os.environ["api_key"]
        self.api_key_secrets = os.environ["api_key_secret"]
        self.access_token = os.environ["access_token"]
        self.access_token_secrets = os.environ["access_token_secret"]
```

Fig 8.3.15 - source code for class initialization for stocktweets

In the __init__ method, the class takes a stock_ticker parameter, specifies which stock ticker data to for. The method then sets up some instance variables for working with the stock data. And load environment variables using the find_dotenv and load_dotenv methods from the dotenv package, and then uses the StockDatapipeline.get_current_dir and StockDatapipeline.load_settings methods to set the path and settings for the class. It also sets up a logger using the StockDatapipeline.app_logger

method. Finally, it sets up several instance variables for storing authentication information, such as the bearer_token, api_key, and access_token.

```
@classmethod
def twitter_auth(cls, access_token, access_token_secret, api_key_secret, api_key):
    try:
        auth = tweepy.OAuth1UserHandler(
            access_token, access_token_secret, api_key_secret, api_key
        )
        api = tweepy.API(auth)
        return api
    except Exception as e:
        raise (f"Cannot Authenticate you: {e}")
```

Fig 8.3.16 - source code for twitter authentication

The twitter_auth method is a method for authenticating with the Twitter API using the tweepy package. It takes four parameters: access_token, access_token_secret, api_key_secret, and api_key. These parameters are used for authenticating with the Twitter API. The method then creates an instance of the tweepy.API class, passing the parameters, and returns the instance. If an error occurs during authentication, the method raises an exception with an error message.

```
@classmethod
def client_auth(cls, bearer_token):
    try:
        return tweepy.Client(bearer_token)
    except Exception as e:
        raise (f"Cannot Authenticate Client: {e}")
```

Fig 8.3.17 - source code for client side authentication

The client_auth is a method for creating an instance of the tweepy.Client class. It takes a single parameter, bearer_token, which is used to create the tweepy.Client object. The method then returns the instance of the Client class. If an error occurs during authentication, the method raises an exception with an error message.

```python
@staticmethod
def preprocess_tweet(sen):
    """Cleans text data up, leaving only 2 or more char long non-stepwords composed of A-Z & a-z only
    in lowercase"""

    sentence = sen.lower()
    sentence = re.sub("RT @\w+: ", " ", sentence)
    sentence = re.sub(
        "(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", sentence
    )
    sentence = re.sub(r"\s+[a-zA-Z]\s+", " ", sentence)
    sentence = re.sub(r"\s+", " ", sentence)
    return sentence
```

Fig 8.3.18 - source code for preprocessing tweets

The preprocess_tweet method is a method for cleaning up and preprocessing tweet text data. It takes a single parameter sen, which represents the sentence to be preprocessed. The method first converts the sentence to lowercase and then uses regular expression patterns to remove certain substrings from the sentence. It also uses regular expression patterns to replace multiple consecutive whitespace characters with a single space. Finally, it returns the preprocessed sentence.

```python
def get_tweets_df(self):
    try:
        self.logger.info(f"Retrieving Tweets for {self.stock_ticker}")
        tweet_list = []
        query = f"#{self.stock_ticker} -is:retweet lang:en"
        paginator = tweepy.Paginator(
            self.__class__.client_auth(
                self.bearer_token).search_recent_tweets,
            query=query,
            max_results=100,
            limit=5,
        )
        for tweet in paginator.flatten():
            tweet_list.append(tweet)
        tweet_list_df = pd.DataFrame(tweet_list, columns=["text"])
        return tweet_list_df
    except Exception as e:
        self.logger.error(f"Cannot find any tweet for {self.stock_ticker}")
```

Fig 8.3.19 - source code for get tweet

The get_tweets_df method is an instance method of the StockTweets class. This method is used for retrieving a list of tweets for a particular stock and returning them as a Pandas DataFrame.

The method begins by logging a message that it is retrieving tweets for the stock specified by the stock_ticker instance variable. It then creates an empty list called tweet_list and constructs a query string for the Twitter API using the stock_ticker instance variable. The query string is used to search for tweets that contain the stock ticker, are not retweets, and are written in English.

The method then creates a tweepy.Paginator object, passing in the client_auth class method, the query string, and the maximum number of results to retrieve and the number of results per page. The method then iterates over the paginator, appending each tweet to the tweet_list list.

Once all the tweets have been retrieved and added to the list, the method creates a DataFrame from the list, using the text column to store the text of each tweet. Finally, the method returns the DataFrame. If an error occurs, the method logs an error message and returns nothing.

```
def get_cleaned_tweets(self):
    try:
        self.logger.info(f"Cleaning tweets for {self.stock_ticker}")
        cleaned_tweets = []
        tweet_list_df = self.get_tweets_df()
        tweet_list_df["cleaned"] = tweet_list_df["text"].apply(
            self.__class__.preprocess_tweet
        )
        # print(tweet_list_df.head())
        return tweet_list_df      You, 4 days ago • App 4.0
    except Exception as e:
        self.logger.error(
            f"Cannot clean any tweets for {self.stock_ticker}")
```

Fig 8.3.20 - source code for cleaning tweets

The get_cleaned_tweets method is an instance method of the StockTweets class. This method is used for retrieving a list of tweets for a particular stock, cleaning the tweets using the preprocess_tweet static method, and returning the cleaned tweets as a Pandas DataFrame.

The method begins by logging a message that it is cleaning tweets for the stock specified by the stock_ticker instance variable. It then creates an empty list called cleaned_tweets and retrieves the list of tweets for the stock by calling the get_tweets_df instance method.

The method then adds a new column to the DataFrame called cleaned, which contains the preprocessed version of the text from the text column. The preprocessing is done using the preprocess_tweet static method. Finally, the method returns the DataFrame with the cleaned tweets. If an error occurs, the method logs an error message and returns nothing.

```
    @staticmethod
    def get_polarity(df, column):
        df[["polarity", "subjectivity"]] = df[column].apply(
            lambda Text: pd.Series(TextBlob(Text).sentiment)
        )

        return df       You, 5 days ago • App 3.0 Refractor

    def get_tweets_polarity(self):
        try:
            tweet_list_df = self.get_cleaned_tweets()
            tweet_list_df = self.get_polarity(tweet_list_df, "cleaned")

            return tweet_list_df
        except Exception as e:
            self.logger.error(f"{e}")
```

Fig 8.3.21 - source code for getting polarity score out of tweets

The get_tweets_polarity method is an instance method of the StockTweets class. This method is used for retrieving a list of cleaned tweets for a particular stock, and adding columns to the DataFrame to store the polarity and subjectivity of each tweet.

The method begins by calling the get_cleaned_tweets instance method to get the DataFrame of cleaned tweets. It then calls the get_polarity static method, passing in the DataFrame and the name of the column containing the cleaned tweets. The get_polarity method adds two new columns to the DataFrame, called polarity and subjectivity, which contain the polarity and subjectivity of each tweet, respectively. These values are calculated using the TextBlob package.

Finally, the get_tweets_polarity method returns the modified DataFrame with the added columns for polarity and subjectivity. If an error occurs, the method logs an error message and returns nothing.

```
@staticmethod
def get_sentiments(df, column):
    try:
        for index, row in df[column].iteritems():
            score = SentimentIntensityAnalyzer().polarity_scores(row)
            neg = score["neg"]
            neu = score["neu"]
            pos = score["pos"]
            comp = score["compound"]
            if comp <= -0.05:
                df.loc[index, "sentiment"] = "negative"
            elif comp >= 0.05:
                df.loc[index, "sentiment"] = "positive"
            else:
                df.loc[index, "sentiment"] = "neutral"
            df.loc[index, "neg"] = neg       You, 4 days ago • App 2.0
            df.loc[index, "neu"] = neu
            df.loc[index, "pos"] = pos
            df.loc[index, "compound"] = comp
        # print(tweet_list_df.head())
        return df
    except OSError:
        raise RuntimeError("Unable to Load df")
```

Fig 8.3.22 - source code for assigning sentiments based on polarity score

The get_sentiments method is a static method of the StockTweets class.This method is used for adding columns to a DataFrame containing tweets, which store the sentiment, negativity, neutrality, positivity, and compound score of each tweet.

The method takes two parameters: df, which is the DataFrame containing the tweets, and column, which is the name of the column in the DataFrame that contains the tweets. The method then iterates over each row in the column of the DataFrame, using the SentimentIntensityAnalyzer from the nltk package to calculate the sentiment, negativity, neutrality, positivity, and compound score of the tweet.

Based on the compound score, the method assigns each tweet a sentiment label of "positive", "negative", or "neutral". It then adds new columns to the DataFrame for the negativity, neutrality, positivity, and compound score, and assigns the calculated values to these columns for each tweet. Finally, the method returns the modified DataFrame with the added columns. If an error occurs, the method raises a RuntimeError with an appropriate error message.

```
def get_tweets_sentiments(self):
    try:
        tweet_list_df = self.get_tweets_polarity()
        tweet_list_df = self.get_sentiments(tweet_list_df, "cleaned")
        return tweet_list_df
    except Exception as e:      You, 4 days ago • App 2.0
        self.logger.error(f"{e}")
```

Fig 8.3.23 - source code for getting tweets sentiments

The get_tweets_sentiments method is an instance method of the StockTweets class. The method is used for retrieving a list of cleaned tweets for a particular stock, adding columns to the DataFrame to store the polarity and subjectivity of each tweet, and then adding further columns to the DataFrame to store the sentiment, negativity, neutrality, positivity, and compound score of each tweet.

The method begins by calling the get_tweets_polarity instance method to get the DataFrame of cleaned tweets with added columns for polarity and subjectivity. It then calls the get_sentiments static method, passing in the DataFrame and the name of the column containing the cleaned tweets. The get_sentiments method adds several new columns to the DataFrame, which contain the sentiment, negativity, neutrality, positivity, and compound score of each tweet. These values are calculated using the SentimentIntensityAnalyzer from the nltk package.

Finally, the get_tweets_sentiments method returns the modified DataFrame with the added columns for sentiment, negativity, neutrality, positivity, and compound score. If an error occurs, the method logs an error message and returns nothing.

```
def get_tweets_sentiments_percentage(self):
    try:
        tweet_list_df = self.get_tweets_sentiments()
        data = self.__class__.__count_values_in_column(
            data=tweet_list_df, feature="sentiment"
        )
        # print(data.head())
        return data
    except Exception as e:
        self.logger.error(f"{e}")

def plot_tweet_sentiment_donut_chart(self):
    try:
        pichart = self.get_tweets_sentiments_percentage()
        names = pichart.index
        size = pichart['Percentage']
        fig = plt.figure(figsize=(10, 7))
        my_circle = plt.Circle((0, 0), 0.7, color="black")
        plt.pie(size, autopct='%1.0f%%', textprops={'fontsize': 16}, labels=names,
                colors=['#fab1a0', '#74b9ff', '#ff7675'])
        p = plt.gcf()
        plt.title(label=f'{self.stock_ticker} Tweets Sentiments')
        plt.legend(loc="best")
        p.gca().add_artist(my_circle)
        # plt.show()
        return fig
    except Exception as e:
        self.logger.error(f"{e}")
```

Fig 8.3.24 - source code for plotting donut chart and calculating percentage ratio

The get_tweets_sentiments_percentage method is an instance method of the StockTweets class. This method is used for retrieving a list of cleaned tweets for a particular stock, adding columns to the DataFrame to store the polarity and subjectivity of each tweet, and then adding further columns to the DataFrame to store the sentiment, negativity, neutrality, positivity, and compound score of each tweet. It then calculates the percentage of tweets with each sentiment and returns the result as a Pandas DataFrame.

The method begins by calling the get_tweets_sentiments instance method to get the DataFrame of cleaned tweets with added columns for polarity, subjectivity, sentiment, negativity, neutrality, positivity, and compound score. It then calls the __count_values_in_column static method, passing in the DataFrame and the name of the column containing the sentiments. The __count_values_in_column method counts the number of tweets with each sentiment and calculates the percentage of tweets with each sentiment.

Finally, the get_tweets_sentiments_percentage method returns the DataFrame with the calculated percentages. If an error occurs, the method logs an error message and returns nothing.

The plot_tweet_sentiment_donut_chart method is an instance method of the StockTweets class. This means that it can only be called on instances of the StockTweets class. The method appears to be used for generating a donut chart to visualise the sentiment percentages of the tweets for a particular stock.

The method begins by calling the get_tweets_sentiments_percentage instance method to get the DataFrame of sentiment percentages. It then uses the matplotlib package to create a donut chart using the plot

```
You, 2 days ago | 1 author (You)
class StockNews(StockTweets):
    def __init__(self, stock_ticker):
        super().__init__(stock_ticker)
        self.finviz_url = 'https://finviz.com/quote.ashx?t='
        #print(date.today(), date.today() - timedelta(days=10))

    def get_news(self):
        try:
            self.logger.info(
                f"Retriving Financial News for {self.stock_ticker}")
            url = self.finviz_url + self.stock_ticker
            req = Request(url=url, headers={
                'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/20100101 Firefox/20.0'})
            response = urlopen(req)
            # Read the contents of the file into 'html'
            html = BeautifulSoup(response, "lxml")
            # Find 'news-table' in the Soup and load it into 'news_table'
            news_table = html.find(id='news-table')
            return news_table
        except Exception as e:
            self.logger.error(f"Didn't find news: {e} for {self.stock_ticker}")
```

Fig 8.3.25 - source code for stocknews class initialization and web scraping news

The StockNews class is a subclass of the StockTweets class.

The __init__ method of the StockNews class takes one parameter: stock_ticker, which is the stock ticker symbol for which to retrieve news. The method first calls the __init__ method of the parent StockTweets class, passing in the stock_ticker parameter. This allows the StockNews class to inherit all of the attributes and methods of the StockTweets class.

The __init__ method then defines a new instance variable called finviz_url, which contains the URL of the Finviz website. This is used later when making a request to the Finviz website to retrieve news articles.

The get_news method is an instance method of the StockNews class. This means that it can only be called on instances of the StockNews class. This method is used for retrieving news articles related to a particular stock ticker symbol.

The method begins by logging a message that it is retrieving financial news for the stock specified by the stock_ticker instance variable. It then constructs the URL for the Finviz website by concatenating the finviz_url instance variable with the stock_ticker variable.

Next, the method makes an HTTP request to the constructed URL, using the `urllib

```
@staticmethod
def parse_news(news_table):
    parsed_news = []
    for x in news_table.findAll('tr'):
        # read the text from each tr tag into text
        # get text from a only
        text = x.a.get_text()
        # splite text in the td tag into a list
        date_scrape = x.td.text.split()
        # if the length of 'date_scrape' is 1, load 'time' as the only element

        if len(date_scrape) == 1:
            time = date_scrape[0]

        # else load 'date' as the 1st element and 'time' as the second
        else:
            date = date_scrape[0]      You, 2 weeks ago • Improved Sentiment Analyzer
            time = date_scrape[1]

        # Append ticker, date, time and headline as a list to the 'parsed_news' list
        parsed_news.append([date, time, text])
        # Set column names
        columns = ['date', 'time', 'headline']
        # Convert the parsed_news list into a DataFrame called 'parsed_and_scored_news'
        parsed_news_df = pd.DataFrame(parsed_news, columns=columns)
        # Create a pandas datetime object from the strings in 'date' and 'time' column
        parsed_news_df['datetime'] = pd.to_datetime(
            parsed_news_df['date'] + ' ' + parsed_news_df['time'])

    return parsed_news_df
```

Fig 8.3.26 - source code for cleaning html table after web scraping

The parse_news method is a static method of the StockNews class.This method is used for parsing news articles retrieved from the Finviz website.

The method takes one parameter: news_table, which is a BeautifulSoup object containing the HTML of the news table from the Finviz website. The method iterates over the tr tags in the news_table object, extracting the text of each news article, as well as the date and time of publication.

Next, the method creates a new list called parsed_news, which will contain the parsed news articles. For each tr tag in the news_table object, the method appends a list containing the date, time, and text of the news article to the parsed_news list.

Once all of the news articles have been parsed, the method creates a Pandas DataFrame from the parsed_news list, using the column names date, time, and headline. It then creates a new column called datetime, which contains a pandas datetime object created from the values in the date and time columns.

Finally, the parse_news method returns the parsed_news_df DataFrame containing the parsed news articles.

```
def get_news_df(self):
    self.logger.info(f"parsing news for {self.stock_ticker}")
    try:
        news_table = self.get_news()
        parsed_news_df = self.parse_news(news_table)
        return parsed_news_df
    except Exception as e:
        self.logger.error(f"Didn't find any news for {self.stock_ticker}")

def get_stock_news_sentiments(self):
    parsed_news_df = self.get_news_df()
    parsed_and_scored_news = super(
        StockNews, self).get_polarity(parsed_news_df, 'headline')

    parsed_and_scored_news = super(StockNews, self).get_sentiments(
        parsed_and_scored_news, 'headline')

    parsed_and_scored_news = parsed_news_df.set_index('datetime')

    parsed_and_scored_news = parsed_and_scored_news.drop(
        columns=['date', 'time'])

    parsed_and_scored_news = parsed_and_scored_news.rename(
        columns={"compound": "sentiment_score"})

    parsed_and_scored_news.sort_index(ascending=False, inplace=True)
    return parsed_and_scored_news
```

Fig 8.3.27 - source code for web scrapping news and calculating polarity score for sentiments based on news

The get_news_df method is an instance method of the StockNews class. This method is used for retrieving a DataFrame containing parsed news articles related to a particular stock ticker symbol.

The method begins by logging a message that it is parsing news for the stock specified by the stock_ticker instance variable. It then calls the get_news method to retrieve the news table from the Finviz website, and passes this news table to the parse_news static method to parse the articles. The parse_news method returns a DataFrame containing the parsed news articles, which is then returned by the get_news_df method.

The get_stock_news_sentiments method is another instance method of the StockNews class. This method appears to be used for retrieving a DataFrame containing news articles and their corresponding sentiment scores.

The method begins by calling the get_news_df method to retrieve the parsed news articles, and then calls the get_polarity method of the parent StockTweets class to calculate the polarity of the articles. This method returns a new DataFrame containing the polarity of each article.

Next, the method calls the get_sentiments method of the parent StockTweets class to calculate the sentiment of each article. This method returns a new DataFrame containing the sentiment of each article.

The method then sets the datetime column of the DataFrame as the index, drops the date and time columns, renames the compound column to sentiment_score, and sorts the DataFrame by the index in descending order. Finally, the method returns the modified DataFrame containing the parsed news articles and their corresponding sentiment scores.

```
def __plot_daily_sentiment(self, parsed_and_scored_news, ticker):

    # Group by date and ticker columns from scored_news and calculate the mean
    df = parsed_and_scored_news
    df2 = df.reset_index()
    df2['datetime'] = df2['datetime'].dt.date
    df2.set_index('datetime', inplace=True)
    df2.sort_i
    color = ['
    fig = plt.figure(figsize=(11, 3))
    sns.barplot(data=df2, x=df2.index,
                y=df2['sentiment_score'], hue=df2['sentiment'], palette=color)
    fig.autofmt_xdate()
    plt.title(label=f"{self.stock_ticker} Daily News Sentiment Scores")
    # plt.show()
    return fig

def plot_daily_sentiment_barchart(self):
    try:
        self.logger.info(
            f"Plotting Daily Sentiment Bar Chart for {self.stock_ticker}")
        parsed_and_scored_news = self.get_stock_news_sentiments()
        ticker = self.stock_ticker
        return self.__plot_daily_sentiment(parsed_and_scored_news, ticker)
    except Exception as e:
        self.logger.error(f"Couldn't plot any chart {e}")
```


Fig 8.3.28 - source code for plotting bar chart for stock news

The StockTweets class provides methods to access and preprocess tweets for a given stock ticker. It defines methods for authenticating with the Twitter API using an access token and API key, preprocessing tweets to remove unnecessary information, and calculating sentiment scores for tweets using the TextBlob library.

The StockNews class extends the StockTweets class to also access and preprocess news articles related to a given stock ticker. It defines a method to scrape news articles from the Finviz website, parse the articles into a DataFrame, and calculate sentiment scores for the articles using the get_polarity and get_sentiments methods from the StockTweets class. It also defines methods to plot the sentiment scores of news articles over time.

The StockNews class has several methods that are used to retrieve, parse, and analyze news data for a given stock ticker. Some of these methods include:

get_news: This method retrieves news data for the given stock ticker from the finviz website.

parse_news: This method parses the retrieved news data and returns a dataframe containing the news headlines, dates, and times.

get_news_df: This method retrieves and parses the news data for the given stock ticker.

get_stock_news_sentiments: This method calculates the sentiment scores for each news headline and adds these scores to the news data dataframe.

plot_daily_sentiment_barchart: This method plots a bar chart showing the daily sentiment scores for the news headlines.

plot_sentiments_with_price: This method plots the sentiment scores for the news headlines on the same chart as the stock price data, allowing you to see how the sentiment of the news affects the stock price. These methods are used together to analyze the sentiment of news headlines for a given stock ticker and visualize the results.

StockPredictionAnalyzer

```
You, 11 hours ago | 1 author (You)
class RNN_Model:
    def __init__(self, stock_ticker):
        self.stock_ticker = stock_ticker
        self.aim = "Close"
        self.scaler = MinMaxScaler()
        self.rnn_neurons = 256
        self.epochs = 100
        self.batch_size = 32
        self.loss = 'mse'
        self.dropout = 0.24
        self.optimizer = 'adam'
        self.data, self.y = self.__get_data()
        self.row_len = round(len(self.data.index) * 0.95)
        self.X_train, self.y_train, self.X_test, self.y_test = self.__scaler_transform()
        self.X_train = np.expand_dims(self.X_train, axis=1)
        self.X_test = np.expand_dims(self.X_test, axis=1)

    def __get_data(self):
        data = StockDatapipeline.get_stock_data_from_ticker(self.stock_ticker)
        data = data[["Date", "Volume", "Open", "Close"]]
        y = data.loc[:, ['Close', 'Date']]
        data = data.drop(['Close'], axis='columns')
        y = y.set_index('Date')
        y.index = pd.to_datetime(y.index, unit='ns')
        data = data.set_index('Date')
        data.index = pd.to_datetime(data.index, unit='ns')
        # print(y)
        return data, y
```

Fig 8.3.29 - source code for RNN model class initialization

This code is used for defining a class called RNN_Model. The __init__ method initializes several attributes of the RNN_Model object, including the stock ticker for which the model will be trained, the desired output variable ("Close"), and various hyperparameters such as the number of epochs and the batch size. It also retrieves stock data and target values using the StockDatapipeline.
get_stock_data_from_ticker method, and scales the data using a MinMaxScaler object. The method then splits the data into training and test sets and reshapes them to be compatible with the RNN model.

```
@staticmethod
def get_callback() -> None:
    callback = tf.keras.callbacks.EarlyStopping(
        monitor="val_loss", patience=3, mode="min", verbose=1
    )
    return callback

def __get_dynamic_train_test_data(self):
    X_train = self.data[:self.row_len]
    X_test = self.data[self.row_len:]
    y_train = self.y[:self.row_len]
    y_test = self.y[self.row_len:]
    #print(X_train, y_train, X_test, y_test)
    return X_train, y_train, X_test, y_test

def __scaler_transform(self):
    X_train, y_train, X_test, y_test = self.__get_dynamic_train_test_data()
    X_train = self.scaler.fit_transform(X_train)
    X_test = self.scaler.fit_transform(X_test)
    y_train = self.scaler.fit_transform(y_train)
    y_test = self.scaler.fit_transform(y_test)
    #print(X_train, y_train, X_test, y_test)

    return X_train, y_train, X_test, y_test
```

Fig 8.3.30 - source code for train-test split , pre processing and early stopping

The get_callback method returns a tf.keras.callbacks.EarlyStopping object that can be used to stop training the RNN model if the validation loss does not improve after a specified number of epochs.

The __get_dynamic_train_test_data method splits the stock data and target values into training and test sets.

The __scaler_transform method scales the training and test sets using the MinMaxScaler object, which will scale the data to have values in the range [0, 1]. This can help the model converge faster and can improve its performance.

```
@staticmethod
def build_RNN_model(input_data, output_size, neurons, activ_func='tanh',
                    dropout=0.21, loss='mse', optimizer='adam'):
    """Recurrent Neural Network Model"""
    model = Sequential()
    model.add(RNN(cell=[SimpleRNNCell(256),
                        SimpleRNNCell(512),
                        SimpleRNNCell(1024)], input_shape=(1, 2)))
    model.add(Dropout(dropout))
    model.add(Dense(units=64*4))
    model.add(Activation("relu"))
    model.add(Dropout(dropout))
    model.add(Dense(units=output_size))
    model.add(Activation(activ_func))

    model.compile(loss=loss, optimizer=optimizer)
    return model
```

Fig 8.3.31 - source code for building RNN model

The build_RNN_model method defines a recurrent neural network model for stock price prediction. The model has three SimpleRNNCell layers, each with a different number of neurons. The first layer has 256 neurons, the second has 512 neurons, and the third has 1024 neurons. The model also has two dropout layers and two dense layers, which are used to make predictions. The compile method is used to specify the loss function and optimizer for the model. The resulting model is then returned.

```
def train_rnn_model(self):
    np.random.seed(1024)
    model = self.build_RNN_model(self.X_train, output_size=1, neurons=self.rnn_neurons,
                                 dropout=self.dropout, loss=self.loss, optimizer=self.optimizer)

    modelfit = model.fit(self.X_train, self.y_train, validation_data=(
        self.X_test, self.y_test), epochs=self.epochs, batch_size=self.batch_size,
        verbose=1, shuffle=True, callbacks=self.get_callback())

    return model, modelfit

def plot_rnn_prediction(self):
    model, modelfit = self.train_rnn_model()
    test_data = self.y[self.row_len:]
    preds = model.predict(self.X_test).squeeze()

    predd = self.scaler.inverse_transform(preds.reshape(len(test_data), 1))
    predd = pd.Series(index=test_data.index, data=predd.flatten())
    return self.line_plot(test_data.Close, predd, 'Actual',
                          'Predicted', stock_ticker=self.stock_ticker, title='RNN')
```

Fig 8.3.32 - source code for training RNN model and plotting prediction

The train_rnn_model method trains the RNN model defined in the build_RNN_model method. It uses the fit method to train the model on the training data and evaluate it on the test data. The get_callback method is used to stop training the model if the validation loss does not improve after 3 epochs. The trained model and the training history are then returned.

The plot_rnn_prediction method uses the trained RNN model to make predictions on the test data. It then uses the line_plot method to visualize the actual and predicted stock prices. This can help users understand how well the model is performing and whether it is making accurate predictions.

```
You, 5 days ago | 1 author (You)
class LSTM_Model(RNN_Model):
    def __init__(self, stock_ticker):
        super().__init__(stock_ticker)
        self.window_len = 5
        self.test_size = 0.2
        self.zero_base = True
        self.lstm_neurons = 256
        self.epochs = 100
        self.batch_size = 32
        self.loss = 'mse'
        self.dropout = 0.24
        self.optimizer = 'adam'

    @staticmethod
    def build_lstm_model(input_data, output_size, neurons, activ_func='linear',
                         dropout=0.2, loss='mse', optimizer='adam'):
        model = Sequential()
        model.add(LSTM(neurons, input_shape=(
            input_data.shape[1], input_data.shape[2])))
        model.add(Dropout(dropout))
        model.add(Dense(units=output_size))
        model.add(Activation(activ_func))

        model.compile(loss=loss, optimizer=optimizer)
        return model
```

Fig 8.3.33 - source code for class inizilization for LSTM model

The LSTM_Model class inherits from the RNN_Model class. It has the same methods and attributes as the RNN_Model class, with the addition of the window_len, test_size, and zero_base attributes. The window_len attribute specifies the number of past data points to use for making predictions. The test_size attribute specifies the proportion of the data that should be used for testing the model. The zero_base attribute specifies whether the data should be normalized to have a zero mean. These attributes can be used to customize the training and testing of the LSTM model.

The build_lstm_model() method is used to create a LSTM (Long Short-Term Memory) model for use in predicting stock prices. It takes the following arguments:

input_data: The input data for the model, in the form of a NumPy array. This should have shape (n_samples, window_len, n_features) where n_samples is the number of samples, window_len is the number of time steps, and n_features is the number of features (e.g. the number of columns in the input data).
output_size: The size of the model's output. This should be set to 1 for a regression task (predicting a single continuous value, such as stock price).

neurons: The number of neurons (or nodes) in the LSTM layer.
activ_func: The activation function to use in the output layer. This should be set to linear for regression tasks.
dropout: The dropout rate to use for the Dropout layers in the model. This is a regularization technique that helps prevent overfitting.
loss: The loss function to use for the model. This should be set to mse (mean squared error) for regression tasks.
optimizer: The optimization algorithm to use for training the model. This is typically set to adam, which is a variant of stochastic gradient descent.
The build_lstm_model() method creates a Sequential model using the keras library. It adds an LSTM layer with the specified number

```
def train_lstm_model(self):
    model = self.build_lstm_model(
        self.X_train, output_size=1, neurons=self.lstm_neurons, dropout=self.dropout, loss=self.loss,
        optimizer=self.optimizer)
    modelfit = model.fit(self.X_train, self.y_train, validation_data=(
        self.X_test, self.y_test), epochs=self.epochs,
        batch_size=self.batch_size, verbose=1, shuffle=True, callbacks=super(LSTM_Model, self).get_callback())

    return model, modelfit      You, 5 days ago • App 3.0 Refractor

def plot_lstm_prediction(self):
    model, modelfit = self.train_lstm_model()
    test_data = self.y[self.row_len:]
    preds = model.predict(self.X_test).squeeze()

    predd = self.scaler.inverse_transform(preds.reshape(len(test_data), 1))
    predd = pd.Series(index=test_data.index, data=predd.flatten())
    return super(LSTM_Model, self).line_plot(test_data.Close, predd, 'Actual',
                                             'Predicted', stock_ticker=self.stock_ticker, title='LSTM')
```

Fig 8.3.34 - source code for training LSTM model and plotting prediction

The LSTM_Model class appears to be a subclass of the RNN_Model class, which is a class that trains a recurrent neural network (RNN) model on a given stock's data to predict its closing value. The LSTM_Model class overrides some of the attributes and methods of the RNN_Model class in order to train a long short-term memory (LSTM) model instead of an RNN model.

The init method of the LSTM_Model class takes in a stock_ticker argument, which is the stock's ticker symbol. It passes this argument to the __init__ method of the RNN_Model class using super(), so that the RNN_Model class can retrieve the stock's data using its get_stock_data_from_ticker method.

The LSTM_Model class then sets some attributes that are specific to training an LSTM model, such as the window_len attribute which specifies the length of the sliding window used to split the data into training and test sets. The train_lstm_model method uses this attribute along with other attributes, such as the lstm_neurons attribute that specifies the number of neurons in the LSTM layer, to train an LSTM

model on the stock's data. The plot_lstm_prediction method uses the trained LSTM model to make predictions on the test set and plots the actual and predicted values.

```
You, 11 hours ago | 1 author (You)
class XGBoostModel:
    def __init__(self, stock_ticker):
        self.test_size = 0.05
        self.valid_size = 0.05
        self.stock_ticker = stock_ticker
        self.data = StockDatapipeline.get_stock_data_from_ticker(stock_ticker)
        self.gamma = 0.001
        self.learning_rate = 0.05
        self.max_depth = 12
        self.n_estimators = 400
        self.random_state = 42      You, last week • fine tuning xgboost
```

Fig 8.3.35 - source code for XGboost class initialization

This class is a definition for an XGBoostModel object. It takes a stock ticker as input and uses it to retrieve stock data using the StockDatapipeline.get_stock_data_from_ticker method. The data, along with some other parameters, are stored as attributes of the XGBoostModel object.

```
def build_and_train_model(self):
    y_train, X_train, y_valid, X_valid, y_test, X_test = self.split_data()
    eval_set = [(X_train, y_train), (X_valid, y_valid)]
    model = xgb.XGBRegressor(gamma=self.gamma, learning_rate=self.learning_rate,
                             max_depth=self.max_depth, n_estimators=self.n_estimators,
                             random_state=self.random_state, objective='reg:squarederror')
    xg = model.fit(X_train, y_train, eval_set=eval_set, verbose=False)
    return xg       You, 2 weeks ago • refractor 1.0
```

Fig 8.3.36 - source code for building XGboost

The build_and_train_model method builds and trains a regression model using the XGBoost algorithm. It takes in parameters for the model, such as the learning rate and the maximum depth of the tree, and uses them to instantiate an XGBRegressor object. It then fits the model to the training data, and returns the fit model.

```python
def split_data(self):
    drop_cols = ['Date']
    df = self.get_data()
    df.reset_index(inplace=True)
    # print(df)
    You, 2 weeks ago • refractor 1.0
    test_split_idx = int(round(df.shape[0] * (1-self.test_size)))
    valid_split_idx = int(round(
        df.shape[0] * (1-(self.valid_size+self.test_size))))

    # print(test_split_idx)
    # print(valid_split_idx)

    train_df = df.loc[:valid_split_idx].copy()
    valid_df = df.loc[valid_split_idx+1:test_split_idx].copy()
    test_df = df.loc[test_split_idx+1:].copy()

    train_df = train_df.drop(columns=drop_cols)
    valid_df = valid_df.drop(columns=drop_cols)
    test_df = test_df.drop(columns=drop_cols)

    y_train = train_df['Close'].copy()
    X_train = train_df.drop(columns=['Close'])

    y_valid = valid_df['Close'].copy()
    X_valid = valid_df.drop(columns=['Close'])

    y_test = test_df['Close'].copy()
    X_test = test_df.drop(columns=['Close'])

    #print(y_train, X_train, y_valid, X_valid, y_test, X_test)
    return y_train, X_train, y_valid, X_valid, y_test, X_test
```

Fig 8.3.37 - source code for split data into train-test split

The XGBoostModel class is a class that represents a model using the XGBoost machine learning algorithm. The class is used for predicting the closing price of stocks based on certain features. The class has several methods, including split_data, which splits the data into train, validation, and test sets, and build_and_train_model, which builds and trains an XGBoost model on the training data.

```
    def get_ypredicted_value(self):
        y_train, X_train, y_valid, X_valid, y_test, X_test = self.split_data()
        model = self.build_and_train_model()
        y_pred = self.y_pred(model=model, X_test=X_test)
        # print(y_pred)
        print(f'mean_squared_error = {mean_squared_error(y_test, y_pred)}')
        return y_pred

    def plot_xgboost_prediction(self):
        df = self.get_data()
        df.reset_index(inplace=True)
        y_pred = self.get_ypredicted_value()
        test_split_idx = int(round(df.shape[0] * (1-self.test_size)))
        y_train, X_train, y_valid, X_valid, y_test, X_test = self.split_data()

        predicted_prices = df.loc[test_split_idx+1:].copy()
        predicted_prices['Close'] = y_pred

        fig = plt.figure(figsize=(7, 3))
        sns.lineplot(x=predicted_prices.Date, y=y_test, color='tab:Red')
        sns.lineplot(x=predicted_prices.Date, y=y_pred, color='#74b9ff')
        plt.legend(['Actual', 'Predicted'], loc='best')
        plt.title(f'{self.stock_ticker} XGBoost Model')
        fig.autofmt_xdate()

        return fig
```

Fig 8.3.38 - source code for training XGboost model and plotting prediction

The __init__ method initializes several class variables such as the stock ticker, test size, validation size, and various XGBoost model hyperparameters. The build_and_train_model method trains an XGBoost model using the stock data and the hyperparameters.

The split_data method splits the stock data into training, validation, and testing sets, and the get_ypredicted_value method uses the trained model to generate predictions on the test data.

Finally, the plot_xgboost_prediction method plots the actual and predicted stock prices.

9. Tests and Results.

In this chapter, the testing and presentation of results of the developed solution will be conducted.

9.1 Tests

In this sub chapter, the test case scenarios are created and performed.

Testing Scenario 1 : User is able to access and select the Stocks manually.

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application. 3. User has access to source code.	All Preconditions are met
2.	User clicks on the dropdown list below the text “Select Stocks”	User is able to see the list of multiple stocks pre added in the application: 1. GOOGL 2. NFLX 3. AAPL 4. MSFT 5. META 6. AMZN 7. TSLA 8. NVDA 9. TSM 10. WMT 11. INFY 12. WIT 13. RELI 14. BABA 15. IBM
3.	User access the source code. (i.e.STOCK_MARLET_PREDICTION_APP >data> data.json)	The list of the companies must match with the list of stocks available in the dropdown list in the application.

Testing Scenario 2 : User is able to download the stock data in .xsls format.

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application. 3. “Download stock data” button is active. 4. User has selected some stock and data is successfully loaded. (E.g. GOOGL) 5. User has access to source code.	All Preconditions are met.
2.	User clicks on the “Download Stock data” button.	The stock data is downloaded as (i.e. “{Stock Name}_stock_data.xlsx”) -User finds this file in the folder (i.e. STOCK_MARLET_PREDICTION_APP) -The downloaded file’s format must be in .xsls

Testing Scenario 3 : User is able to download the stock info in .CSV format.

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application. 3. “Download stock info” button is active. 4. User has selected some stock and data is successfully loaded. (E.g. GOOGL) 5. User has access to source code.	All Preconditions are met.
2.	User clicks on the “Download Stock info” button.	The stock info is downloaded as (i.e. “{Stock Name}_stock_info.csv”) -User finds this file in the folder (i.e. STOCK_MARLET_PREDICTION_APP) -The downloaded file’s format must be in .CSV

Testing Scenario 4 : User see all the prices in USD

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application. 3. User has selected some stocks and data is successfully loaded. (E.g., GOOGL)	All Preconditions are met.
2.	User observe the three prices for each stock on the header of the application.	All the stock prices displayed on the header of the application must be in the USD. 1. Stock Price 2. Open 3. Volume

Testing Scenario 5 : User is able to see the real time stocks

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application. 3. User has selected some stocks and data is successfully loaded. (E.g. GOOGL)	All Preconditions are met.
2.	User observe the three prices for each stock on the header of the application.	User must observe all the prices for real time from Yahoo Finance. (Verification: User can verify the prices by going to the link: https://finance.yahoo.com/quote/GOOG?p=GOOG&.tsrc=fin-srch) All the prices must be fetched from the link above for: 1. Stock Price 2. Open 3. Volume (Note: Decimals are ignored)

Testing Scenario 6: User is able to see "Technical Analysis" bar with all the appropriate buttons.

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application. 3. User has selected some stocks and data is successfully loaded. (E.g. GOOGL)	All Preconditions are met.
2.	User observe the "Technical Analysis" bar.	Technical Analysis bar must be present after all the prices of stock with all the buttons: 1. RSI 2. MACD 3. 50D SMA 4. 100 D SMA 5. 200 D SMA

Testing Scenarios 7: User is able to see the updated graphs with RSI (End-to-End Testing)

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application.	All Preconditions are met.
2.	The user Selects the “GOOGL” stocks and clicks on the RSI from “Technical Analysis”.	The data is successfully loaded, and the user is able to see the following charts: 1. RSI (Line) 2. Stock Price Chart (Line) 3. Two Side View (Line) 4. Tweet Sentiment View (Donut) 5. Daily News Sentiment Scores (Bar Graph) 6. Daily News Affecting Price (Line Chart) 7. LSTM Prediction (Line Chart) 8. XGBoost Prediction (Line Chart).
3.	The user Selects the “NFLX” stocks and clicks on the RSI from “Technical Analysis”.	The data is successfully updated, and the user is able to see the updated charts: 1. RSI (Line) 2. Stock Price Chart (Line) 3. Two Side View (Line) 4. Tweet Sentiment View (Donut) 5. Daily News Sentiment Scores (Bar Graph) 6. Daily News Affecting Price (Line Chart) 7. LSTM Prediction (Line Chart) 8. XGBoost Prediction (Line Chart).

Testing Scenarios 8: User is able to see the updated graphs with MACD (End-to-End Testing)

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application.	All Preconditions are met.
2.	The user Selects the “AAPL” stocks and clicks on the MACD from “Technical Analysis”.	The data is successfully loaded, and the user is able to see the following charts: 1. RSI (Line) 2. Stock Price Chart (Line) 3. Two Side View (Line) 4. Tweet Sentiment View (Donut) 5. Daily News Sentiment Scores (Bar Graph) 6. Daily News Affecting Price (Line Chart) 7. LSTM Prediction (Line Chart) 8. XGBoost Prediction (Line Chart).
3.	The user Selects the “NVDA” stocks and clicks on the MACD from “Technical Analysis”.	The data is successfully updated, and the user is able to see the updated charts: 1. RSI (Line) 2. Stock Price Chart (Line) 3. Two Side View (Line) 4. Tweet Sentiment View (Donut) 5. Daily News Sentiment Scores (Bar Graph) 6. Daily News Affecting Price (Line Chart) 7. LSTM Prediction (Line Chart) 8. XGBoost Prediction (Line Chart).

Testing Scenarios 9: User is able to see the updated graphs with 50D SMA (End-to-End Testing)

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application.	All Preconditions are met.
2.	The user Selects the “AMZN” stocks and clicks on the 50D SMA from “Technical Analysis”.	The data is successfully loaded, and the user is able to see the following charts: 1. RSI (Line) 2. Stock Price Chart (Line) 3. Two Side View (Line) 4. Tweet Sentiment View (Donut) 5. Daily News Sentiment Scores (Bar Graph) 6. Daily News Affecting Price (Line Chart) 7. LSTM Prediction (Line Chart) 8. XGBoost Prediction (Line Chart).
3.	The user Selects the “CS” stocks and clicks on the 50D SMA from “Technical Analysis”.	The data is successfully updated, and the user is able to see the updated charts: 1. RSI (Line) 2. Stock Price Chart (Line) 3. Two Side View (Line) 4. Tweet Sentiment View (Donut) 5. Daily News Sentiment Scores (Bar Graph) 6. Daily News Affecting Price (Line Chart) 7. LSTM Prediction (Line Chart) 8. XGBoost Prediction (Line Chart).

Testing Scenarios 10: User is able to see the updated graphs with 100D SMA (End-to-End Testing)

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application.	All Preconditions are met.
2.	The user Selects the "META" stocks and clicks on the 100D SMA from "Technical Analysis".	The data is successfully loaded, and the user is able to see the following charts: 1. RSI (Line) 2. Stock Price Chart (Line) 3. Two Side View (Line) 4. Tweet Sentiment View (Donut) 5. Daily News Sentiment Scores (Bar Graph) 6. Daily News Affecting Price (Line Chart) 7. LSTM Prediction (Line Chart) 8. XGBoost Prediction (Line Chart).
3.	The user Selects the "TSM" stocks and clicks on the 100D SMA from "Technical Analysis".	The data is successfully updated, and the user is able to see the updated charts: 1. RSI (Line) 2. Stock Price Chart (Line) 3. Two Side View (Line) 4. Tweet Sentiment View (Donut) 5. Daily News Sentiment Scores (Bar Graph) 6. Daily News Affecting Price (Line Chart) 7. LSTM Prediction (Line Chart) 8. XGBoost Prediction (Line Chart).

Testing Scenarios 11: User is able to see the updated graphs with 200D SMA (End-to-End Testing)

Step	Action	Expected Result
1.	Preconditions: 1. User has access to the stock application. 2. User is on the main page of the application.	All Preconditions are met.
2.	The user Selects the “TSLA” stocks and clicks on the 200D SMA from “Technical Analysis”.	The data is successfully loaded, and the user is able to see the following charts: 1. RSI (Line) 2. Stock Price Chart (Line) 3. Two Side View (Line) 4. Tweet Sentiment View (Donut) 5. Daily News Sentiment Scores (Bar Graph) 6. Daily News Affecting Price (Line Chart) 7. LSTM Prediction (Line Chart) 8. XGBoost Prediction (Line Chart).
3.	The user Selects the “GS” stocks and clicks on the 200D SMA from “Technical Analysis”.	The data is successfully updated, and the user is able to see the updated charts: 1. RSI (Line) 2. Stock Price Chart (Line) 3. Two Side View (Line) 4. Tweet Sentiment View (Donut) 5. Daily News Sentiment Scores (Bar Graph) 6. Daily News Affecting Price (Line Chart) 7. LSTM Prediction (Line Chart) 8. XGBoost Prediction (Line Chart).

9.2 Results

In this chapter the design of the application is defined and all the structure of the application by the use case diagram along with the User layer where we will go through the application's front end and the components present in our application.

9.2.1 User Layer

This subchapter covers the working of the application on the browser. where the user layer can be seen along with the Screenshot of the application.

This application just has one page and primarily uses data from Twitter, Finviz, and Yahoo Finance.

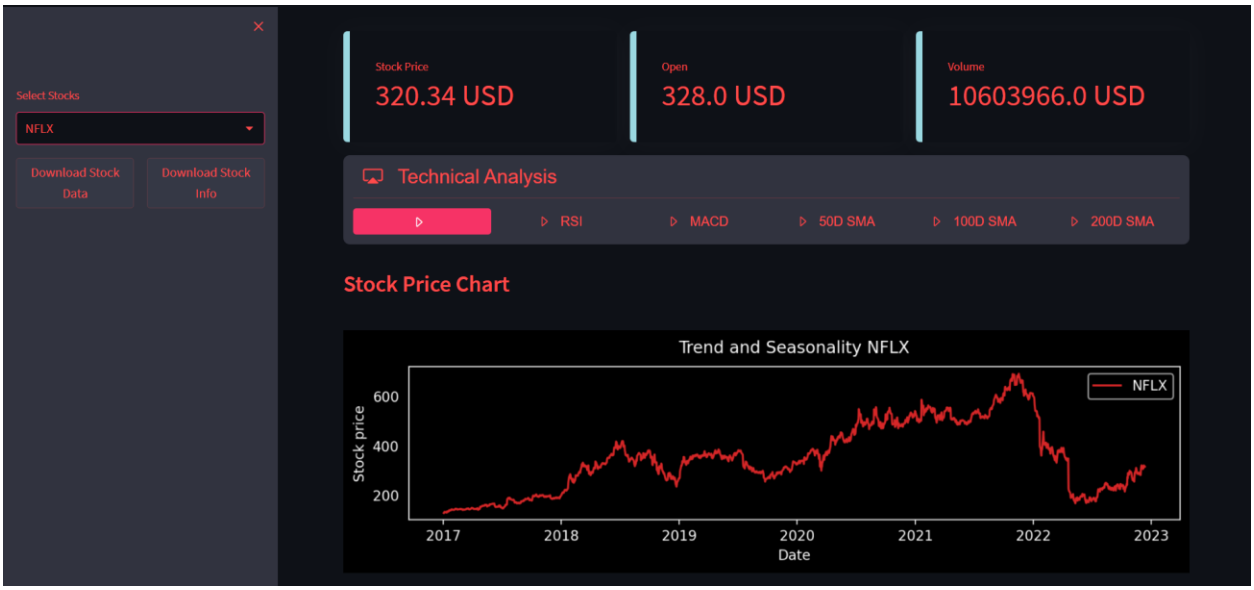


Fig 9.2.1.1 – home page of the application

In fig 9.2.1.1 the homepage of the application is displayed As can be seen, the web application has a side bar, a dashboard for real-time stock information, a dashboard for examining various forms of technical analysis graphs related to a stock, and numerous sorts of graphs and charts are displayed, mostly including predictions and sentiment analysis. This structure is incredibly straightforward and not at all complicated, which is exactly what the application is designed to be. Once the user chooses a stock from the sidebar's list of stocks, an application is supposed to execute and display the results. Lets now go through each components present in our application.

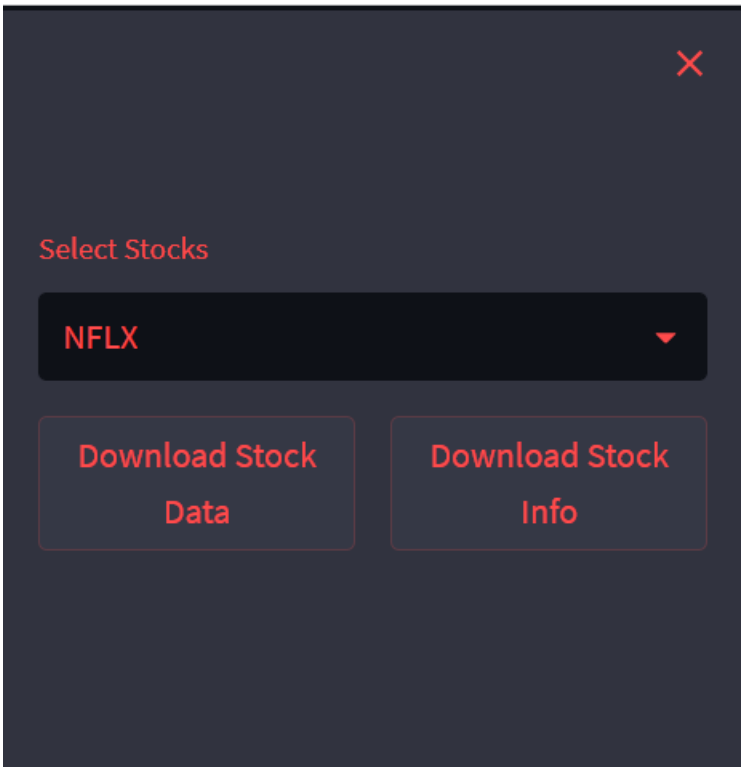


Fig 9.2.1.2 – sidebar of the application

In fig 9.2.1.2 the sidebar of the application is displayed As seen, the user would be able to choose a stock from the list of stocks available and download stock data and stock information for the chosen stock using this side bar.

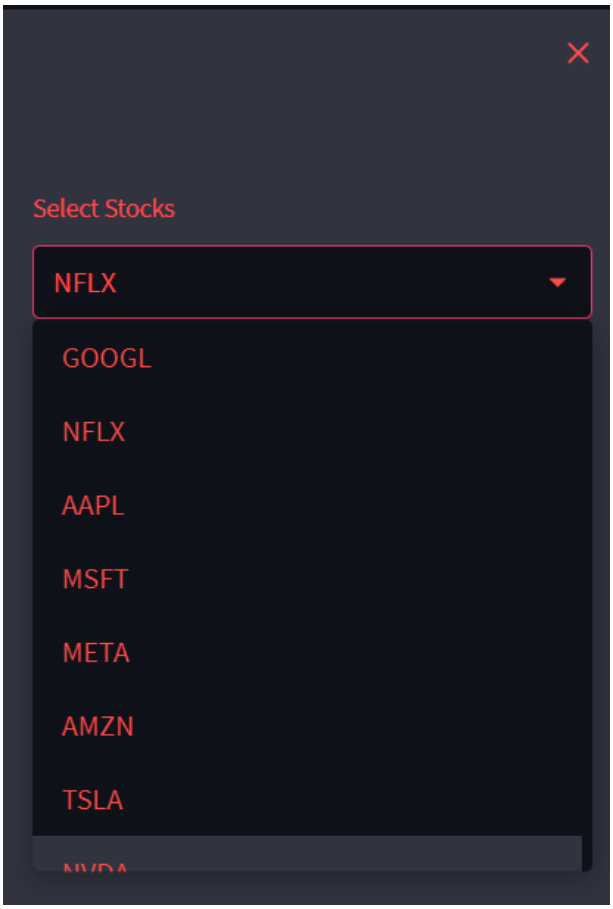


Fig 9.2.1.3 – list of stocks

In fig 9.2.1.3 the drop down is displayed As can be seen, the user can choose a stock from a list of available stocks by utilizing this drop-down menu. We have listed above 50 plus Fall back Major firms from the USA, Europe, and Asia are represented in the list of stocks by their stock tickers.

Some of the Stocks mentioned in drop down list include:

- GOOGL- Google LLC.
- NFLX - Netflix Inc
- AAPL - Apple Inc
- MSFT - Microsoft Corporation
- META - Meta Platforms, Inc
- AMZN - Amazon.com, Inc
- TSLA - Tesla Inc
- NVDA - NVIDIA Corporation
- TSM - Taiwan Semiconductor Mfg. Co. Ltd
- WMT - Walmart Inc
- INFY - Infosys Limited
- WIT - Wipro Limited
- RELI - Reliance Global Group Inc
- BABA - Alibaba Group Holding Limited
- IBM - International Business Machines Corporation(IBM)

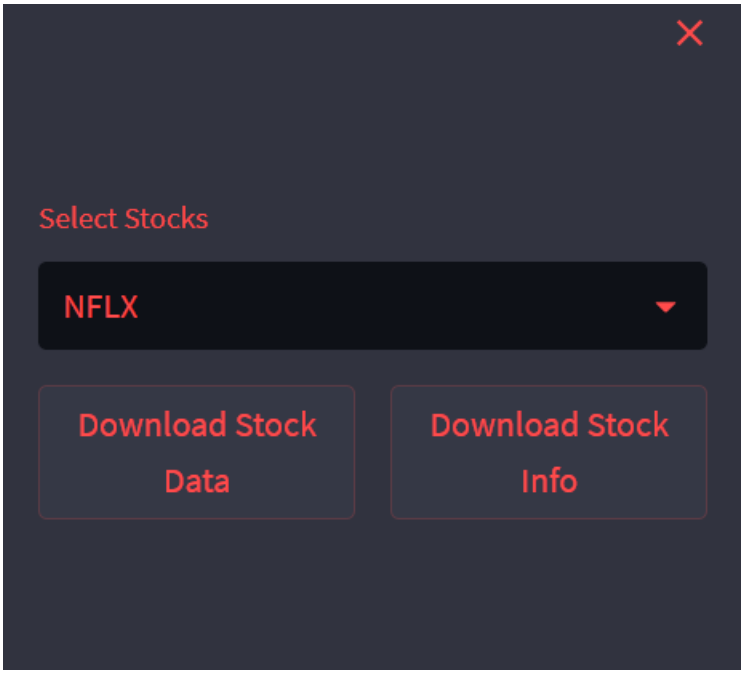


Fig 9.2.1.4 – Download stock data and stock info

In fig 9.2.1.4 the (Download stock data and stock info) two buttons which helps users to download stock data and stock info is displayed as can be seen, when the user clicks on the "download stock data" button, the entire market information related to the chosen stock will be downloaded. The complete historical data for a stock is downloaded when a user clicks "download stock info".

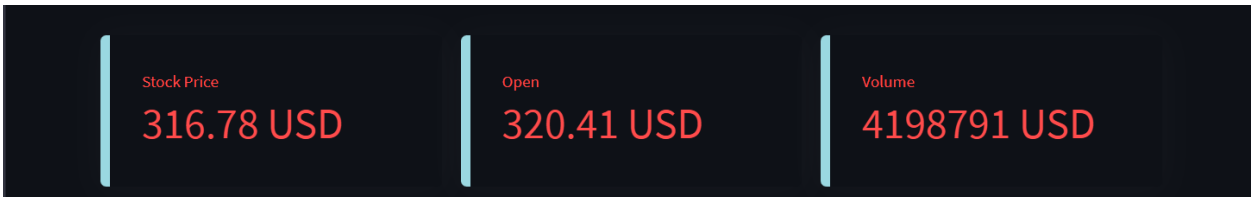


Fig 9.2.1.5 dashboard for real time data

In fig 9.2.1.5 the dashboard for real time data is displayed As can be seen, the user can check the opening price, real-time stock price, and volume of the stock they have chosen from the drop-down menu with the use of this dashboard.

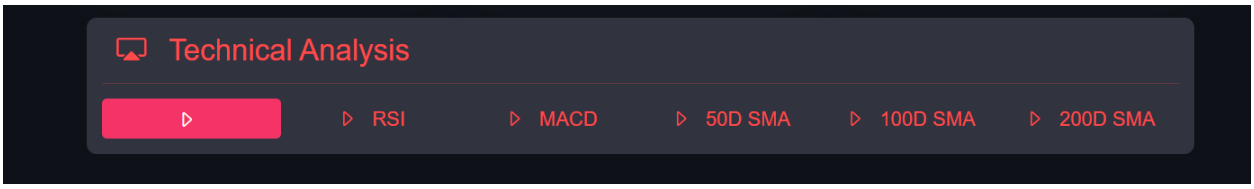


Fig 9.2.1.6 dashboard for technical analysis

In fig 9.2.1.6 the dashboard for technical analysis is displayed As seen, the user would be able to view various sorts of technical analysis charts with this dashboard, including

RSI-Relative strength index

MACD- moving average convergence/divergence

50D SMA -50-Day Simple Moving Average

100D SMA-100-Day Simple Moving Average

200D SMA-200-Day Simple Moving Average

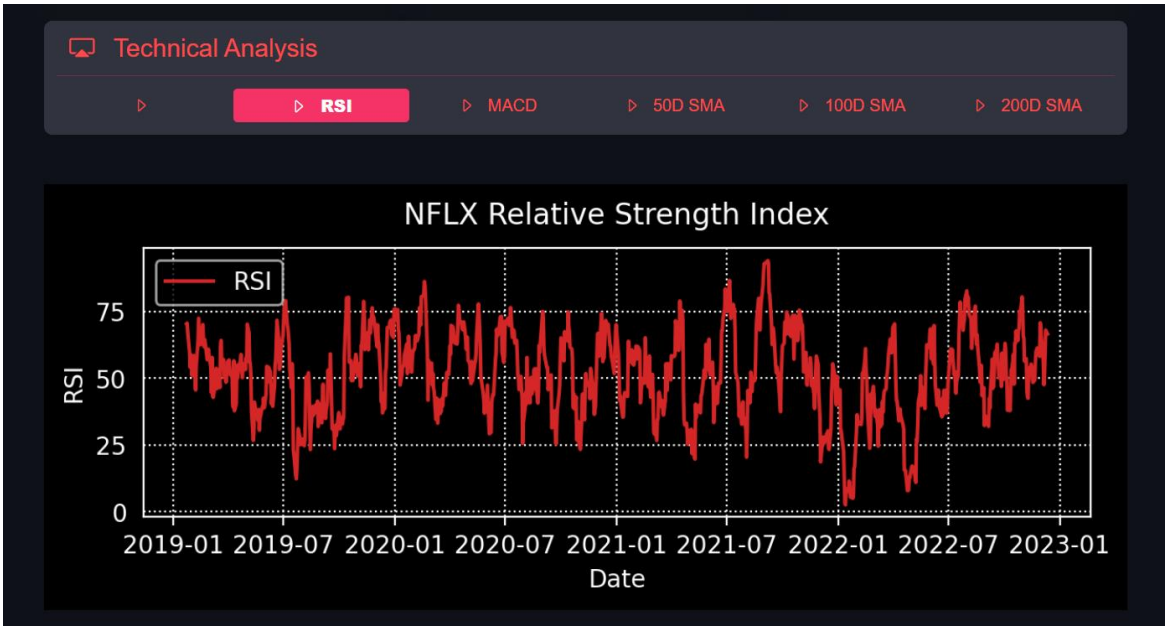


Fig 9.2.1.7 RSI-Relative strength index

In fig 9.2.1.7 the Relative strength index of Netflix stock is displayed. As can be seen, the user can gauge the speed and size of the stock's recent price changes using this chart.

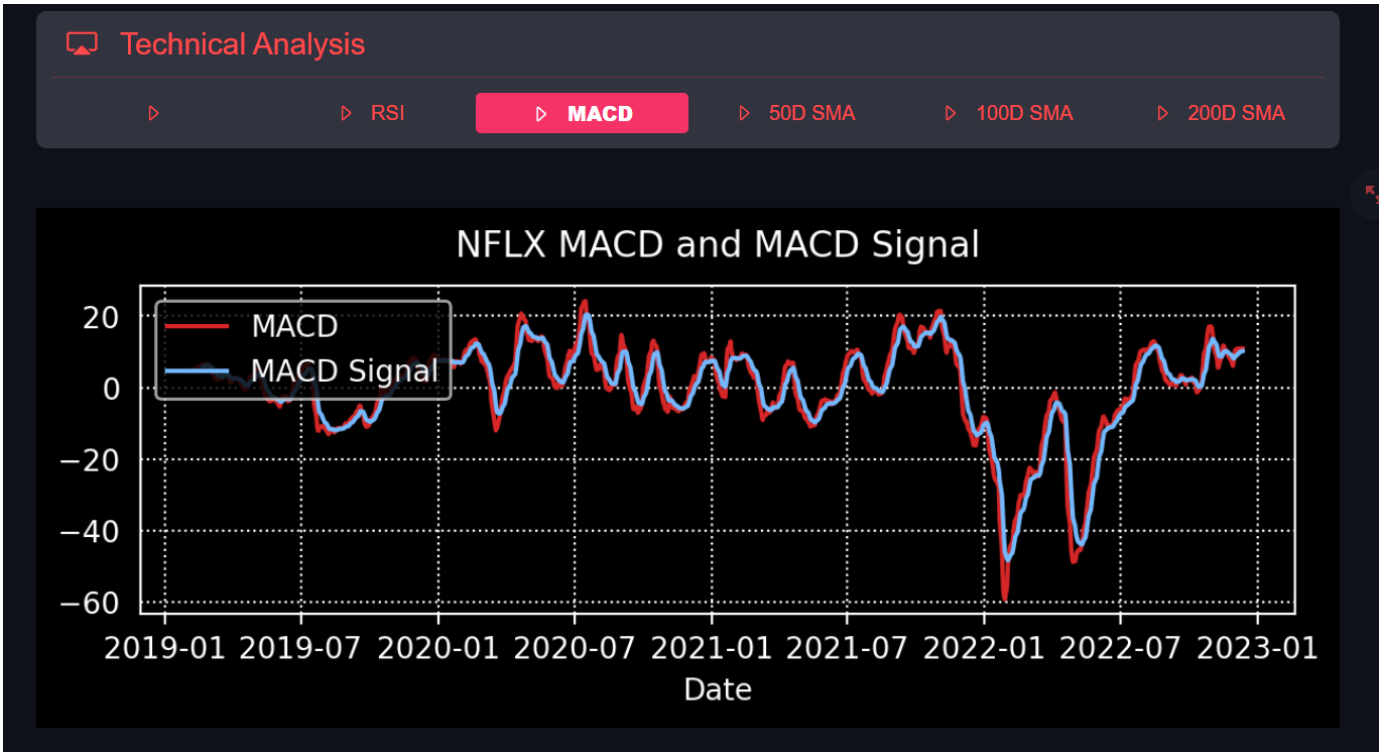


Fig 9.2.1.8 MACD - moving average convergence/divergence

In fig 9.2.1.8 the MACD of Netflix stock is displayed. As can be seen, the user can view a trend-following momentum indicator i.e. the relationship between two exponential moving averages of a stock's price using this chart.

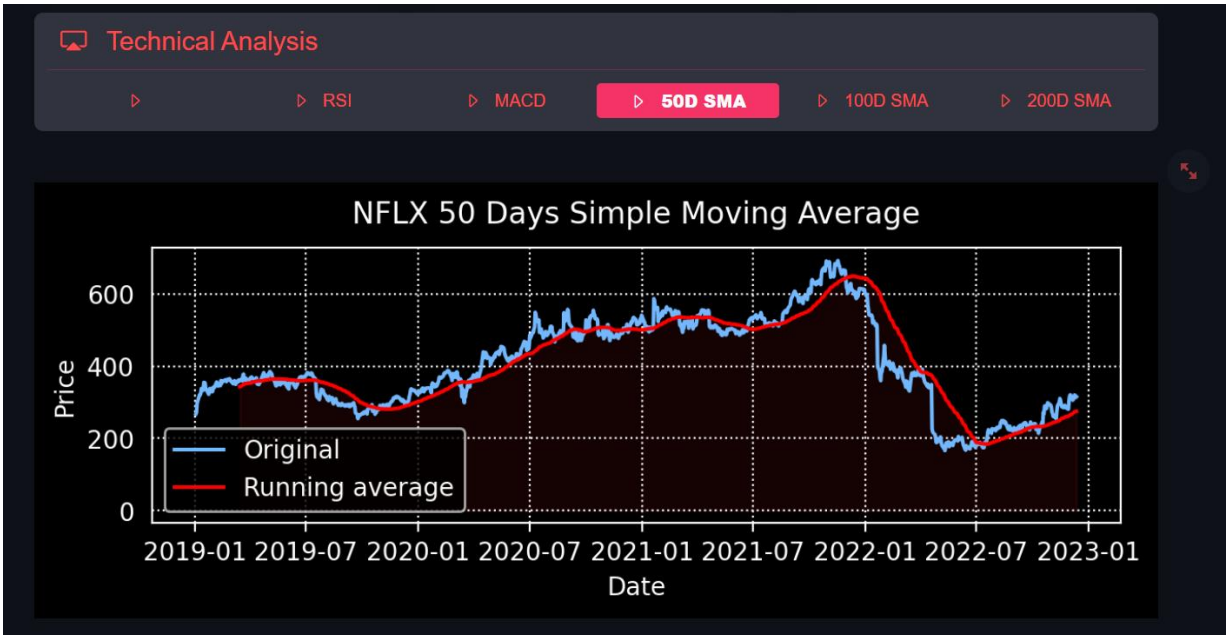


Fig 9.2.1.9 50D SMA -50-Day Simple Moving Average

In fig 9.2.1.9 the 50-Day Simple Moving Average of Netflix stock is displayed As can be seen, the user would be able to view the average of 50 days' worth of closing prices for a stock, plotted across time, as seen with the aid of this chart. Traders use it as a reliable trend indicator. It may also reveal the strength or weakness of a stock. Because it assigns the same weight to each day's closing price, it is known as basic.

Fig 9.2.1.10 100D SMA -100 Day Simple Moving Average

In fig 9.2.1.10 the 100-Day Simple Moving Average of Netflix stock is displayed As can be seen, the user would be able to view the average of 100 days' worth of closing prices for a stock, plotted across time, as seen with the aid of this chart. Traders use it as a reliable trend indicator. It may also reveal the strength or weakness of a stock. Because it assigns the same weight to each day's closing price, it is known as basic.

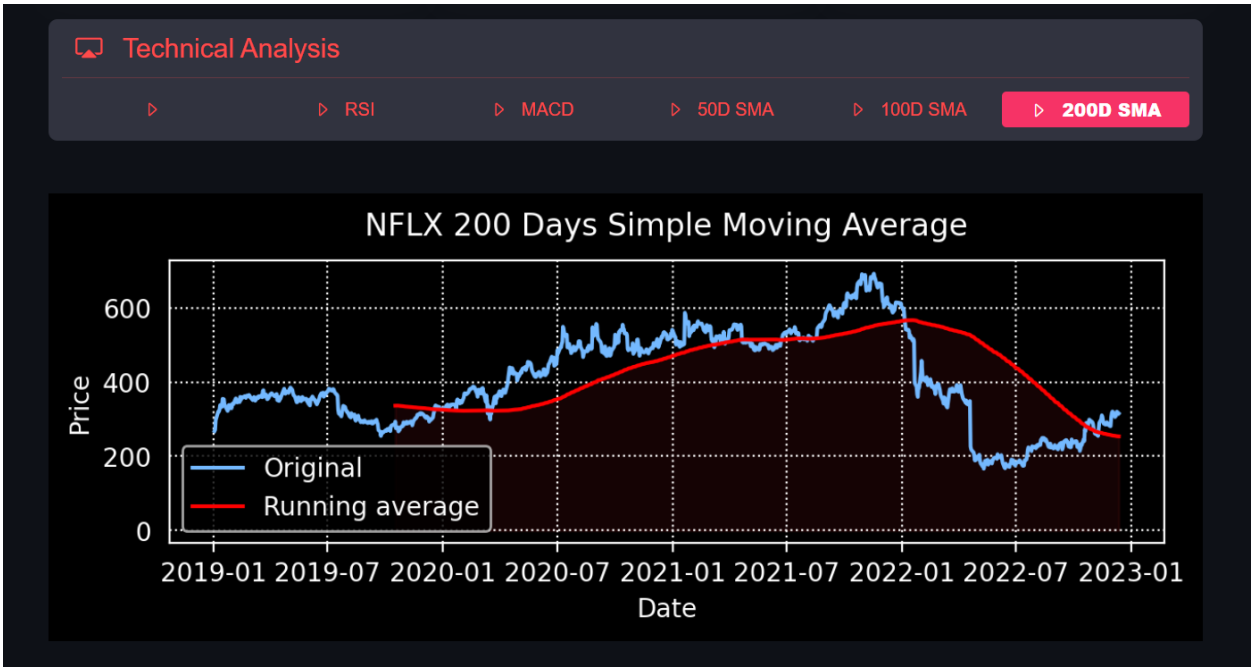


Fig 9.2.1.11 200D SMA -200 Day Simple Moving Average

In fig 9.2.1.11 the 200-Day Simple Moving Average of Netflix stock is displayed As can be seen, the user would be able to view the average of 200 days' worth of closing prices for a stock, plotted across time, as seen with the aid of this chart. Traders use it as a reliable trend indicator. It may also reveal the strength or weakness of a stock. Because it assigns the same weight to each day's closing price, it is known as basic.

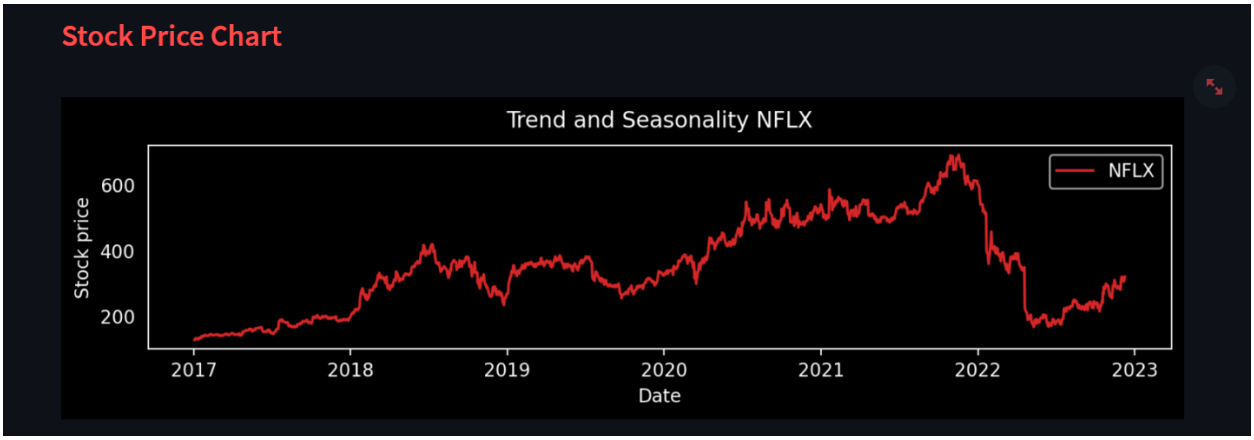


Fig 9.2.1.12 Trend and seasonality chart

In fig 9.2.1.12 the Trend and seasonality chart of Netflix stock is displayed As shown, the user would be able to gain historical perspective on performance tendencies with the use of this chart. By looking for bullish setups when the seasonal patterns are strongly bullish and bearish settings when the seasonal patterns are strongly bearish, the user can use these seasonal patterns to strengthen their edge. It is best to utilize seasonal charts in conjunction with other analysis methods.

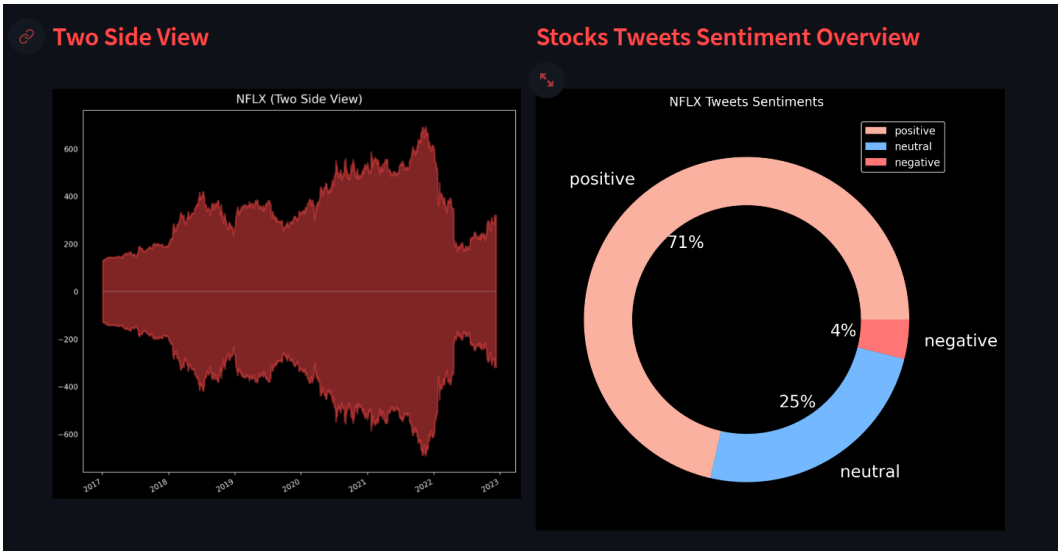


Fig 9.2.1.13 Two Side view and Tweets Sentiment Overview

In fig 9.2.1.13 the Two Side view and Tweets Sentiment Overview of Netflix stock is displayed. The user can learn about a stock's price trend by looking at it from two sides. And As shown in Tweets Sentiment Overview donut chart, the user would be able to gain perception of public sentiment about a stock from twitter. The opinions of Twitter users regarding a stock are categorized into three categories, including positive, negative, and neutral, in the Tweets Sentiment Overview Donut Chart

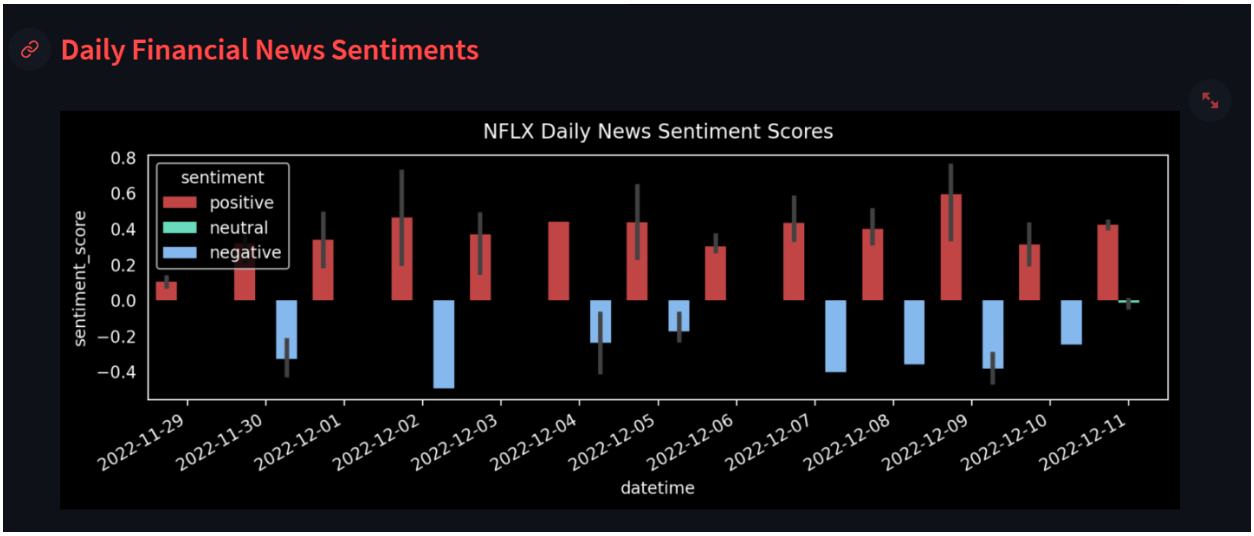


Fig 9.2.1.14 Daily news sentiments

In fig 9.2.1.14 the Daily news sentiments of Netflix stock is displayed. as noticed The news that affects a stock is divided into three categories, such as positive, negative, and neutral, in the Daily News Sentiment Chart. This provides a clearer picture of the current perception of a stock and whether news about it is good, negative, or neutral.

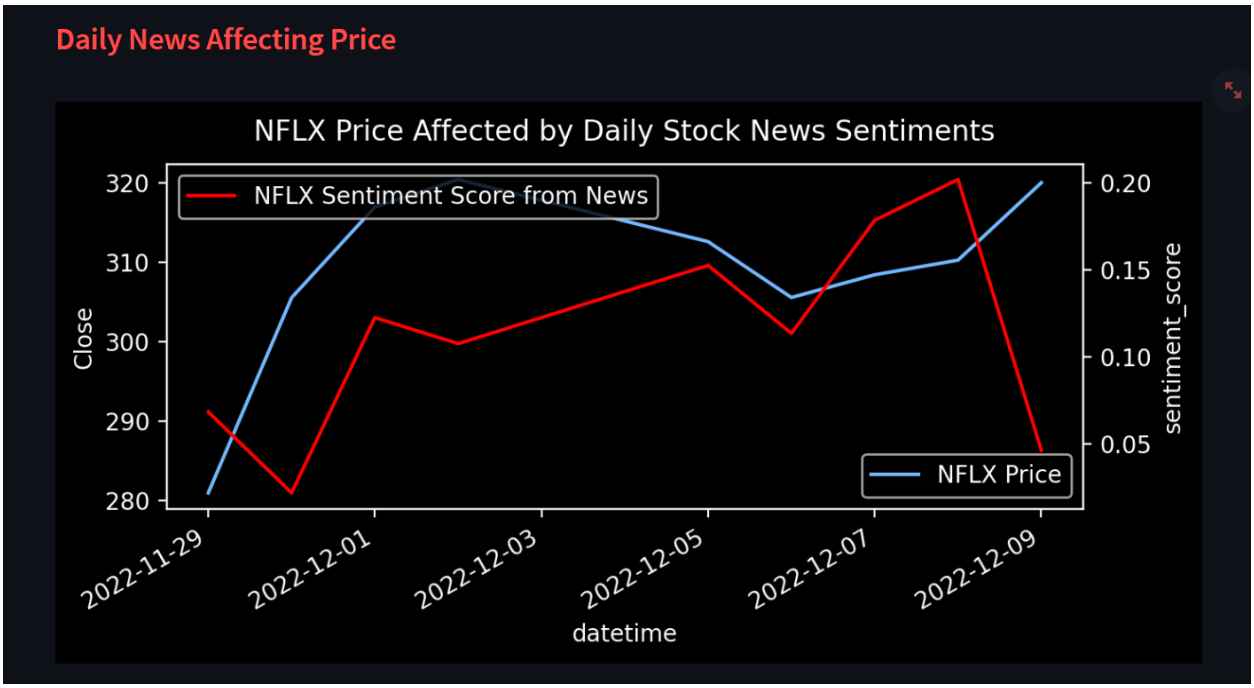


Fig 9.2.1.15 Daily news affecting price

In fig 9.2.1.15 price vs sentiment score from news related to Netflix stock is displayed. As seen, it provides users with a clear picture of how the sentiment score from news is impacting a stock's price.

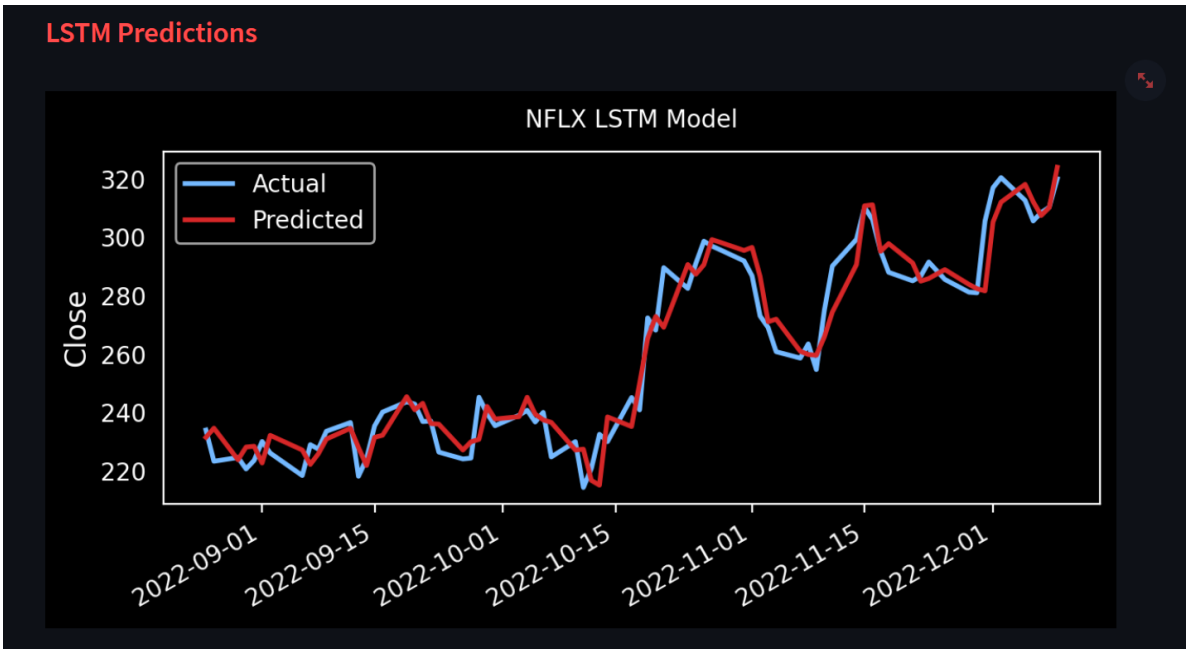


Fig 9.2.1.14 LSTM prediction

In fig prediction of Netflix stock from machine learning model i.e. LSTM model is displayed. As can be seen from the fig, it shows the user both the stock's current price movement and its forecast price trend. And by looking at this graph, the customer may determine to choose whether or not buy the stock.

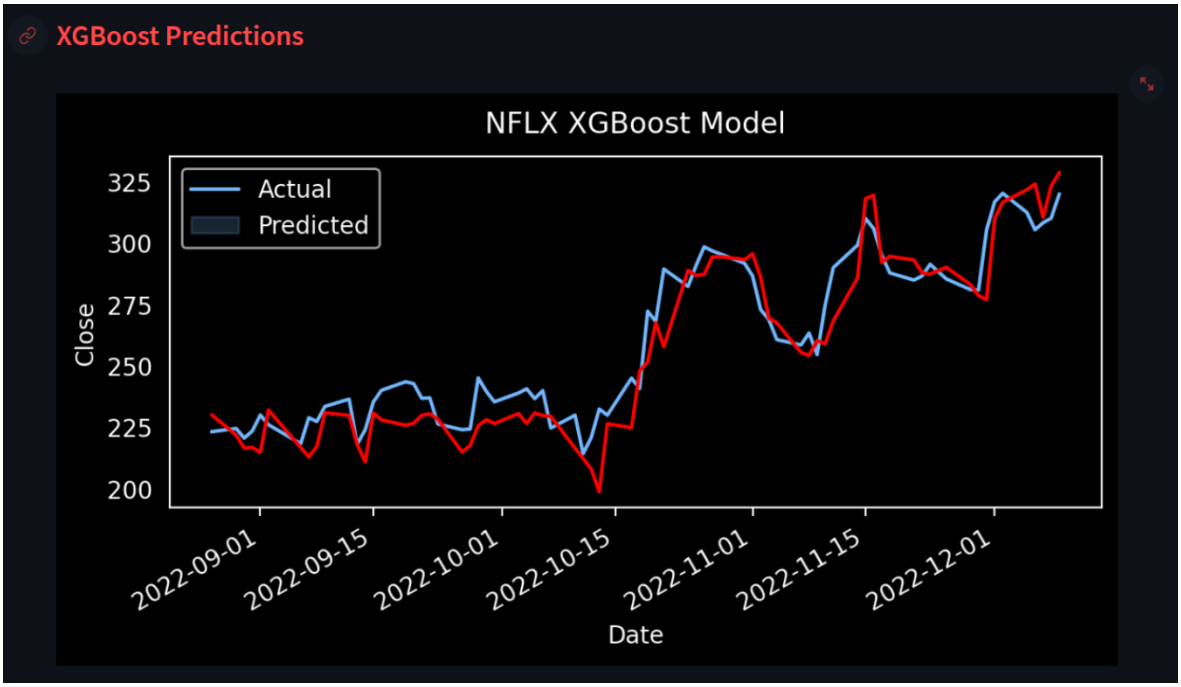


Fig 9.2.1.15 XGBoost prediction

In fig prediction of Netflix stock from machine learning model i.e. XGBoost model is displayed. As can be seen from the fig, it shows the user both the stock's current price movement and its forecast price trend. And by looking at this graph, the customer may determine to choose whether or not buy the stock.

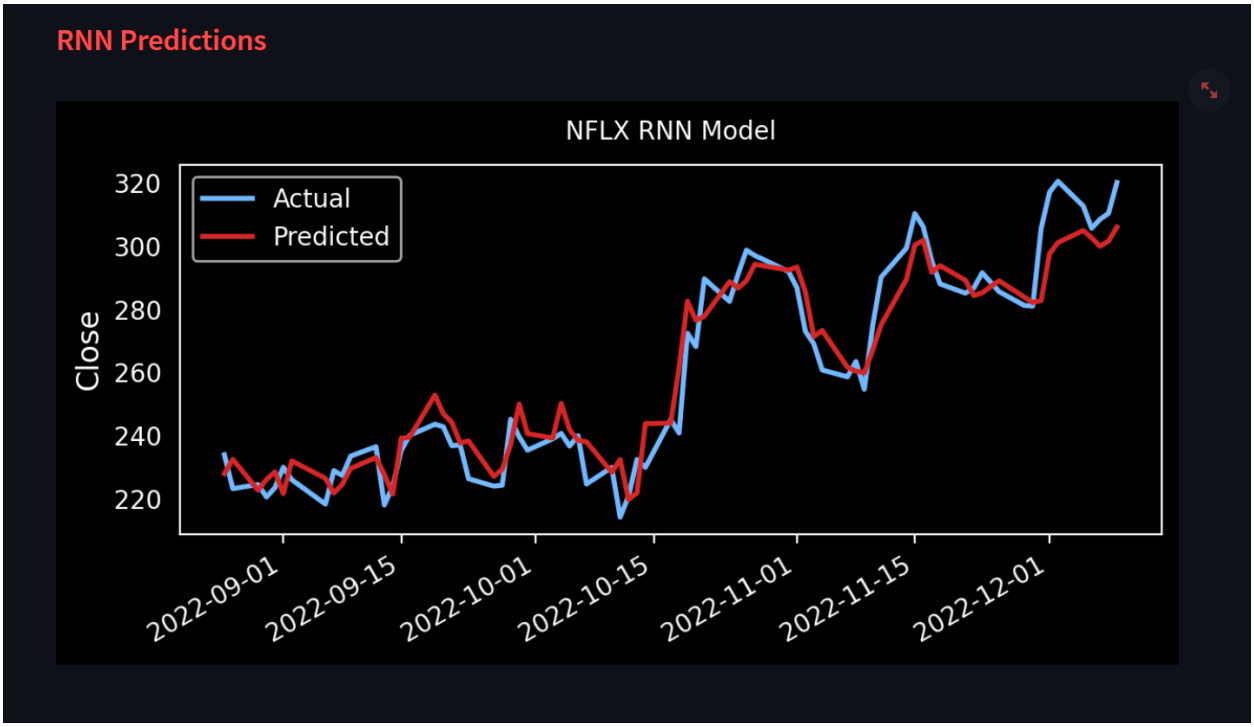


Fig 9.2.1.16 RNN prediction

In fig prediction of Netflix stock from machine learning model i.e. RNN model is displayed. As can be seen from the fig, it shows the user both the stock's current price movement and its forecast price trend. And by looking at this graph, the customer may determine to choose whether or not buy the stock

10. Summary

10.1 Conclusion

This project has launched a web application that shows real-time stock information, makes stock predictions for the future, and analyzes stock sentiment based on news and tweets that are now available. People today constantly go to the stock market to safely invest their money and to profit from it because it is a volatile and risky market. As a result, there is a constant need for tools to help people properly invest in the stock market. This stock market app helps new and experienced investors to invest securely.

The application was built from the ground up in an effort to understand the challenge and solve it through the application. After understanding the structure, officially evaluating the use cases, and completing all application requirements, the authors developed the implemented solution from scratch, whether it be the front end or the back end of the application.

Numerous experiments were carried out before, during, and after the implementation, in addition to the test cases we previously described on the chapter (Tests and Results). In the application We chose to employ hyperparameter tuning for LSTM,RNN and XGBoost.And we have discovered the ideal parameters for models using grid search CV, and we implemented them as a result. We employ such variables, tested on several markets and found which are effective with the majority of the stocks that we provide. LSTM and RNN was successfully implement, however XGBoost had some underfitting, but with the aid of Grid Search CV, we were able to identify the ideal parameters for XGBoost that are effective for the majority of stocks.

For sentiment analysis, we evaluated the news and tweet data quality to see if it was reliable and relevant to stocks or not. as it turned out, it exceeded our expectations. Along with data quality checks, we also examined the data we obtained from the data sources we fetched using pandas. The majority of the news and tweets were accurately analysed. After we thoroughly reviewed the polarity scores several times to see whether NLTK was supplying the right polarity score or not.

We have also been effective in implementing a number of techniques that provide stock technical analysis.

The proper testing was done Everything that we tested and found to be in functioning order and satisfied all of the specifications and standards.This demonstrates that the team project's goal was met.

It is planned to develop the application in the future and make it more robust. In chapter(10.2) the future direction of development is discussed .

It is difficult to reach a definitive conclusion about the effectiveness of system.Our system use advanced machine learning and natural language processing techniques to analyze large amounts of

data and identify trends, but the accuracy of their predictions can vary. In general, our system can provide valuable insights and help users make informed decisions, but they should not be relied upon solely for making investment decisions. It is important for investors to do their own research and carefully evaluate the risks and potential rewards of any investment.

10.2 Future direction of development

There are many potential directions for future work on our project. One direction could be to improve the performance of the deep learning and machine learning algorithms used for prediction. This could involve fine-tuning the hyperparameters of the algorithms, incorporating additional data sources, or using more advanced algorithms such as deep reinforcement learning. Another potential direction can be to expand the scope of the sentiment analysis to include more data sources and a wider range of emotions and sentiments. This could help the system to better capture the dynamics of the market. Another potential direction can be to explore the use of other natural language processing techniques, such as topic modeling and named entity recognition, to further enhance the analysis of the textual data. Another potential direction can be to improve the deep learning and machine learning algorithms used for prediction to give out pridiction for upcoming 30 days. Additionally in the web application we can add more technical analysis and make all the charts pretty intractive.

Overall, there are many opportunities for future work on this project, and the potential for continued improvement is exciting.

USER DOCUMENTATION

1.Introduction

This chapter covers the instructions to run the source code.

2.System requirements

This subchapter covers the system requirements and installation procedure of the softwares which is required for running the project.

Requirement 1:

Minimum System Requirements:

Processor: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, 2496 Mhz, 4 Core(s)

System Type: x64-based PC

OS Name: Microsoft Windows 10 Home

Physical Memory (RAM): 8.00 GB

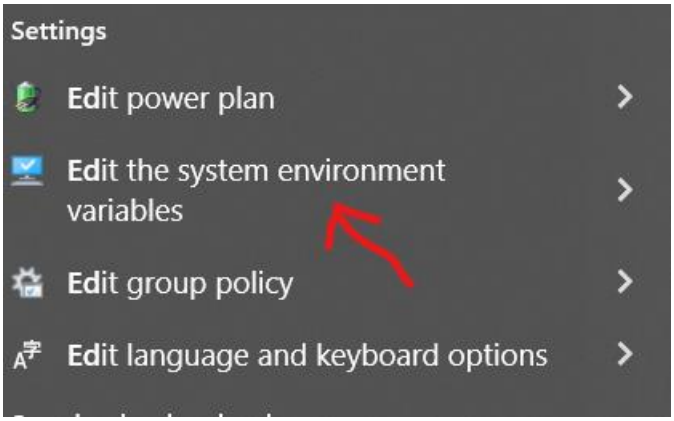
Graphics Card: Nvidia 1050 GTX

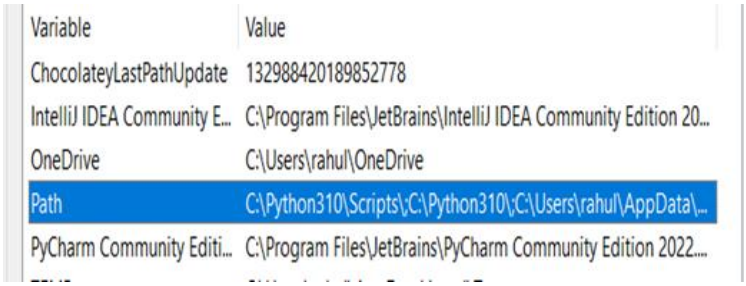
Requirement 2:

Python Version Required: 3.10.0 and above

-> **https://www.python.org/downloads/**

Click on the link, Download Python and Install in your system, once installed set python into env variable path





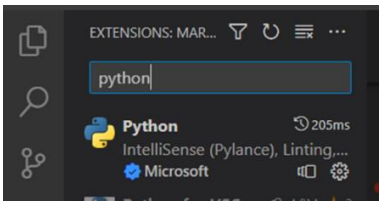
Variable	Value
ChocolateyLastPathUpdate	132988420189852778
IntelliJ IDEA Community E...	C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 20...
OneDrive	C:\Users\rahul\OneDrive
Path	C:\Python310\Scripts\;C:\Python310\;C:\Users\rahul\AppData\...
PyCharm Community Editi...	C:\Program Files\JetBrains\PyCharm Community Edition 2022....

Requirement 3 :

Visual Studio Version: 1.74.0 and above

-> **https://code.visualstudio.com/**
Download visual studio from above link and install in the system

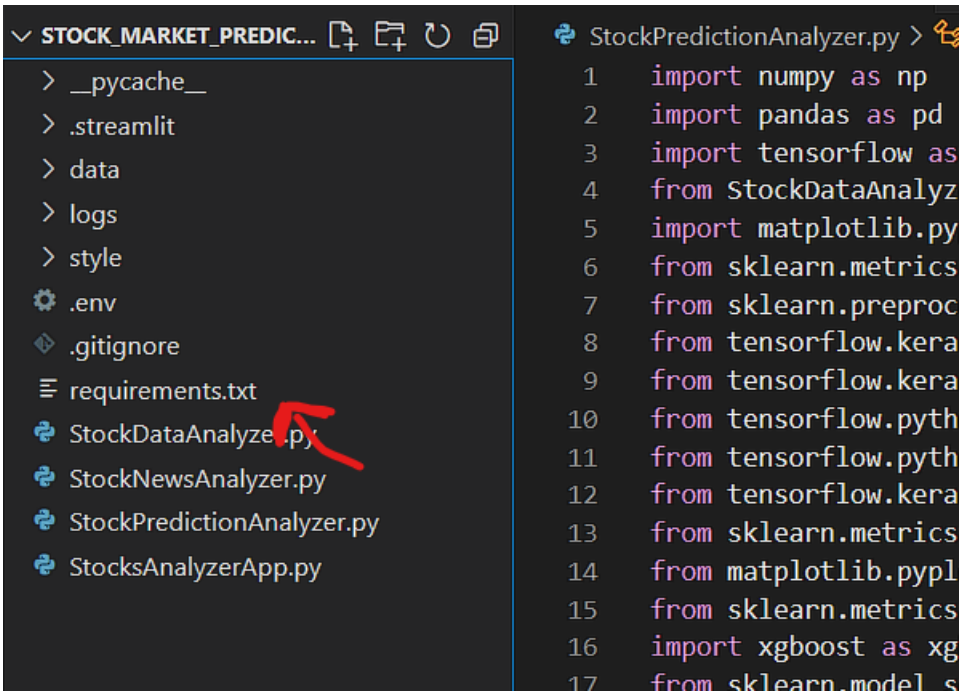
Once downloaded and installed visual studio code. Open visual studio code and download all required Python Extension in Visual Studio from Extension MarketPlace and Let the Visual Studio Code detect the python interpreter.

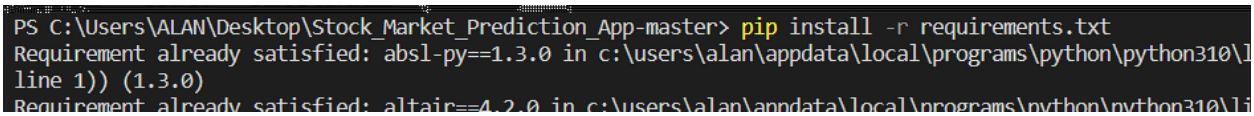


3.Installation procedure

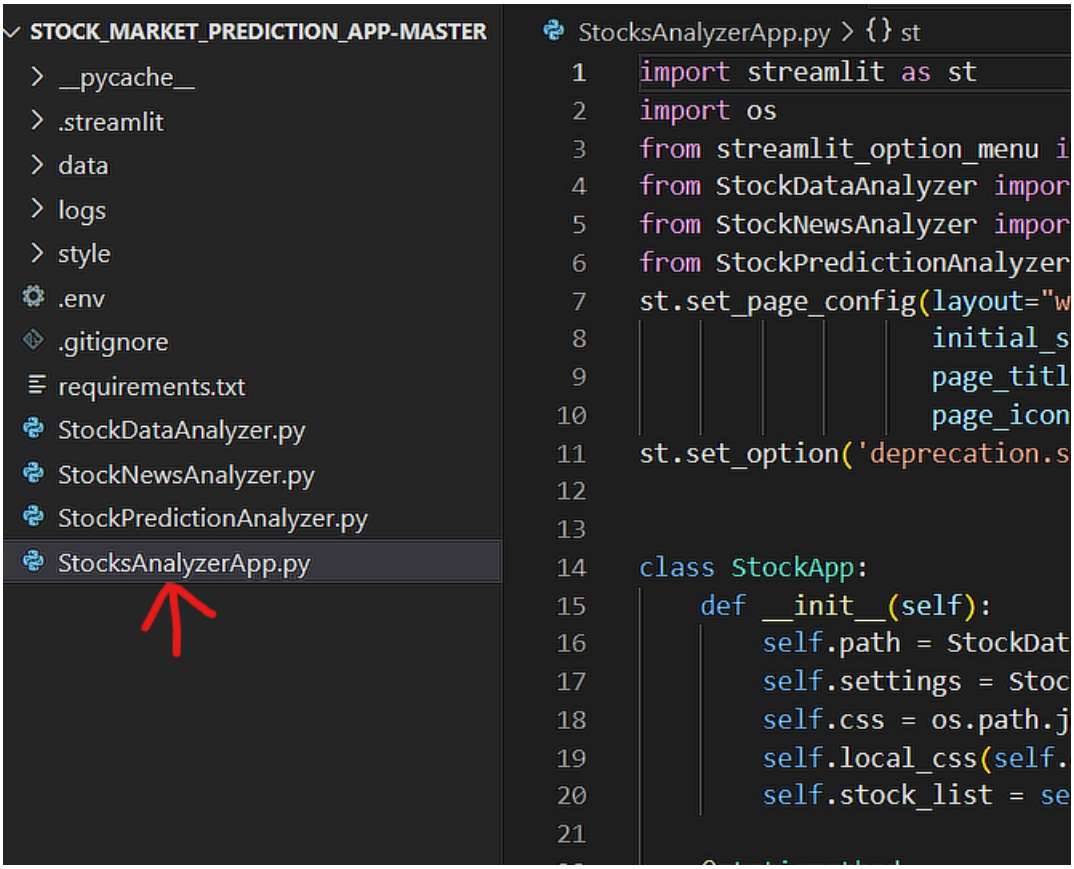
This subchapter covers installation procedure of all the libraries required and detailed information about how to run the source code.

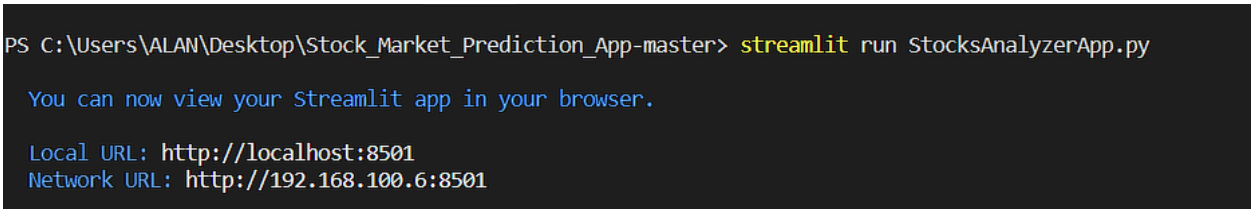
once all set application folder has requirement.txt file, install requirement.txt into system





Path to application\ **pip install -r requirements.txt** -> use this command to install all the required modules





-> path to application\ **streamlit run StocksAnalyzerApp.py**

use this command like shown in image and run streamlit app in the terminal, once all the requirements met completely

*note- use only python IDE and python IDE terminal to excecute all commands and to run application

*note- for smooth running of the applicationa and installation procedure use visual studio code

References

- International Research Journal of Engineering and Technology (IRJET)
- International academic conferences on management and economics
- Indian Institute of Management project taken as reference by Naliniprava Tripathy
- Avrim L, and Pat Langley, 1997. Selection of relevant features and examples in
- machine learning, Artificial Intelligence, 97(1–2):.245-271
- Chen Wun-Hua., and Shih Jen Ying., 2006. Comparison of support vector machines
- and back propagation neural networks in forecasting the six major Asian stock markets,
- International Journals Electronics Finance, 1(1):49-67.
- Huang, W., Nakamori, Y. and Wang, S.Y. 2005. Forecasting Stock Market Movement
- Direction with Support Vector Machine, Computers and Operations Research,
- 32(10):2513–2522
- Huang, Z., Chen, H., Hsu, C. J., Chen, W. H., and Wu, S. 2004.Credit rating analysis
- with support vector machines and neural networks: a market comparative study.
- Decision support systems, 37(4):543-558
- Hsu S. H., Hsieh J. J. P. A., Chih, T. C., and Hsu, K. C. 2009. A two-stage architecture
- for stock price forecasting by integrating self-organizing map and support vector
- regression. Expert Systems with Applications, 36(4):7947–7951
- Gavrishchaka Valeriy V. and Supriya B. Ganguli, 2003. Volatility forecasting from
- multiscale and high-dimensional market data, Neurocomputing, 55(1-2):285-305
- Fan, A. and Palaniswami,M., 2001. Stock selection using support vector machines,
- IJCNN'01: International Joint Conference on Neural Networks, 3:1793-1798
- Keerthi S. S.Duan, K.B., Shevade, S.K., Poo, A.N., 2005. A Fast Dual Algorithm for
- Kernel Logistic Regression, Machine Learning, 61(1–3):151–165.
- https://www.markettrak.com/about.html