Machine learning based Stock Predictor

Zagazig University
Faculty of Computer Science and Information
Information Technology And Computer Science Department

2020 - 2021

Supervisor
Prof. Dr. Ahmed Salah

ZAGAZIG University

Faculty of computers and Informatics

Machine learning based stock predictor

Supervised By

Prof.Dr. Ahmed Salah

Eng/Mahmoud Mahdi

Department of computer science

With

Department of Information technology

Faculty of Computers and informatics –ZAGAZIG University

2020-2021

Prepared By:

- ✓ Amr El-Sayed El-Hossini
- ✓ Ashraf Mohamed El-Nagar
- ✓ Hossam Hamed Saad
- ✓ Karim Mohamed Abo-Elazm
- ✓ Mahmoud Mohamed Ateia
- ✓ Mohamed Magdy Saber
- ✓ Mohamed Fawzi Elsaidy

The success of this project depends largely on the encouragement and guidelines of our professor and team work of all the members

Acknowledgement

It gives our great pleasure to present this progress report on ‘Machine Learning Stock Market Predictions’. We are grateful to the Faculty of Computers and Informatics, ZAGAZIG University, especially to our doctor (Dr. Ahmed Salah) for providing us and giving us this wonderful opportunity to do this project, which has invaluable assistance to our project with his advice and suggestions, and we would like to introduce a special thanks to (Dr.Ahmed Fathallah) who helped us a lot in this project. We would like to thank also (Dr. Mohamed Abdel Basset), Head of the Department .We also extend our thanks and appreciation to all our professors in the Departments of Computer Science CS) and Information Technology (IT) at ZAGAZIG University.

Abstract

Stock price forecasting is a popular and important topic in financial and academic studies. Share Market is an untidy place for predicting since there are no significant rules to estimate or predict the price of a share in the share market. Many methods like technical analysis, fundamental analysis, time series analysis, and statistical analysis, etc. are all used to attempt to predict the price in the share market but none of these methods are proved as a consistently acceptable prediction tool.

In this project, we **implemented a web site and Application** to predict stock market prices and **optimized** a Predictive Modeling and Technical Indicators Analysis approach by developing an automated stock data collection and predictive analysis tools addition to using **the sentiment analysis approach**, after that, we **made voting** between all of them to select the best accuracy.

Predictive Modeling is very effectively implemented in forecasting stock prices, returns, and stock modeling, and the most frequent methodologies are **LSTM**, **Transformer**, **Linear regression**, **Ridge regression**, **SVR**, **Prophet,** and the **ARIMA** models.

Learned Skills

Skills Gained From This Project

- How to collect information about the problem.
- How to analyze information.
- How to select sufficient information.
- How to solve the problem.
- How to deal with external environment.
- Deal with new technologies.
- Increase our programming skills specially in machine learning.
- How to read international researches and add your enhancement notes.
- How to be a member of teamwork, and we learned the experience of co-operation.
- How to search about a specific information, tool, technique or technology that may help us in achieving our project by reading books, exploring the internet, or asking anyone.
- • Requirements of the market.

Table of Contents

1 Introduction

1.1 Introduction

Stock Market prediction and analysis is the act of trying to determine the future value of company stock or other financial instrument traded on an exchange. Both investors and industry are involved in the stock market and want to know whether some stock will rise or fall over a certain period of time. The stock market is the primary source for any company to raise funds for business expansions. It is based on the concept of demand and supply. If the demand for a company's stock is higher, then the company share price increases and if the demand for the company's stock is low then the company share price decrease. Due to the involvement of many industries and companies, it contains very large sets of data from which it is difficult to extract information and analyze their trend of work manually. Stock market analysis and prediction will reveal the market patterns and predict the time to purchase stock.

The successful prediction of a stock's future price could yield **significant profit**. so, we implemented a website and Application to predict stock market prices and optimized a Predictive Modeling and Technical Indicators Analysis approach by developing an automated stock data collection and predictive analysis tools in addition to using sentiment analysis, after that, we made voting between all of them to select the best accuracy.

1.2 Problem definition

The stock market is very vast and difficult to understand. It is being considered too uncertain to a predictable due to huge fluctuation of the market. The stock market prediction task is interesting as well as divides researchers and academics into two groups, those who believe that we can devise mechanisms to predict the market and those who believe that the market is efficient and whenever new information comes up the market absorbs it by correcting itself, thus there is no space for prediction.

Investing in a good stock but at a bad time can have disastrous results, while investing in a stock at the right time can bear profits. Financial investors of today are facing this problem of trading as they do not properly understand which stocks to buy or which stocks to sell in order to get optimum results. Investors prefer stock market investments as they have the opportunity of highest return over other schemes. The need for an **automated user-friendly trading predictor system**, which predicts stock price upward/downward movements, is necessary for the stock market, given the explosion of algorithmic trading, being one of the most prominent trends in the global financial industry.

1.3 Documentation structure

In chapter one, we are introducing a brief about our project(the idea and the problem definition).
In chapter two, you will get a brief about the concept of trading and the machine learning models used in forecasting. In chapter three the planning phase in the project, how we started, and how we got the requirements to begin the Implementation of this project.
After that, you will go through to see the system analysis of the project in chapter four. the methodology that we applied, the diagrams of what is the input of the system, and what is the output.
In chapter five you will find the machine learning model's implementation and its results also, the implementation of the website, and the application.
chapter six, we are trying to make a prediction for stock prices based on the tweets on Twitter using the sentiment analysis technique.
In chapter seven, the conclusion of the project and the future work. at the end(*chapter eight*) the references of our articles and the scientific papers that we have been read.

2 ML and Trading overview

2.1 Trading overview

The first step to understanding the stock market is knowing the lingo. Here are a few commonly used words and phrases:

- **Earnings per Share**: The total company profit divided by the number of stock shares outstanding.
- **Going Public**: Slang for when a company plans to have an IPO of its stock.
- **IPO**: Short for Initial Public Offering, when a company sells its shares of stock for the first time.
- **Market Cap**: Short for Market Capitalization, the amount of money you would have to pay if you bought every single share of stock in a company. To calculate market cap, multiply the number of shares by the price per share.
- **Share**: A share, or a single common stock, represents one unit of an investor's ownership in a share of the profits, losses, and assets of a company. A company creates shares when it carves itself into pieces and sells them to investors in exchange for cash.
- **Ticker Symbol**: A short group of letters that represents a particular stock as listed on the stock market. For example, The Coca-Cola Company has a ticker symbol of KO, and Johnson & Johnson has a ticker symbol of JNJ.
- **Underwriter**: The financial institution or investment bank that does all of the paperwork and orchestrates a company's IPO.

2.1.1 Introduction to the Stock Market

The workings of the stock market can be confusing. Some people believe investing is a form of gambling and feel that, if you invest, you will likely end up losing your money.

These fears can stem from the personal experiences of family members and friends who suffered similar fates or lived through the Great Depression. These feelings are understandable but aren't

grounded in facts. Someone who believes in this line of thinking may not have an in-depth understanding of the stock market, why it exists, and how it works.

2.1.2 Investing by Following the Crowd

Other people believe that they should invest for the long run but don't know where to begin. Before learning about how the stock market works, they look at investing like some sort of magic that only a few people know how to use. More often than not, they leave their financial decisions up to professionals and cannot tell you why they own a particular stock or mutual fund.

This investment style could be called blind faith, or perhaps it's limited to a sentiment such as, "This stock is going up—we should buy it." Though it may not seem so on the surface, this group is in far more danger than the first. They tend to invest by following the masses and then wonder why they only achieve mediocre, or, in some cases, devastating results.

2.1.3 Learning to Invest

Upon learning a few techniques, the average investor can evaluate the balance sheet of a company and, following a few relatively simple calculations, arrive at their own interpretation of the real value of a company and its stock.

This allows an investor to look at a stock and know that it is worth, for instance, $40 per share. This gives each investor the freedom to determine when the market has undervalued stocks, increasing their long-term returns substantially, or overvalued it, making it a poor investment candidate.

Why Do Companies Sell Stock?

When learning how to value a company, it helps to understand the nature of a business and the stock market. Almost every large corporation started as a small, mom-and-pop operation and, through growth, became a financial giant.

Consider Walmart, Amazon, and McDonald's. Walmart was originally a single-store business in Arkansas. Amazon.com began as an online bookseller in a garage. McDonald's was once a small restaurant that no one outside of San Bernardino, California, had ever heard of. How did these small companies grow from tiny, hometown enterprises to three of the largest businesses in the American economy? They raised capital by selling stocks.

2.1.4 Making the Decision to Sell Shares

Ten years later, the business has grown rapidly. The couple has managed to pay off the company's debt and have profits of more than $500,000 per year. Convinced that ABC Furniture could do as well in several larger neighboring cities, the couple decides they want to open two new branches.

They research their options and find out that they need over $4 million to expand. Not wanting to borrow money and make debt and interest payments again, they decide to raise funds by offering equity to potential shareholders, so they sell stock in their company.

2.1.5 Deciding How Much of Their Business to Sell

The young couple, now in their 30s, must decide how much of the company they are willing to sell. Right now, they own 100% of the business. The more company shares they sell, the more cash they'll raise, but they must keep in mind that by selling more, they'll be giving up a larger part of their ownership. As the company grows, that ownership will be worth more, so a wise entrepreneur would not sell more than he or she had to.

After discussing it, the couple decides to keep 60% of the company and sell the other 40% to the public as stock. When you do the math, this means that they will keep $7.8 million worth of the business (60% of the $13 million value). Because they own a majority of the stock, greater than 50%, they will still be in control of the store.

The other 40% of the stock that they want to sell to the public has a value of $5.2 million. The underwriter finds investors who want to buy the stock and gives a check for $5.2 million to the couple.

2.1.6 The Benefits of Selling and Owning Shares

With this example, it's easy to see how small businesses seem to explode in value when they go public. The original owners of the company, in a sense, become wealthier overnight. Before, the amount they could take out of the business was limited to the profit that was generated. Now, the owners can sell their shares in the company at any time, raising cash quickly.

This process forms the basis of Wall Street. The stock market functions as a large auction where ownership in companies just like ABC Furniture is sold to the highest bidder each day. Because of human nature and the emotions of fear and greed, a company can sell for far more or for far less than its intrinsic value. A good investor learns to identify those companies currently selling below their true worth so that they can buy as many shares as possible.

2.2 ML and statistical methods for forecasting

Machine Learning (ML) methods have been proposed in the academic literature as alternatives to statistical ones for time series forecasting. Yet, scant evidence is available about their relative performance in terms of accuracy and computational requirements. The purpose of this paper is to

evaluate such performance across multiple forecasting horizons using a large subset of 1045 monthly time series used in the M3 Competition. After comparing the post-sample accuracy of popular ML methods with that of eight traditional statistical ones, we found that the former are dominated across both accuracy measures used and for all forecasting horizons examined. Moreover, we observed that their computational requirements are considerably greater than those of statistical methods. The paper discusses the results, explains why the accuracy of ML models is below that of statistical ones and proposes some possible ways forward. The empirical results found in our research stress the need for objective and unbiased ways to test the performance of forecasting methods that can be achieved through sizable and open competitions allowing meaningful comparisons and definite conclusions.

We all must have heard that people are saying that the price of different objects has decreased or increased with time, these different objects could be anything like petrol, diesel, gold, silver, edible things, etc.

Another example is, the rate of interest fluctuates in banks and different for different kinds of loans. What are all this data, how useful it is? These types of data are time-series data that go through analysis for forecasts.

Because of the tremendous variety of conditions, time-series analysis is used by both nature and human beings for communication, description, and data visualizations.

Also, time is the physical quantity, and elements, coefficients, parameters, and characteristics of time-series data are mathematical quantities, so time-series can have real-time or real-world interpretations as well.

A **time series data** is the set of measurements taking place in a constant interval of time, here time acts as independent variable and the objective (to study changes in a characteristics) is dependent variables.

For example, one can measure

- Consumption of energy per hour
- Sales on daily basis
- Company's profits per quarter
- Annual changes in a population of a country.

The time series data is of three types:

- **Time series data:** A set of observations contains values, taken by variable at different times.
- **Cross-sectional data:** Data values of one or more variables, gathered at the same time-point.
- **Pooled data:** A combination of time series data and cross-sectional data.

Time-Series Analysis through various mode of data visualization

2.2.1 ML Methods For Time-Series Forecasting

1. In the ***Univariate Time-series Forecasting method***, forecasting problems contain only two variables in which one is time and the other is the field we are looking to forecast.

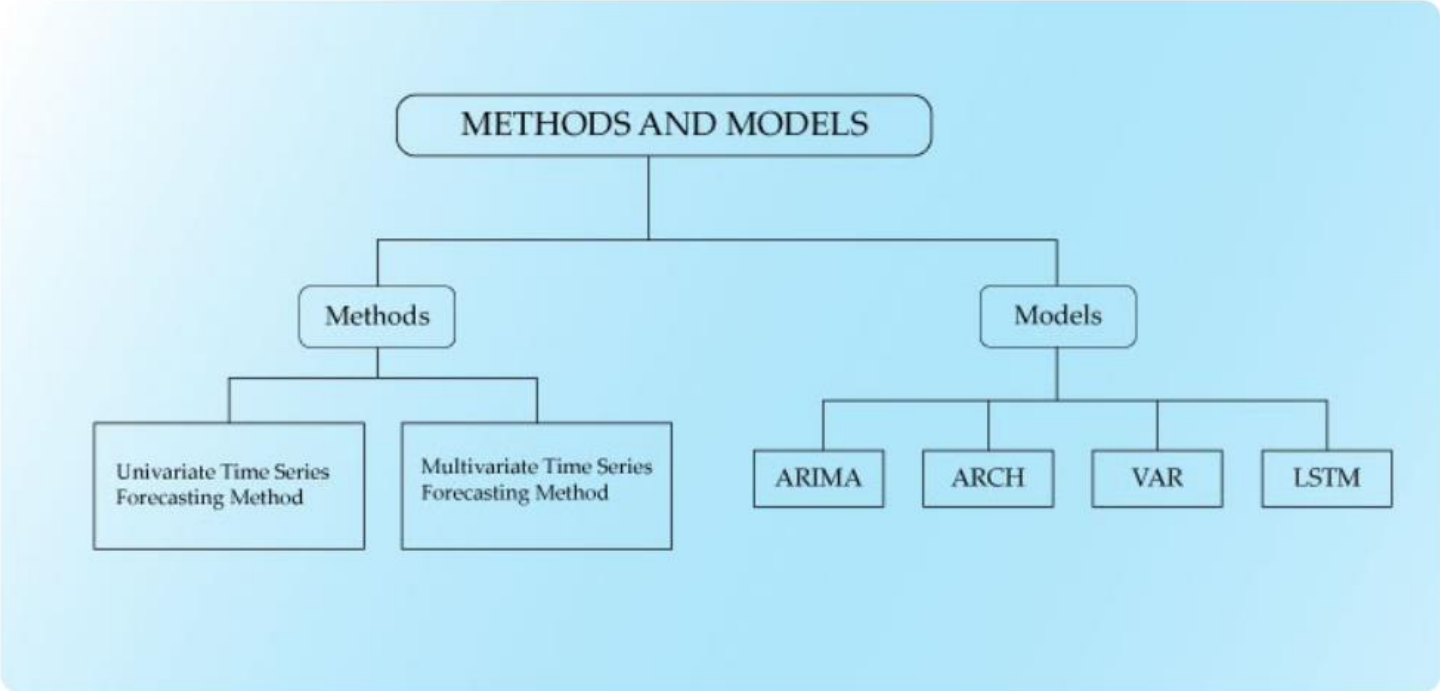
- For example, if you want to predict the mean temperature of a city for the coming week, now one parameter is time(week) and the other is a city.
- Another example could be when measuring a person's heart rate per minute through using past observations of heart rate only. Now one parameter is time(minute) and the other is a heart rate.

2. On the other hand, in the ***Multivariate Time-series Forecasting method***, forecasting problems contain multiple variables keeping one variable as time fixed and others will be multiple in parameters.

Consider the same example, predicting the temperature of a city for the coming week, the only difference would come here now temperature will consider impacting factors such as

- *Rainfall and time duration of raining,*
- *Humidity,*
- *Wind speed,*
- *Precipitation,*
- *Atmospheric pressure, etc,*

And then the temperature of the city will be predicted accordingly. All these factors are related to temperature and impact it vigorously.



Time-Series Forecasting: Methods and Models in Machine Learning

- **ARIMA Model:** As mentioned in the above section, it is a combination of three different models itself, AR, MA and I, where

  1. "AR" reflects the evolving variable of interest is regressed on its own prior values,
  2. "MA" infers that the regression error is the linear combination of error terms values happened at various stages of time priorly, and
  3. "I" shows the data values are replaced by the difference between their values and the previous values.

  Combinedly "ARIMA" tries to fit the data into the model, and also ARIMA depends on the accuracy over a broad width of time series.

- **ARCH/GARCH Model:** Being the extended model of its common version GARCH, Autoregressive Conditional Heteroscedasticity (ARCH) is the most volatile model for time series forecasting, and are well trained for catching dynamic variations of volatility from time series.

- **Vector Autoregressive Model or VAR model:** It gives the independencies between various time-series data which as a generalization of the Univariate Autoregression Model.

- **LSTM:** Long-short term memory(LSTM)is a deep learning model, it is a kind of Recurrent Neural Network(RNN) to read the sequence dependencies.

  It enables us to handle long structures during training the dataset and creates predictions according to previous data.

3 Proposed System

3.1 Planning

This chapter discusses how organizations evaluate and select projects to undertake from the many available projects. Once a project has been selected, the project manager plans the project. Project management involves selecting a project methodology, creating the project work plan, identifying project staffing requirements, and preparing to manage and control the project. These steps produce important project management deliverables, including the work plan, staffing plan, standards list, project charter, and risk assessment.

3.1.1 Project plan

Task	Period by (weeks)	Deliverables	Notes
Knowledge acquisition: **· RNN and LSTM.** **Transformers model** **Moving average and ARIMA models**	6	**1. A working model for each algorithm on the standard dataset.** **2. A presentation of each algorithm (10-15 slides each).**	**You should deliver models for standard datasets I will provide to you.**
Data collection Phase: **1. quandl python** **2. MetaTrader** **3. yfinance**	4	**A working code that takes the finical instrument name the time window as inputs, and downloaded the corresponding data as output.**	**We need to compare the data collected by these packages. If we can find better package that is a plus.**
UI: **· Tradingview app/web.** **· NetDania** **· Trade Interceptor.** **· Bloomberg Business Mobile App.** **· thinkorswim Mobile.**	4	**· User/market research report.** **· UX wireframe.**	**We need to study the design of these app/web applications to design our own solution UI.**

Apply the facebook Prophet model to our data.	1		***https://www.kaggle.com/janiobachmann/s-p-500-time-series-forecasting-with-prophet***
Sentiment Analysis:	2	**Understand the existing codes in Kaggle or github and select the best one. Then, try to modify the code to set it in the best form.**	
Ensemble learning:	1	**We should try the tools in the note section.**	
Tuning the performance	3		

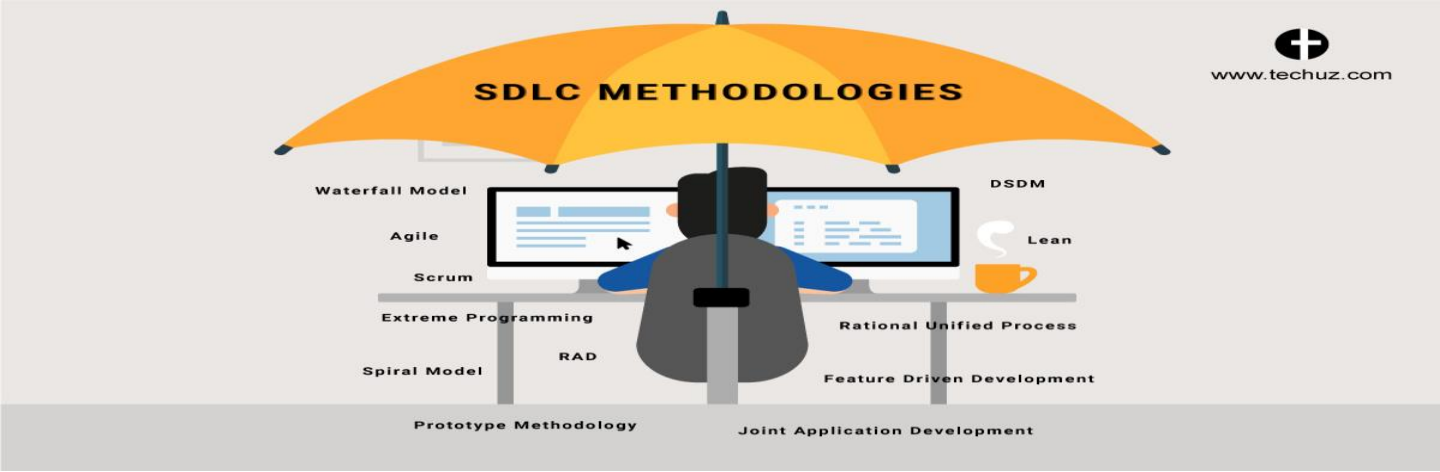
3.2 Scientific papers and articles

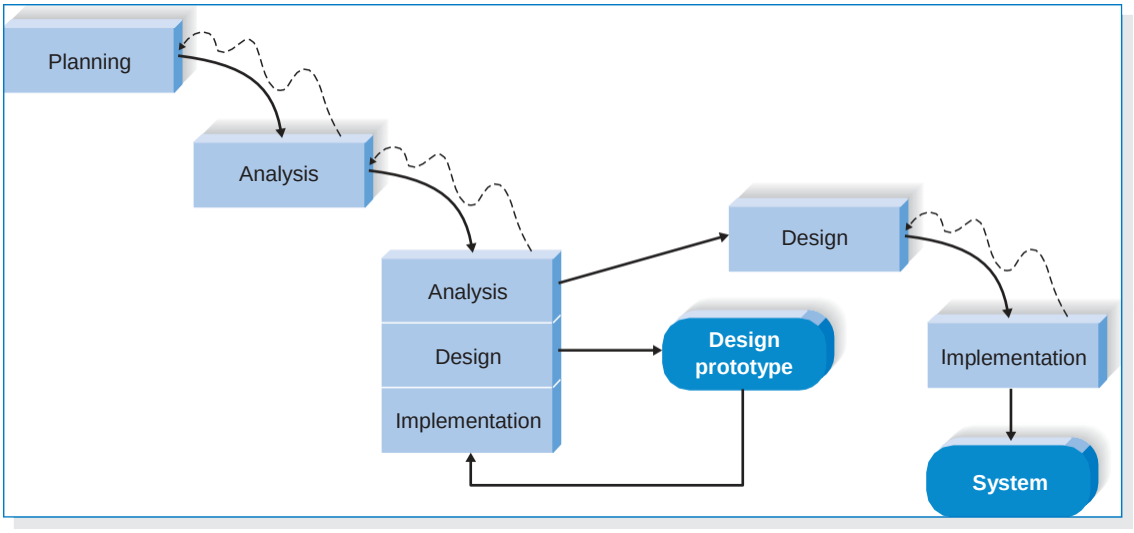
we have prepared for this project and established a solid background which helped us very much to implement the tasks in this project.

1. **RNN and LSTM. [1]**
2. **Transformers model.[2]**
3. **Moving average and ARIMA models.[3]**
4. **Prophet model.[4]**
5. **Sentiment Analysis.[5]**
6. **Ensemble learning.[6]**

3.3 Project Methodology Options

The Systems Development Life Cycle (SDLC) provides the foundation for the processes used to develop an information system. A *method- ology* is a formalized approach to implementing the SDLC (i.e., it is a list of steps and deliverables). There are many different systems development methodologies, and they vary in terms of the progression that is followed through the phases of the SDLC. Some methodologies are formal standards used by government agencies, while others have been developed by consulting firms to sell to clients. Many organizations have their own internal methodologies that have been refined over the years, and they explain exactly how each phase of the SDLC is to be performed in that company. Here we will review several of the predominant methodologies that have evolved over time.


SDLC METHODOLOGIES
www.techuz.com
Waterfall Model
Agile
Scrum
Extreme Programming
Spiral Model
RAD
Prototype Methodology
DSDM
Lean
Rational Unified Process
Feature Driven Development
Joint Application Development


Planning
Analysis
Analysis
Design
Implementation
Design
Design prototype
Implementation
System

Throwaway prototyping

Includes the development of prototypes, but uses the prototypes primarily to explore design alternatives rather than as the actual new system (as in system prototyping). As shown in the Figure above, throwaway prototyping has a fairly thorough analysis phase that is used to gather requirements and to develop ideas for the system concept. Many of the features suggested by the users may not be well understood, however, and there may be challenging technical issues to be solved. Each of these issues is examined by analyzing, designing, and building a *design prototype.* A design prototype is not intended to be a working system. It contains only enough detail to enable users to understand the issues under consideration. For example, suppose that users are not completely clear on how an order entry system should work.

A system that is developed by this type of methodology probably requires several design prototypes during the analysis and design phases. Each of the prototypes is used to minimize the risk associated with the system by confirming that important issues are understood before the real system is built. Once the issues are resolved, the project moves into design and implementation. At this point, the design proto- types are thrown away, which is an important difference between this approach and system prototyping, in which the prototypes evolve into the final system.

As the previous section shows, there are many methodologies. The first challenge faced by project managers is to select which methodology to use. Choosing a methodology is not simple, because no one methodology is always best. (**If it were, we'd simply use it everywhere!**) Many organizations have standards and policies to

guide the choice of methodology. You will find that organizations range from hav- ing one “approved” methodology to having several methodology options to having no formal policies at all.

The Figure below summarizes some important methodology selection criteria. One important item not discussed in this figure is the degree of experience of the analyst team. Many of the RAD and agile development methodologies require the use of new tools and techniques that have a significant learning curve. Often, these tools and tech- niques increase the complexity of the project and require extra time for learning. Once they are adopted and the team becomes experienced, the tools and techniques can sig- nificantly increase the speed in which the methodology can deliver a final system.

Usefulness in Developing Systems	Waterfall	Parallel	V-Model	Iterative	System Prototyping	Throwaway Prototyping	Agile Development
with unclear user requirements	Poor	Poor	Poor	Good	Excellent	Excellent	Excellent
with unfamiliar technology	Poor	Poor	Poor	Good	Poor	Excellent	Poor
that are complex	Good	Good	Good	Good	Poor	Excellent	Poor
that are reliable	Good	Good	Excellent	Good	Poor	Excellent	Good
with short time schedule	Poor	Good	Poor	Excellent	Excellent	Good	Excellent
with schedule visibility	Poor	Poor	Poor	Excellent	Excellent	Good	Good

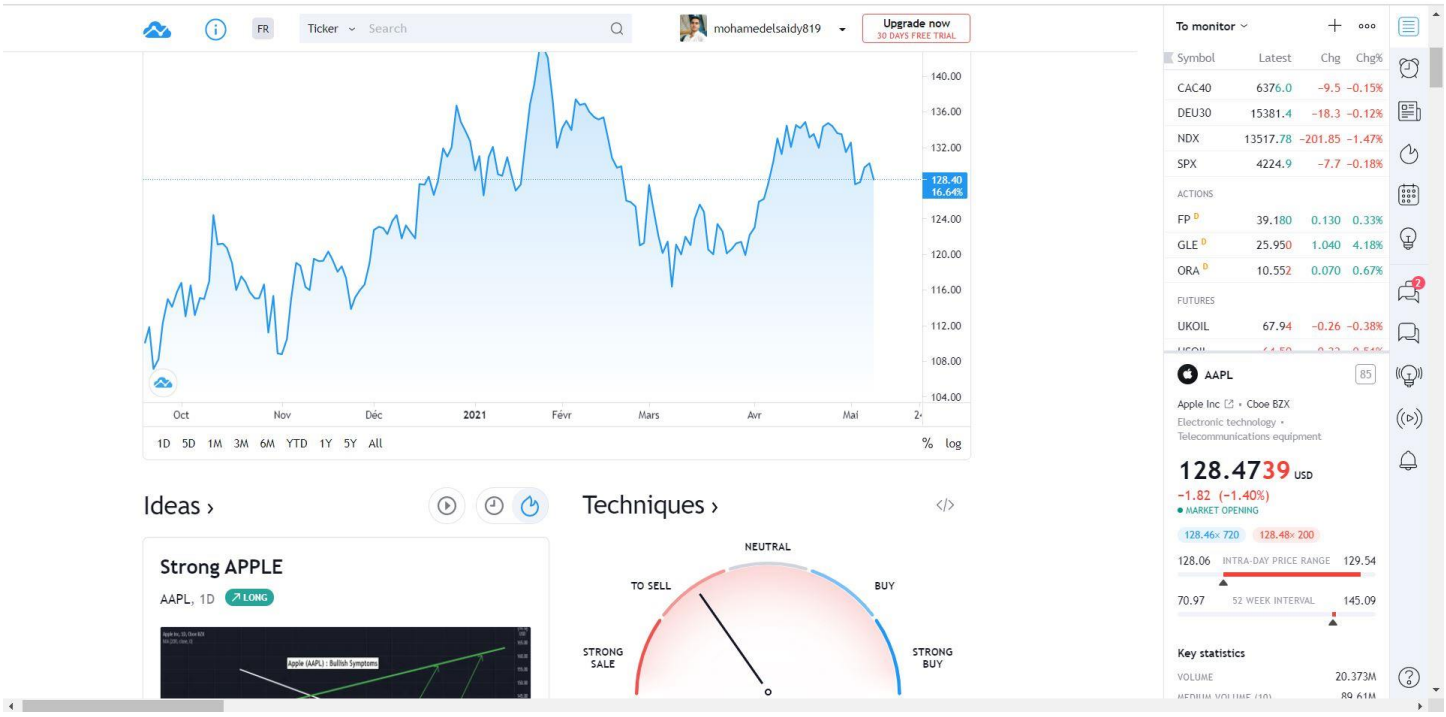
➢ **Our prototyping**

TradingView is a cloud-based charting and social-networking software for both beginner and advanced active investment traders. Basic charting, research, and analysis information are available with a free account. Still, most trades must be made outside the platform because only select brokerages are linked to TradingView at this time.

We took a close look at TradingView to see how it works. Here's what we found.

3.3 TradingView Features

Price	$0-59.95/month
Charting	Yes
Research and Analysis	Yes
Notifications and Alerts	Yes
Access	Web, iOS, Android
Customer Service	Live Chat: 24/7
Promotions	None



3.4 Estimating the Project Time Frame

As the previous section illustrated, some development methodologies have evolved in an attempt to accelerate the project through the SDLC as rapidly as possible while still producing a quality system. Regardless of whether time is a critical issue on a project or not, the project manager will have to develop a preliminary estimate of the amount of time the project will take. *Estimation*16 is the process of assigning projected values for time and effort.

There are **two basic ways** to estimate the time required to build a system. The simplest method uses the amount of time spent in the planning phase to predict the time required for the entire project. The idea is that a simple project will require little planning, and a complex project will require more planning; so using the amount of

time spent in the planning phase is a reasonable way to estimate overall project time requirements.

	Planning	Analysis	Design	Implementation
Typical industry standards for business applications	15%	20%	30%	35%
Estimates based on actual figures for first stage of SDLC	Actual: 7 person-months	Estimated: 2 person-months	Estimated: 3 person-months	Estimated: 7 person-months

SDLC = systems development life cycle.

4 System Analysis

4.1 Introduction

Systems analysis is a process of collecting factual data, understand the processes involved, identifying problems and recommending feasible suggestions for improving the system functioning. This involves studying the business processes, gathering operational data, understand the information flow, finding out bottlenecks and evolving solutions for overcoming the weaknesses of the system so as to achieve the organizational goals. System Analysis also includes subdividing of complex process involving the entire system, identification of data store and manual processes. The major objectives of systems analysis are to find answers for each business process: What is being done, how is it being done, who is doing it, when is he doing it, why is it being done and How can it be improved? It is more of a thinking process and involves the creative skills of the System Analyst. It attempts to give birth to a new efficient system that satisfies the current needs of the user and has scope for future growth within the organizational constraints. The result of this process is a logical system design. Systems analysis is an iterative process that continues until a preferred and acceptable solution emerges.

4.2. Requirement Analysis

After the extensive analysis of the problems in the system, we are familiarized with the requirement that the current system needs. The requirement that the system needs is categorized into the functional and non-functional requirements. These requirements are listed below:

4.2.1 Functional Requirements

Functional requirement are the functions or features that must be included in any system to satisfy the business needs and be acceptable to the users. Based on this, the functional requirements that the system must require are as follows:

- The system should be able to generate an approximate share price.
- The system should collect accurate data from the Yahoo finance website in consistent manner.

4.2.2 Non-Functional Requirements

Non-functional requirement is a description of features, characteristics and attribute of the system as well as any constraints that may limit the boundaries of the proposed system. The non- functional requirements are essentially based on the performance, information, economy, control and security efficiency and services. Based on these
the non-functional requirements are as follows:

- The system should provide better accuracy.
- The system should have simple interface for users to use.
- To perform efficiently in short amount of time.

4.3 System Design

System Design Based on the user requirements and the detailed analysis of the existing system, the new system must be designed. This is the phase of system designing. It is the mostcrucial phase in the developments of a system. The logical system design arrived at as a resultof systems analysis is converted into physical system design.

Normally, **the design proceeds in two stages:**

- **Preliminary or General Design**

  In the preliminary or general design, the features of the new system are specified. The costs of implementing these features and the benefits to be derived are estimated. If the project is still considered to be feasible, we move to the detailed design stage.

- **Structured or Detailed Design**

  In the detailed design stage, computer-oriented work begins in earnest. At this stage, the design of the system becomes more structured. Structure design is a blue print of a computer system solution to a given problem having the same components and inter-relationships among the same components as the original problem. Input, output, databases, forms, codification schemes and processing specifications are drawn up in detail. In the design stage, the programming language and the hardware and software platform in which the new system will run are also decided.

One of the most powerful graphical notation to express the design of software projects is **UML**, It's a very important part of developing objects-oriented software and the software development process. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. It is a means for representing a system using some kind of graphical notation, which is now almost always based on notations in the (Unified Modeling Language).

A UML deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. Deployment diagrams is a kind of structure diagram used in modeling the physical aspects of

an object-oriented system. They are often being used to model the static deployment view of a system (topology of the hardware). Deployment diagrams are important for visualizing, specifying, and documenting embedded, client/server, and distributed systems and also for managing executable systems through forward and reverse engineering.

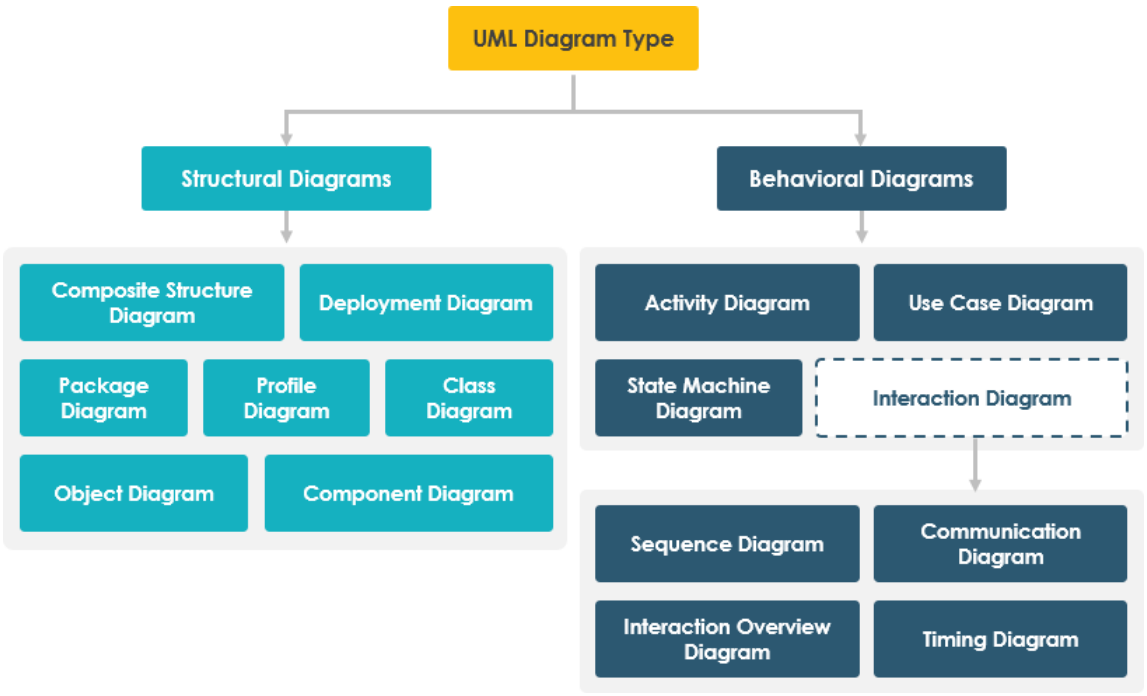


Figure 3.1 – UML types

In this chapter we will show steps of DS-ECG:

- Use case diagram.
- DFD diagram.
- ERD diagram.

Let's go to explore these Diagrams:

4.4 UMLs (Unified Modeling Language)

4.4.1 Use case diagram

To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating. Only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML, there are five diagrams available to model the dynamic nature and use case diagram is one of them. Now as we

have to discuss that the use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. Use case diagrams consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

Hence to model the entire system, a number of use case diagrams are used:

Purpose of use case diagrams

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and State chart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modelled to present the outside view in brief, the purposes of use case diagrams can be said to be as follows:

- Used to get an outside view of a system.
- Used to gather the requirements of a system.
- Show the interaction among the requirements are actors.
- Identify the external and internal factors influencing the system.

In the next picture, a use case for both the user and the system, and what functions and powers are allowed

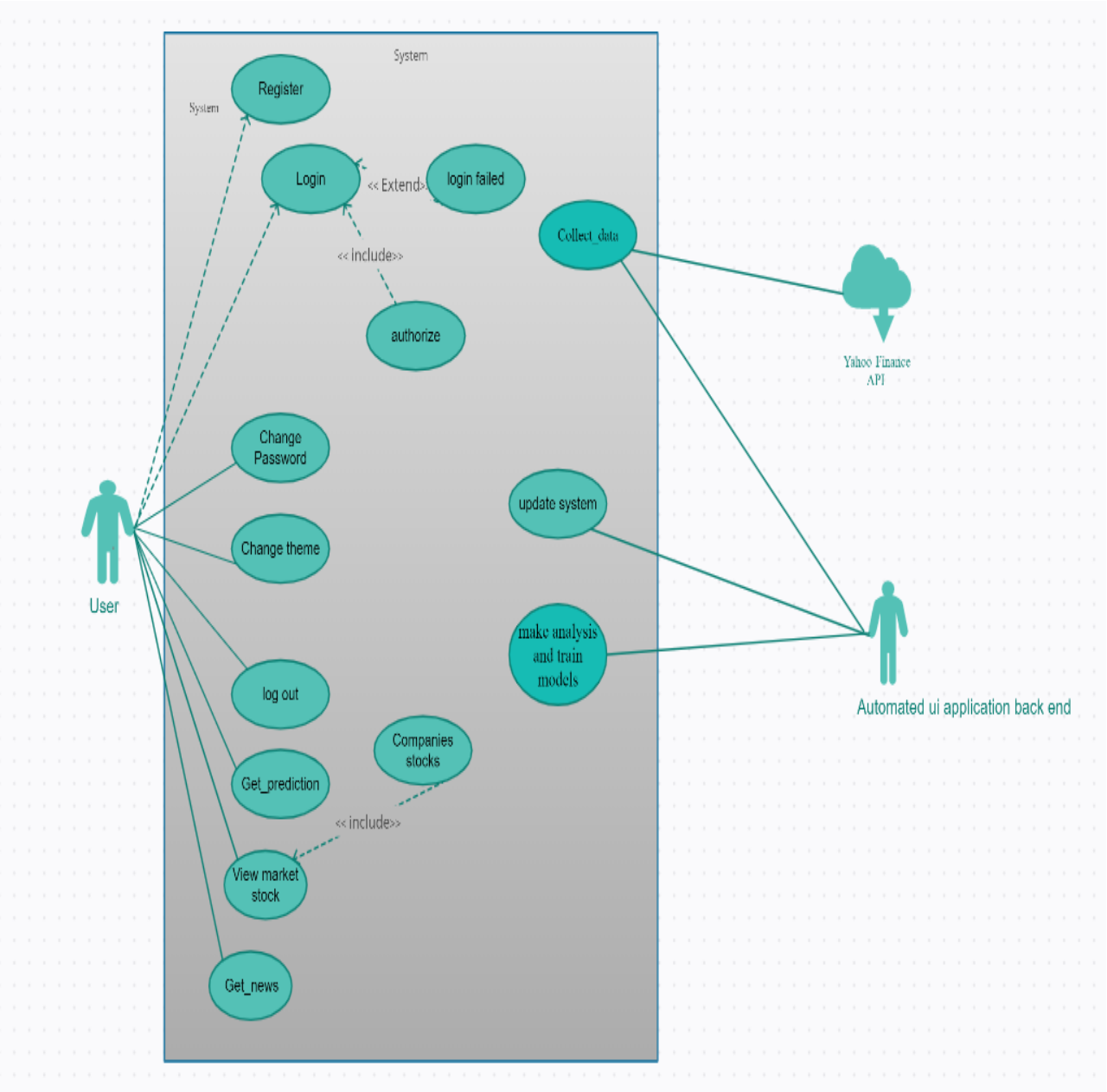


Figure 3.2 – user use case

Use case description:

Use Case Name: **Get prediction**	ID: **UC-1**	Priority: **High**
Actor: **User**		
Description: **After logging in to the site, the user chooses the name of the stock that he wants to expect to rise or fall during a certain period of time that he also determines, and then the site performs all operations based on well-trained models using artificial intelligence techniques to predict the share price**.		
Trigger: **User wants to make stock price forecast.** Type: ✓ External Temporal		
Preconditions: 1. **The user identity is authenticated.** 2. **Artificial intelligence models are well trained and give high accuracy.** 3. **The dataset is up-to-date and on-line.**		
Normal Course: **1.0 The user chooses the name of the stock that he wants to expect to rise or fall during a certain period of time that he also determines.** 1. **The site performs all operations based on well-trained models using artificial intelligence techniques to predict the share price**. 2. **He can go to follow the latest stock news.**		
Postconditions: 1. **The indicator gives the user a buy or sell advice based on the prediction produced by the model.**		
Exceptions: **1. The system asks the user if he wants to request another prediction or to exit** **2. The user asks to make another pred** **3. The system starts Normal Course again .** **4. The user asks to exit** **5. The system terminates the use case**		

4.4.2 DFD Diagram

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled.

They can be used to analyze an existing system or model a new one. Like all the best diagrams and charts, a DFD can often visually “say” things that would be hard to explain in words, and they work for both technical and nontechnical audiences, from developer to CEO. That’s why DFDs remain so popular after all these years. While they work well for data flow software and systems, they are less applicable nowadays to visualizing interactive, real-time or database-oriented software or systems.

Symbols and Notations Used in DFDs

Three common systems of symbols are named after their creators:

- Yourdon and Coad
- Yourdon and DeMarco
- Gane and Sarson

One main difference in their symbols is that Yourdon-Coad and Yourdon-DeMarco use circles for processes, while Gane and Sarson use rectangles with rounded corners, sometimes called lozenges. There are other symbol variations in use as well, so the

important thing to keep in mind is to be clear and consistent in the shapes and notations you use to communicate and collaborate with others.

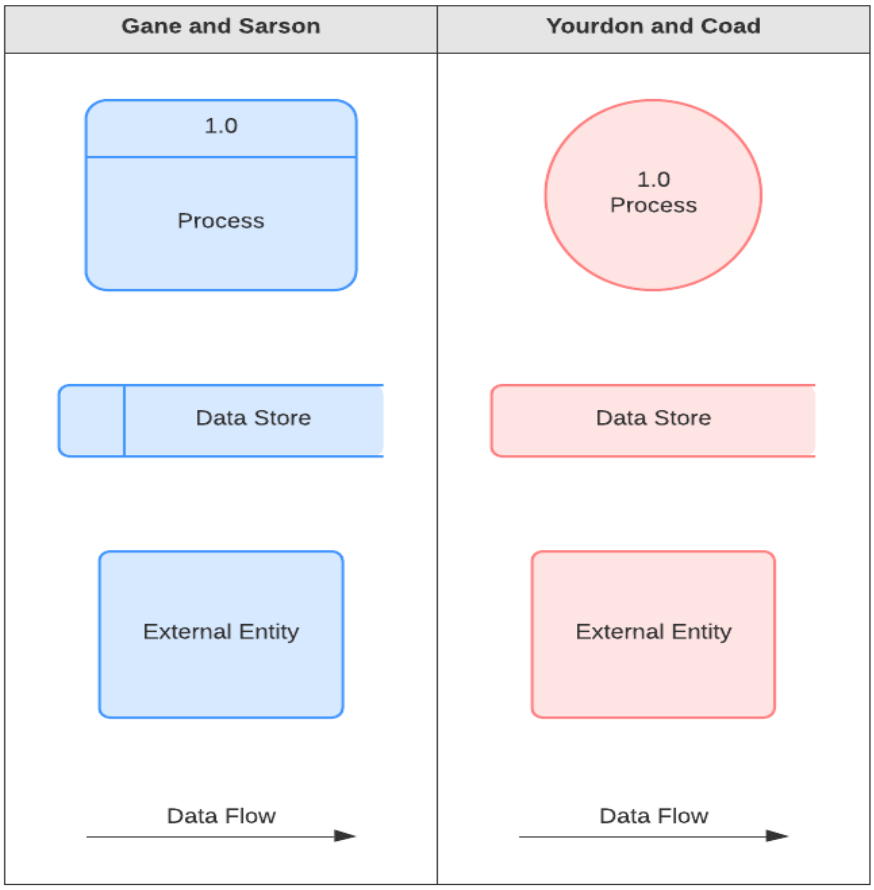
1. **External entity:** an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a

   business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

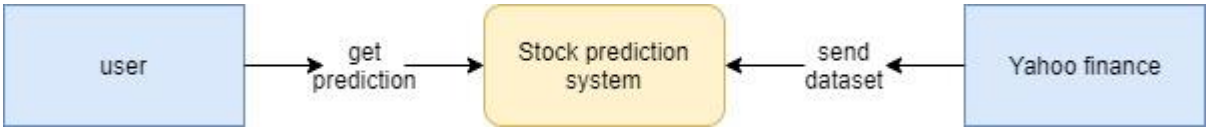
2. **Process:** any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules. A short label is used to describe the process, such as “Submit payment.”

3. **Data store:** files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label, such as “Orders.”

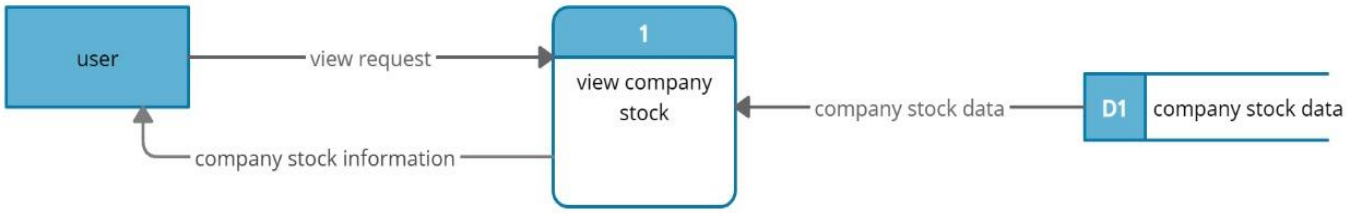
4. **Data flow:** the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like “Billing details”.

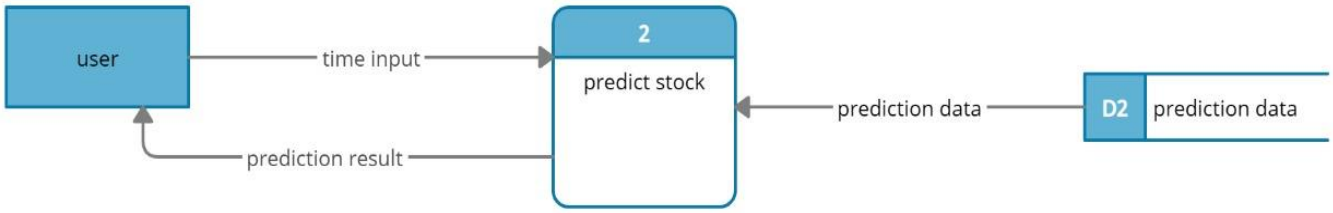


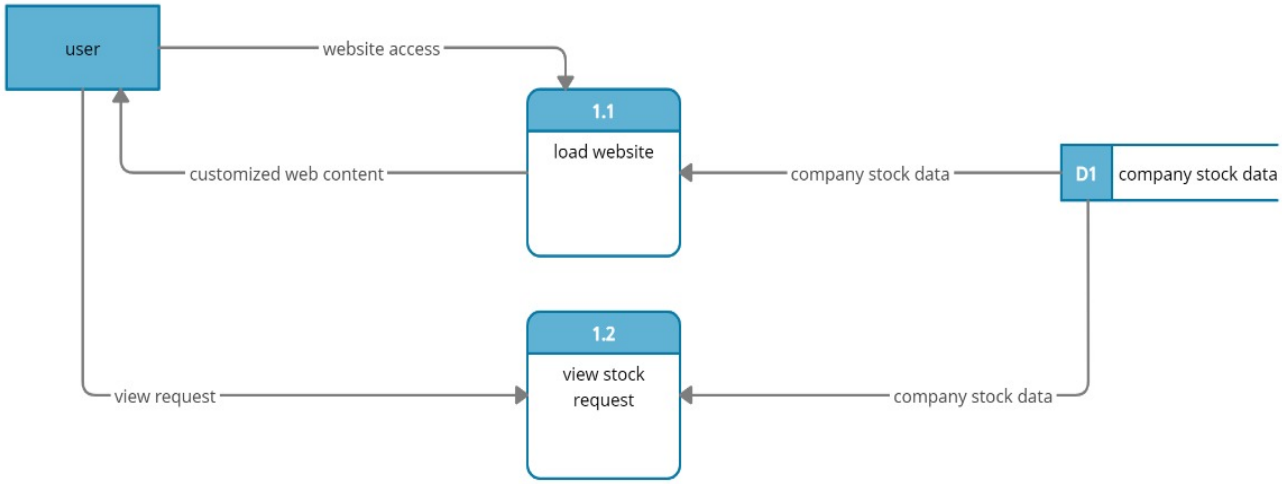
Data Flow Diagram (DFD)

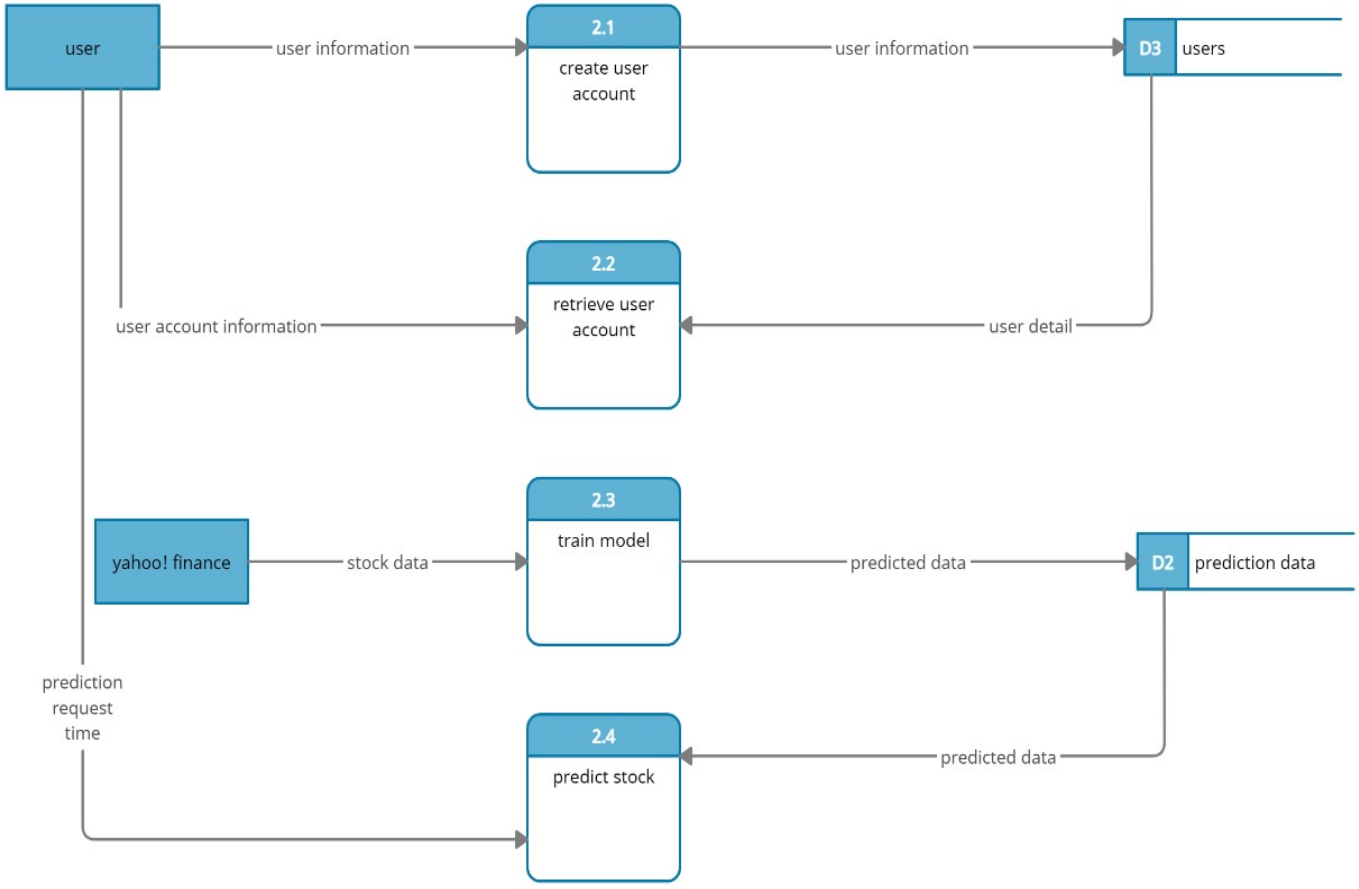


DFD level 0



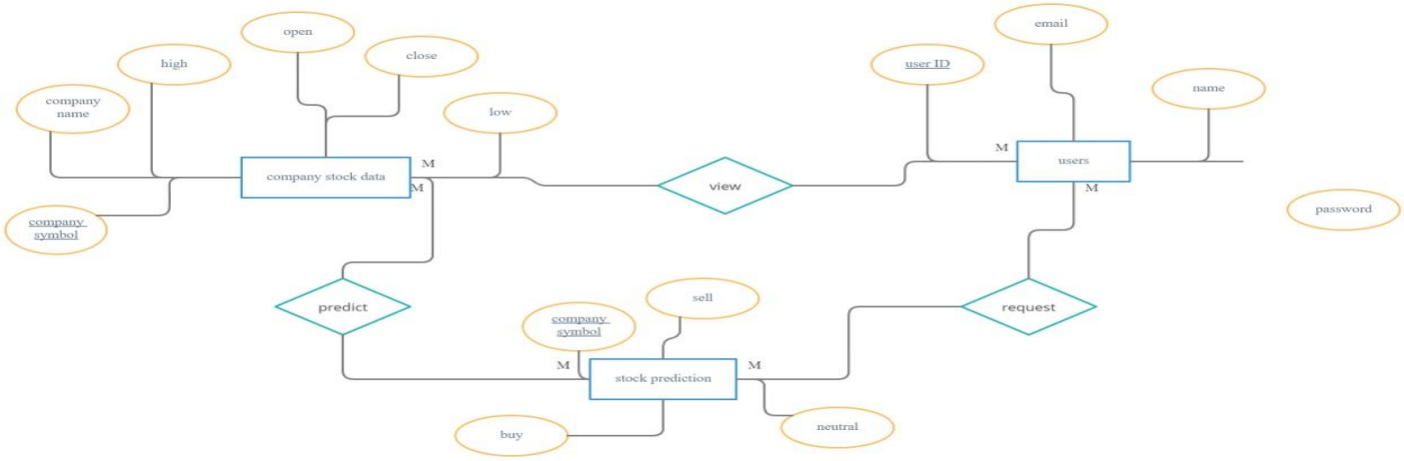



user
website access
1.1
load website
customized web content
company stock data
D1
company stock data
1.2
view stock request
view request
company stock data


user
user information
2.1
create user account
user information
D3
users
2.2
retrieve user account
user account information
user detail
yahoo! finance
stock data
2.3
train model
predicted data
D2
prediction data
prediction request time
2.4
predict stock
predicted data

4.4.3 ER Diagram:

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs. ER diagrams are related to data structure diagrams (DSDs), which focus on the relationships of elements within entities instead of relationships between entities themselves. ER diagrams also are often used in conjunction with data flow diagrams (DFDs), which map out the flow of information for processes or systems.



4.4.4 sequence diagram

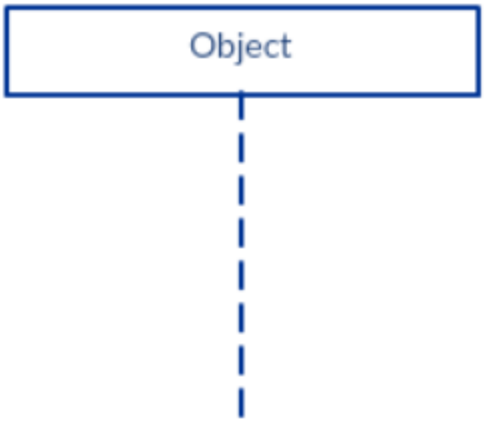
A **sequence diagram** shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development. Sequence diagrams are sometimes called **event diagrams** or **event scenarios**.

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

Sequence Diagram Notations

A sequence diagram is structured in such a way that it represents a timeline which begins at the top and descends gradually to mark the sequence of interactions. Each object has a column and the messages exchanged between them are represented by arrows.

Lifeline Notation



A sequence diagram is made up of several of these lifeline notations that should be arranged horizontally across the top of the diagram. No two lifeline notations should overlap each other. They represent the different objects or parts that interact with each other in the system during the sequence.

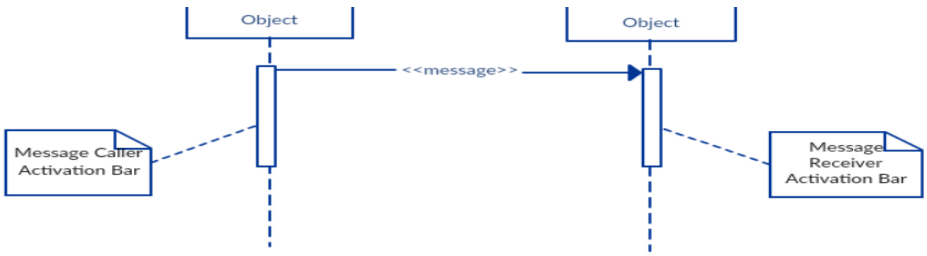
A lifeline notation with an actor element symbol is used when the particular sequence diagram is owned by a use case.

A lifeline with an entity element represents system data. For example, in a customer service application, the Customer entity would manage all data related to a customer.

Activation Bars

Activation bar is the box placed on the lifeline. It is used to indicate that an object is active (or instantiated) during an interaction between two objects. The length of the rectangle indicates the duration of the objects staying active.

In a sequence diagram, an interaction between two objects occurs when one object sends a message to another. The use of the activation bar on the lifelines of the Message Caller (the object that sends the message) and the Message Receiver (the object that receives the message) indicates that both are active/is instantiated during the exchange of the message.



Message Arrows

An arrow from the Message Caller to the Message Receiver specifies a message in a sequence diagram.A message can flow in any direction; from left to right, right to left or back to the Message Caller itself. While you can describe the message being sent from one object to the other on the arrow, with different arrowheads you can indicate the type of message being sent or received.

The message arrow comes with a description, which is known as a message signature, on it. The format for this message signature is below. All parts except the message_name are optional.

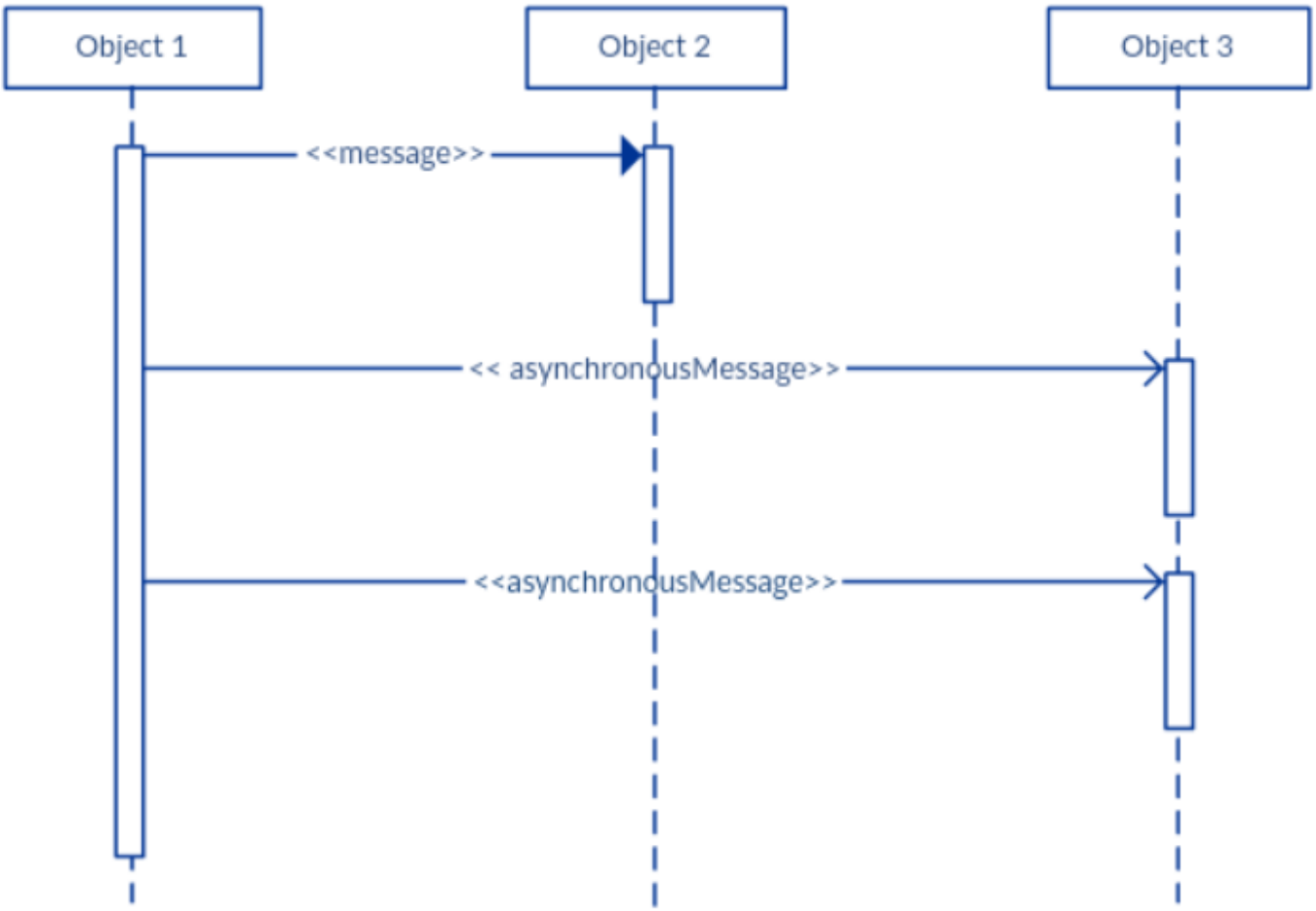
attribute = message_name (arguments): return_type

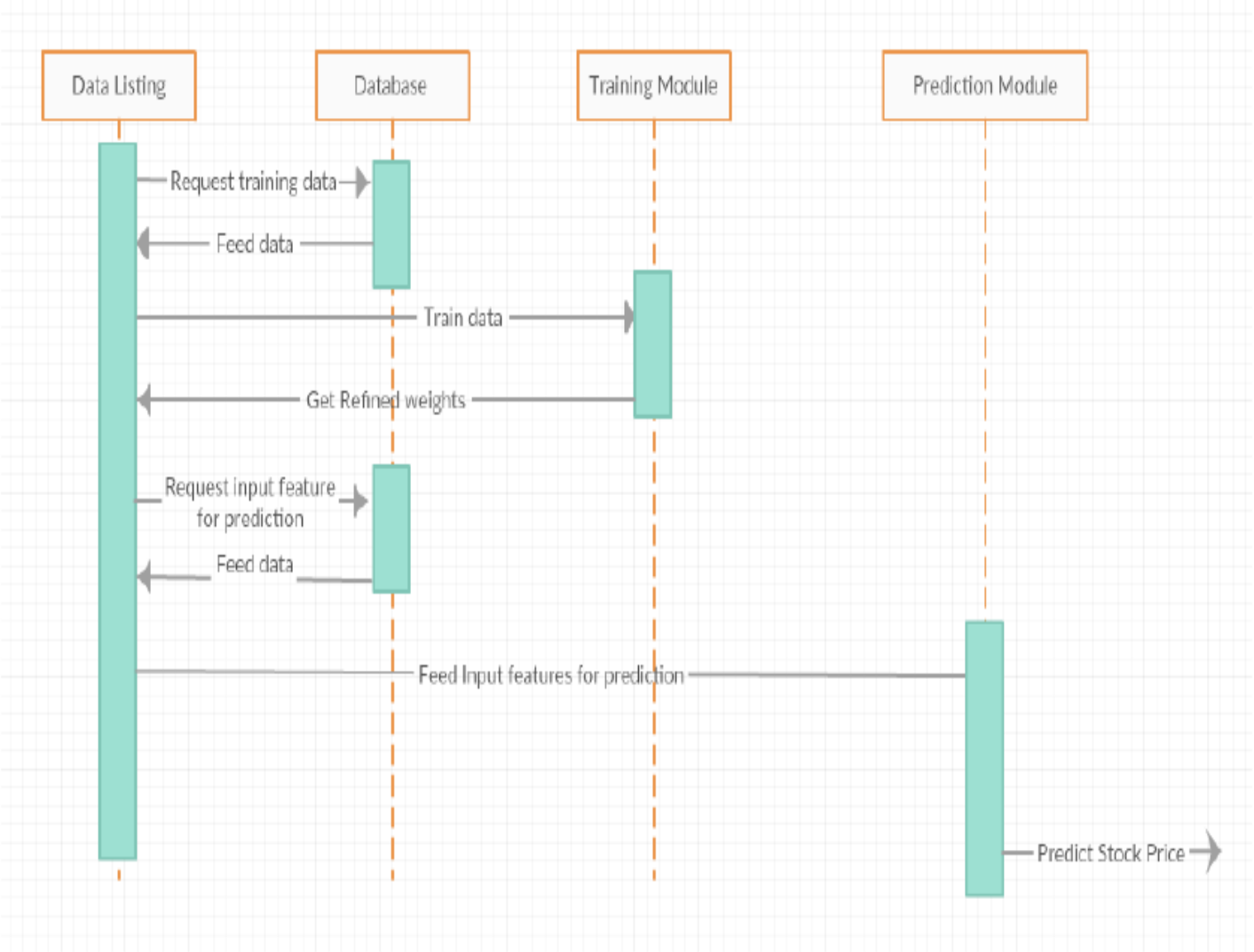
- ***Synchronous message***

As shown in the activation bars example, a synchronous message is used when the sender waits for the receiver to process the message and return before carrying on with another message. The arrowhead used to indicate this type of message is a solid one, like the one below.

Asynchronous message

An asynchronous message is used when the message caller does not wait for the receiver to process the message and return before sending other messages to other objects within the system. The arrowhead used to show this type of message is a line arrow like shown in the example below.



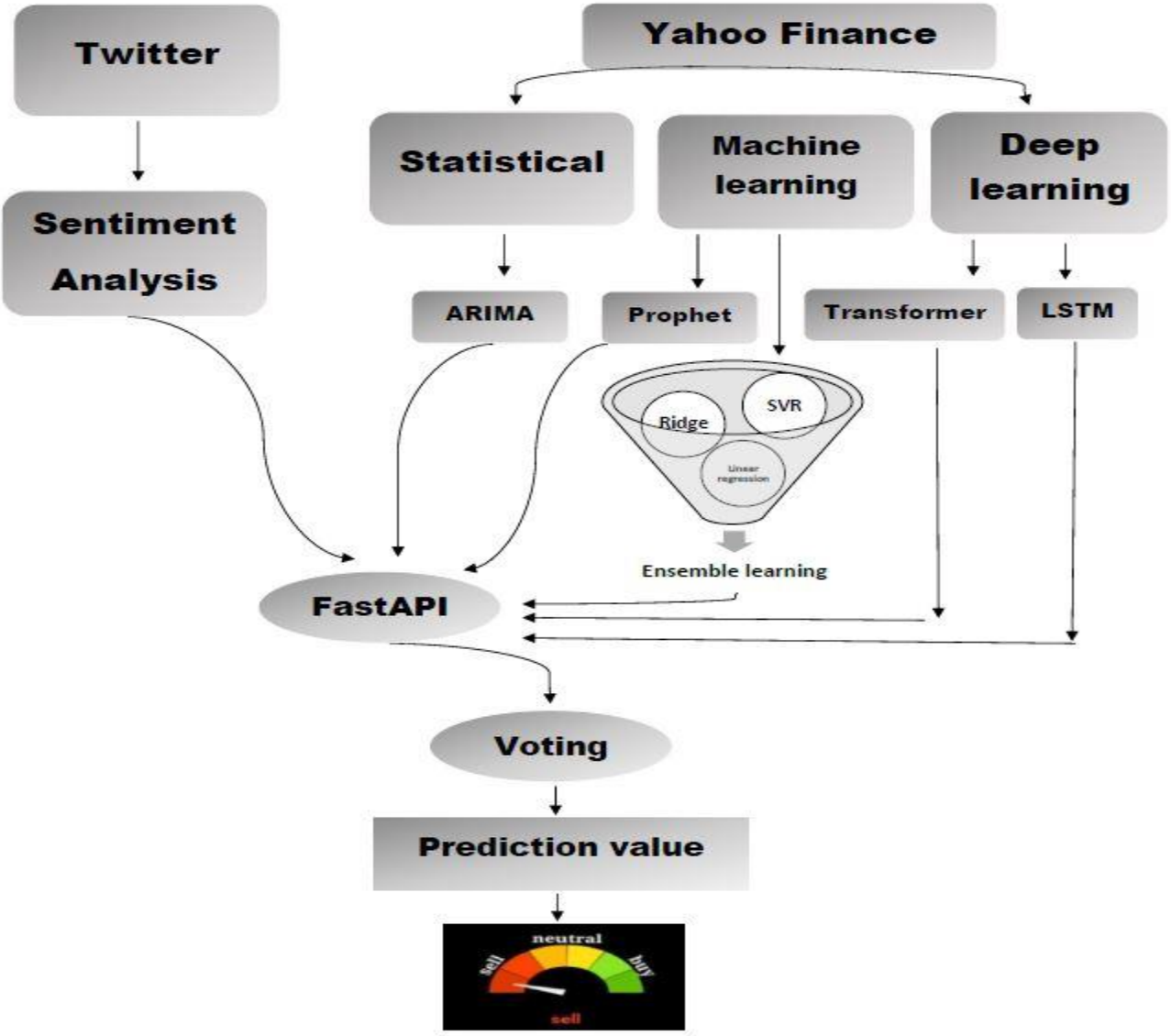


Sequence diagram for stock price prediction

5 Machine learning model's Implementation and results

we are trying here in this chapter to show the implementation of this project from a-z. first of all, we will show the implementation of machine learning team. after that, we will show the website implementation and finally the App.

5.1Machine learning



Block diagram of the ML Models

The programming languages and tools:

1. Python (programming language)

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

2.TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

Tensorflow is a symbolic math library based on dataflow and differentiable programming. It is used for both research and production at Google.

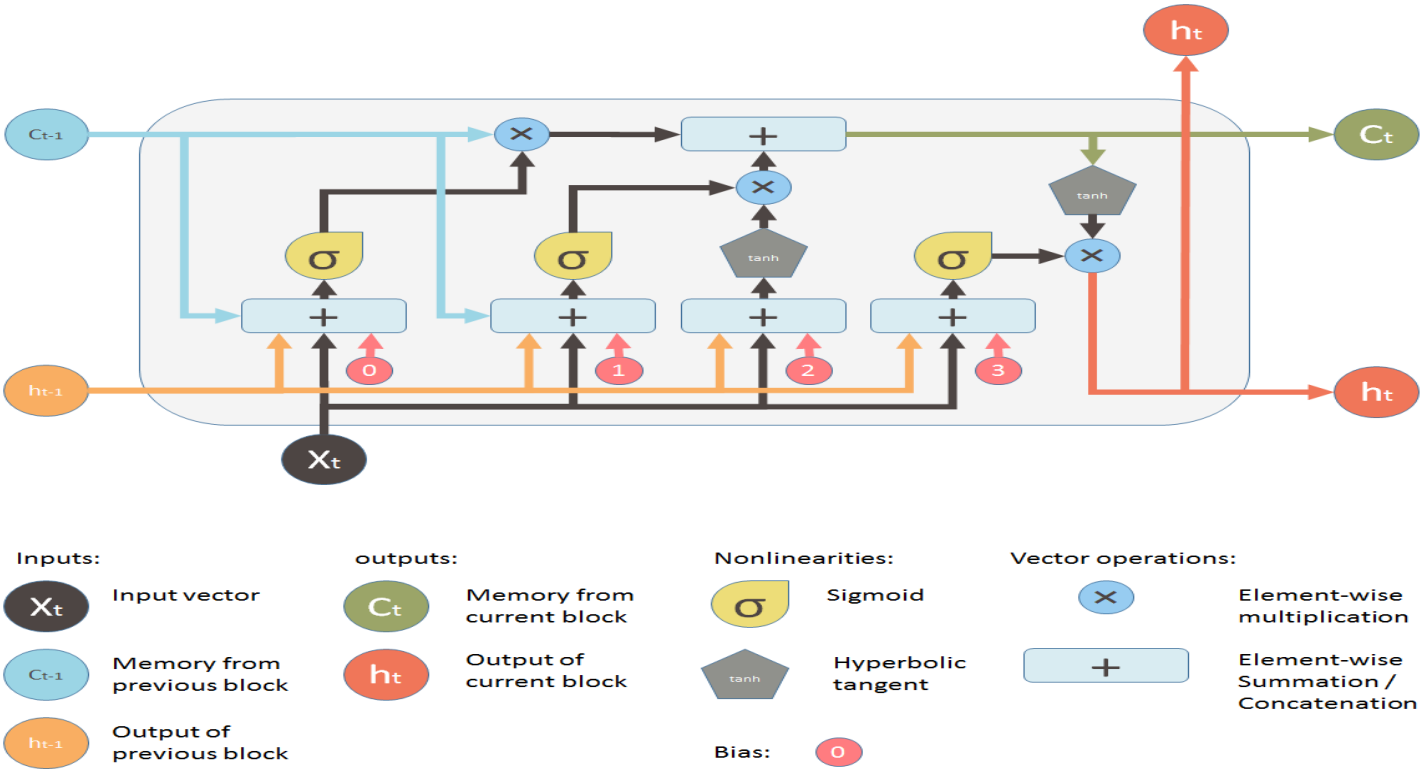
TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 in 2015.

3. scikit-learn

Scikit-learn is largely written in Python, and uses NumPy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible.

Models:

- Long Short term memory(LSTM)



Long short-term memory (**LSTM**) is an artificial recurrent neural network (RNN) architecture[1] used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition,[2] speech recognition[3][4] and anomaly detection in network traffic or IDSs (intrusion detection systems).

A common LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate**. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.

1) Get data and manipulation

```python
import math
import pandas_datareader as pdr
from .visuals import plotting
import numpy as np


def get_data(data_set='AAPL', start='2014-01-01', end='2019-09-30', look_back=60):
    """
    Getting the desired data from yahoo, then doing some data manipulation
    such as plotting, train-test-split and data
    reshaping
    Args:
        (str) data_set - the ticker of desired dataset (company)
        (str) start - the start date of the desired dataset
        (str) end - the end date of the desired dataset
        (int) look_back - number of days the forecast is based on
    Returns:
        (np array) x_train, y_train : reshaped arrays to train the model with
        (np array) x_test - reshaped array to test the model with
        (np array) y_test - to validate the model on
        (int) training_data_len - the number to split the data with into train and test
        close_df - a data frame of the close price after resetting the index
    """
    # getting the data
    df = pdr.DataReader(data_set, data_source='yahoo', start=start, end=end)
    print("Data set shape:", df.shape)

    # plotting the data
    plotting(df['Close'], title="Close Price History", x_label='Date', y_label='Close Price US ($)')

    # creating a new df with Xt - Xt-1 values of the close prices
    close_df = df['2012-01-01':].reset_index()['Close']
    plotting(close_df, title="Close price trend ",
             x_label="Date", y_label="Price")

    close_diff = close_df.diff().dropna()
    # plotting the differenced data
    plotting(close_diff, "close price after data differencing", "Date")

    # splitting the data 80% for training and 20% for testing
    training_data_len = math.ceil(len(close_diff) * 0.8)
    data = np.array(close_diff).reshape(-1, 1)

    # for doing a forecast based on a specific period (in days),,,
    # creating x_train having the first specified period data in column 1 and so on
    # creating y_train having the value of the day next to the specified period
    train_data = data[0:training_data_len, :]
    x_train = []
    y_train = []
    for i in range(look_back, len(train_data)):
        x_train.append(train_data[i - look_back:i, 0])
        y_train.append(train_data[i, 0])
    print("Training set shape:", train_data.shape)

    # converting x_train, y_train to np arrays
    x_train, y_train = np.array(x_train), np.array(y_train)

    # reshaping the data to 3D to be accepted by our LSTM model
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

    # creating the test data set
    test_data = data[training_data_len - look_back:, :]
    x_test = []
    y_test = data[training_data_len:, :]
    print("Testing set shape:", test_data.shape)
    for i in range(60, len(test_data)):
        x_test.append(test_data[i - look_back:i, 0])

    # converting x_test to be a 3D np array to be accepted by the LSTM model
    x_test = np.array(x_test)
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

    return x_train, y_train, x_test, y_test, training_data_len, close_df
```

2) get model

```python
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
import keras
from tensorflow.keras.models import load_model


def get_model(x_train, units=None, dropout=0.2, loss='mean_squared_error', metrics='MAE'):
    """
    Building the LSTM architecture to have three LSTM layer and dense layer with 1 neuron as output
layer
    Args:
        (np array) x_train - a 3D shaped array input to the model
        (list) units - number of units in each layer
        (int) dropout - Dropout percentage to control over-fitting during training
        (str) loss - the loss function to be used
        (str) metrics - the metric to be used
    Returns:
        model - the model after being compiled
    """
    if units is None:
        units = [40, 40, 50]
    model = Sequential()
    model.add(LSTM(units=units[0], return_sequences=True, input_shape=(x_train.shape[1], 1)))
    model.add(Dropout(dropout))
    model.add(LSTM(units=units[1], return_sequences=True))
    model.add(Dropout(dropout))
    model.add(LSTM(units=units[2]))
    model.add(Dropout(dropout))
    model.add(Dense(units=1))
    lr_dc = keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=0.001, decay_steps=10000,
decay_rate=0.6)
    opt = keras.optimizers.Adam(learning_rate=lr_dc)
    model.compile(optimizer=opt, loss=loss, metrics=[metrics])
    model.save('LSTM.hdf5')
    model = load_model('LSTM.hdf5')
    return model
```

3) LSTM model

```
# Commented out IPython magic to ensure Python compatibility.
from google.colab import drive
drive.mount('/content/drive')
# %cd drive/MyDrive/LSTM
!ls

# Commented out IPython magic to ensure Python compatibility.
# %load_ext autoreload
# %reload_ext autoreload
# %autoreload 2

from utilities.data_manipulation import get_data
x_train, y_train, x_test, y_test, training_data_len, close_df = get_data()

from utilities.get_model import get_model
from tensorflow.keras.callbacks import EarlyStopping
model = get_model(x_train)
# training the model
early_stopping = EarlyStopping(monitor='loss', patience=10, restore_best_weights = True, verbose=10)
lstm_history = model.fit(x=x_train, y=y_train, batch_size=64,
epochs=40,shuffle=False,validation_split=0.1,
                         callbacks=[early_stopping])

from utilities.prediction import predict
validation_df = predict(model.predict(x_test), y_test, training_data_len, close_df)

from utilities.visuals import plotting
plotting(validation_df["real data"], title="Actual VS predicted", y_label="Price",
         data2=validation_df['predictions'], legend_d1="Real", legend_d2="Predicted", save_plot=False,
plot_name="test.png")

from tensorflow.keras.models import load_model

lstm_model=load_model("LSTM.hdf5")

pred=lstm_model.predict(x_test)

x_test
```

4) Prediction

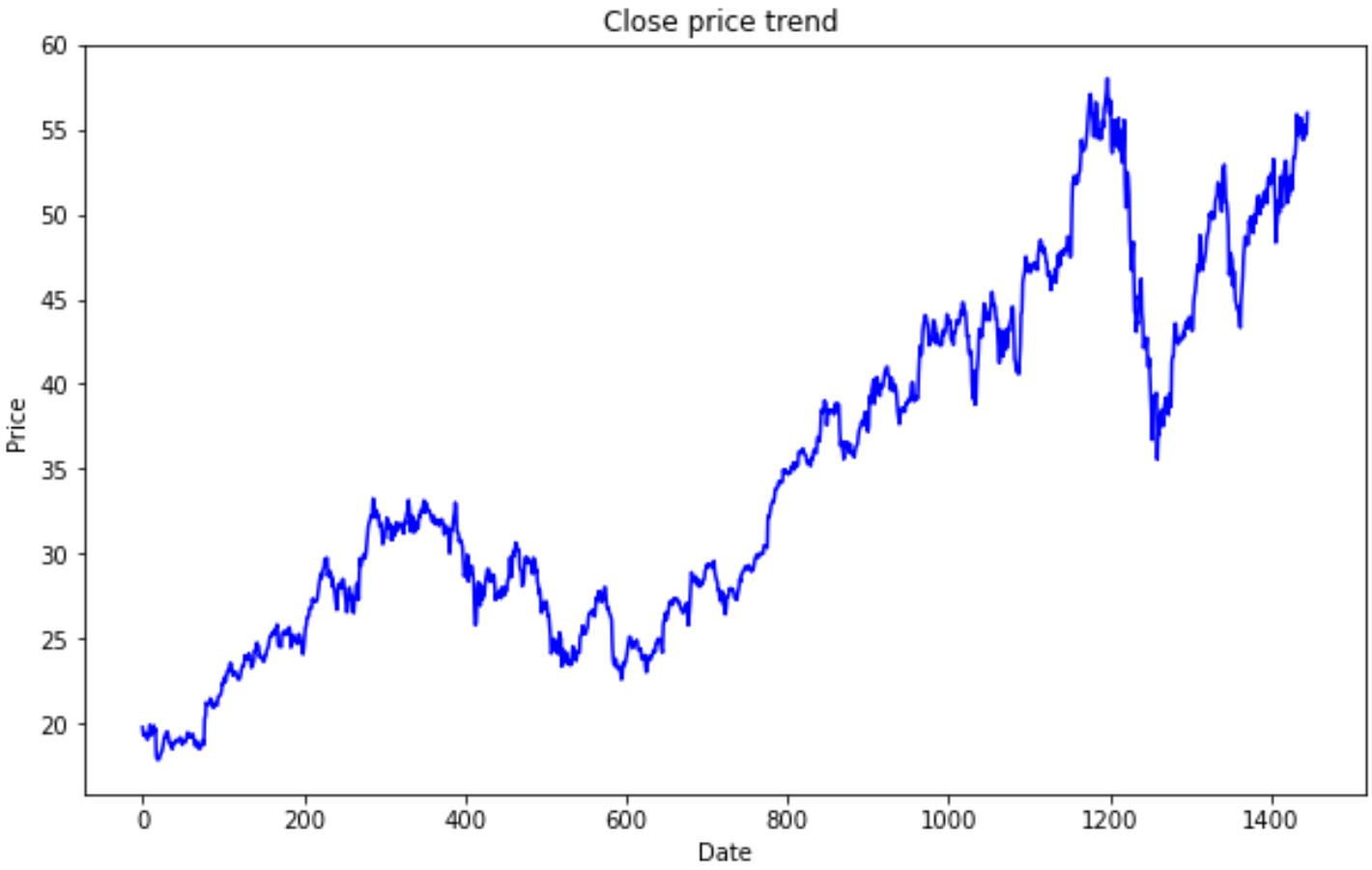
```python
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error


def predict(predictions, y_test, training_data_len, close_df):
    """
    Testing the model and validating its predictions
    Args:
        (np array) predictions -  variable to store the result of (model.predict(test_data))
        (np array) x_test - reshaped array to test the model with
        (np array) y_test - to validate the model on
        (int) training_data_len - the number to split the data with into train and test
        close_df - a data frame of the close price after resetting the index
    Returns:
        validation_df - a df contains the predicted prices and the real data
    """

    # getting the real prediction values instead of the price change in each prediction....
    # reshaping the close_df to be the same shape as the model output
    close_df = np.array(close_df).reshape(-1, 1)
    # real test data without last value
    test_df = np.delete(close_df[training_data_len:, :], -1, 0)
    # real test data shifted
    test_df_shifted = close_df[training_data_len+1:, :]
    # the logic of reversing the data from difference to real
    real_data_prediction = predictions + test_df

    # Calculate/Get the value of MSE
    mse = mean_squared_error(predictions, y_test)
    print("MSE value:", mse)
    # Calculate/Get the value of MAE
    mae = mean_absolute_error(predictions, y_test)
    print("MAE value:", mae)

    # creating a new df to assign the predictions to its equivalent days and comparing them to the real
data
    validation_df = pd.DataFrame(real_data_prediction, columns=["predictions"])
    validation_df['real data'] = test_df_shifted
    print(validation_df)

    return validation_df
```

5) visualization

```python
import matplotlib.pyplot as plt


def plotting(data1, title, x_label="Date", fig_size=(10, 6), y_label=None, data2=None, legend_d1=None,
legend_d2=None,
             save_plot=False, plot_name=None):
    """
    Plotting the given data into 10 X 6 figure
    Args:
        data1 - data to be plotted
        (str) title - the figure title
        (str) x_label - x axis label
        (str) y_label - y axis label (if given)
        data2 - data to be plotted (if given)
        (str) legend_d1 - legend specifies data1
        (str) legend_d2 - legend specifies data2
        (bool) save plot - if not False, will save the plot given specific name
        (str) plot_name - the desired name to save the plot with
    Returns:
        a plot with the specified args
    """

    plt.figure(figsize=fig_size)
    plt.plot(data1, color="blue")
    plt.title(title)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    if data2 is not None:
        plt.plot(data2, color="red")
        plt.legend([legend_d1, legend_d2], loc="upper left")
    if save_plot is not False:
        plt.savefig(plot_name)
    return plt.show()
```

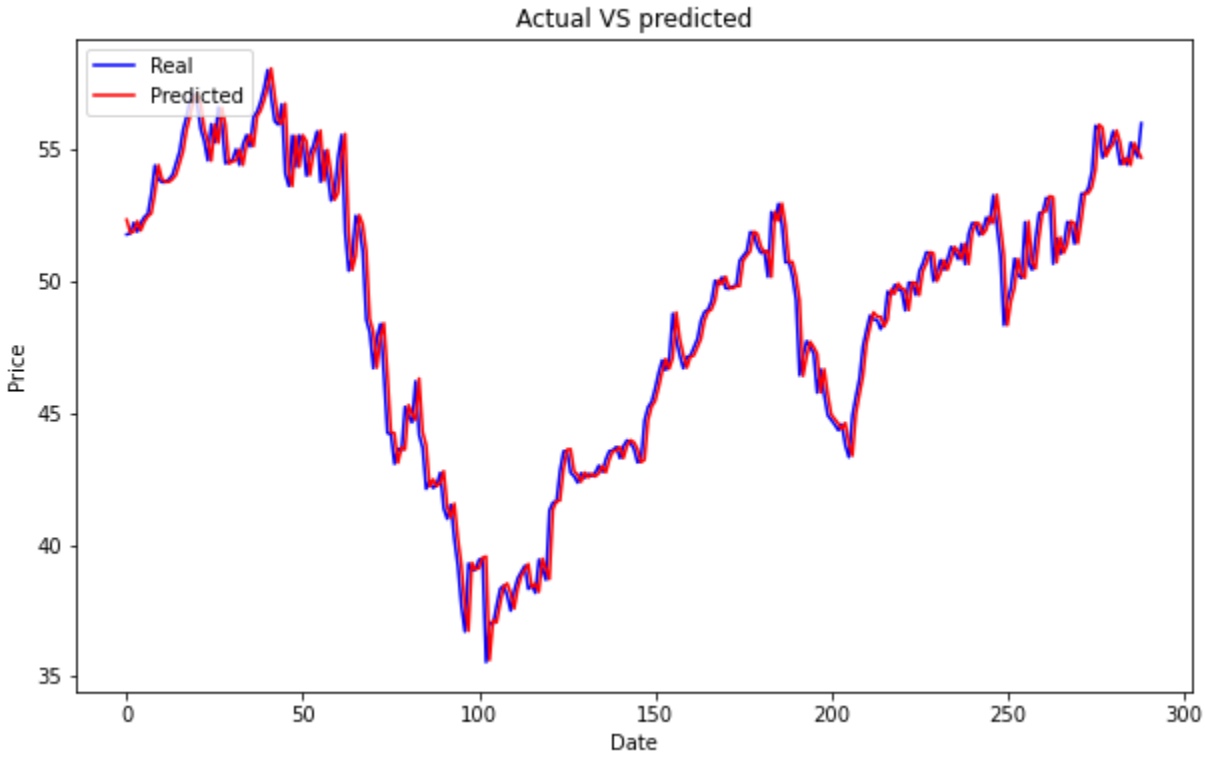
The output and results:

1) The history data

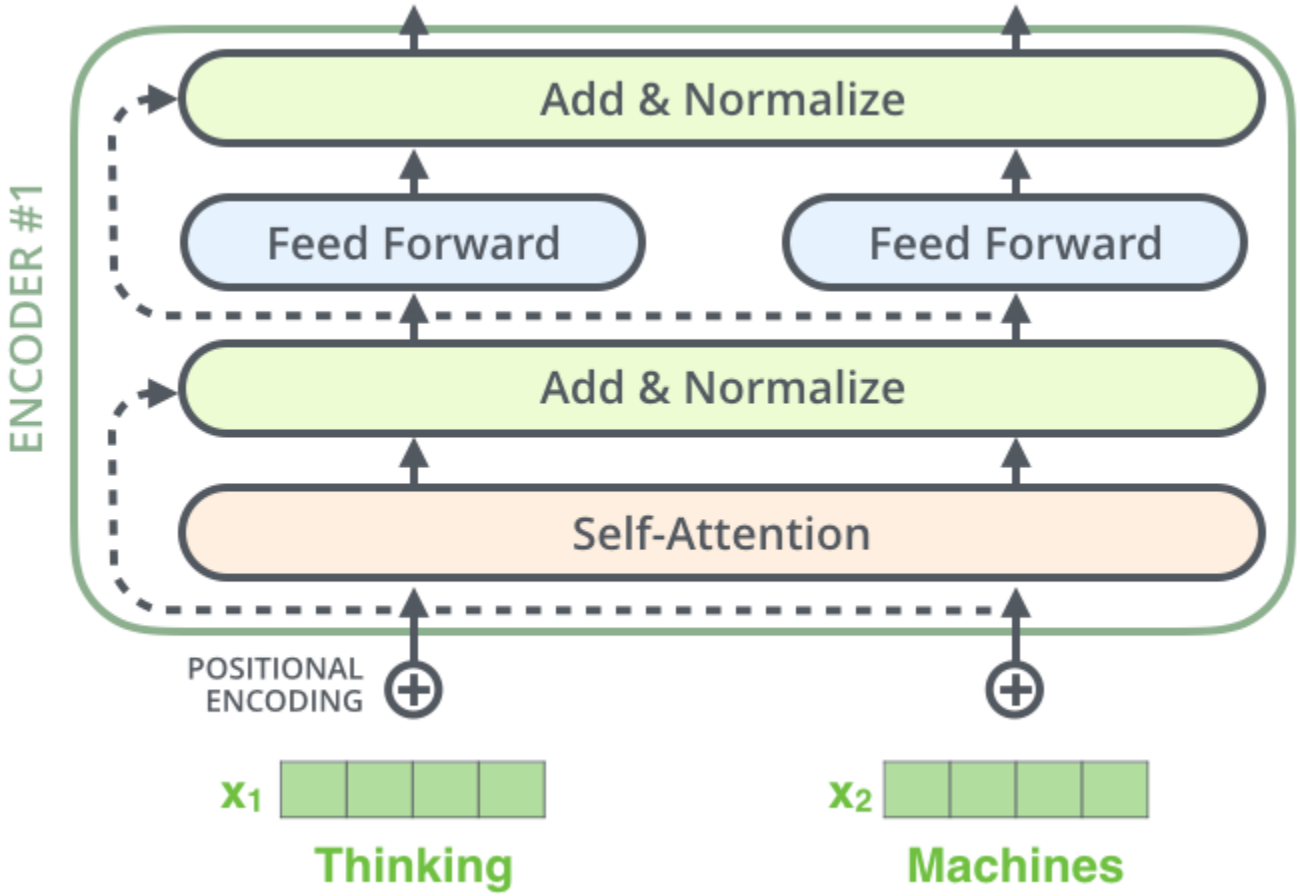


2) Difference check

3) Test


Actual VS predicted
Real
Predicted
Price
55
50
45
40
35
0
50
100
150
200
250
300
Date

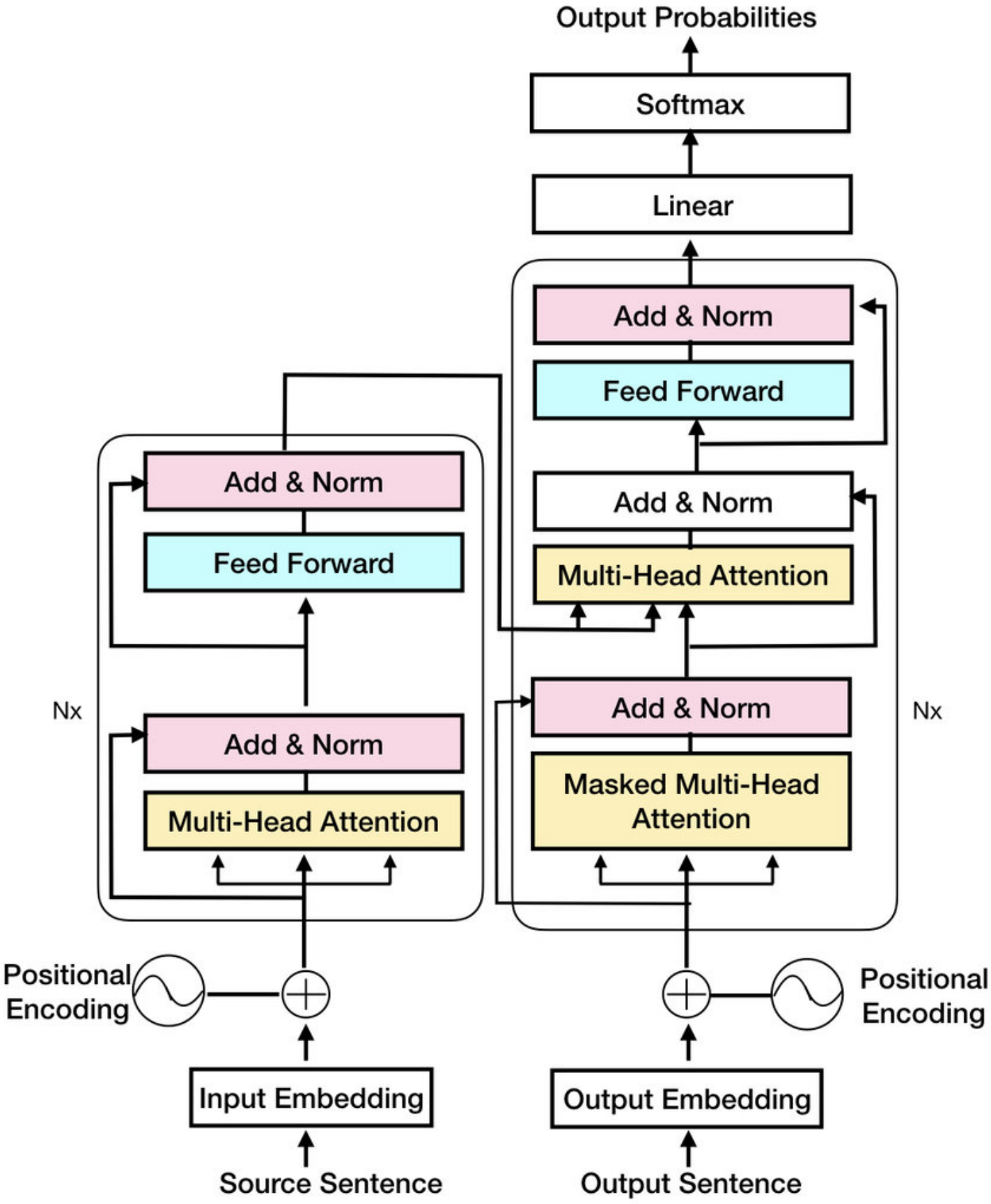
- **Transformer**



A **transformer** is a deep learning model that adopts the mechanism of attention, differentially weighing the significance of each part of the input data. It is used primarily in the field of natural language processing (NLP) and in computer vision (CV).

Like recurrent neural networks (RNNs), transformers are designed to handle sequential input data, such as natural language, for tasks such as translation and text summarization. However, unlike RNNs, transformers do not necessarily process the data in order. Rather, the attention mechanism provides context for any position in the input sequence. For example, if the input data is a natural language sentence, the transformer does not need to process the beginning of the sentence before the end. Rather, it identifies the context that confers meaning to each word in the sentence. This feature allows for more parallelization than RNNs and therefore reduces training times.

Transformers are the model of choice for NLP problems,[j] replacing RNN models such as long short-term memory (LSTM). The additional training parallelization allows training on larger datasets than was once possible. This led to the development of pretrained systems such as BERT (Bidirectional Encoder Representations from transformers) and GPT (Generative Pre-trained Transformer), which were trained with large language datasets, such as Wikipedia Corpus and Common Crawl, and can be fine-tuned for specific tasks


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Source Sentence
Output Sentence

1) get data

```python
import pandas_datareader as pdr
import numpy as np
from sklearn.preprocessing import MinMaxScaler

def get_data(seq_len = 32):
    df = pdr.DataReader('F',data_source='yahoo' ,start='2012-01-01', end='2021-05-1')

    df.drop('Volume',axis=1,inplace=True)

    scaler = MinMaxScaler(feature_range=(0,1))
    #df[df.columns] = scaler.fit_transform(df)
    df=df[['High','Low','Open','Adj Close','Close']]

    '''Create training, validation and test split'''

    times = sorted(df.index.values)
    last_10pct = sorted(df.index.values)[-int(0.1*len(times))] # Last 10% of series
    last_20pct = sorted(df.index.values)[-int(0.2*len(times))] # Last 20% of series

    df_train = df[(df.index < last_20pct)]  # Training data are 80% of total data
    df_val = df[(df.index >= last_20pct) & (df.index < last_10pct)]
    df_test = df[(df.index >= last_10pct)]

    train_data = df_train.values
    val_data = df_val.values
    test_data = df_test.values
    print('Training data shape: {}'.format(train_data.shape))
    print('Validation data shape: {}'.format(val_data.shape))
    print('Test data shape: {}'.format(test_data.shape))
    train_data_len=len(train_data)
    val_data_len=len(train_data)+len(val_data)

    # Training data
    X_train, y_train = [], []
    for i in range(seq_len, len(train_data)):
        X_train.append(train_data[i-seq_len:i]) # Chunks of training data with a length of 128 df-rows
        y_train.append(train_data[:, 4][i]) # Value of 4th column (Close Price) of df-row 128+1
    X_train, y_train = np.array(X_train), np.array(y_train)

    # Validation data
    X_val, y_val = [], []
    for i in range(seq_len, len(val_data)):
        X_val.append(val_data[i-seq_len:i])
        y_val.append(val_data[:, 4][i])
    X_val, y_val = np.array(X_val), np.array(y_val)

    # Test data
    X_test, y_test = [], []
    for i in range(seq_len, len(test_data)):
        X_test.append(test_data[i-seq_len:i])
        y_test.append(test_data[:, 4][i])
    X_test, y_test = np.array(X_test), np.array(y_test)

    print('Training set shape', X_train.shape, y_train.shape)
    print('Validation set shape', X_val.shape, y_val.shape)
    print('Testing set shape' ,X_test.shape, y_test.shape)

    return df, X_train, y_train, X_val, y_val, X_test, y_test,train_data_len,val_data_len,scaler
```

2) Get model

```
import tensorflow as tf
from tensorflow.keras import Model, Input
from tensorflow.keras.layers import Concatenate, Dense, Dropout, GlobalAveragePooling1D
import keras
from .Encoder import TransformerEncoder
from .Time2Vector import Time2Vector

def get_model(seq_len, d_k, d_v, n_heads, ff_dim, encoder_stack_size = 3, loss='mse',
output_activation='linear'):
    '''Initialize time and transformer layers'''
    time_embedding = Time2Vector(seq_len)
    Encoder_layer = TransformerEncoder(d_k, d_v, n_heads, ff_dim)

    '''Construct model'''
    in_seq = Input(shape=(seq_len, 5))
    time_embedding_layer = time_embedding(in_seq)
    x = Concatenate(axis=-1)([in_seq, time_embedding_layer])

    for _ in range(encoder_stack_size):
        x = Encoder_layer((x, x, x))

    x = GlobalAveragePooling1D(data_format='channels_first')(x)
    x = Dropout(0.1)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.1)(x)
    out = Dense(1, activation=output_activation)(x) # sigmoid

 lr_dc=keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=0.001,decay_steps=10000,decay_
rate=0.3)
    #optimizer=tf.keras.optimizers.Adam(
    #learning_rate=0.001,decay=0.3)
    optimizer=tf.keras.optimizers.Adam(lr_dc)

    model = Model(inputs=in_seq, outputs=out)
    model.compile(loss=loss, optimizer=optimizer, metrics=['mae'])
    return model
```

3) Attention

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.activations import linear
from tensorflow.keras.layers import Layer, Dense

class SingleAttention(Layer):
    def __init__(self, d_k, d_v):
        super(SingleAttention, self).__init__()
        self.d_k = d_k
        self.d_v = d_v

    def build(self, input_shape):
        self.query = Dense(self.d_k, input_shape=input_shape, kernel_initializer='glorot_uniform',
bias_initializer='glorot_uniform')
        self.key = Dense(self.d_k, input_shape=input_shape, kernel_initializer='glorot_uniform',
bias_initializer='glorot_uniform')
        self.value = Dense(self.d_v, input_shape=input_shape, kernel_initializer='glorot_uniform',
bias_initializer='glorot_uniform')

    def call(self, inputs): # inputs = (in_seq, in_seq, in_seq)
        q = self.query(inputs[0])
        k = self.key(inputs[1])

        attn_weights = tf.matmul(q, k, transpose_b=True)
        attn_weights = tf.map_fn(lambda x: x/np.sqrt(self.d_k), attn_weights)
        attn_weights = tf.nn.softmax(attn_weights, axis=-1)

        v = self.value(inputs[2])
        attn_out = tf.matmul(attn_weights, v)
        return attn_out

class MultiAttention(Layer):
    def __init__(self, d_k, d_v, n_heads):
        super(MultiAttention, self).__init__()
        self.d_k = d_k
        self.d_v = d_v
        self.n_heads = n_heads
        self.attn_heads = list()

    def build(self, input_shape):
        for n in range(self.n_heads):
          self.attn_heads.append(SingleAttention(self.d_k, self.d_v))
        self.linear = Dense(7, input_shape=input_shape, kernel_initializer='glorot_uniform',
bias_initializer='glorot_uniform')

    def call(self, inputs):
        attn = [self.attn_heads[i](inputs) for i in range(self.n_heads)]
        concat_attn = tf.concat(attn, axis=-1)
        multi_linear = self.linear(concat_attn)
        return multi_linear
```

4) Encoder

```python
import tensorflow as tf
from tensorflow.keras.layers import LayerNormalization, Dropout, Conv1D
from tensorflow.keras.layers import Layer

from .Attention import MultiAttention

class TransformerEncoder(Layer):
    def __init__(self, d_k, d_v, n_heads, ff_dim, dropout=0.1, **kwargs):
        super(TransformerEncoder, self).__init__()
        self.d_k = d_k
        self.d_v = d_v
        self.n_heads = n_heads
        self.ff_dim = ff_dim
        self.attn_heads = list()
        self.dropout_rate = dropout

    def build(self, input_shape):
        self.attn_multi = MultiAttention(self.d_k, self.d_v, self.n_heads)
        self.attn_dropout = Dropout(self.dropout_rate)
        self.attn_normalize = LayerNormalization(input_shape=input_shape, epsilon=1e-6)

        self.ff_conv1D_1 = Conv1D(filters=self.ff_dim, kernel_size=1, activation='relu')
        self.ff_conv1D_2 = Conv1D(filters=7, kernel_size=1) # input_shape[0]=(batch, seq_len, 7),
input_shape[0][-1]=7
        self.ff_dropout = Dropout(self.dropout_rate)
        self.ff_normalize = LayerNormalization(input_shape=input_shape, epsilon=1e-6)

    def call(self, inputs): # inputs = (in_seq, in_seq, in_seq)
        attn_layer = self.attn_multi(inputs)
        attn_layer = self.attn_dropout(attn_layer)
        attn_layer = self.attn_normalize(inputs[0] + attn_layer)

        ff_layer = self.ff_conv1D_1(attn_layer)
        ff_layer = self.ff_conv1D_2(ff_layer)
        ff_layer = self.ff_dropout(ff_layer)
        ff_layer = self.ff_normalize(inputs[0] + ff_layer)
        return ff_layer

    def get_config(self):  # Needed for saving and loading model with custom layer
        config = super().get_config().copy()
        config.update({'d_k': self.d_k,
                       'd_v': self.d_v,
                       'n_heads': self.n_heads,
                       'ff_dim': self.ff_dim,
                       'attn_heads': self.attn_heads,
                       'dropout_rate': self.dropout_rate})
        return config
```

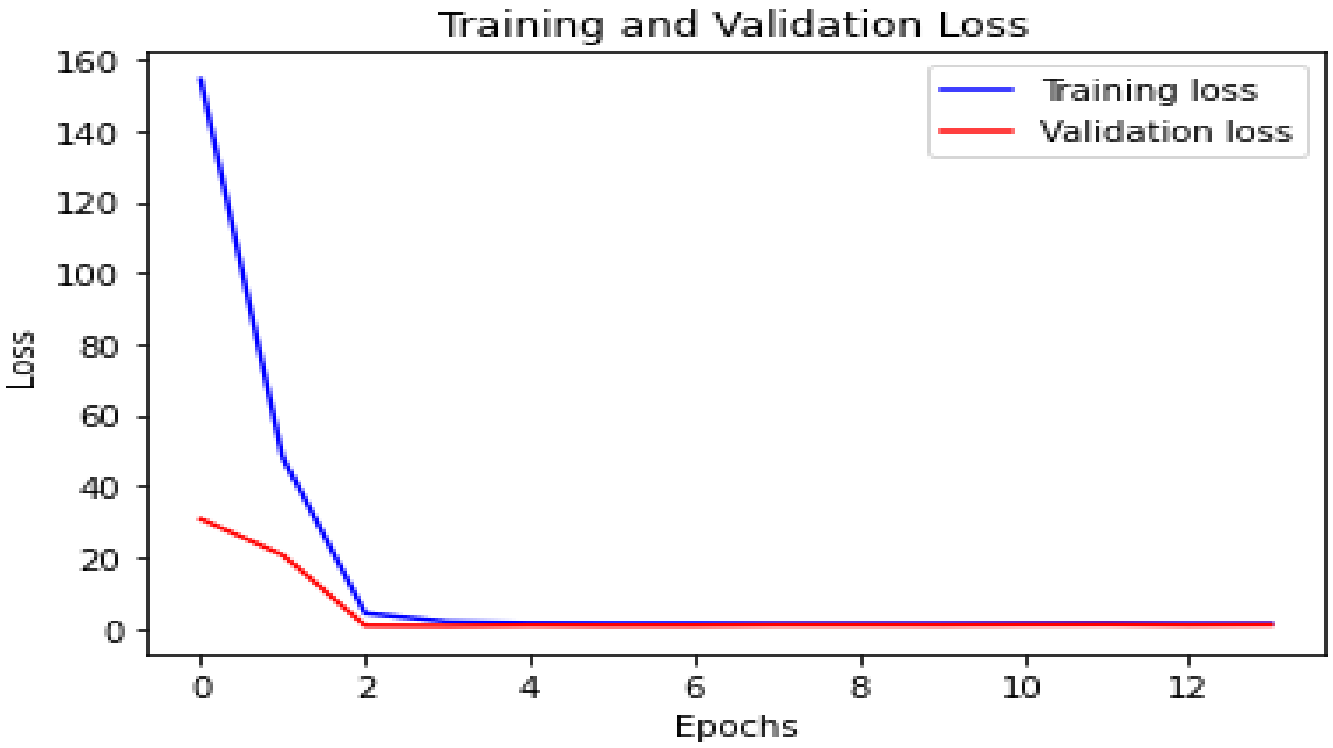
5) Train the model

```python
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
from utils.Time2Vector import Time2Vector
from utils.Attention import MultiAttention, SingleAttention
from utils.Encoder import TransformerEncoder
from utils.get_data import get_data

def train_model(model, epoches=50, batch=32):
    seq_len = 128
    df, X_train, y_train, X_val, y_val, X_test, y_test,train_data_len,val_data_len,scaler = get_data()

    CP = ModelCheckpoint(filepath='Transformer+TimeEmbedding.hdf5',
                         monitor='val_loss',
                         save_best_only=True,
                         verbose=1)

    ES = EarlyStopping(monitor='val_loss',
                       mode='min',
                       patience=10)

    history = model.fit(X_train, y_train,
                        batch_size=batch,
                        epochs=epoches,
                        callbacks=[CP, ES],
                        verbose=1,
                        validation_data=(X_val, y_val))

    model = load_model('Transformer+TimeEmbedding.hdf5',
                       custom_objects={'Time2Vector': Time2Vector,
                                       'SingleAttention': SingleAttention,
                                       'MultiAttention': MultiAttention,
                                       'TransformerEncoder': TransformerEncoder})


    #Print evaluation metrics for all datasets
    train_eval = model.evaluate(X_train, y_train, verbose=0)
    val_eval = model.evaluate(X_val, y_val, verbose=0)
    test_eval = model.evaluate(X_test, y_test, verbose=0)

    print('\nEvaluation metrics')
    print('Training Data - Loss: {:.4f}, MAE: {:.4f}'.format(train_eval[0], train_eval[1]))
    print('Validation Data - Loss: {:.4f}, MAE: {:.4f}'.format(val_eval[0], val_eval[1]))
    print('Test Data - Loss: {:.4f}, MAE: {:.4f}'.format(test_eval[0], test_eval[1]))
    # Visualize Loss Vs. epochs

    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs = range(len(loss))
    plt.figure()
    plt.plot(epochs, loss, "b", label="Training loss")
    plt.plot(epochs, val_loss, "r", label="Validation loss")
    plt.title("Training and Validation Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()
```

6) Transformer

```
# %load_ext autoreload
# %reload_ext autoreload
# %autoreload 2

from utils.get_data import get_data
from sklearn.metrics import mean_absolute_error,mean_squared_error
import matplotlib.pyplot as plt

df, X_train, y_train, X_val, y_val, X_test, y_test,train_data_len,val_data_len,scaler = get_data()
print(df.head())
print(df.tail())

from utils.get_model import get_model
from utils.train import train_model

n = 256
model = get_model(seq_len=32,
                  d_k = n, d_v = n,
                  n_heads = 12, ff_dim = n,
                  encoder_stack_size = 1,
                  loss='mse',
                  output_activation='linear')

train_model(model, epoches=250, batch=32)

from utils.Time2Vector import Time2Vector
from utils.Attention import MultiAttention, SingleAttention
from utils.Encoder import TransformerEncoder
from tensorflow import keras
from keras.models import load_model
custom_objects = {"Time2Vector":Time2Vector,
                      "MultiAttention":MultiAttention,
                      'TransformerEncoder':TransformerEncoder}
with keras.utils.custom_object_scope(custom_objects):
      final_model=load_model('Transformer+TimeEmbedding.hdf5')
```

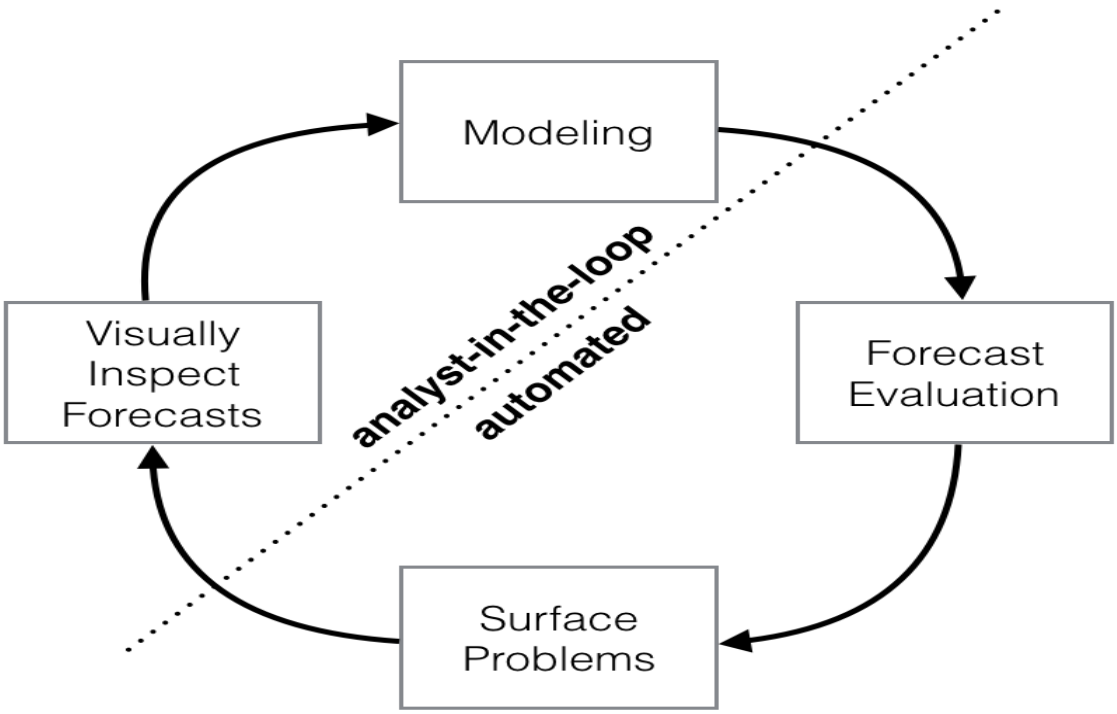
7) Training and validation loss



- Profit

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

Prophet is open source software released by Facebook's Core Data Science team. It is available for download on CRAN and PyPI.



1) Data gathering and manipulation

```
# getteing ford data set
df = pdr.DataReader("F", data_source='yahoo', start='2015-01-01')

# visualize Line pattern of 'Close' Feature to see how it looks like
plt.figure(figsize=(10, 6))
plt.plot(df['Close'])
plt.title("Close Price History")
plt.xlabel('Date', fontsize=15)
plt.ylabel('Close Price US ($)', fontsize=15)
plt.show()
# Convert the date index into a datetime
df.index = pd.to_datetime(df.index, format="%Y/%m/%d")

#Adding Holidays to the model
import holidays
holiday = pd.DataFrame([])
for date, name in sorted(holidays.UnitedStates(years=[2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019,
2020, 2021]).items()):
    holiday = holiday.append(pd.DataFrame({'ds': date, 'holiday': "US-Holidays"}, index=[0]),
ignore_index=True)
holiday['ds'] = pd.to_datetime(holiday['ds'], format='%Y-%m-%d', errors='ignore')
```

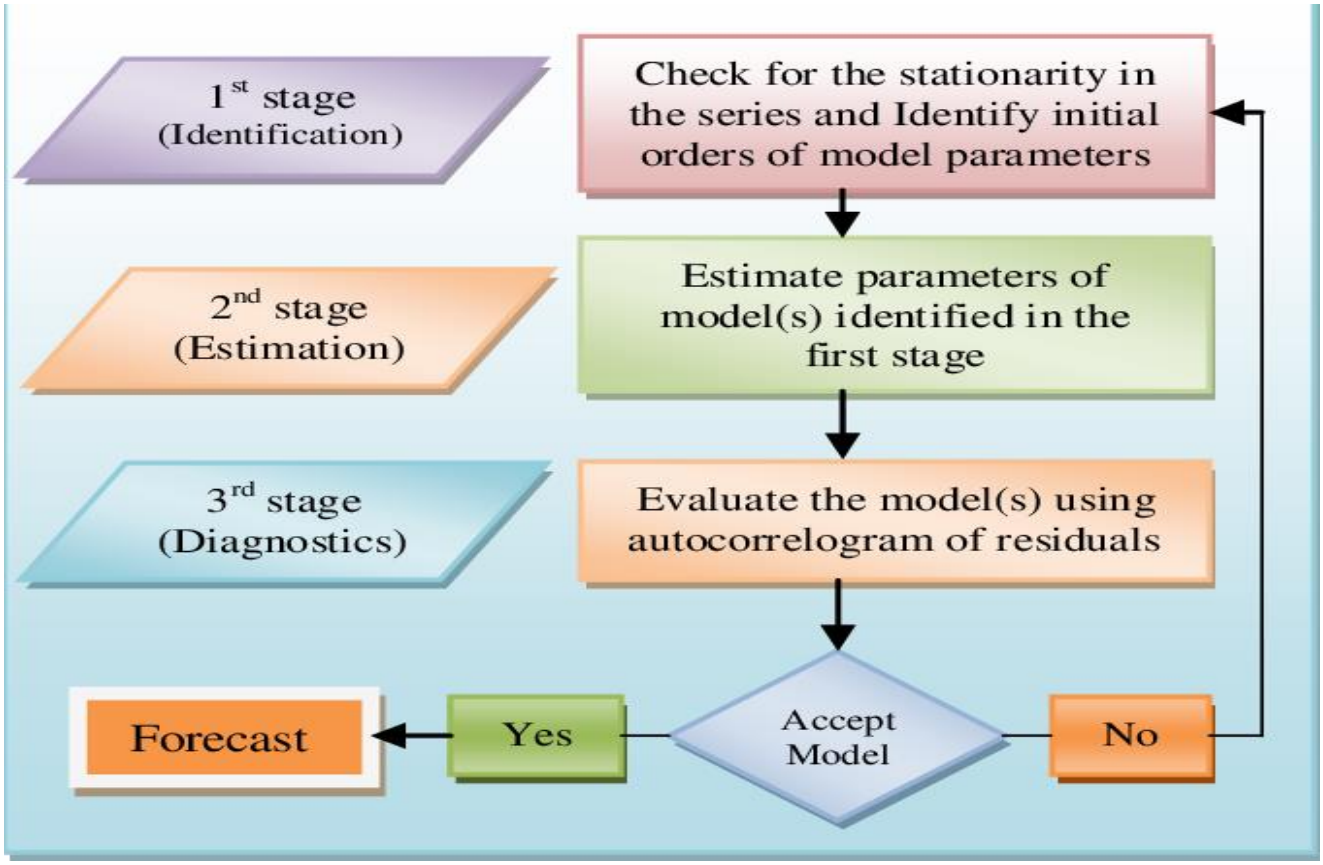
2) Model training

```
import json
from prophet.serialize import model_to_json, model_from_json
m = Prophet(holidays=holiday,seasonality_mode='additive', changepoint_prior_scale = 0.1,
seasonality_prior_scale=0.01)
m.fit(data)

with open('F_prophet.json', 'w') as fout:
    json.dump(model_to_json(m), fout)  # Save model

#Diagnostics and evaluation
from prophet.diagnostics import cross_validation
# setting 80 % of the data as initial (training data),
# and 20 days to forecast per iteration
# For each ordered validation iteration, FBProphet will forecast between
# the cutoff (period) and the cutoff + horizon and then add the period to get the next cutoff
data_cv = cross_validation(m, initial='1752 days', period = '10 days', horizon = '20 days'
,parallel="threads")
```

3) Prophet model

```
def prophet (ticker):
  """
  Forcasting using prophet ! by Getting the desired data from yahoo, then doing some data manipulation,
then the comes the prophet's turn
  Args:
      (str) ticket - the ticker of desired dataset (company)
  Returns:
      (float) prophet_output - the model out-put (the prediction of the next day)
  """

  # data_gathering
  df = pdr.DataReader(ticker, data_source='yahoo', start='2015-01-01')

  # data manipulation
  holiday = pd.DataFrame([])
  for date, name in sorted(holidays.UnitedStates(
    years=[2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021]).items()):
      holiday = holiday.append(pd.DataFrame({'ds': date, 'holiday': "US-Holidays"}, index=[0]),
ignore_index=True)
  holiday['ds'] = pd.to_datetime(holiday['ds'], format='%Y-%m-%d', errors='ignore')

  # data frame modification to be accepted by prophet
  data = df['Close'].reset_index()
  data.columns = ['ds', 'y']

  # model building
  m = Prophet(holidays=holiday,seasonality_mode='additive', changepoint_prior_scale = 0.1,
seasonality_prior_scale=0.01)
  m.fit(data)

  # model predictions
  future = m.make_future_dataframe(periods=1)
  model_prediction = m.predict(future)
  prophet_prediction = float(model_prediction[ 'yhat'][-1:])
  return prophet_prediction
```

- **Autoregressive Integrated Moving Average(ARIMA)**



In statistics and econometrics, and in particular in time series analysis, an **autoregressive integrated moving average (ARIMA)** model is a generalization of an autoregressive moving average (ARMA) model. Both of these models are fitted to time series data either to better understand the data or to predict future points in the series (forecasting). ARIMA models are applied in some cases where data show evidence of non-stationarity in the sense of mean (but not variance/autocovariance), where an initial differencing step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate the non-stationarity of the mean function

(i.e., the trend).[1] When the seasonality shows in a time series, the seasonal-differencing[2] could be applied to eliminate the seasonal component. Since the ARMA model, according to the Wold's decomposition theorem,[3][4][5] is theoretically sufficient to describe a **regular** (a.k.a. purely nondeterministic) wide-sense stationary time series, we are motivated to make stationary a non-stationary time series, e.g., by using differencing, before we can use the ARMA model. Note that if the time series contains a **predictable** sub-process (a.k.a. pure sine or complex-valued exponential process), the predictable component is treated as a non-zero-mean but periodic (i.e., seasonal) component in the ARIMA framework so that it is eliminated by the seasonal differencing.

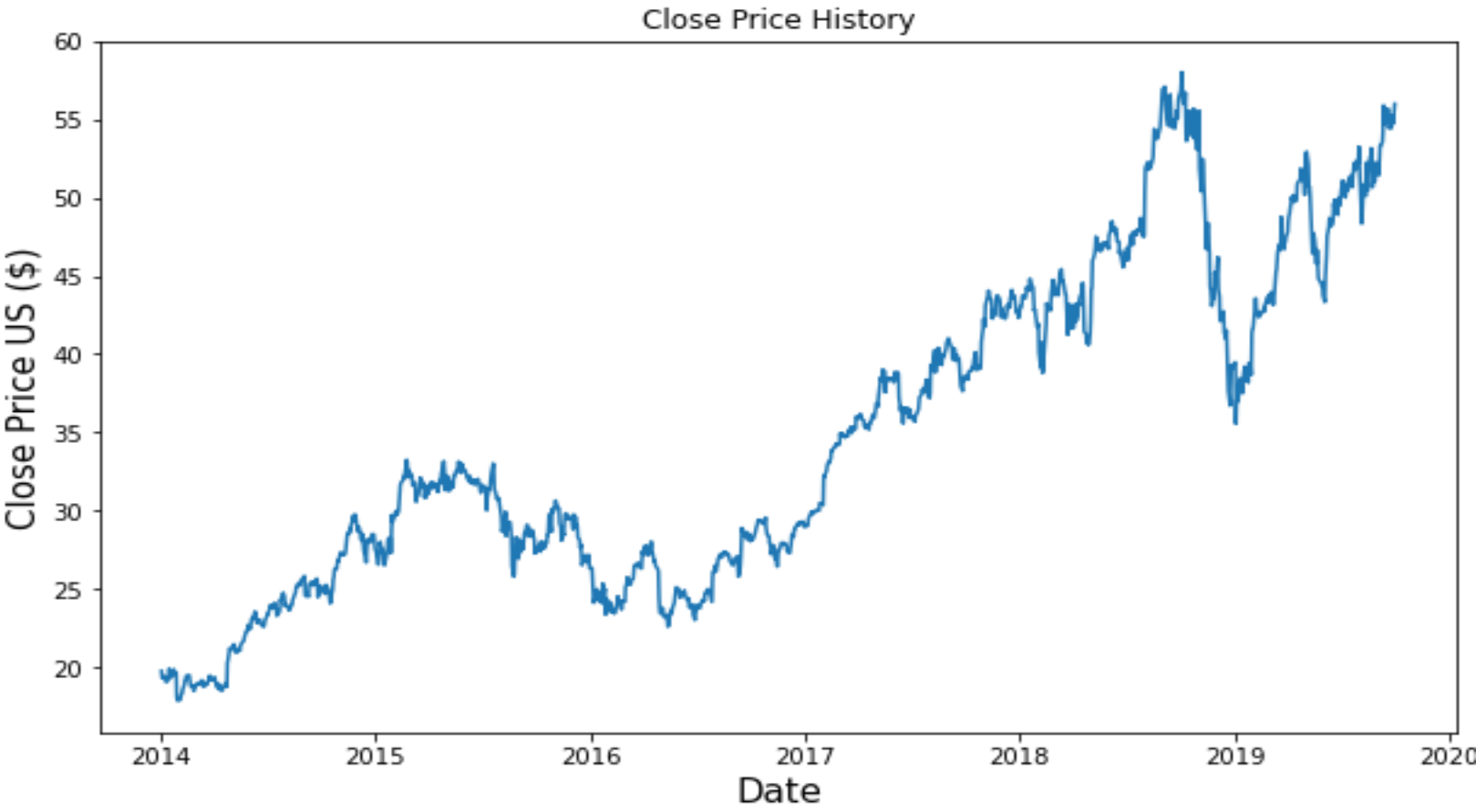
The AR part of ARIMA indicates that the evolving variable of interest is regressed on its own lagged (i.e., prior) values. The MA part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past. The I (for "integrated") indicates that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once). The purpose of each of these features is to make the model fit the data as well as possible. Non-seasonal ARIMA models are generally denoted ARIMA(p,d,q)

where parameters *p*, *d*, and *q* are non-negative integers, *p* is the order (number of time lags) of the autoregressive model, *d* is the degree of differencing (the number of times the data have had past values subtracted), and *q* is the order of the moving-average model. Seasonal ARIMA models are usually denoted ARIMA(p,d,q)(P,D,Q)$_m$, where *m* refers to the number of periods in each season, and the uppercase *P*,*D*,*Q* refer to the autoregressive, differencing, and moving average terms for the seasonal part of the ARIMA model.

When two out of the three terms are zeros, the model may be referred to based on the non-zero parameter, dropping "AR", "I" or "MA" from the acronym describing the model. For example, is AR(1), is I(1), and is MA(1).

1) Data gathering

```
import pandas_datareader as pdr
import pandas as pd
import matplotlib.pyplot as plt
# from datetime import date
# end_date = date.today()
# print(end_date)
# start_date = end_date - datetime.timedelta(days=32)
# print(start_date)

def get_data(data_set='AAPL'):

    df = pdr.DataReader(data_set, data_source='yahoo', start='2014-01-01')

    # visualize Line pattern of 'Close' Feature to see how it looks like
    # plt.figure(figsize=(10, 6))
    # plt.plot(df['Close'])
    # plt.title("Close Price History")
    # plt.xlabel('Date', fontsize=15)
    # plt.ylabel('Close Price US ($)', fontsize=15)
    # plt.show()
    # Convert the date index into a datetime
    df.index = pd.to_datetime(df.index, format="%Y/%m/%d")

    # Convert to Series to run Dickey-Fuller test
    df = pd.Series(df['Close'])

    return df
```

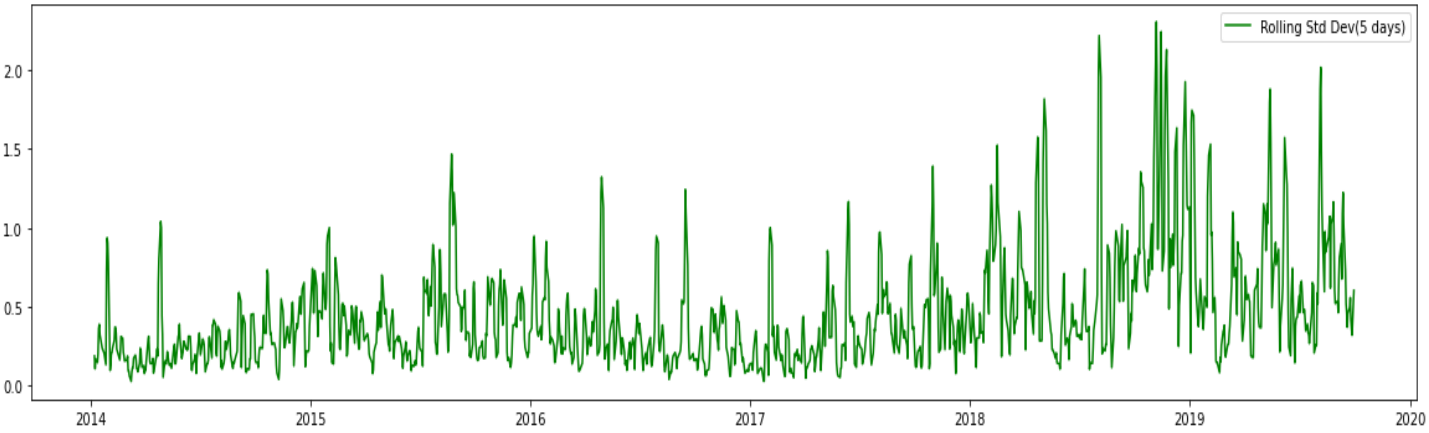


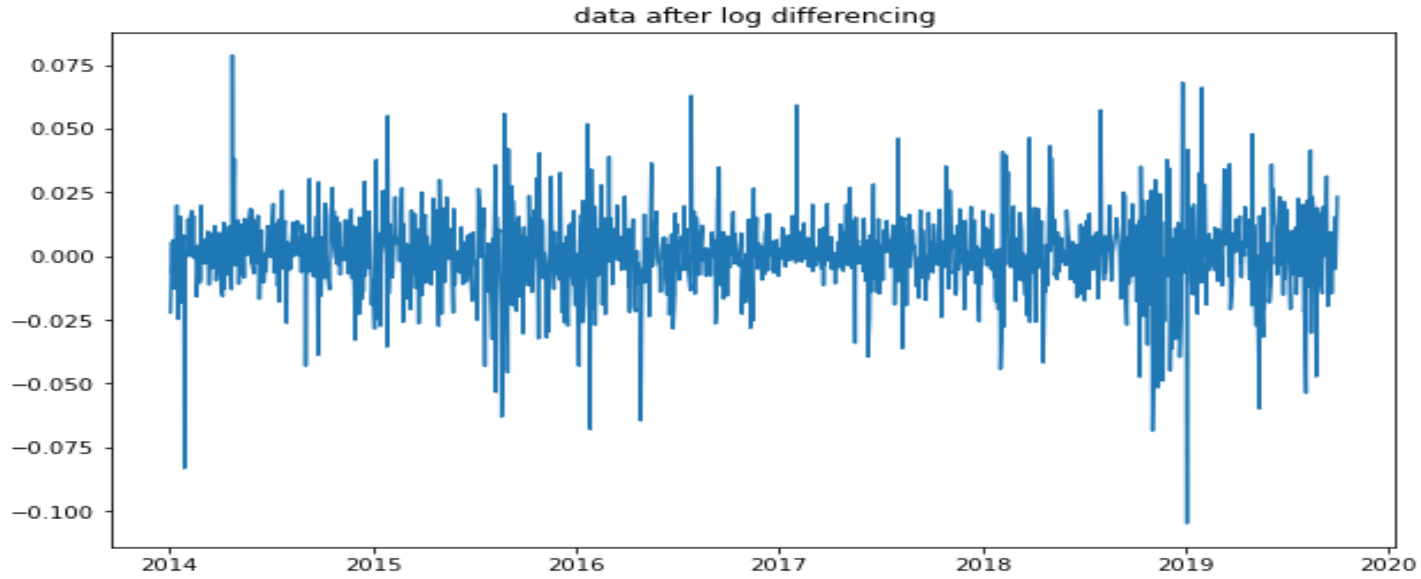
2) Stationarity check

A stationary time series' properties do not depend on the time at which the series is observed. Specifically, for a wide-sense stationary time series, the mean and the variance/autocovariance keep constant over time. **Differencing** in statistics is a transformation applied to a non-stationary time-series in order to make it stationary in the mean sense (viz., to remove the non-constant trend), but having nothing to do with the non-stationarity of the variance/autocovariance. Likewise, the **seasonal differencing** is applied to a seasonal time-series to remove the seasonal component.

```python
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf
import statsmodels.api as sm


# Check Stationary by many methods like diff , log
def rolling_stats(ts_data):
    roll_mean = ts_data.rolling(30).mean()
    roll_std = ts_data.rolling(5).std()

    # Plot rolling statistics
    # fig = plt.figure(figsize=(20, 10))
    # plt.subplot(211)
    # plt.plot(ts_data, color='black', label='Original Data')
    # plt.plot(roll_mean, color='red', label='Rolling Mean(30 days)')
    # plt.legend()
    # plt.subplot(212)
    # plt.plot(roll_std, color='green', label='Rolling Std Dev(5 days)')
    # plt.legend()


def dickey_fuller(ts_data):
    print('Dickey-Fuller test results\n')
    df_test = adfuller(ts_data, regresults=False)
    test_result = pd.Series(df_test[0:4], index=['Test Statistic', 'p-value', '# of lags', '# of obs'])
    print(test_result)
    for k, v in df_test[4].items():
        print('Critical value at %s: %1.5f' % (k, v))


def log_diff(ts_data):
    df_final_log = np.log(ts_data)
    df_final_log_diff = df_final_log - df_final_log.shift()
    df_final_log_diff.dropna(inplace=True)
    # plt.figure(figsize=(10, 6))
    # plt.plot(df_final_log_diff)
    # plt.title("data after log differencing")
    # plt.show()


def auto_correlation(ts_data):
    df_final_diff = ts_data - ts_data.shift()
    df_final_diff.dropna(inplace=True)
    df_acf = acf(df_final_diff)

    # Partial autocorrelation function
    df_pacf = pacf(df_final_diff)

    # visualizing acf and pacf
    # fig1 = plt.figure(figsize=(20, 10))
    # ax1 = fig1.add_subplot(211)
    # fig1 = sm.graphics.tsa.plot_acf(df_acf, ax=ax1)
    # ax2 = fig1.add_subplot(212)
    # fig1 = sm.graphics.tsa.plot_pacf(df_pacf, ax=ax2)
```


Original Data
Rolling Mean(30 days)
60
55
50
45
40
35
30
25
20
2014
2015
2016
2017
2018
2019
2020


Rolling Std Dev(5 days)
2.0
1.5
1.0
0.5
0.0
2014
2015
2016
2017
2018
2019
2020


data after log differencing
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
2014
2015
2016
2017
2018
2019
2020

3) Best model

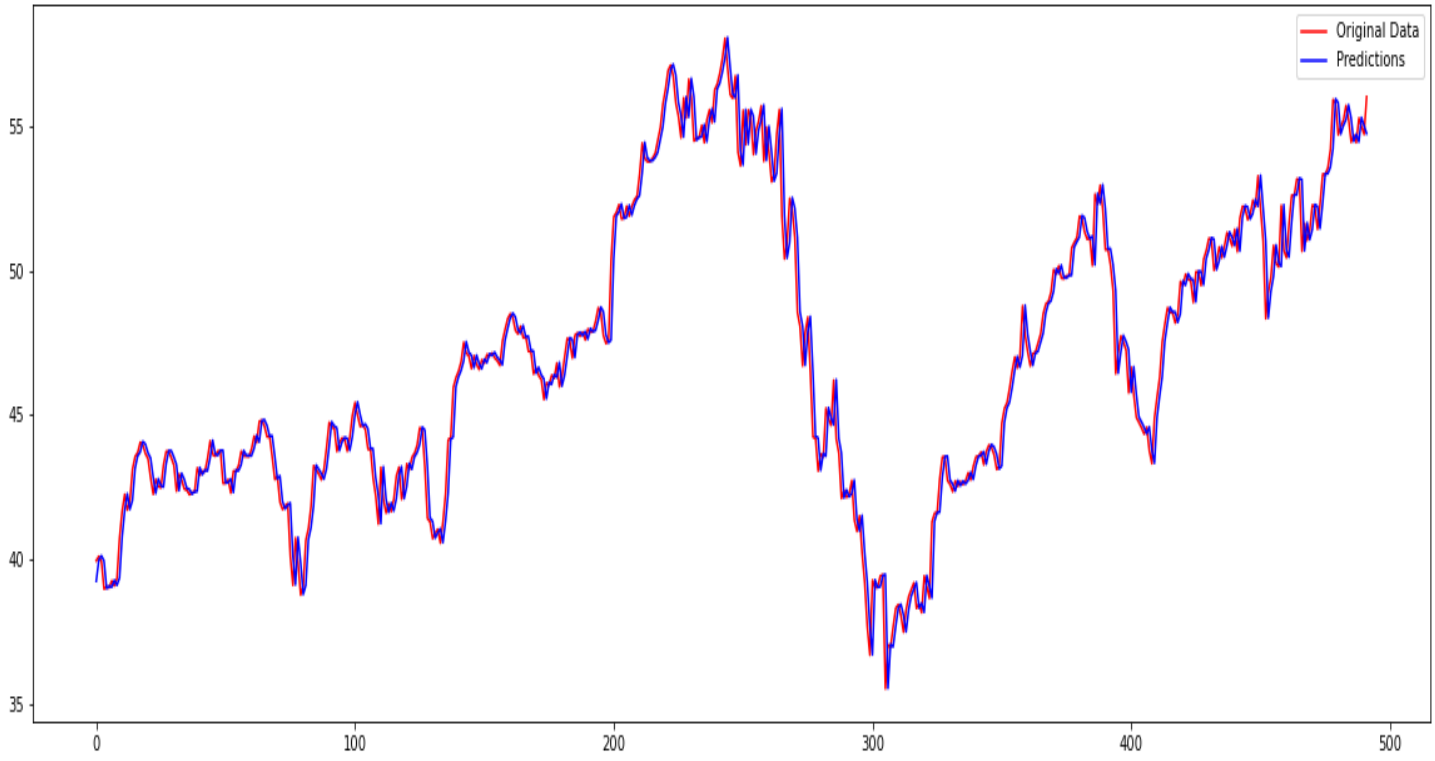
```
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima_model import ARIMA


# evaluate an ARIMA model for a given order (p,d,q)
def evaluate_arima_model(X, arima_order):
    # prepare training dataset
    train_size = int(len(X) * 0.66)
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    # make predictions
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
        model_fit = model.fit(disp=0)
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])

    # calculate out of sample error
    error = mean_squared_error(test, predictions)
    return error


# evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(dataset, p_values, d_values, q_values):
    dataset = dataset.astype('float32')
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p, d, q)
                try:
                    mse = evaluate_arima_model(dataset, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    print('ARIMA%s MSE=%.3f' % (order, mse))
                except:
                    continue
    print('Best ARIMA%s MSE=%.3f' % (best_cfg, best_score))
    return best_cfg
© 2021 GitHub, Inc.
```

4) prediction

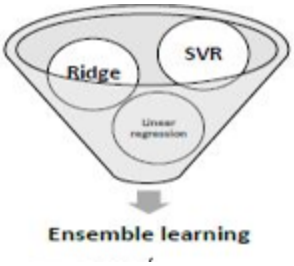
```python
from math import sqrt
import pandas as pd
from statsmodels.tsa.arima_model import ARIMA
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from statsmodels.tsa.arima_model import ARIMAResults

def one_step_prediction(df, best_model):
    df_values = df.values
    size = int(len(df_values) * 0.66)
    train, test = df_values[0:size], df_values[size:len(df_values)]
    history = [x for x in train]
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=best_model)
        model_fit = model.fit(disp=0)
        model_fit.save('Arima_Ford.pkl')
        model_fit = ARIMAResults.load('Arima_Ford.pkl')
        output = model_fit.forecast()

        # model_fit = load_model('Arima.hdf5')
        yhat = output[0]
        predictions.append(yhat)
        obs = test[t]
        history.append(obs)

        print('predicted=%f, expected=%f' % (yhat, obs))


    error = mean_squared_error(test, predictions)
    print("error : ", error)
    predictions_series = pd.Series(predictions)

    # # prediction vs actual
    # fig = plt.figure(figsize=(20, 15))
    # plt.subplot(211)
    # plt.plot(test, color='red', label='Original Data')
    # plt.plot(predictions, color='blue', label='Predictions')
    # plt.legend()
    return predictions_series, test


def accuracy_metrics(predictions_series, test):
    # Accuracy metrics
    mape = mean_absolute_error(predictions_series, test)
    mse = mean_squared_error(predictions_series, test)
    rmse = sqrt(mse)
    print("mape value --> ", mape)
    print("mse value --> ", mse)
    print("rmse value --> ", rmse)
```

5) ARIMA Model

```
get_ipython().run_line_magic('load_ext', 'autoreload')
get_ipython().run_line_magic('reload_ext', 'autoreload')
get_ipython().run_line_magic('autoreload', '2')


#Data Gathering

from arima_utils.data_gathering import get_data
df = get_data('F')


#stationarity tests

import warnings
warnings.filterwarnings("ignore")
from arima_utils.stationarity_tests import rolling_stats
rolling_stats(df)
from arima_utils.stationarity_tests import dickey_fuller
dickey_fuller(df)
from arima_utils.stationarity_tests import log_diff
log_diff(df)
from arima_utils.stationarity_tests import auto_correlation
auto_correlation(df)


#searching for best ARIMA order

warnings.filterwarnings("ignore")
# evaluate parameters
p_values = range(0, 3)
d_values = range(0, 3)
q_values = range(0, 3)
from arima_utils.best_model import evaluate_models
best_model = evaluate_models(df, p_values, d_values, q_values)


#using best arima found in making walk-forward predictions

from arima_utils.predictions import one_step_prediction
predictions_series, test = one_step_prediction(df, best_model)


#accuracy metrics

from arima_utils.predictions import accuracy_metrics
accuracy_metrics(predictions_series, test)
```


Original Data
Predictions
55
50
45
40
35
0
100
200
300
400
500

- Ensemble Learning (Linear regression, SVR, Ridge)

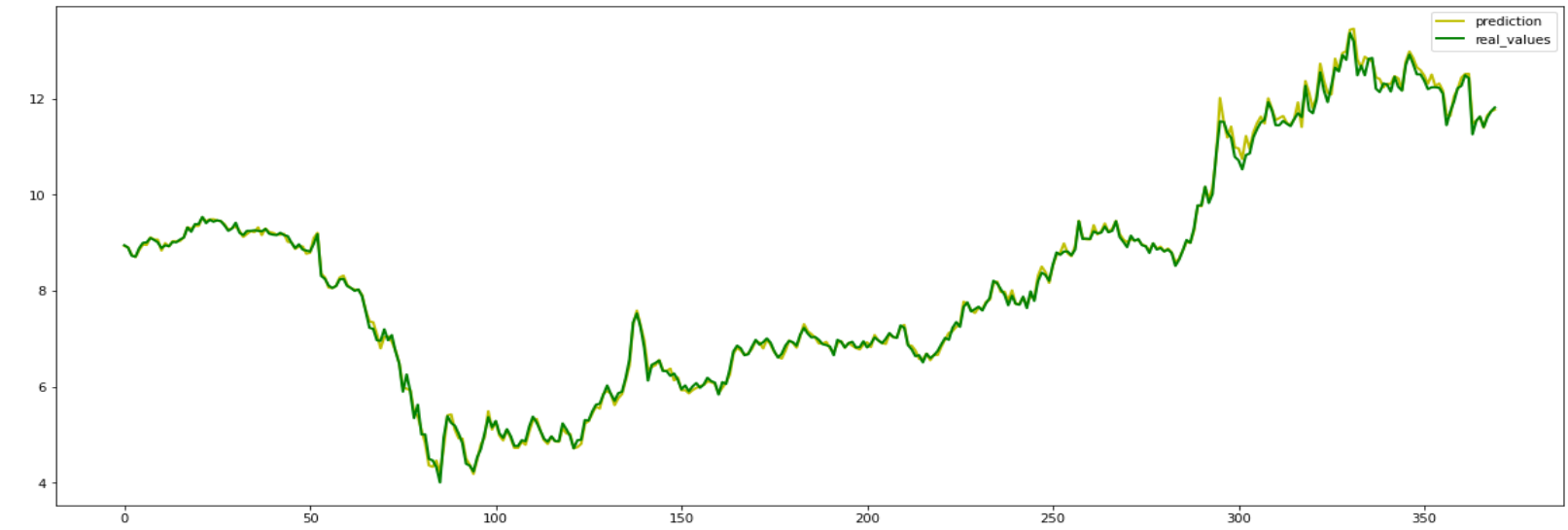


1) **Linear regression**

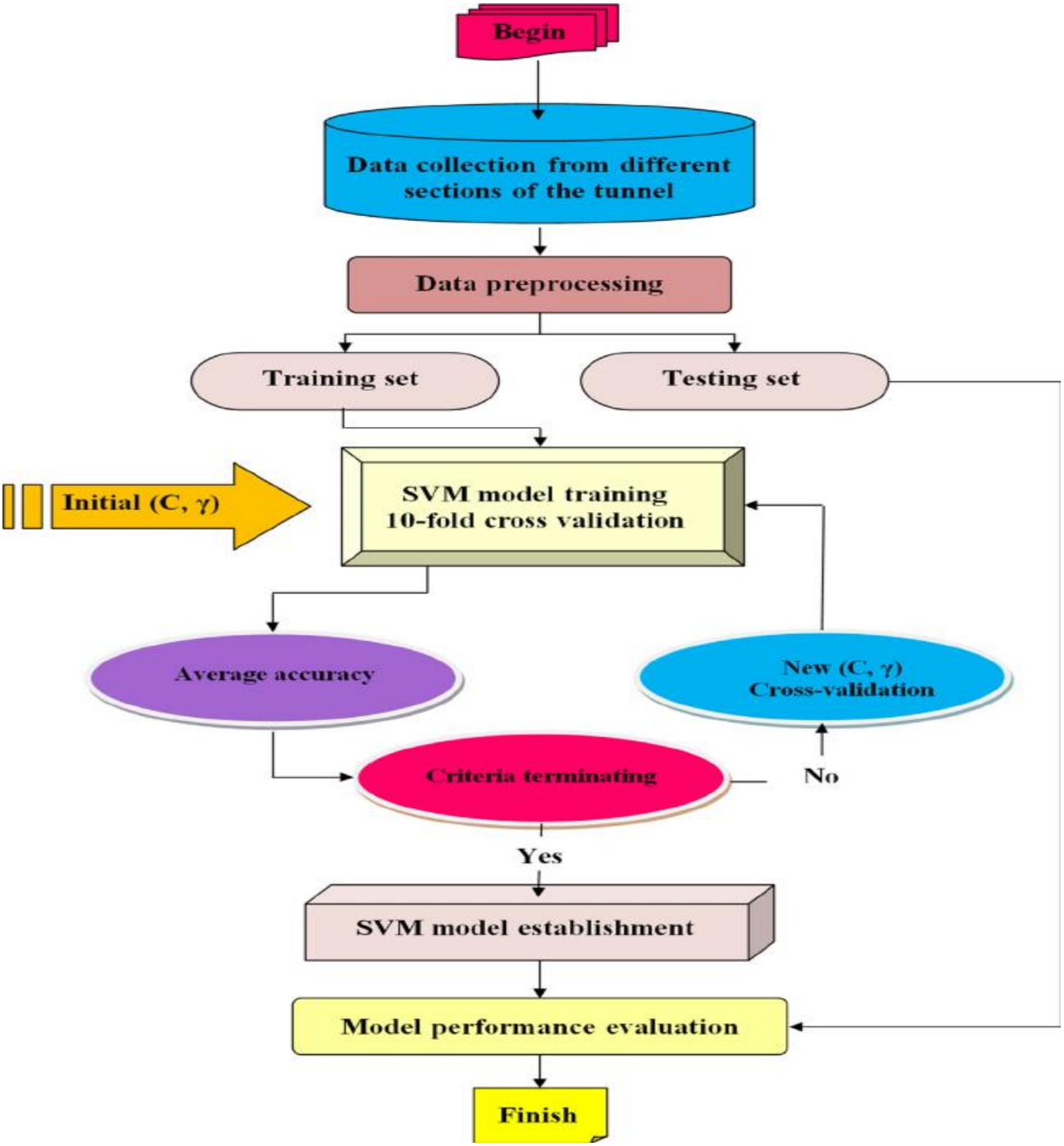
In statistics, **linear regression** is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The case of one explanatory variable is called *simple linear regression*; for more than one, the process is called **multiple linear regression**. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.[3] Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values; less commonly, the conditional median or some other quantile is used. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis.

```python
# Linear Regression Model

def LR (X_train,X_test,y_train,y_test):

  reg=LinearRegression()
  reg.fit(X_train,y_train)

  cv = cross_val_score(reg, X_train, y_train, cv = ts_cross_val, scoring= "neg_mean_squared_error")
  print('Cross Val Score {}'.format(cv))

  LR_predictions=reg.predict(X_test)


  plt.figure(figsize=(20,8))
  plt.plot(reg.predict(X_test[-400:]), "y", label="prediction", linewidth=2.0)
  plt.plot(y_test.values[-400:], "g", label="real_values", linewidth=2.0)
  plt.legend(loc="best")

  MAE_LR=mean_absolute_error(LR_predictions,y_test)
  MSE_LR=mean_squared_error(LR_predictions,y_test)
  print('MAE for Linear regression {} and MSE  is {}'.format(MAE_LR,MSE_LR))

  return MAE_LR,MSE_LR,reg

LR_MAE , LR_MSE ,reg= LR(X_train,X_test,y_train,y_test)
```



2) Support Vector machine (SVR)



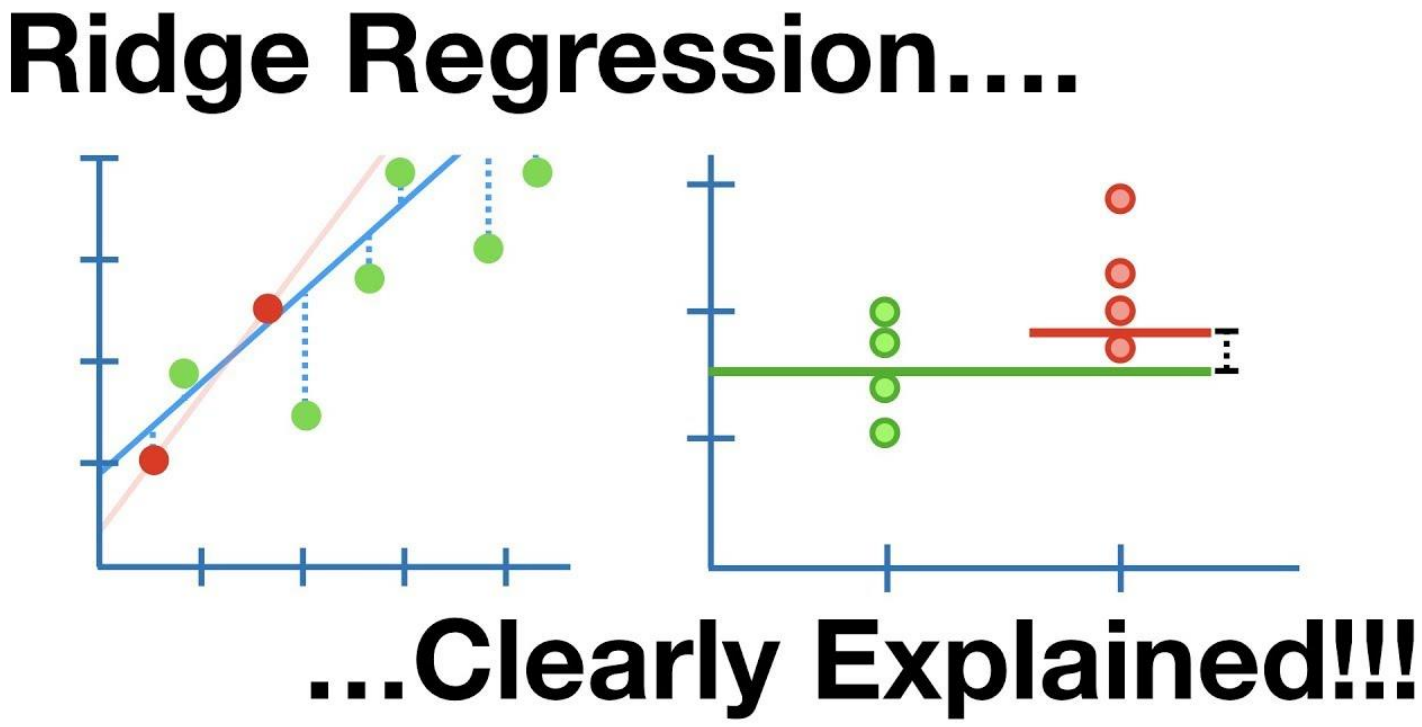
In machine learning, **support-vector machines** (**SVMs**, also **support-vector networks**[]) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Developed at AT&T Bell Laboratories by Vladimir Vapnik with colleagues (Boser et al., 1992, Guyon et al., 1993, Vapnik et al., 1997), SVMs are one of the most robust prediction methods, being based on statistical learning frameworks or VC theory proposed by Vapnik (1982, 1995) and Chervonenkis (1974). Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data are unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The **support-vector clustering** algorithm, created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.

```
# Support Vector Regression Model

def SVR (X_train,X_test,y_train,y_test):
  from sklearn.svm import SVR
  svr = SVR(kernel='linear', C=1000, gamma=1)
  svr.fit(X_train, y_train)

  #cv_svr = cross_val_score(svr, X_train, y_train, cv = ts_cross_val, scoring=
"neg_mean_squared_error")
  #cv_svr

  SVR_predictions =svr.predict(X_test)


  plt.figure(figsize=(20,8))
  plt.plot(svr.predict(X_test[-400:]), "y", label="prediction", linewidth=2.0)
  plt.plot(y_test.values[-400:], "g", label="real_values", linewidth=2.0)
  plt.legend(loc="best")

  MAE_SVR=mean_absolute_error(SVR_predictions,y_test)
  MSE_SVR=mean_squared_error(SVR_predictions,y_test)
  print('MAE for Support Vector regressor {}  and MSE  is {}'.format(MAE_SVR,MSE_SVR))

  return MAE_SVR ,MSE_SVR,svr

SVR_MAE ,SVR_MSE,svr =SVR(X_train,X_test,y_train,y_test)
```

3) Ridge model



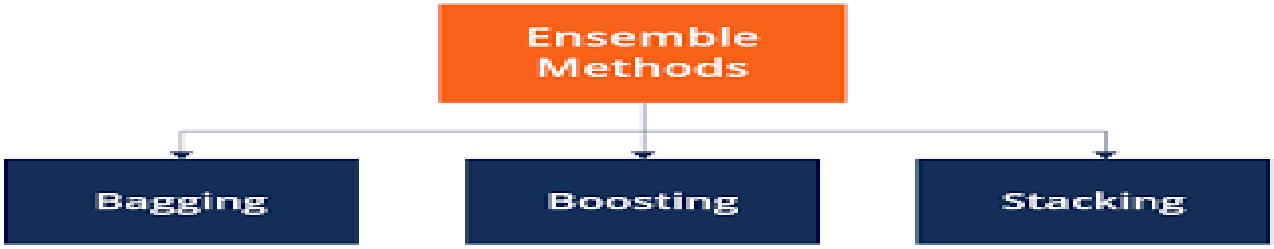
Ridge regression is a method of estimating the coefficients of multiple-regression models in scenarios where independent variables are highly correlated. It has uses in fields including econometrics, chemistry, and engineering.

The theory was first introduced by Hoerl and Kennard in 1970 in their *Technometrics* papers "RIDGE regressions: biased estimation of nonorthogonal problems" and "RIDGE regressions: applications in nonorthogonal problems". This was the result of ten years of research into the field of ridge analysis.

Ridge regression was developed as a possible solution to the imprecision of least square estimators when linear regression models have some multicollinear (highly correlated) independent variables—by creating a ridge regression estimator (RR). This provides a more precise ridge parameters estimate, as its variance and mean square estimator are often smaller than the least square estimators previously derived.

```python
## Ridge Model

def Ridge (X_train,y_train,X_test,y_test):

  from sklearn.linear_model import Ridge
  ridge = Ridge(alpha=0.1)
  ridge.fit(X_train, y_train)#fitting model

  cv_ridge = cross_val_score(ridge, X_train, y_train, cv = ts_cross_val, scoring=
"neg_mean_squared_error")
  print('Cross Val Score {}'.format(cv_ridge))

  #y_pred_ridge_train=ridge.predict(X_train)
  Ridge_predictions=ridge.predict(X_test)

  plt.figure(figsize=(20,8))
  plt.plot(ridge.predict(X_test[-400:]), "y", label="prediction", linewidth=2.0)
  plt.plot(y_test.values[-400:], "g", label="real_values", linewidth=2.0)
  plt.legend(loc="best")

  MAE_ridge=mean_absolute_error(Ridge_predictions,y_test)
  MSE_ridge=mean_squared_error(Ridge_predictions,y_test)
  print('MAE for Ridge regressor {} and MSE  is {}'.format(MAE_ridge,MSE_ridge))

  return MAE_ridge,MSE_ridge,ridge

Ridge_MAE , Ridge_MSE,ridge = Ridge(X_train,y_train,X_test,y_test)
```

- Prepare the data and ensemble code



Empirically, ensembles tend to yield better results when there is a significant diversity among the models. Many ensemble methods, therefore, seek to promote diversity among the models they combine. Although perhaps non-intuitive, more random algorithms (like random decision trees) can be used to produce a stronger ensemble than very deliberate algorithms (like entropy-reducing decision trees). Using a variety of strong learning algorithms, however, has been shown to be more effective than using techniques that attempt to *dumb-down* the models in order to promote diversity.

```python
#Import packages
import pandas_datareader as pdr
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Import Models
from sklearn.linear_model import Ridge
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import VotingRegressor,BaggingRegressor
# Import Scalling and Normalization functions
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

#Import Metrices
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score,mean_absolute_error,r2_score,mean_squared_error
ts_cross_val = TimeSeriesSplit(n_splits=5)

"""# Prepare Data"""

def prepare_data (X):

    df = pdr.DataReader(X,data_source='yahoo' ,start='2014-01-01') #read data
    df.drop('Volume',axis='columns',inplace=True)
    print(df.info())
    print('-----------------')
    print(df.head())
    print('-----------------')
    print('Null values in data = {}'.format(df.isna().sum().sum())) # See if there are null values

    #df=df.reset_index() #drop Date Column from input
    #scaler = MinMaxScaler(feature_range=(0,1))
    #df[df.columns] = scaler.fit_transform(df)  #^^^^^^^^^^^^^
    X=df[['High','Low','Open','Adj Close']]#input columns
    y=df[['Close']] #output column

    #Scalling input data
    #sc_x = StandardScaler()
    #x = sc_x.fit_transform(X)
    #X = pd.DataFrame(x,columns=[X.columns])



    test_size=0.2
    test_index = int(len(X)*(1-test_size))
    X_train = X.iloc[:test_index]
    y_train = y.iloc[:test_index]
    X_test = X.iloc[test_index:]
    y_test = y.iloc[test_index:]

    return X_train, y_train, X_test, y_test
```

```
# Average Ensamble


def Ensemble_Learing(X_train,X_test,y_train,y_test):
  reg_voting=VotingRegressor(estimators=[('reg',reg),
                                         ('svr',svr),
                                         ('ridge',ridge)],
                                         weights=[5,0,5])
  reg_voting.fit(X_train,y_train)
  y_pred=reg_voting.predict(X_test)
  MAE_ENS=mean_absolute_error(y_pred,y_test)
  MSE_ENS=mean_squared_error(y_pred,y_test)
  print('MAE for Support Vector regressor {}  and MSE  is {}'.format(MAE_ENS,MSE_ENS))
  return MAE_ENS,MSE_ENS,reg_voting

ENS_MAE,ENS_MSE,ENS =Ensemble_Learing(X_train,X_test,y_train,y_test)

"""## Bagging Ensemble"""

reg_bag=BaggingRegressor(base_estimator=ENS,n_estimators=2)
reg_bag.fit(X_train,y_train)
pred=reg_bag.predict(X_test)
print(mean_absolute_error(pred,y_test))
print(mean_squared_error(pred,y_test))
r2_score(pred,y_test)

ENS_MSE=mean_squared_error(pred,y_test)
ENS_MAE=mean_absolute_error(pred,y_test)
```

The Deployment

We used FastAPI to deploy our models.

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.

The key features are:

- **Fast**: Very high performance, on par with **NodeJS** and **Go** (thanks to Starlette and Pydantic). One of the fastest Python frameworks available.
- **Fast to code**: Increase the speed to develop features by about 200% to 300%. *
- **Fewer bugs**: Reduce about 40% of human (developer) induced errors. *
- **Intuitive**: Great editor support. Completion everywhere. Less time debugging.
- **Easy**: Designed to be easy to use and learn. Less time reading docs.
- **Short**: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
- **Robust**: Get production-ready code. With automatic interactive documentation.
- **Standards-based**: Based on (and fully compatible with) the open standards for APIs: OpenAPI (previously known as Swagger) and JSON Schema.

```python
app = FastAPI()


@app.get('/')
def index():
    return {'message': 'This is your fav stock predictor!'}


@app.post('/predict')
async def predict_price(data: str):

    if data == 'F':

      prophet_prediction = float(prophet(data))
      arima_prediction, diff = arima(data)
      model_prediction, lstm_prediction = lstm(data)
      reg_prediction,reg_diff = Regression(data)
      trans_prediction, trans_difference = Transformer(data)
      close=real(data)

      indicators=np.array(indicator(reg_prediction[0],arima_prediction[0]
                                    ,lstm_prediction,prophet_prediction
                                    ,trans_prediction,close))
      weights=np.array([0.3,0.3,0.3,0.05,0.05])


      final_decision=indicators*weights
      final_decision=final_decision.sum()
      final_decision=final_decision.item(0)
      decision_array=np.array([-1,final_decision,1],dtype=np.float)
      decision_array=d3_scale(decision_array)
      final_decision=round(decision_array[1],1)*10

      return {
        'Decision': final_decision

            }

    else:
      return {"the ticker not supported yet"}


# App startup
ngrok_tunnel = ngrok.connect(8090)
print('Public URL:', ngrok_tunnel.public_url)
nest_asyncio.apply()
uvicorn.run(app, port=8090)
```

```python
def lstm(data_set):
    """
    Getting the desired data from yahoo, then doing some data manipulation such as data
    reshaping
    Args:
        (str) data_set - the ticker of desired dataset (company)
    Returns:
        (float) diff_prediction - the model out-put (the prediction of the next day)
        (float) real_prediction - the model output + today's price (real price of tomorrow)
    """

    # data gathering
    df = pdr.DataReader(data_set, data_source='yahoo', start=date.today() - timedelta(100))

    # data manipulation

    # creating a new df with Xt - Xt-1 values of the close prices (most recent 60 days)
    close_df = df['2012-01-01':].reset_index()['Close'][-61:]
    close_diff = close_df.diff().dropna()
    data = np.array(close_diff).reshape(-1, 1)

    # reshaping the data to 3D to be accepted by our LSTM model
    model_input = np.reshape(data, (1, 60, 1))

    # loading the model and predicting
    loaded_model = load_model("lstm_f_60.hdf5")
    model_prediction = float(loaded_model.predict(model_input))
    real_prediction = model_prediction + df['Close'][-1]


    return model_prediction, real_prediction
```

```
""" Models used"""

def arima(ticker):
  """
  Forcasting using ARIMA ! by Getting the desired data from yahoo,
  then finding the best order of arima params then the comes the ARIMA's turn
  Args:
      (str) ticket - the ticker of desired dataset (company)
  Returns:
      (float) arima_output - the model out-put (the prediction of the next day)
      (float) diff - the model output - today's price (the diff between tomorrow's prediction and
today's real value)
  """

  # data gathering
  df = pdr.DataReader(ticker, data_source='yahoo', start='2016-01-01')
  df.index = pd.to_datetime(df.index, format="%Y/%m/%d")
  df = pd.Series(df['Close'])
  last_day=df[-1]

  # finding the best order
  auto_order = pm.auto_arima(df, start_p=0, start_q=0, test='adf', max_p=3, max_q=3,
m=1,d=None,seasonal=False
                     ,start_P=0,D=0,
trace=True,error_action='ignore',suppress_warnings=True,stepwise=True)
  best_order = auto_order.order

  # model fitting
  model = ARIMA(df, order=best_order)
  model_fit = model.fit(disp=0)
  arima_prediction ,se, conf = model_fit.forecast(1)

  diff = arima_prediction - last_day

  return arima_prediction , diff
```

```python
def Regression(ticker):
  """
  Forcasting using an ensambled model between SVR, Ridge and Linear regression! by Getting the desired
data from yahoo,
  then doing some data manipulation
  Args:
      (str) ticket - the ticker of desired dataset (company)
  Returns:
      (float) arima_output - the model out-put (the prediction of the next day)
      (float) diff - the model output - today's price (the diff between tomorrow's prediction and
today's real value)
  """
  start_date = datetime.now() - timedelta(7)
  start_date = datetime.strftime(start_date, '%Y-%m-%d')

  df = pdr.DataReader(ticker, data_source='yahoo', start=start_date)  # read data
  df.drop('Volume', axis='columns', inplace=True)

  if df.index[-1]==datetime.now().strftime('%Y-%m-%d') :
    df=df.iloc[-2:]
  else :
    df =df.iloc[-1:]
  X = df[['High', 'Low', 'Open', 'Adj Close']]  # input columns
  y = df[['Close']]  # output column
  input = X
  loaded_model = pickle.load(open('regression_model.pkl', 'rb'))
  reg_prediction = loaded_model.predict(input)
  reg_diff=reg_prediction-df.Close[-1]

  return  reg_prediction,reg_diff
```

```
""" Models used"""

def prophet (ticker):
  """
  Forcasting using prophet ! by Getting the desired data from yahoo, then doing some data manipulation,
then the comes the prophet's turn
  Args:
      (str) ticket - the ticker of desired dataset (company)
  Returns:
      (float) prophet_output - the model out-put (the prediction of the next day)
  """

  # data_gathering
  df = pdr.DataReader(ticker, data_source='yahoo', start='2015-01-01')

  # data manipulation
  holiday = pd.DataFrame([])
  for date, name in sorted(holidays.UnitedStates(years=[2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019,
2020, 2021]).items()):
      holiday = holiday.append(pd.DataFrame({'ds': date, 'holiday': "US-Holidays"}, index=[0]),
ignore_index=True)
  holiday['ds'] = pd.to_datetime(holiday['ds'], format='%Y-%m-%d', errors='ignore')

  # data frame modification to be accepted by prophet
  data = df['Close'].reset_index()
  data.columns = ['ds', 'y']

  # model building
  m = Prophet(holidays=holiday,seasonality_mode='additive', changepoint_prior_scale = 0.1,
seasonality_prior_scale=0.01)
  m.fit(data)

  # model predictions
  future = m.make_future_dataframe(periods=1)
  model_prediction = m.predict(future)
  prophet_prediction = float(model_prediction[ 'yhat'][-1:])
  return prophet_prediction
```

- Voting

```
def
indicator(reg_pred,arima_pred,lstm_pred,prophet_pred,trans_pred,real):
  percent=real*0.5/100
  predictions=[reg_pred,arima_pred,lstm_pred,prophet_pred,trans_pred]
  decisions=[]
  for i in predictions:
    if i>=real+percent:
      action=1
    elif i<real-percent:
      action=-1
    else:
      action=0
    decisions.append(action)
  return decisions
```

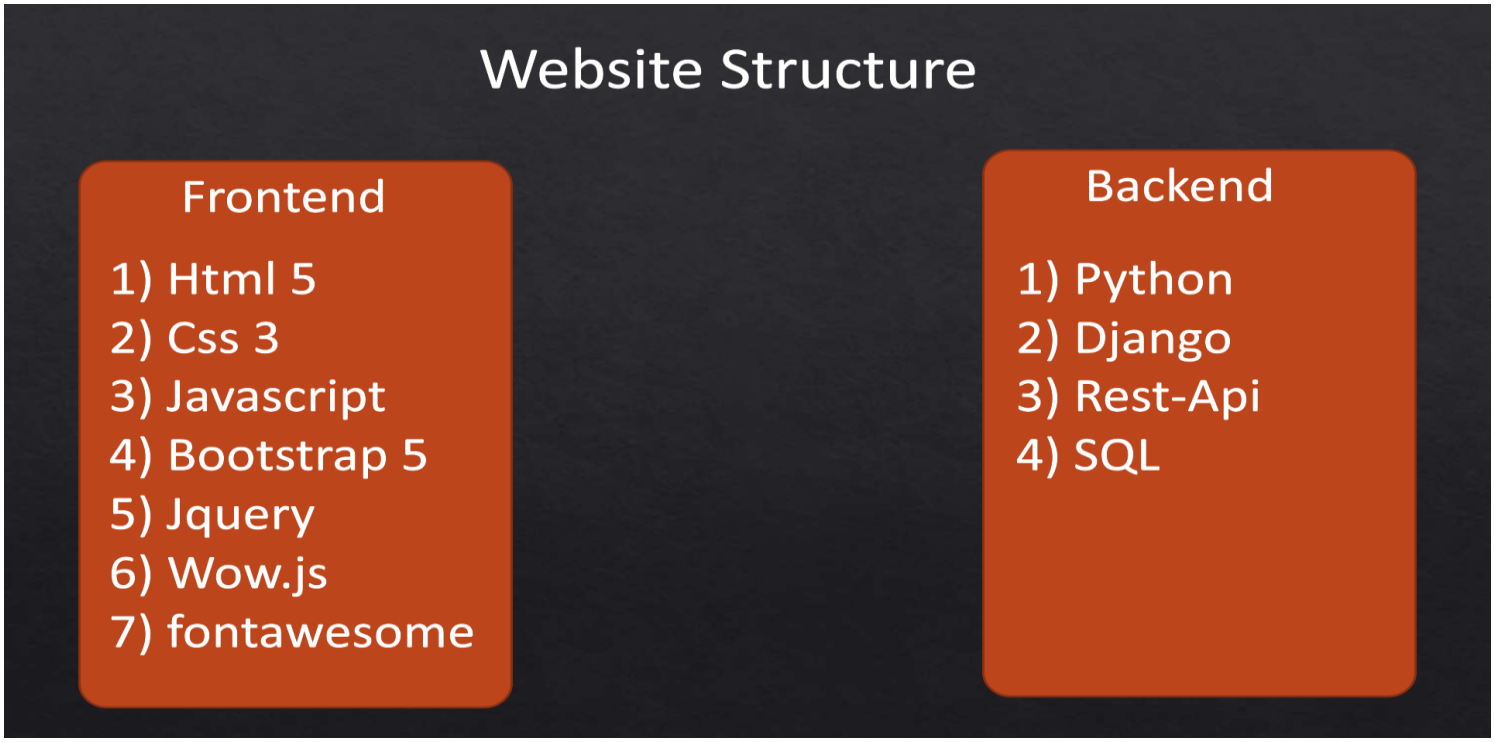
```
def d3_scale(data, out_range=(0, 1)):
  '''
    Scale values from 0 to 1
    Args:
        (float) data - The unscaled data
        (tuple) data - The range of the new scaled data

  Returns:
      (Array) array that has values from 0 into 1  '''
  domain = [np.min(data, axis=0), np.max(data, axis=0)]

  def interp(x):
      return out_range[0] * (1.0 - x) + out_range[1] * x

  def uninterp(x):
      b = 0
      if (domain[1] - domain[0]) != 0:
          b = domain[1] - domain[0]
      else:
          b =  1.0 / domain[1]
      return (x - domain[0]) / b

  return interp(uninterp(data)
```

5.2 Evaluation matrix

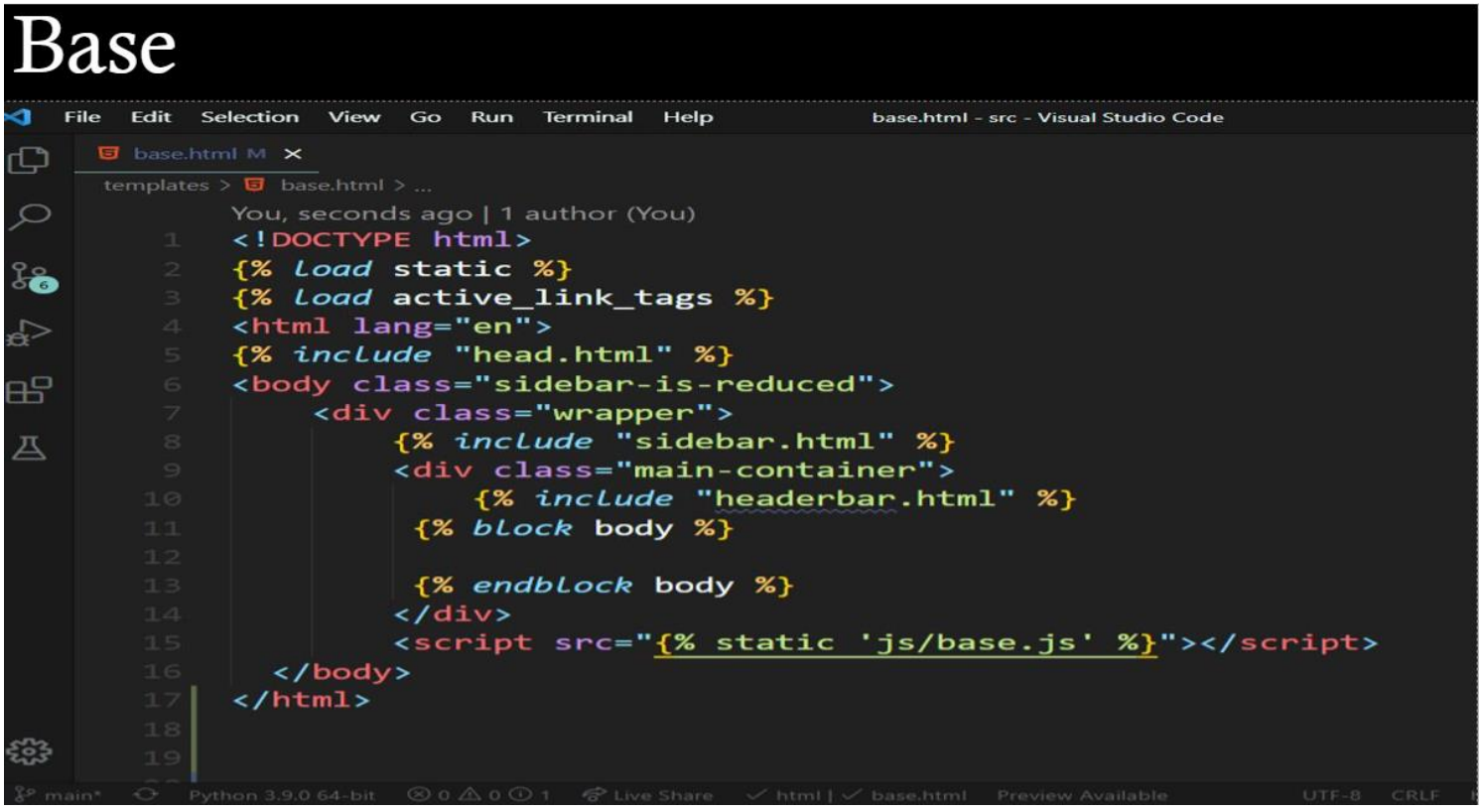
Here the evaluation matrix between all the models that we used.

Model name	MAE
Ensemble	0.0637
Linear regression	0.0683
Ridge	0.0685
SVR	0.0807
ARIMA	0.5781
LSTM	0.6708
Prophet	1.7468
Transformer	1.8491

5.2 User interface



- **HTML code**



Splash home content

splash > templates > splash.html > ...

You, seconds ago | 1 author (You)

```
{% extends "splash_home.html" %}
{% load static %}
{% block title %} Home {% endblock title %}
{% block splash_content %}
    <video autoplay muted loop id="myVideo">
        <source src="{% static 'img/videos/back2.mp4' %}" type="video/mp4">
    </video>
    <div class="box" style="clip-path: polygon(50% 50%, 100% 70%, 100% 99%, 0 100%, 0 70%);
    padding-top:150px; margin-bottom: 0;opacity: 0.9;">
        <div class="content">
          <span class="icon_logo"><i class="fa fa-bar-chart" aria-hidden="true"
            style="font-size: 50px;color: #3498db"></i></span>
            <h1>FIN<span>ANCE ST</span>OCK</h1>
            <div class="btns" style="margin-top: 80px;">
               <span><a href="{% url 'login' %}" class="btn third">Get Start</a></span>
               <span><a href="{% url 'accounts:register' %}" class="btn third">Sign Up</a></span>
            </div>
        </div>
    </div>
{% endblock splash_content %}
```

About Us extends

SRC
- about_us
  - __pycache__
  - migrations
  - templates
    - about_us... M

```
{% extends 'base.html' %}
  {% block title %} About Us {% endblock title %}
  {% block body %}
    {% include "about_us_content.html" %}
  {% endblock body %}
```

About Us content

SRC
- about_us
  - __pycache__
  - migrations
  - templates
    - about_us... M
    - about_us_... U
    - about_us.html
  - __init__.py
  - admin.py
  - apps.py
  - models.py
  - tests.py
  - urls.py
  - views.py

```
{% load static %}
  <div class="user-box first-box">···
  </div>
  <script>
  var p = $('.project_intro');
  var text = p.text().split(' ');
  for (var i = 0, len = text.length; i < len; i = i + 1) {
      text[i] = '<mark>' + text[i] + '</mark>';
  }
  p.html(text.join(' '));
  </script>
```

Stock News content

```
<!-- stock news component -->
{% block body %}
<!-- stock page content -->
<div class="news_row" style="margin-top:40px;">
    <!-- add followed stocks that => [in profile with user adding ]-->
    <!-- display srocks news for companies -->
<div class="news_card " style="margin-bottom:30px;">
    <div class="tradingview-widget-container">···
    </div>
</div>
</div>
{% endblock body %}
```

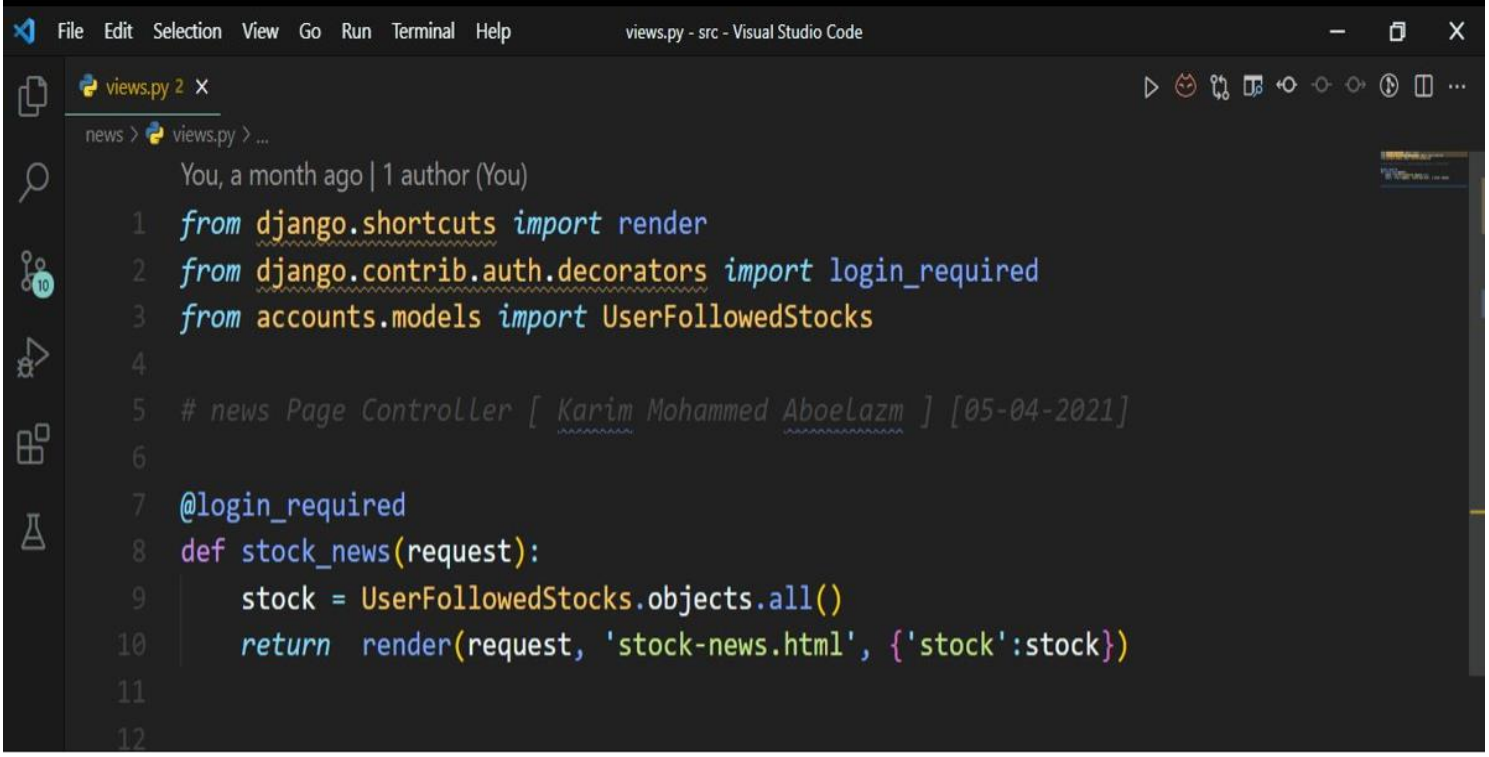
Profile Page content cont.

accounts > templates > content > pro_continent.html > div.user-box.first-box

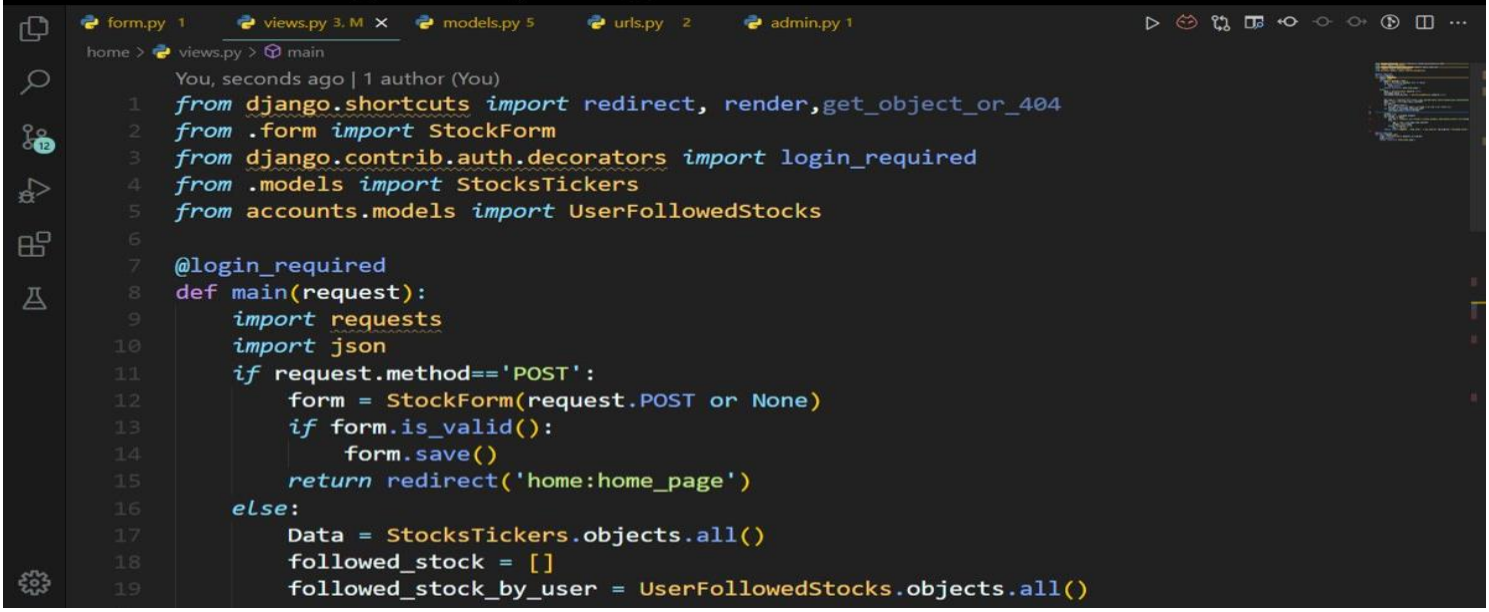
```
            {% if profile.state and profile.city %}
            <li><div><span class="tag activity-link active">Address    : </span>
                <span class="span_dec">{{profile.state|capfirst}} - {{profile.city}}</span>
            </div></li>
            {% else %}
            <li><div><span class="tag activity-link active">Address    : </span>
                <span>........</span>
            </div></li>
            {% endif %}
        </ul>
    </div>

    <div class="discount card user_img_container user_pic" style="--delay: .4s">
        {% if profile.image %}
        <div class="user_img">
            <img src="{{profile.image.url}}" alt="This is The User Image .. " style="width: 280px;h
        </div>
        {% else %}
        <div class="user_img">
            <img src="{% static 'img/user.png' %}" alt="This is The User Image .. " style="width: 2
        </div>
        {% endif %}
    </div>
    You, a month ago • first commit
```

- **Django code**

News Page Django views



Home Page Django views



```
        api_req_2 = requests.get("http://api.marketstack.com/v1/eod?access_key=8c161f
        api2 = json.loads(api_req_2.content)
        all_api=[]
        for j in api2["data"]:
            all_api.append([j['open'],j['high'],j['low'],j['close']]);
        for follow in followed_stock_by_user:
            followed_stock.append(follow)
        output = []
        my_ziplist = zip(Data,output)
        for ticker in Data:
            api_req = requests.get("https://cloud.iexapis.com/stable/stock/"+str(tick
            try:
                api = json.loads(api_req.content)
                output.append(api)
            except Exception as e:
                api = "Error"
                context={'my_ziplist':my_ziplist,
                         'followed_stock':followed_stock,'all_api':all_api}
        return render(request, 'home.html', context)

@login_required
def dell(request,id):
    item = StocksTickers.objects.get(id=id)
    item.delete()
    return redirect('home:home_page')
```

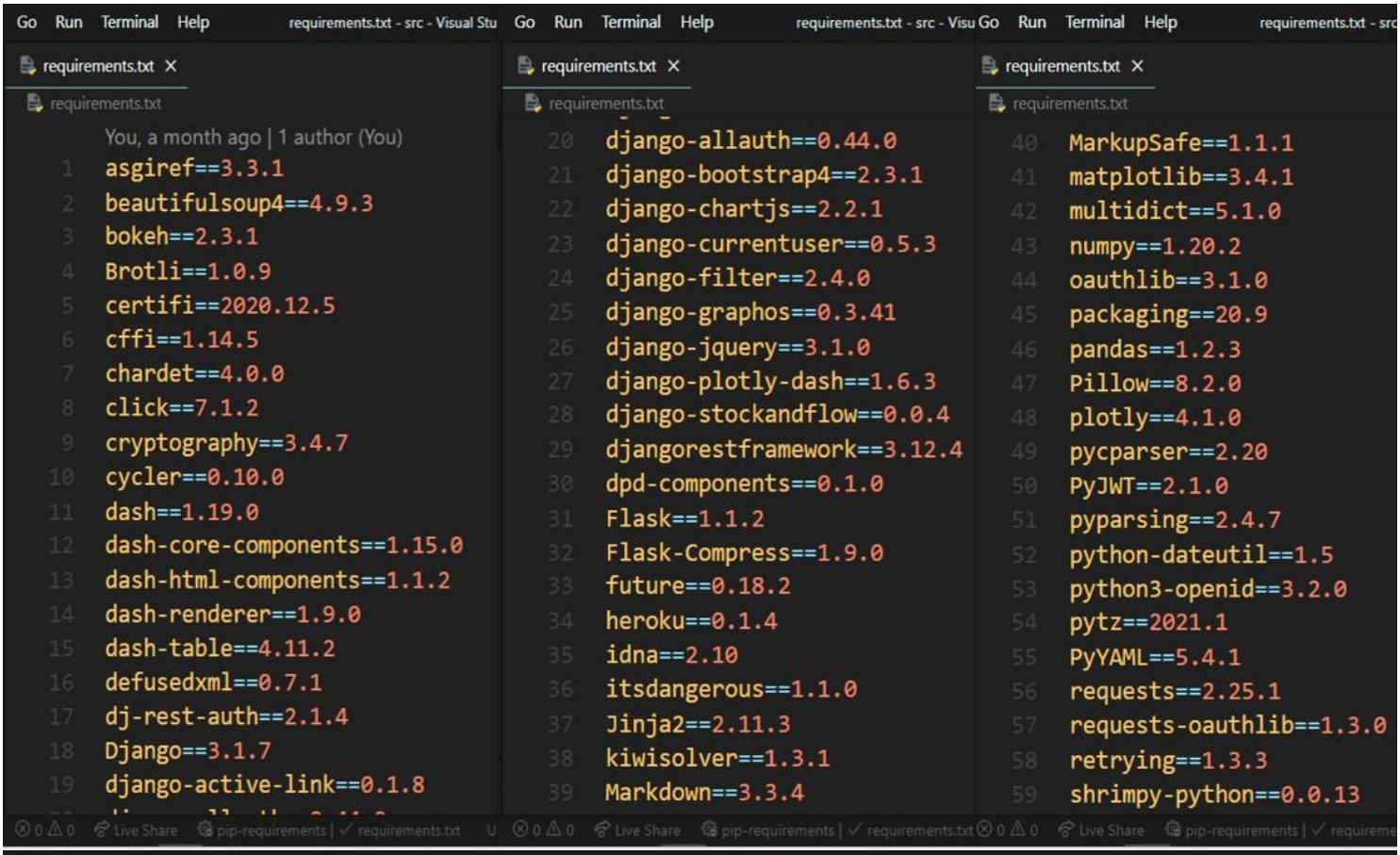
Home Page Django Form

```
You, seconds ago | 1 author (You)
from django.contrib.auth.models import User
from django.db import models
from django.contrib.auth.models import User
from django_currentuser.middleware import (get_current_user,
                                           get_current_authenticated_user)
from django_currentuser.db.models import CurrentUserField

You, a month ago | 1 author (You)
class StocksTickers(models.Model):
    ticker = models.CharField(max_length=200,unique=True)
    created_at = models.DateTimeField(blank=True, null=True,auto_now_add=True)
    created_by = CurrentUserField()

    def __str__(self):
        return str(self.ticker)
```

Home Page Django Models

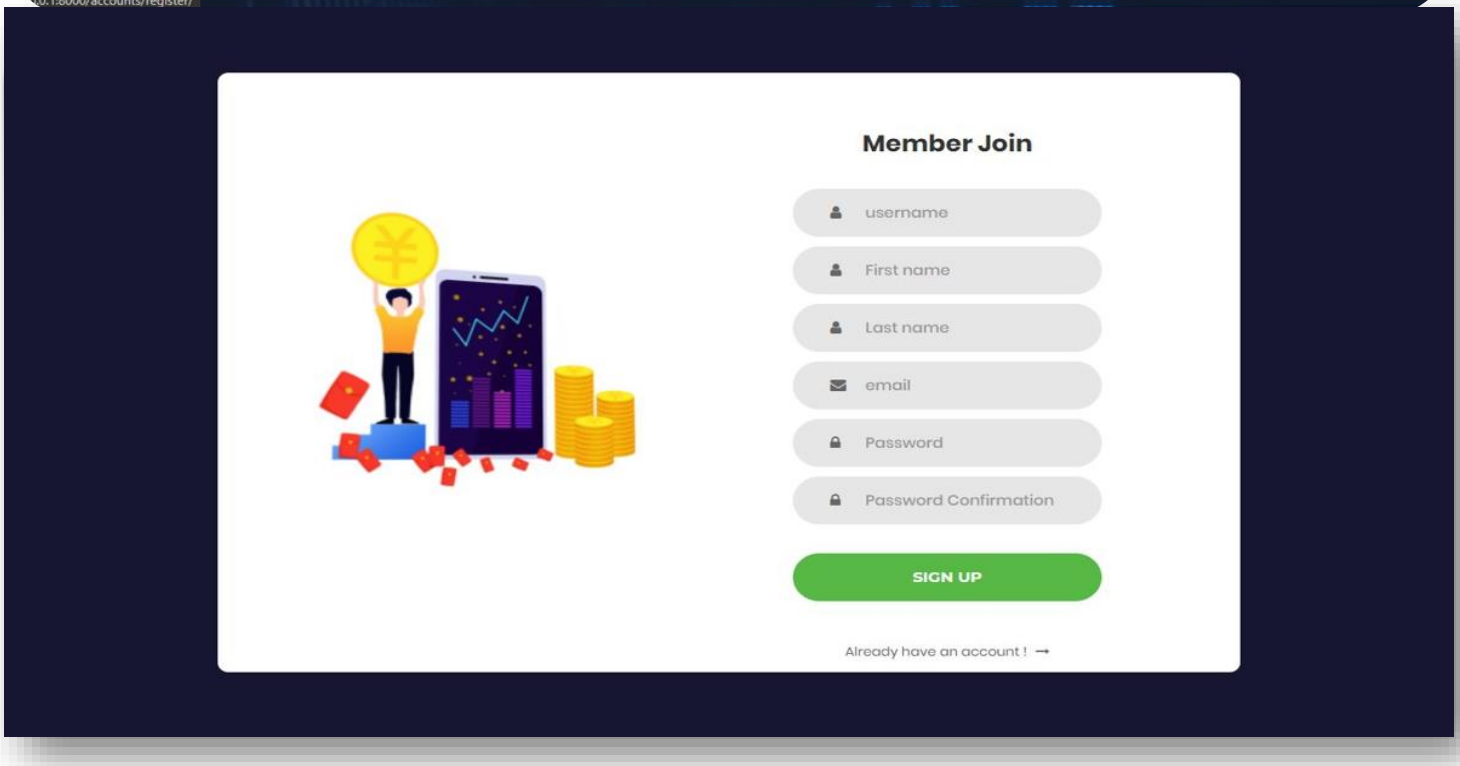
```
You, seconds ago | 1 author (You)
from django import forms
from . models import StocksTickers
You, seconds ago | 1 author (You)
class StockForm(forms.ModelForm):
    You, seconds ago | 1 author (You)
    class Meta:
        model = StocksTickers
        fields = '__all__'
```

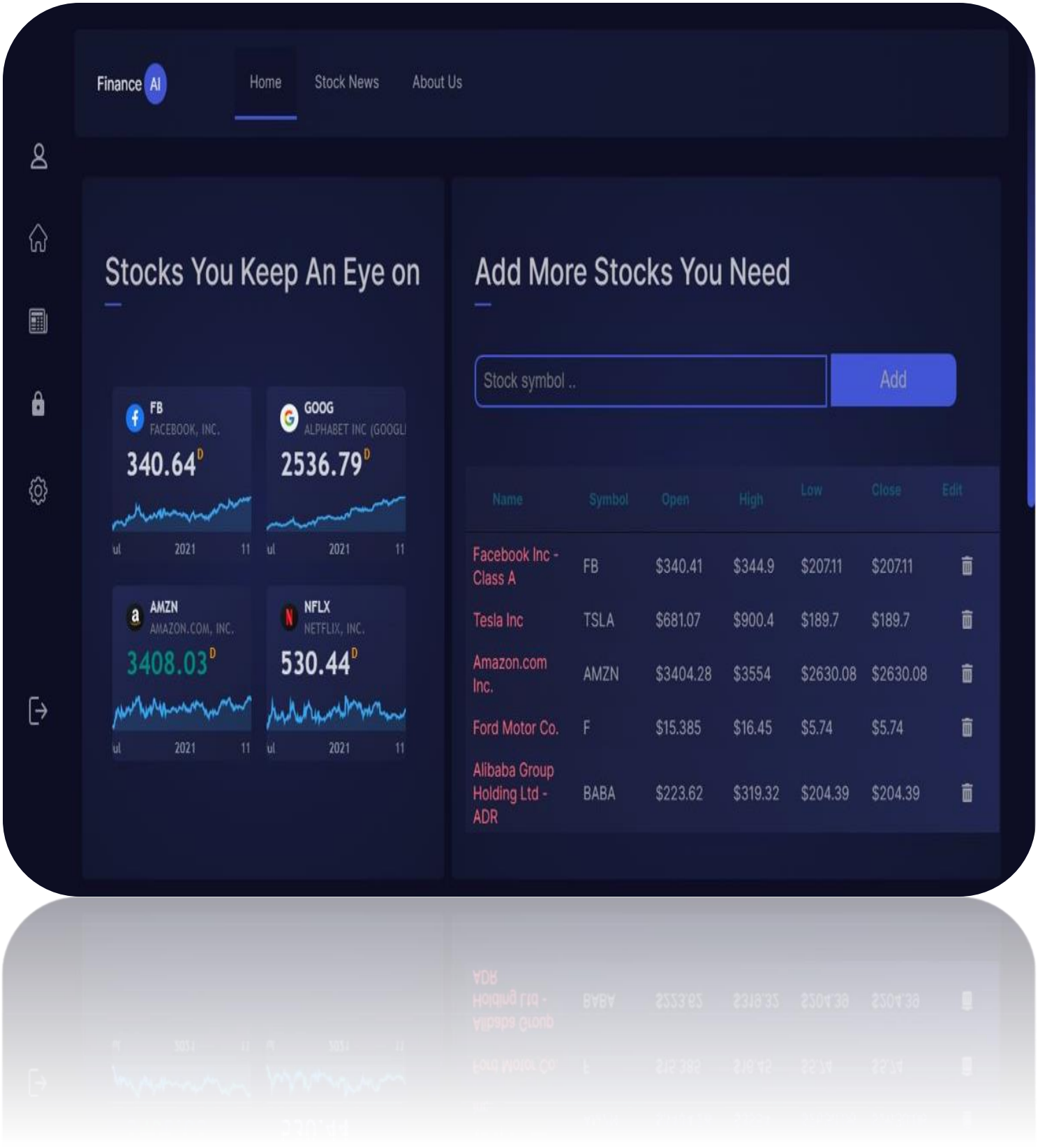
Django Libraries I had been Used

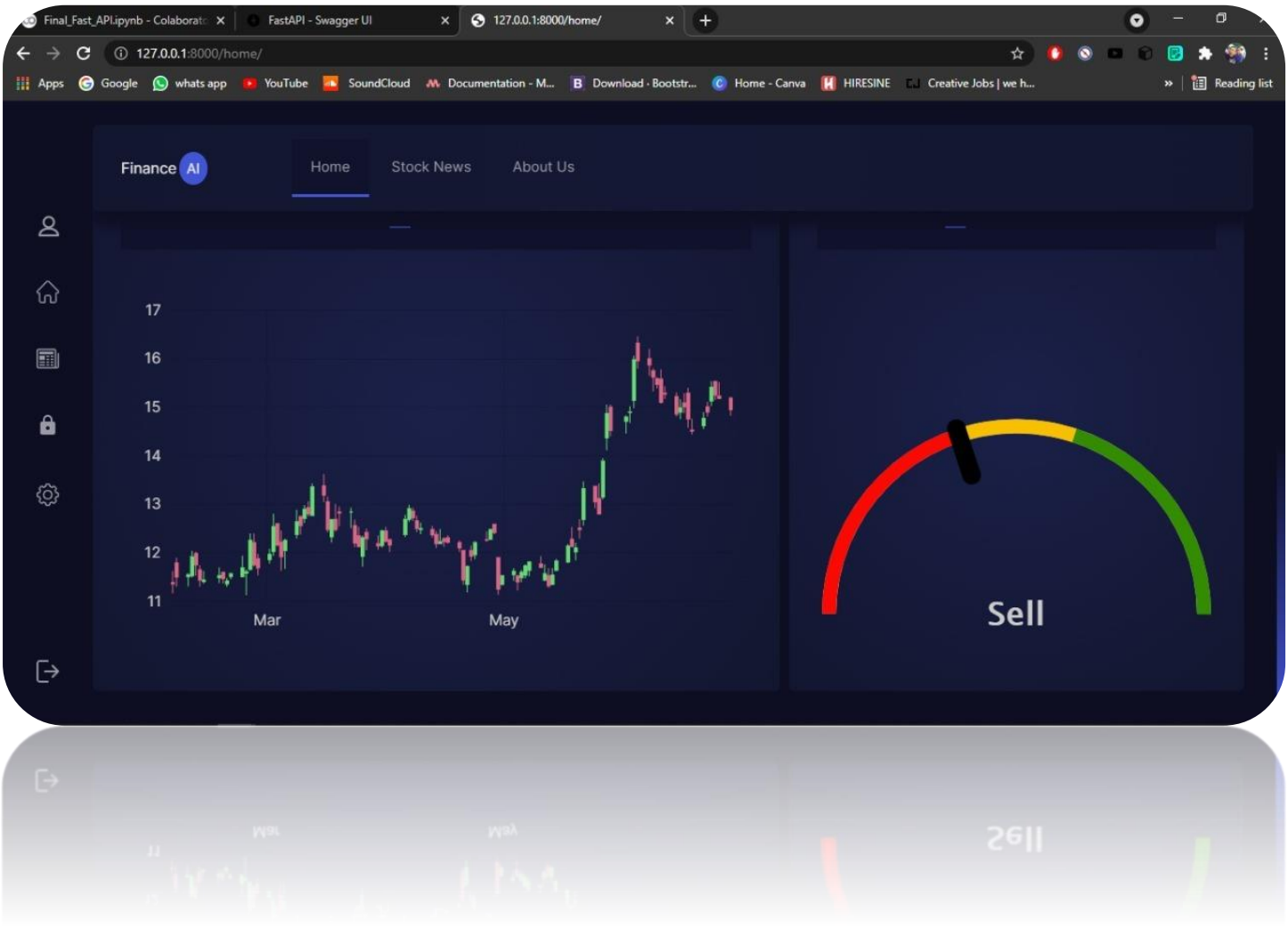


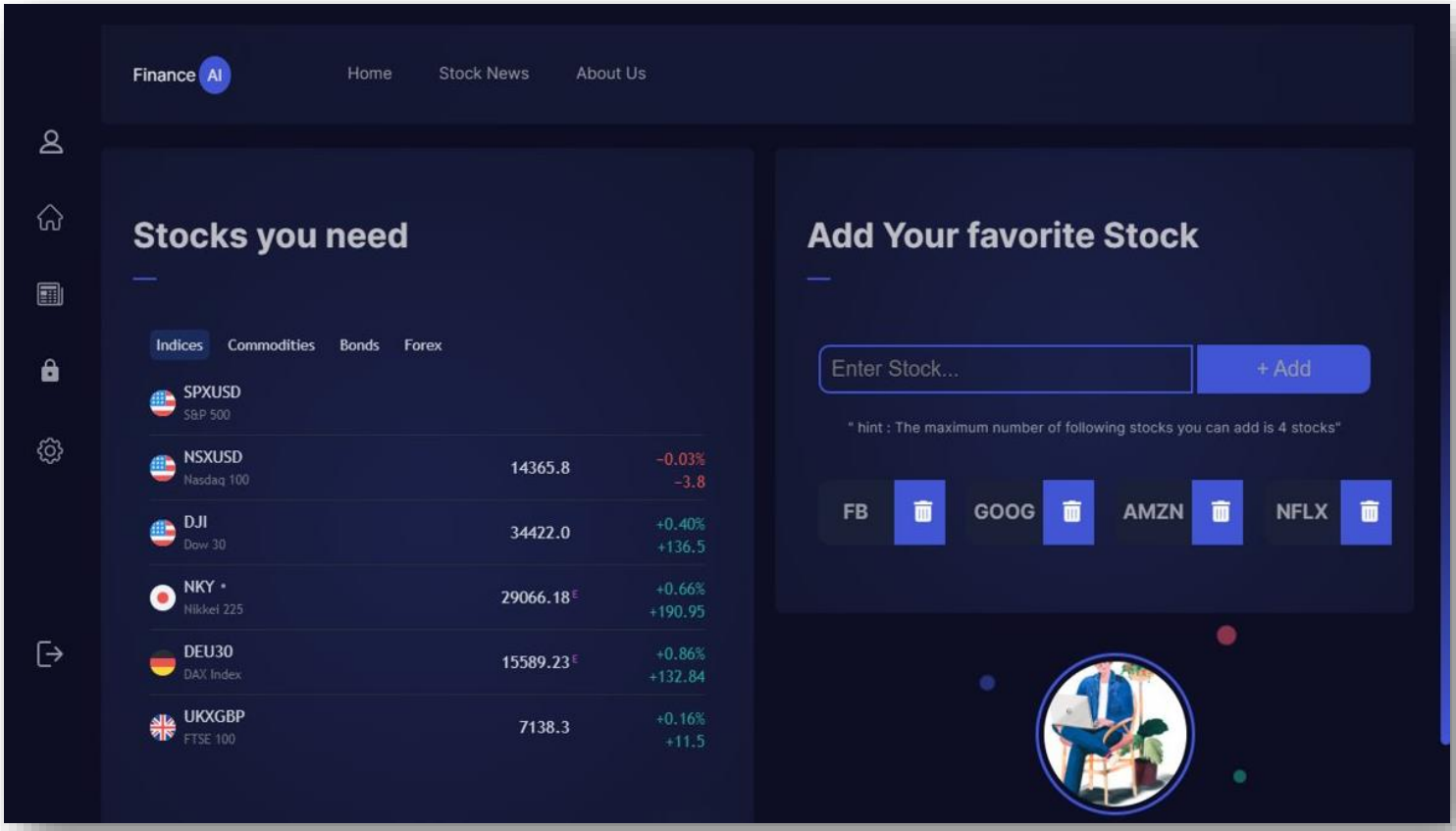
```
asgiref==3.3.1
beautifulsoup4==4.9.3
bokeh==2.3.1
Brotli==1.0.9
certifi==2020.12.5
cffi==1.14.5
chardet==4.0.0
click==7.1.2
cryptography==3.4.7
cycler==0.10.0
dash==1.19.0
dash-core-components==1.15.0
dash-html-components==1.1.2
dash-renderer==1.9.0
dash-table==4.11.2
defusedxml==0.7.1
dj-rest-auth==2.1.4
Django==3.1.7
django-active-link==0.1.8
django-allauth==0.44.0
django-bootstrap4==2.3.1
django-chartjs==2.2.1
django-currentuser==0.5.3
django-filter==2.4.0
django-graphos==0.3.41
django-jquery==3.1.0
django-plotly-dash==1.6.3
django-stockandflow==0.0.4
djangorestframework==3.12.4
dpd-components==0.1.0
Flask==1.1.2
Flask-Compress==1.9.0
future==0.18.2
heroku==0.1.4
idna==2.10
itsdangerous==1.1.0
Jinja2==2.11.3
kiwisolver==1.3.1
Markdown==3.3.4
MarkupSafe==1.1.1
matplotlib==3.4.1
multidict==5.1.0
numpy==1.20.2
oauthlib==3.1.0
packaging==20.9
pandas==1.2.3
Pillow==8.2.0
plotly==4.1.0
pycparser==2.20
PyJWT==2.1.0
pyparsing==2.4.7
python-dateutil==1.5
python3-openid==3.2.0
pytz==2021.1
PyYAML==5.4.1
requests==2.25.1
requests-oauthlib==1.3.0
retrying==1.3.3
shrimpy-python==0.0.13
six==1.15.0
soupsieve==2.2.1
sqlparse==0.4.1
tornado==6.1
typing-extensions==3.7.4.3
urllib3==1.26.4
websockets==8.1
Werkzeug==1.0.1
whitenoise==5.2.0
yarl==1.1.0
```

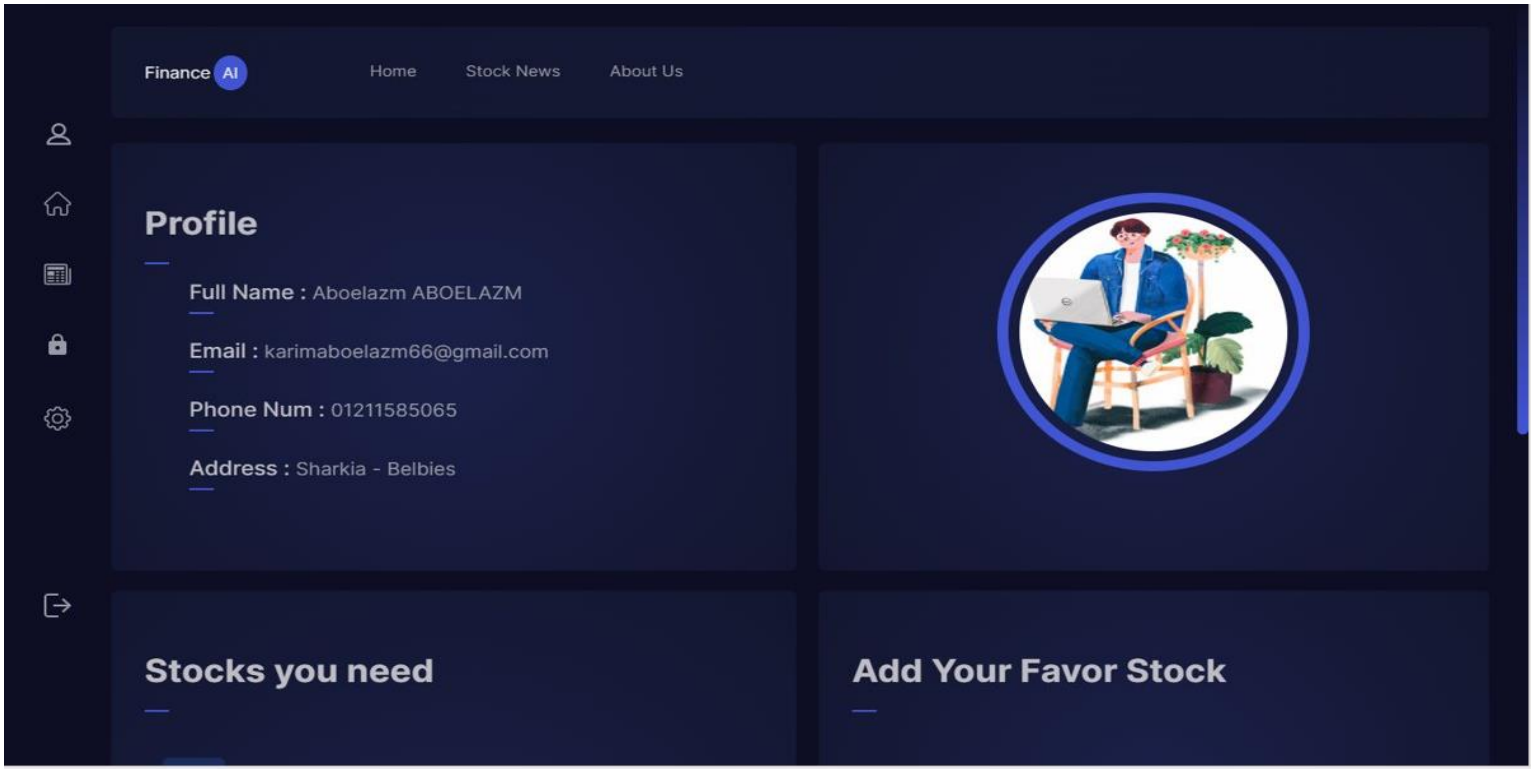
GUI Of Our Project

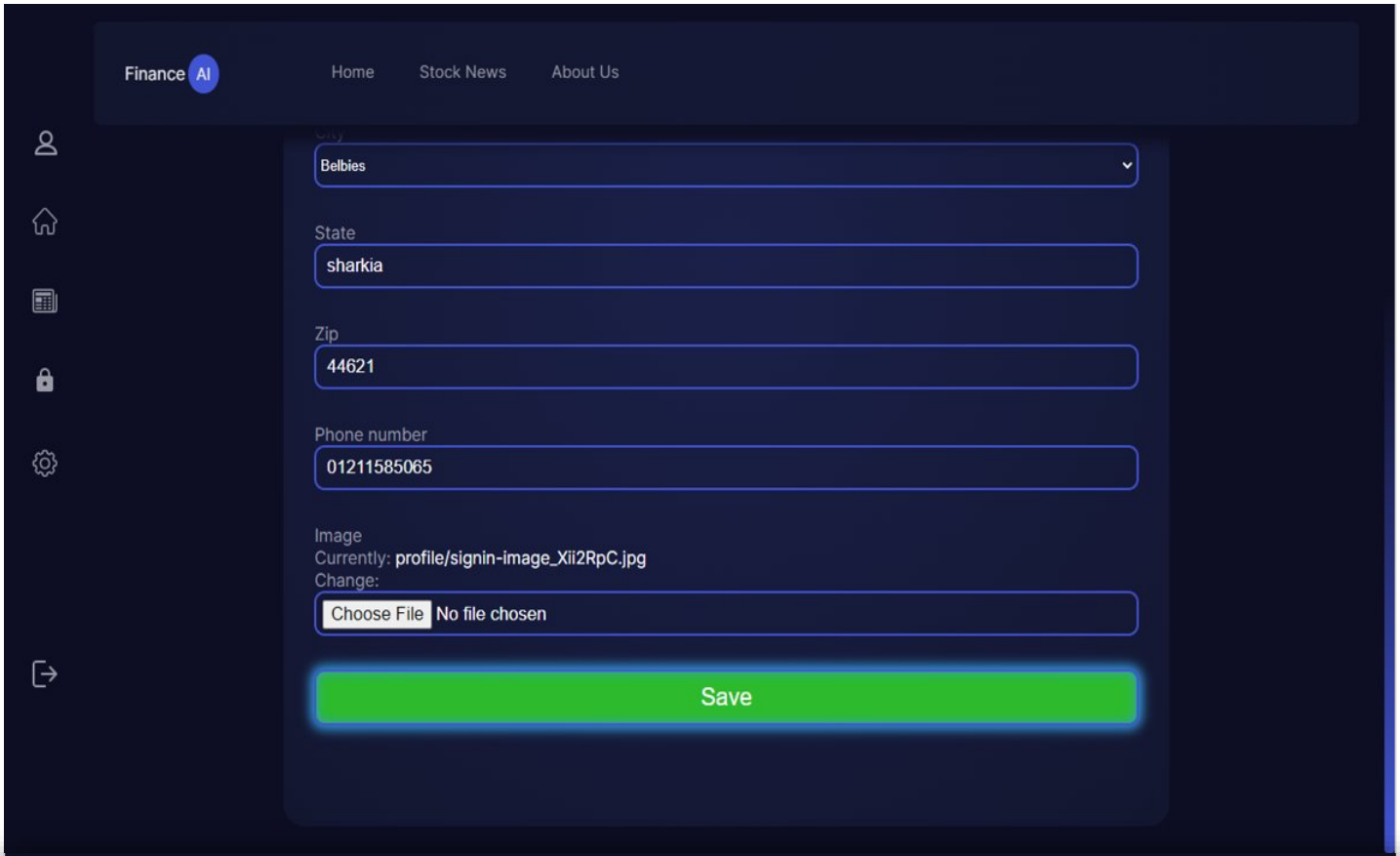


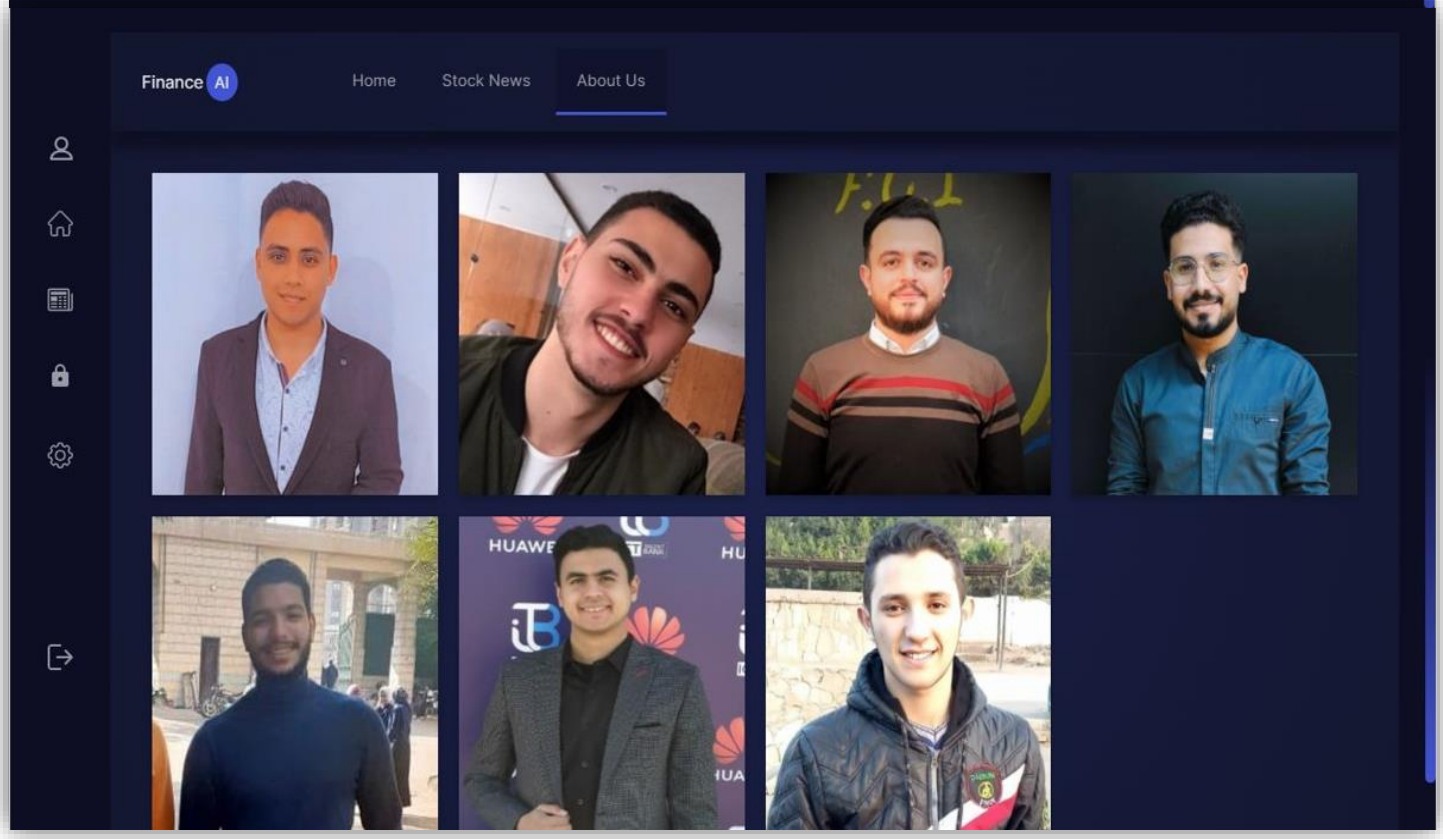

Finance AI
Home
Stock News
About Us
Stocks You Keep An Eye on
FB
FACEBOOK, INC.
340.64
GOOG
ALPHABET INC (GOOGL
2536.79
AMZN
AMAZON.COM, INC.
3408.03
NFLX
NETFLIX, INC.
530.44
2021
Add More Stocks You Need
Stock symbol ..
Add
Name
Symbol
Open
High
Low
Close
Edit
Facebook Inc - Class A
FB
$340.41
$344.9
$207.11
$207.11
Tesla Inc
TSLA
$681.07
$900.4
$189.7
$189.7
Amazon.com Inc.
AMZN
$3404.28
$3554
$2630.08
$2630.08
Ford Motor Co.
F
$15.385
$16.45
$5.74
$5.74
Alibaba Group Holding Ltd - ADR
BABA
$223.62
$319.32
$204.39
$204.39


Final_Fast_API.ipynb - Colaborato
FastAPI - Swagger UI
127.0.0.1:8000/home/
127.0.0.1:8000/home/
Apps
Google
whats app
YouTube
SoundCloud
Documentation - M...
Download - Bootstr...
Home - Canva
HIRESINE
Creative Jobs | we h...
Reading list
Finance AI
Home
Stock News
About Us
17
16
15
14
13
12
11
Mar
May
Sell


Finance AI
Home
Stock News
About Us
Stocks you need
Indices
Commodities
Bonds
Forex
SPXUSD
S&P 500
NSXUSD
Nasdaq 100
14365.8
-0.03%
-3.8
DJI
Dow 30
34422.0
+0.40%
+136.5
NKY
Nikkei 225
29066.18
+0.66%
+190.95
DEU30
DAX Index
15589.23
+0.86%
+132.84
UKXGBP
FTSE 100
7138.3
+0.16%
+11.5
Add Your favorite Stock
Enter Stock...
+ Add
" hint : The maximum number of following stocks you can add is 4 stocks"
FB
GOOG
AMZN
NFLX


Finance AI
Home
Stock News
About Us
Profile
Full Name : Aboelazm ABOELAZM
Email : karimaboelazm66@gmail.com
Phone Num : 01211585065
Address : Sharkia - Belbies
Stocks you need
Add Your Favor Stock


Finance AI
Home
Stock News
About Us
Belbies
State
sharkia
Zip
44621
Phone number
01211585065
Image
Currently: profile/signin-image_Xii2RpC.jpg
Change:
Choose File
No file chosen
Save


Finance AI
Home
Stock News
About Us

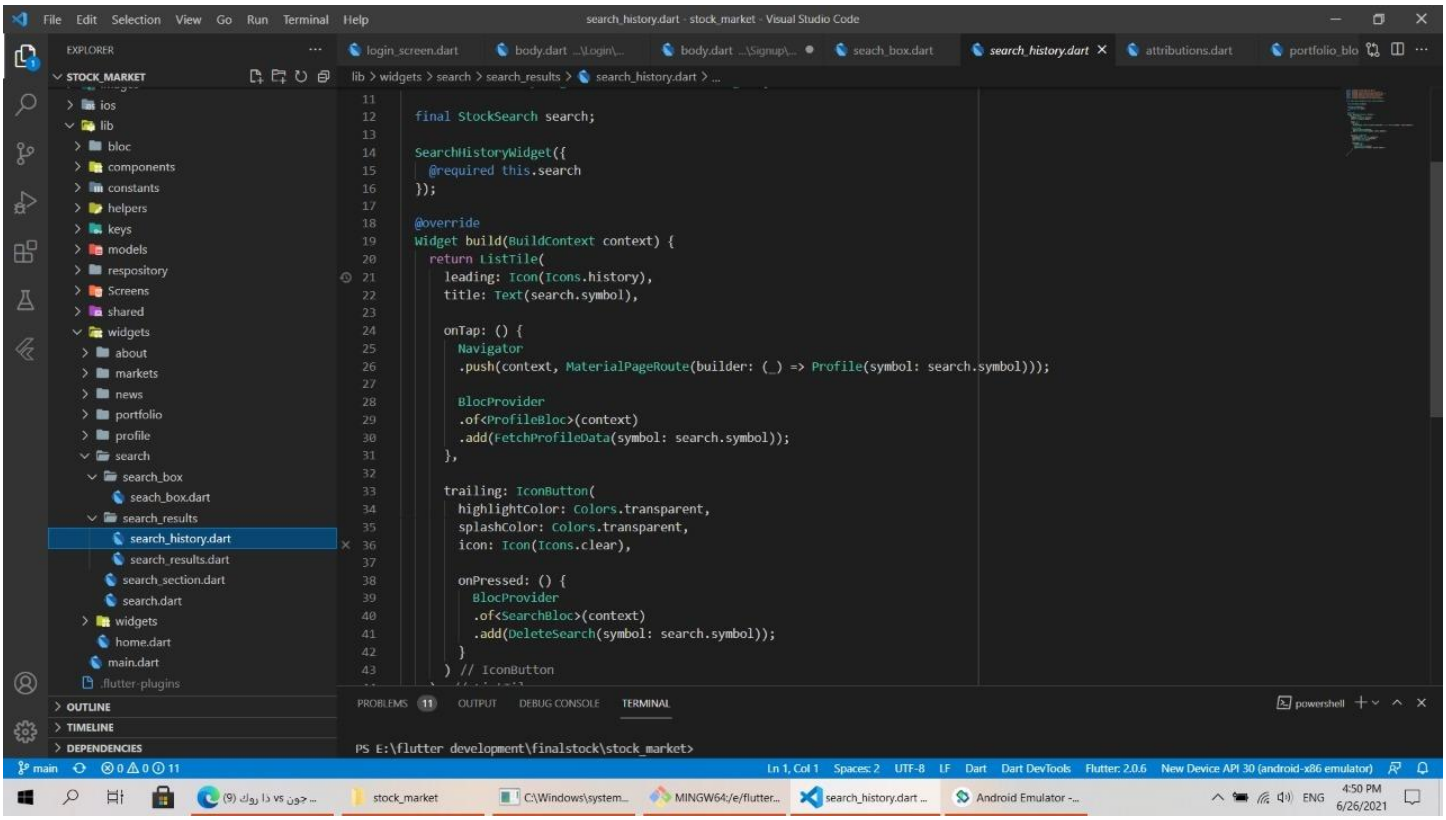
The App code and UI

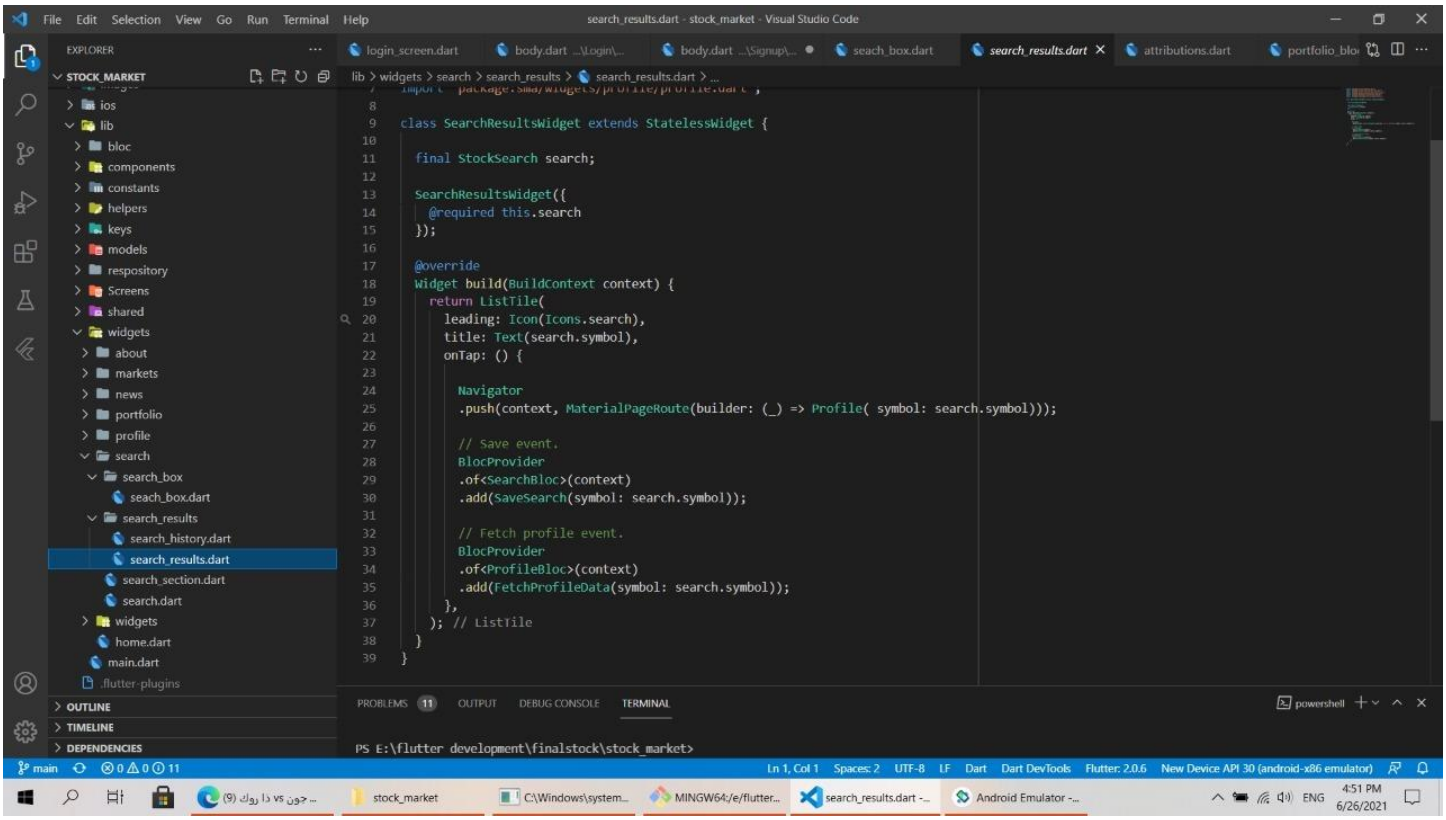
Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, and the web from a single codebase.

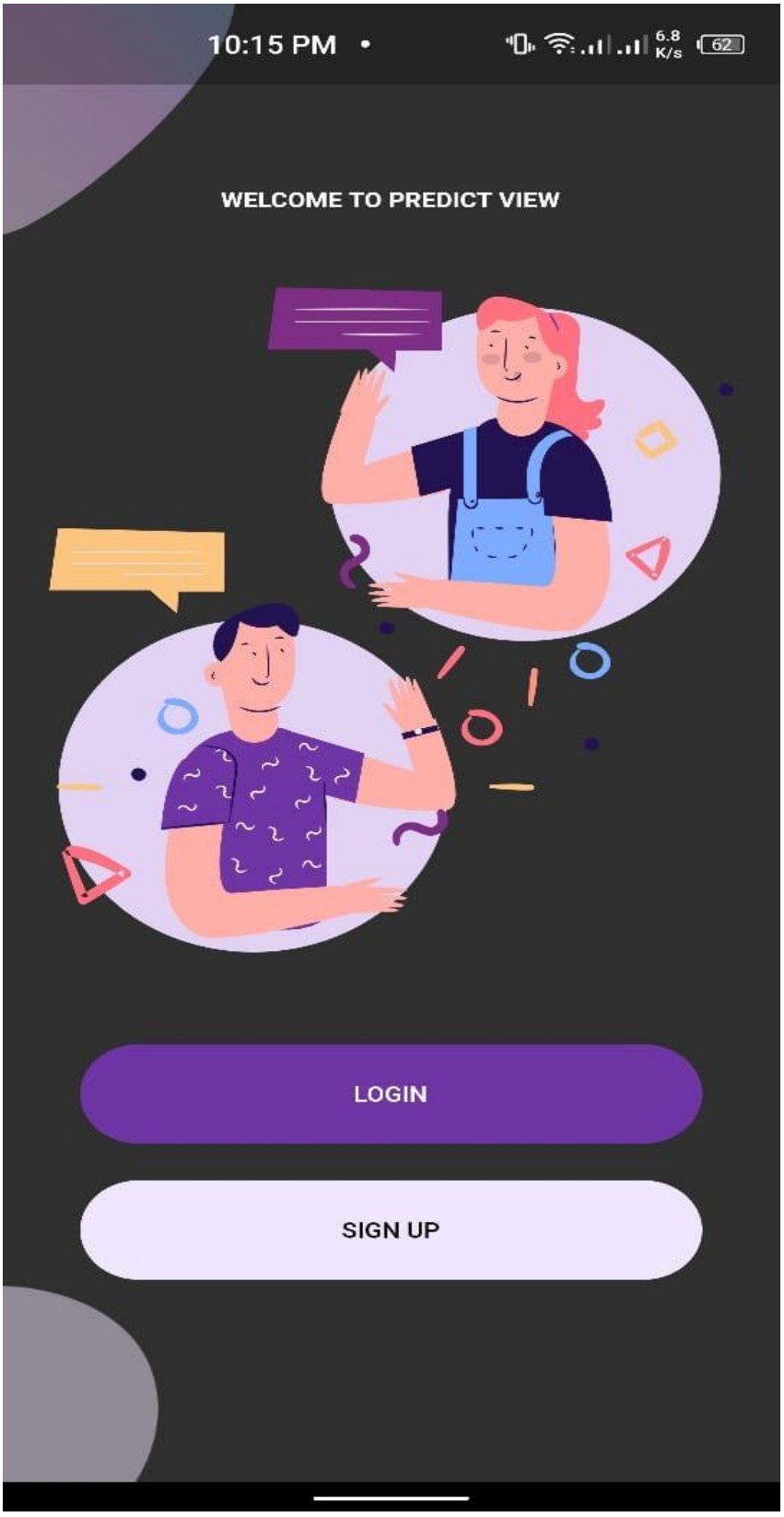
The first version of Flutter was known as codename "Sky" and ran on the Android operating system. It was unveiled at the 2015 Dart developer summit with the stated intent of being able to render consistently at 120 frames per second. During the keynote of Google Developer Days in Shanghai, Google announced Flutter Release Preview 2, which is the last big release before Flutter 1.0. On December 4, 2018, Flutter 1.0 was released at the Flutter Live event, denoting the first "stable" version of the Framework. On December 11, 2019, Flutter 1.12 was released at the Flutter Interactive event.

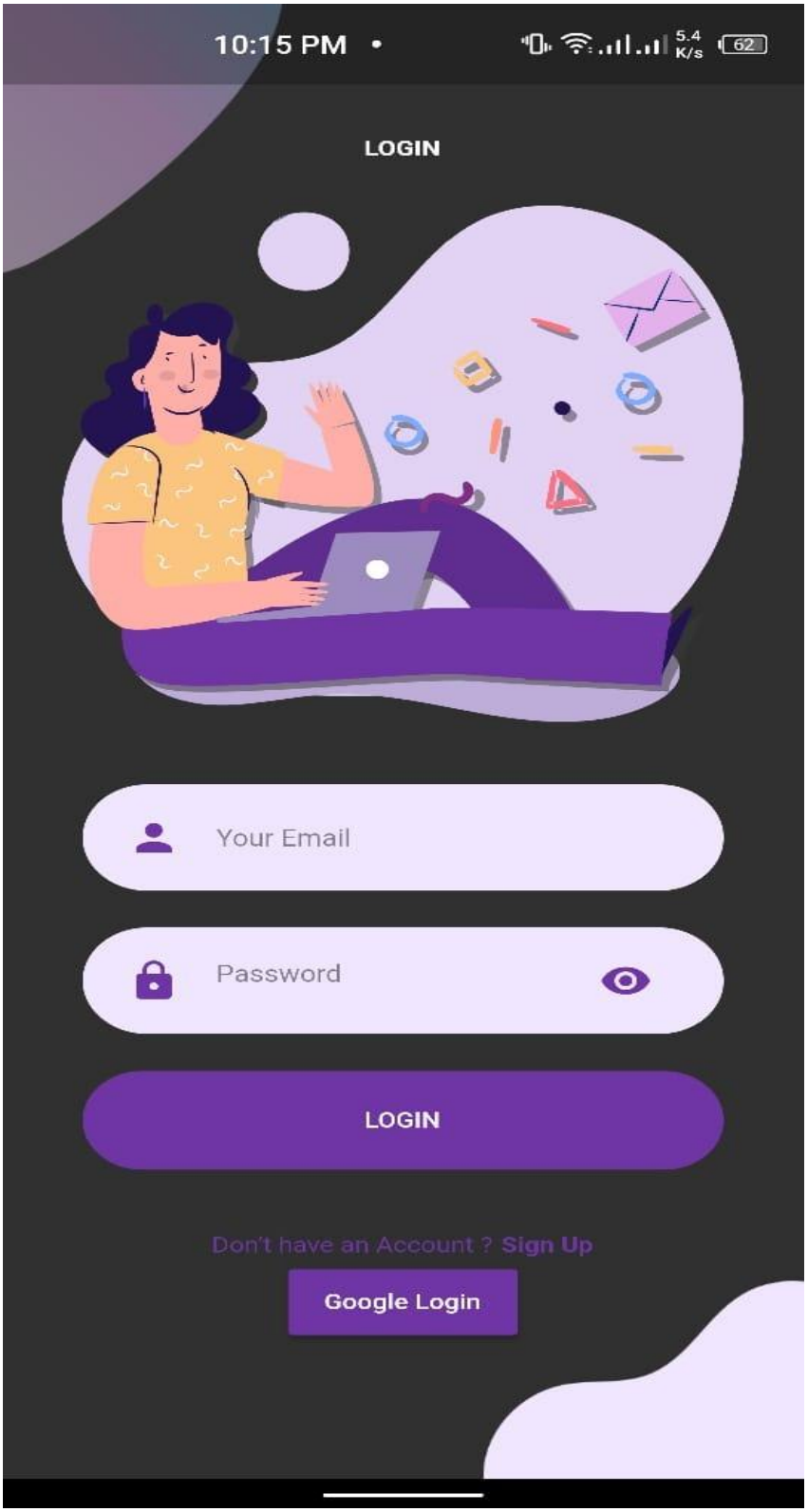
On May 6, 2020, the Dart software development kit (SDK) in version 2.8 and the Flutter in version 1.17.0 were released, where support was added to the Metal API, improving performance on iOS devices (approximately 50%), new Material widgets, and new network tracking.

On March 3, 2021, Google released Flutter 2 during an online Flutter Engage event. This major update brought official support for web-based applications with new CanvasKit renderer and web specific widgets, early-access desktop application support for Windows, macOS, and Linux and improved Add-to-App APIs.

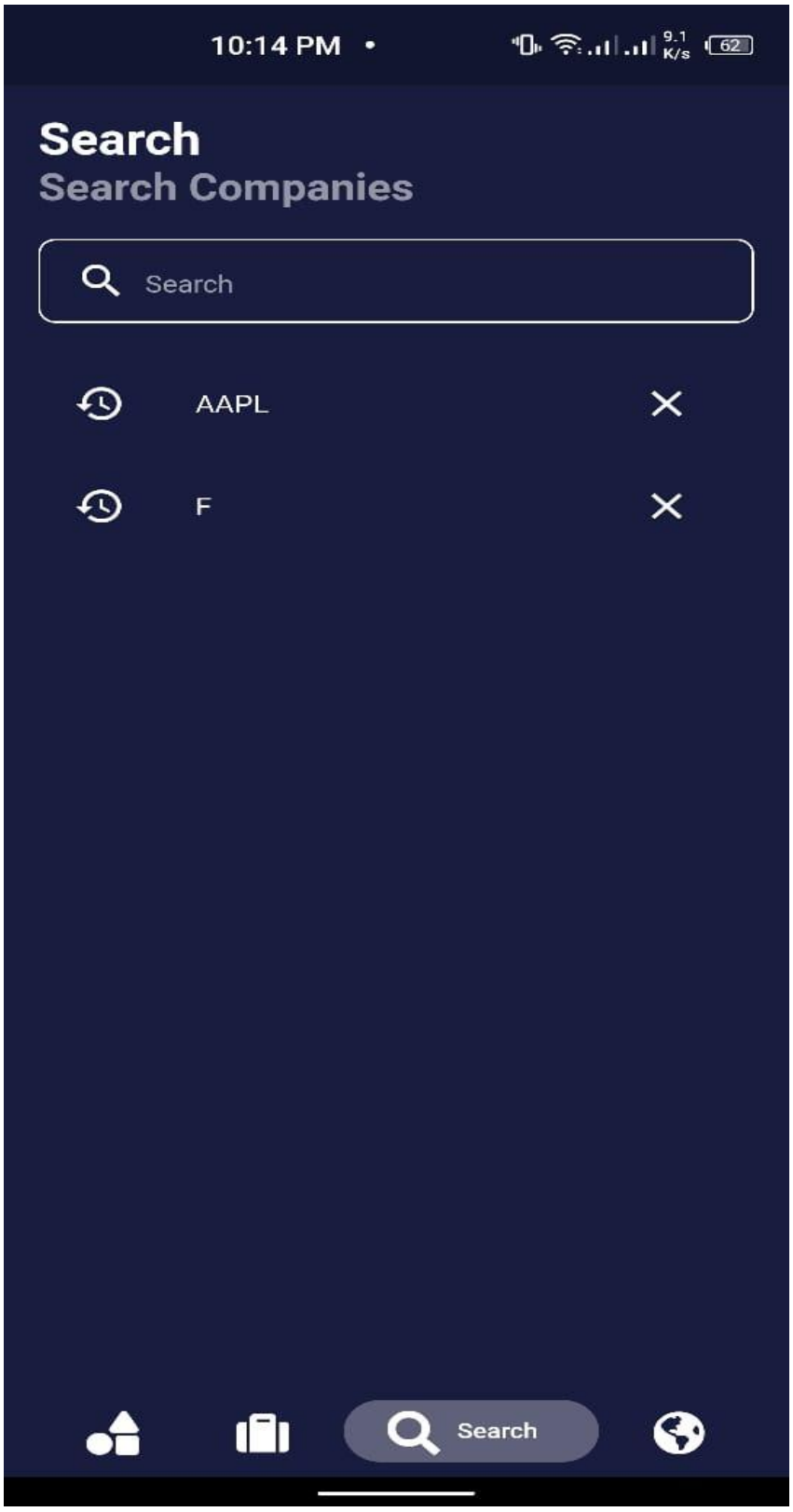

search_history.dart - stock_market - Visual Studio Code
EXPLORER
STOCK_MARKET
lib > widgets > search > search_results > search_history.dart > ...
final StockSearch search;
SearchHistoryWidget({
@required this.search
});
@override
Widget build(BuildContext context) {
return ListTile(
leading: Icon(Icons.history),
title: Text(search.symbol),
onTap: () {
Navigator
.push(context, MaterialPageRoute(builder: (_) => Profile(symbol: search.symbol)));
BlocProvider
.of<ProfileBloc>(context)
.add(FetchProfileData(symbol: search.symbol));
},
trailing: IconButton(
highlightColor: Colors.transparent,
splashColor: Colors.transparent,
icon: Icon(Icons.clear),
onPressed: () {
BlocProvider
.of<SearchBloc>(context)
.add(DeleteSearch(symbol: search.symbol));
}
) // IconButton
PS E:\flutter development\finalstock\stock_market>

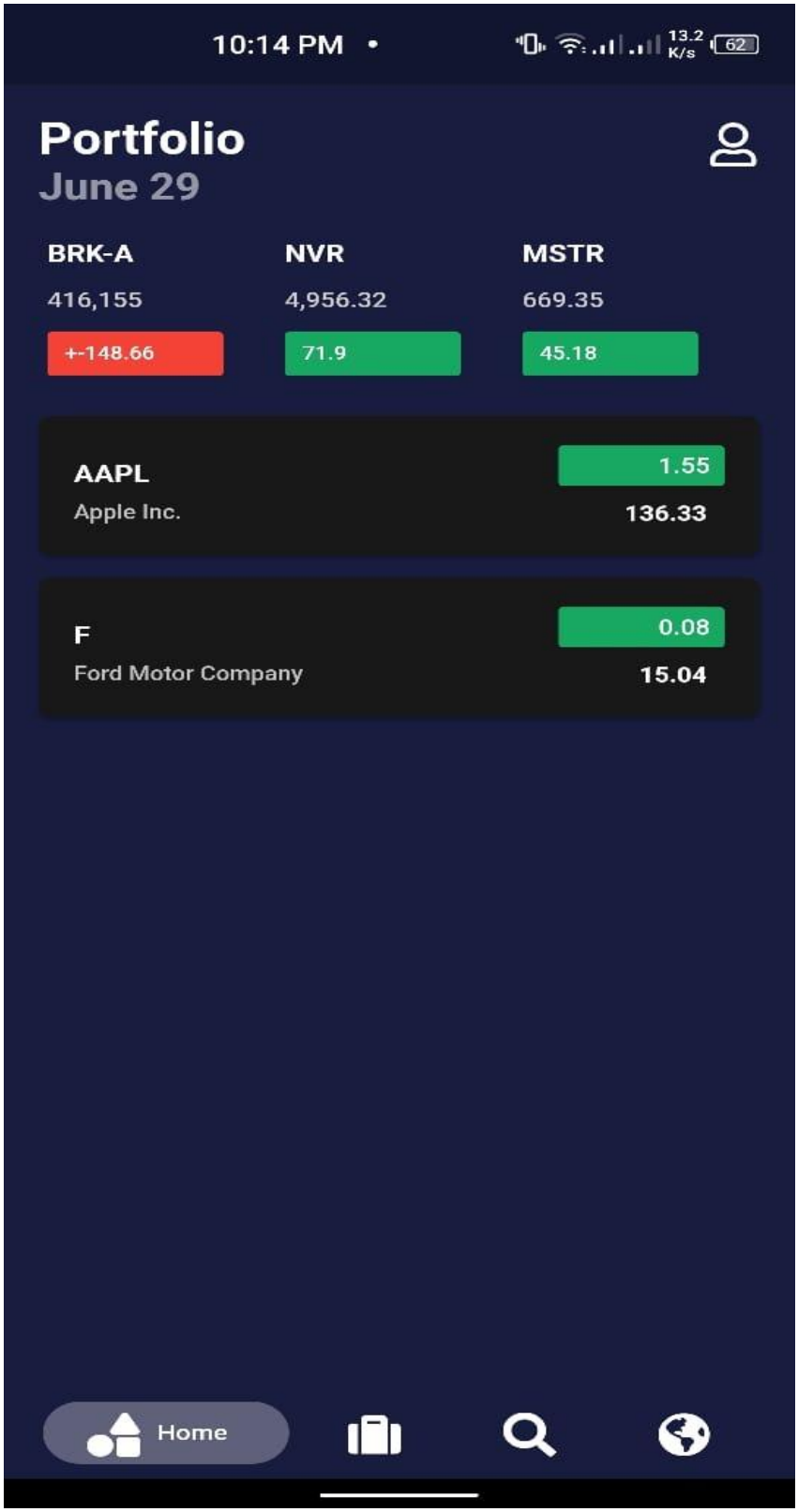

search_results.dart - stock_market - Visual Studio Code
EXPLORER
STOCK_MARKET
lib > widgets > search > search_results > search_results.dart > ...
class SearchResultsWidget extends StatelessWidget {
final StockSearch search;
SearchResultsWidget({
@required this.search
});
@override
Widget build(BuildContext context) {
return ListTile(
leading: Icon(Icons.search),
title: Text(search.symbol),
onTap: () {
Navigator
.push(context, MaterialPageRoute(builder: (_) => Profile(symbol: search.symbol)));
// Save event.
BlocProvider
.of<SearchBloc>(context)
.add(SaveSearch(symbol: search.symbol));
// Fetch profile event.
BlocProvider
.of<ProfileBloc>(context)
.add(FetchProfileData(symbol: search.symbol));
},
); // ListTile
}
}
PS E:\flutter development\finalstock\stock_market>


10:15 PM
6.8 K/s
62
WELCOME TO PREDICT VIEW
LOGIN
SIGN UP


10:15 PM
LOGIN
Your Email
Password
LOGIN
Don't have an Account ? Sign Up
Google Login


10:15 PM
SIGNUP
5.8 K/s
62
Your Name
Your Email
Password
Repeat your password
SIGNUP
Already have an Account ? Sign In


10:14 PM
Search
Search Companies
Search
AAPL
F
Search


10:14 PM
13.2 K/s
62
Portfolio
June 29
BRK-A
416,155
+-148.66
NVR
4,956.32
71.9
MSTR
669.35
45.18
AAPL
Apple Inc.
1.55
136.33
F
Ford Motor Company
0.08
15.04
Home


Ford Motor Company
$15.04
0.08 (+0.53%)
20
15
10
5
0
Jun 29
Nov 29
Apr 29
Here you can get stock prediction
neutral
sell
buy
6.0
Natural
Summary
Open
15.1
Outstanding Shares
3.99B


10:14 PM
11.5 K/s
62
U.S Markets
Sector Performance
Most Active
BRK-A
(-0.04%)
NVR
(+1.47%)
MSTR
(+7.24%)
Top Gainers
BSQR
(+205.73%)
CERE
(+138.66%)
MRIN
(+128.47%)
Top Losers
DMAC
(-33.89%)
RSLS
(-28.27%)
MDWD
(-28.19%)
Information Technology
0.70%
Consumer Discretionary
0.24%
Health Care
0.07%
Materials
-0.04%
Markets

6

Stock price prediction using Sentiment Analysis

6.1 Introduction

Sentiment analysis (also known as **opinion mining** or **emotion AI**) is the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine.

Step1: Data Collection Tweets on Microsoft, Google, AAPL, are extracted from twitter API. The tweets will have collected using Twitter API and filtered using keywords like $ MSFT, # Microsoft, #Windows etc.

Step2: Data Pre-Processing Stock prices data collected is not complete understandably because of weekends and public holidays when the stock market does not function. The missing data is approximated using a simple technique.

Step 3: Sentiment Analysis Sentiment analysis task is very much field specific. Tweets are classified as positive, negative and neutral based on the sentiment present. Out of the total tweets are examined by humans and annotated as 1 for Positive, 0 for Neutral and 2 for Negative emotions.

Step4: Feature Extraction Textual representations can be done using n-grams. N-gram Representation: N-gram representation is known for its specificity to match the corpus of text being studied. In these techniques a full corpus of related text is parsed which are tweets in the present work, and every appearing word sequence of length n is extracted from the tweets to form a dictionary of words and phrases. For example, the text “Microsoft is launching a new product” has the following 3-gram word features: “Microsoft is launching”, “is launching a”, “launching a new” and “a new product”. In our case, N-grams for all the tweets form the corpus. In this representation, tweet is split into N-grams and the features to the model are a string of 1s and 0s where 1 represents the presence of that N-gram of the tweet in the corpus and a 0 indicates the absence.

Step5: Model Training The features extracted using the above methods for the tweets are fed to the classifier and trained using classification methods like Logistic Regression, Decision Tree, SVM and KNN to estimate the movement of the change in stock market price vs the volume as well as sentiment of news articles and tweets.

6.2 Results

The problem that we faced related to the limitation of the Twitter account that we used. It doesn't give us the ability to extract tweets from more than one user so, we made the steps and implemented the code for preprocessing, etc. and in future work, we will solve this problem and we will integrate it into the system.

1) Preprocessing the text

```
# Define Clean Function to fix text
def Clean(text):

  # Frist converting all letters to lower case
  text= text.lower()
  # removing unwanted digits ,special chracters from the text
  text= ' '.join(re.sub("(@[A-Za-z0-9]+)", " ", text).split()) #tags
  text= ' '.join(re.sub("^@?(\w){1,15}$", " ", text).split())
  text= ' '.join(re.sub("(\w+:\/\/\S+)", " ", text).split())   #Links
  text= ' '.join(re.sub("http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\), ]|(?:%[0-9a-fA-F][0-9a-fA-
F]))+"," ", text).split())
  text= ' '.join(re.sub(r'http\S+', '',text).split())
  text= ' '.join(re.sub(r'www\S+', '',text).split())
  text= ' '.join(re.sub("\s+", " ",text).split()) #Extrem white Space
  text= ' '.join(re.sub("[^-9A-Za-z ]", "" ,text).split()) #digits
  text= ' '.join(re.sub('-', ' ', text).split())
  text= ' '.join(re.sub('_', ' ', text).split()) #underscore
  # removing stopwards and numbers from STRING library
  table= str.maketrans('', '', string.punctuation+string.digits)
  text = text.translate(table)
  # Split Sentence as tokens words
  tokens = word_tokenize(text)
  # converting words to their root forms by STEMMING THE WORDS
  #stemmed1 = [lemmatizer.lemmatize(word) for word in tokens] #Covert words to their actual root
  stemmed2 = [porter.stem(word) for word in tokens] # Covert words to their rootbut not actual
  # Delete each stop words from English stop words
  # words = [w for w in stemmed1 if not w in n_words] #n_words contains English stop words
  words = [w for w in stemmed2 if not w in n_words] #n_words contains English stop words
  text  = ' '.join(words)

  return text
```

2) KNN model

```
def KNN_Ngram(n):

    x_train, x_test, y_train, y_test = train_test_split(Data['text'], Data['Emotion'],
                                                        test_size=0.2, random_state=50)

    vect      = CountVectorizer(max_features=1000 , ngram_range=(n,n))
    train_vect= vect.fit_transform(x_train)
    test_vect = vect.transform(x_test)

    for k in [1,3,5,7,10]:

        model = KNeighborsClassifier(n_neighbors=k,algorithm='brute')
        t0        = time.time()
        model.fit(train_vect, y_train)
        t1        = time.time()
        predicted = model.predict(test_vect)
        t2        = time.time()
        time_train= t1-t0
        time_pred = t2-t1

        accuracy  = model.score(train_vect, y_train)
        predicted = model.predict(test_vect)

        report = classification_report(y_test, predicted, output_dict=True)


def KNN_TFIDF():

    x_train, x_test, y_train, y_test = train_test_split(Data['text'], Data['Emotion'],
                                                        test_size=0.2, random_state=50)

    vect      = TfidfVectorizer(min_df = 5, max_df =0.8, sublinear_tf = True, use_idf = True)
    train_vect= vect.fit_transform(x_train)
    test_vect = vect.transform(x_test)

    for k in [1,3,5,7,10]:

        model = KNeighborsClassifier(n_neighbors=k,algorithm='brute')
        t0        = time.time()
        model.fit(train_vect, y_train)
        t1        = time.time()
        predicted = model.predict(test_vect)
        t2        = time.time()
        time_train= t1-t0
        time_pred = t2-t1

        accuracy  = model.score(train_vect, y_train)
        predicted = model.predict(test_vect)

        report = classification_report(y_test, predicted, output_dict=True)
```

```
#Fit KNN Model Based on Different Techniques

''' KNN Models '''
# with Bi-gram
KNN2=KNN_Ngram(2)
# with Tri-gram
KNN3=KNN_Ngram(3)
# with TFIDF
knn_tfidf=KNN_TFIDF()


''' Result '''

Models with  2 -grams :

k = 1 :
Accuracy score train set : 0.8084461637653128
Accuracy score test set  : 0.776173285198556

k = 3 :
Accuracy score train set : 0.7159252095422308
Accuracy score test set  : 0.6890149561629706

k = 5 :
Accuracy score train set : 0.7968407479045777
Accuracy score test set  : 0.7851985559566786

k = 7 :
Accuracy score train set : 0.7943907156673115
Accuracy score test set  : 0.7849406910778752

k = 10 :
Accuracy score train set : 0.7893617021276595
Accuracy score test set  : 0.7849406910778752


 ----------------------------------------------------------------------------------------------------

Models with  3 -grams :

k = 1 :
Accuracy score train set : 0.7754996776273372
Accuracy score test set  : 0.759927797833935

k = 3 :
Accuracy score train set : 0.7751128304319793
Accuracy score test set  : 0.7705002578648789

k = 5 :
Accuracy score train set : 0.7709219858156028
Accuracy score test set  : 0.7692109334708612

k = 7 :
Accuracy score train set : 0.7705996131528047
Accuracy score test set  : 0.7686952037132543

k = 10 :
Accuracy score train set : 0.7682785299806576
Accuracy score test set  : 0.7684373388344508

 ----------------------------------------------------------------------------------------------------

Models with Tfidf :

k = 1 :
Accuracy score train set : 0.9999355254674404
Accuracy score test set  : 0.8705518308406395

k = 3 :
Accuracy score train set : 0.8847195357833656
Accuracy score test set  : 0.8357400722021661

k = 5 :
Accuracy score train set : 0.8397163120567376
Accuracy score test set  : 0.8115007735946365

k = 7 :
Accuracy score train set : 0.8085751128304319
Accuracy score test set  : 0.7962867457452295

k = 10 :
Accuracy score train set : 0.8012250161186332
Accuracy score test set  : 0.7944816915936049
```

3) N-Gram model function

```
def NgramModels(Model , txt, n):

    x_train, x_test, y_train, y_test = train_test_split(Data['text'], Data['Emotion'],
                                                        test_size=0.2, random_state=50)

    vect      = CountVectorizer(max_features=1000 , ngram_range=(n,n))
    train_vect= vect.fit_transform(x_train)
    test_vect = vect.transform(x_test)

    model     = Model
    t0        = time.time()
    model.fit(train_vect, y_train)
    t1        = time.time()
    predicted = model.predict(test_vect)
    t2        = time.time()
    time_train= t1-t0
    time_pred = t2-t1

    accuracy  = model.score(train_vect, y_train)
    predicted = model.predict(test_vect)

    report = classification_report(y_test, predicted, output_dict=True)
    print("Models with " , n , "-grams :\n")
    print('********************** \n')
    print(txt)
    print("Training time: %fs; Prediction time: %fs \n" % (time_train, time_pred))
    print('Accuracy score train set :', accuracy)
    print('Accuracy score test set  :', accuracy_score(y_test, predicted),'\n')
    print('Positive: ', report['1'])
    print('Neutral : ', report['0'])
    print('Negative: ', report['2'])
```

4) TFIDF models

```
def NgramModels(Model , txt, n):

    x_train, x_test, y_train, y_test = train_test_split(Data['text'], Data['Emotion'],
                                                        test_size=0.2, random_state=50)

    vect      = CountVectorizer(max_features=1000 , ngram_range=(n,n))
    train_vect= vect.fit_transform(x_train)
    test_vect = vect.transform(x_test)

    model     = Model
    t0        = time.time()
    model.fit(train_vect, y_train)
    t1        = time.time()
    predicted = model.predict(test_vect)
    t2        = time.time()
    time_train= t1-t0
    time_pred = t2-t1

    accuracy  = model.score(train_vect, y_train)
    predicted = model.predict(test_vect)

    report = classification_report(y_test, predicted, output_dict=True)
    print("Models with " , n , "-grams :\n")
    print('********************** \n')
    print(txt)
    print("Training time: %fs; Prediction time: %fs \n" % (time_train, time_pred))
    print('Accuracy score train set :', accuracy)
    print('Accuracy score test set  :', accuracy_score(y_test, predicted),'\n')
    print('Positive: ', report['1'])
    print('Neutral : ', report['0'])
    print('Negative: ', report['2'])
```

5) SVC model

```
#Fit Support Vector Classifier Model Based on Different Techniques

''' Support Vector Classifier Models '''

# linear Kernal for SVC
SupportVectorClassifier=svm.SVC(kernel='linear')

# with Bi-gram
svm_Bi = NgramModels(Model=SupportVectorClassifier ,
                  txt='Support Vectoer Classifier Model', n=2)
# with Tri-gram
svm_tri = NgramModels(Model=SupportVectorClassifier ,
                  txt='Support Vectoer Classifier Model', n=3)

# with TFIDF
svm=TFIDFModels(Model=SupportVectorClassifier,
                txt='Support Vector Classifier Model')


''' Result '''

Models with  2 -grams :


Accuracy score train set : 0.8073500967117988
Accuracy score test set  : 0.8042805569881382

 ----------------------------------------------------------------------------------------------------

Models with  3 -grams :

Accuracy score train set : 0.7780786589297227
Accuracy score test set  : 0.7774626095925735

 ----------------------------------------------------------------------------------------------------

Models with Tfidf :
Accuracy score train set : 0.9889748549323018
Accuracy score test set  : 0.9783393501805054
```

6) logistic regression model

```
#Fit Logistic Model Based on Different Techniques

''' Logistic Regression Models '''

# with Bi-gram
LogReg_Bi = NgramModels(Model=LogisticRegression(),
                        txt='Logistic Regression Model', n=2)

# with Tri-gram
LogReg_tri = NgramModels(Model=LogisticRegression(),
                        txt='Logistic Regression Model', n=3)

# with TFIDF
LogReg=TFIDFModels(Model=LogisticRegression(),
                   txt='Logistic Regression Model')


''' Result ''' :

Models with  2 -grams :


Accuracy score train set : 0.8079303675048356
Accuracy score test set  : 0.8032490974729242

 ---------------------------------------------------------------------------------------------------

Models with  3 -grams :

Accuracy score train set : 0.7780786589297227
Accuracy score test set  : 0.7753996905621454

 ---------------------------------------------------------------------------------------------------

Models with Tfidf :

Accuracy score train set : 0.9706640876853643
racy score test set  : 0.9522949974213513
```

7) Decision tree classifier

```
#Fit Decision Tree Classifier Model Based on Different Techniques

''' Decision Tree Classifier Models '''
# with Bi-gram
DecTree_Bi = NgramModels(Model=tree.DecisionTreeClassifier(),
                         txt='Decision Tree Classifier Model', n=2)
# with Tri-gram
DecTree_tri = NgramModels(Model=tree.DecisionTreeClassifier(),
                         txt='Decision Tree Classifier Model', n=3)
# with TFIDF
DecTree=TFIDFModels(Model=tree.DecisionTreeClassifier(),
                    txt='Decision Tree Classifier Model')


''' Result '''

Models with  2 -grams :


Accuracy score train set : 0.8228884590586718
Accuracy score test set  : 0.8050541516245487

 ----------------------------------------------------------------------------------------------------

Models with  3 -grams :

Accuracy score train set : 0.7823984526112185
Accuracy score test set  : 0.7759154203197525

 ----------------------------------------------------------------------------------------------------

Models with Tfidf :

Accuracy score train set : 0.9999355254674404
Accuracy score test set  : 0.9863331614234141
```

8) Detect the emotion

```
''' Detect Emotions for each text Form TextBlob Library '''

detectEmotion=[]
detectPolarity=[]

for txt in Data.text:

    analysis=TextBlob(txt)
    Polarity=analysis.sentiment.polarity

    if Polarity  <0:
        emotion='2'  #Negative
    elif Polarity>0:
        emotion='1'  #Positive
    else:
        emotion='0'  #Neutral

    detectEmotion.append(emotion)
    detectPolarity.append(Polarity)

# detectEmotion=pd.DataFrame()

Data['Polarity']=detectPolarity
Data['Emotion'] =detectEmotion

#check for valid string only to detect languages

TextValid=[]

for i in range(len(Data)):
    TextValid.append(bool(re.match('^(?=.*[a-zA-Z])', Data.iloc[i,0])))

'''Detect languages for each text to filter into specific Lang'''

languages = []

# Loop over the sentences in the data and detect their language
for row in range(len(Data)):
    languages.append(detect_langs(Data.iloc[row, 0]))

# print('The detected languages are: ', languages) >>> ['en':'N']
languages = [str(lang).split(':')[0][1:] for lang in languages]

# Assign the list to a new feature
Data['language'] = languages
```

7

Conclusion && Future Work

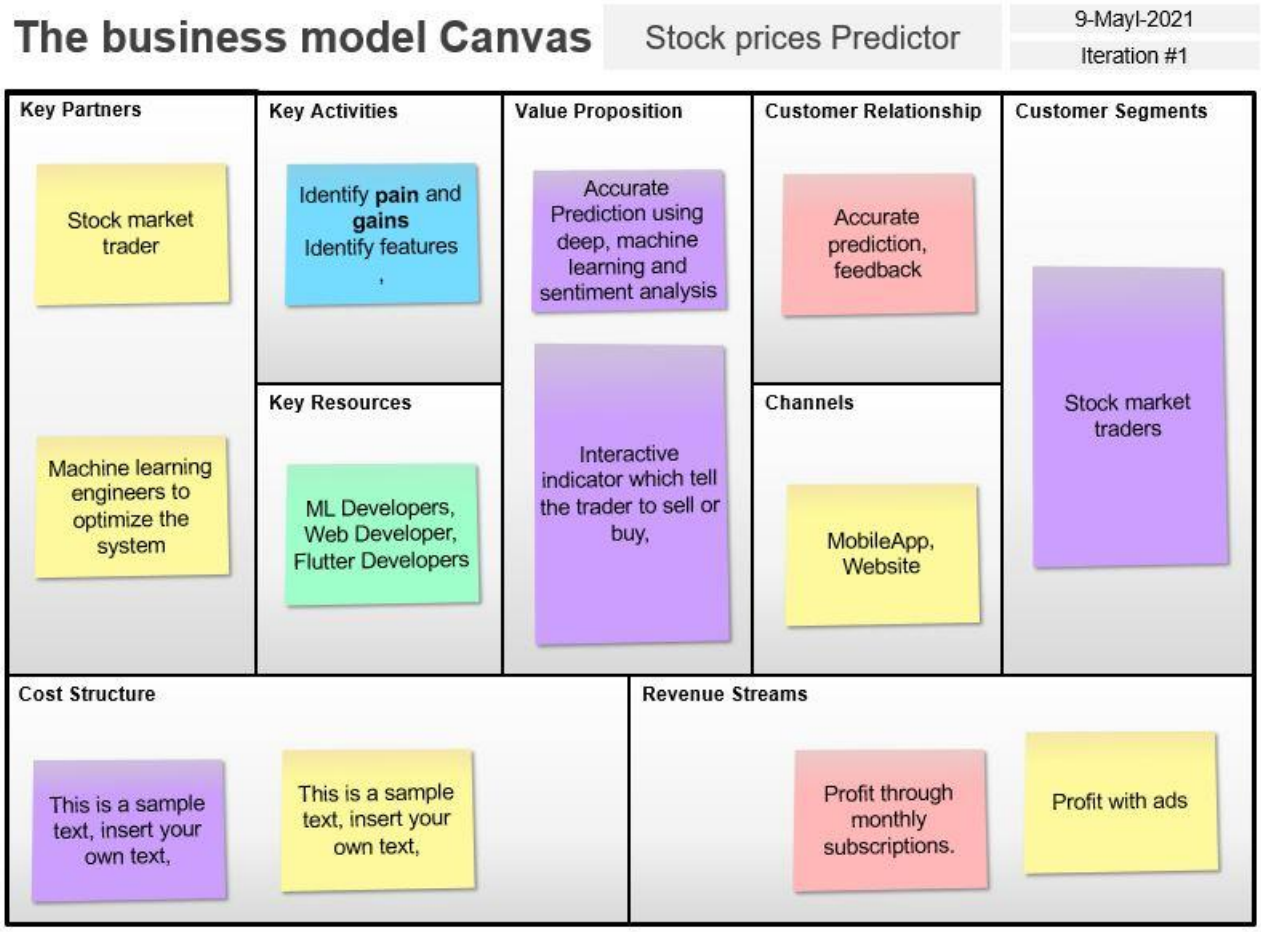
7.1 Conclusion

we **implemented a web site and Application** to predict stock market prices and **optimized** a Predictive Modeling and Technical Indicators Analysis approach by developing an automated stock data collection and predictive analysis tools addition to using **the sentiment analysis**, after that, we **made voting** between all of them to select the best accuracy.

Predictive Modeling is very effectively implemented in forecasting stock prices, returns, and stock modeling, and the most frequent methodologies are **LSTM**, **Transformer**, **Linear regression**, **Ridge regression**, **SVR**, **Prophet,** and the **ARIMA** models.

7.2 Future Work

1) Integrate the sentiment analysis with the system.
2) Establishing a startup based on the idea of this project and the image bellow show our business model to begin the startup.



8 References

1) https://medium.com/@buseyarentkn/introduction-to-rnn-and-lstm-with-keras-522562032c35

2) https://medium.com/inside-machinelearning/what-is-a-transformerd07dd1fbec04
https://towardsdatascience.com/adeep-dive-into-the-transformerarchitecture-the-development-oftransformer-models-acbdf7ca34e0

3) https://www.machinelearningplus.com/time-series/arima-model-time-seriesforecastingpython/#:~:text=Introduction%20to%20ARIMA%20Models,-So%20what%20exactly&text=ARIMA%2C%20short%20for%20'Auto%20Regressive,used%20to%20forecast%20future%20values

4) https://www.kaggle.com/janiobachmann/s-p-500-time-series-forecasting-with-prophet

5) https://www.kaggle.com/davidwallach/financial-tweets/home

6) https://www.google.com/search?q=github+finicial+tweets&rlz=1C1CHBF_enUS877US877&oq=github+finicial+tweets&aqs=chrome..69i57j69i64.3877j0j4&sourceid=chrome&ie=UTF-8

7) http://github.com/business-science/modeltime.ensemble

https://github.com/carloschavez9/ensemble-time-series

https://github.com/cnaimo/hybrid-ARIMA-LSTM-model