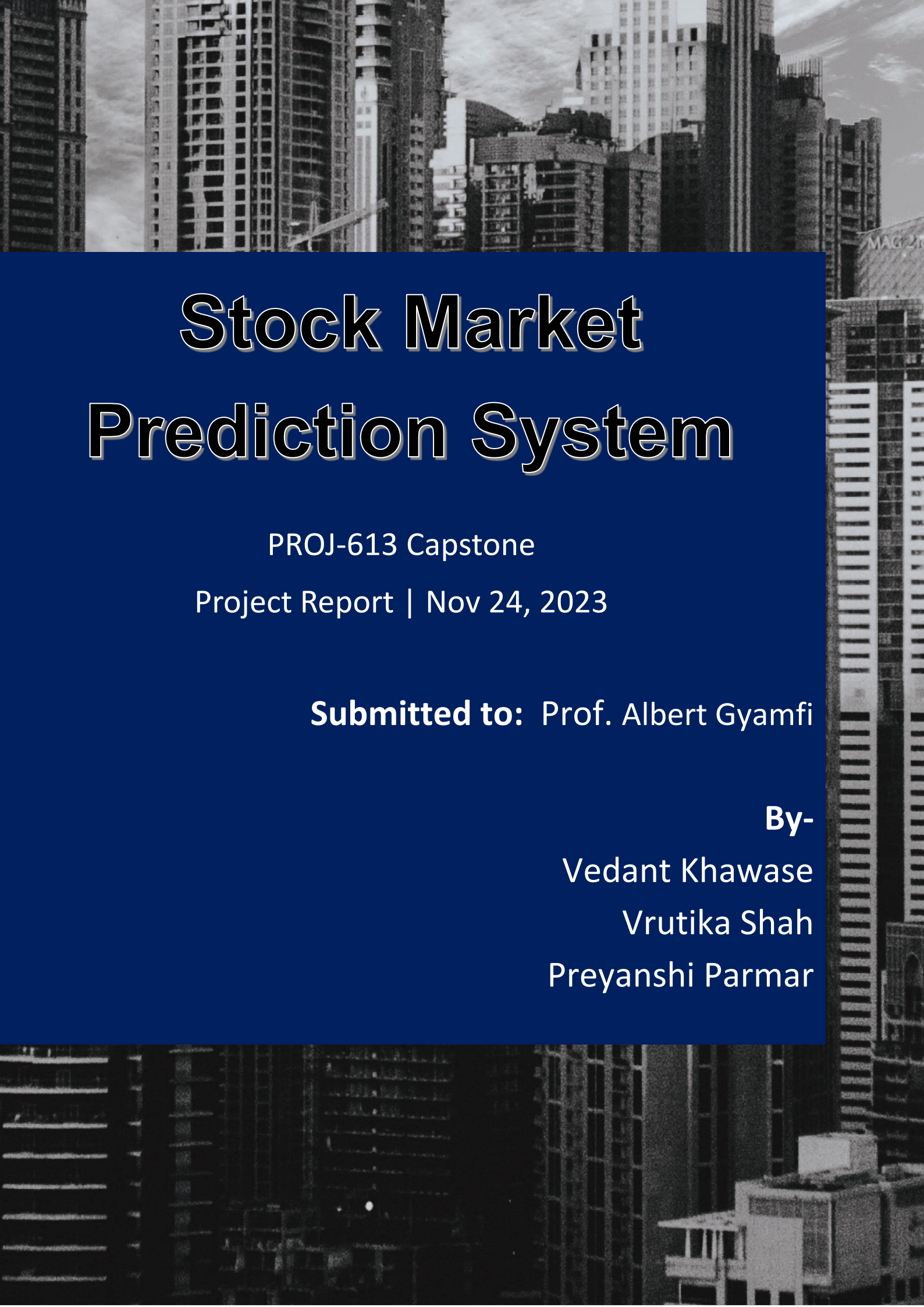


Stock Market Prediction System

PROJ-613 Capstone

Project Report | Nov 24, 2023

Submitted to: Prof. Albert Gyamfi

By-

Vedant Khawase

Vrutika Shah

Preyanshi Parmar

Contents

1. Acknowledgement

I would like to thank each member of our project team for their important efforts. This project required teamwork to be completed successfully, and each team member was essential in providing their special talents and knowledge. Vrutika is responsible for Data analysis in-depth and exploration, preprocessing, Model planning, selection and model building and implementation of the project's algorithm ensuring its accuracy and efficiency. Write valuable information into report for Model knowledge and overall implementation. Vedant has contributed to the idea of the Project, create website framework and development of the website. Make sure report review and write details about website framework and other details. Preyanshi worked on project documentation and presentation initially. All are participated to finalize the Presentation and Project report. We would also like to take this chance to thank Prof. Albert Gyamfi, our main Instructor, for his expert advice throughout the project development. I would also want to express my thanks to my parents for their emotional and financial support over this academic decade.

2. Abstract

For a very long time, people all around the world have believed that stock market investing is extremely risky. The goal of this research is to understand historical stock market data and extract insights from it to close the knowledge gap between investors and market behaviour. The Stock Market Prediction System is a reliable and interactive web application developed to provide users with significant knowledge and precise predictions, assisting both beginner and experienced stock market investors in making wise investment decisions. The inaccuracy of the analysis is the problem with the present setup. Thus, there's a considerable probability of losing money. The system makes use of historical stock market data and machine learning algorithms with the primary objective of producing accurate forecasts and insightful analysis.

3. Introduction

We have examined the stock performance of four distinct companies—Amazon, Walmart, Microsoft, and Apple—for this study. Nevertheless, we used data spanning the last ten years to the present. Beginning with the fundamentals, we removed any extraneous information and concentrated on projecting future stock values using the patterns from the previous month. Recurrent Neural Network (RNN), a sophisticated technology, was used to train a model to interpret this financial narrative.

3.1 The Challenges of Stock Market Investing

Stock market investments have long been considered essential to financial planning and wealth accumulation. Despite its huge earning potential, handling the intricacy of this financial environment is no simple feat. Investors face many different and intricate challenges. The primary obstacle is the excess of data. Emotional Decision-Making: The impact of emotions on judgment is another significant barrier. Fear and greed are often the driving forces behind investment decisions, leading to reckless trading, excitement, or panic selling. Moreover, one of the obstacles is market volatility. Prices are subject to major fluctuations in responses to news events, movements in the economy, or variations in market sentiment.

3.2 The Solution: Stock Market Prediction System

We offer the idea of a Stock Market Prediction System as a creative and innovative solution to these obstacles, offering investors and market players the information and guidance they require to make more informed and intelligent decisions.

Deep Understanding: The Stock Market Prediction System focuses on past stock prices. By integrating this diverse data, users can acquire an in-depth knowledge of the market situation and make more informed choices.

Accurate Forecasts: The capacity to provide precise forecasts on stock prices and trends in the market.

User-Centric Design: Stock Market Prediction System understands the value of the user experience, it includes an easy-to-use online interface. Participation by users with the algorithm allows for the customization of estimates based on personal interests, risk tolerances, and investment goals.

4. Problem Statement

The stock market is a complicated and highly volatile environment that is influenced by a wide range of factors, including economic statistics, geopolitical happenings, corporate performance, and investor attitude. Accurate stock price forecasts are useful to traders, financial institutions, and investors who desire to manage their portfolios effectively and make educated judgments. Generating an accurate and dependable technique to forecast future stock prices and market patterns is a challenge.

5. Background:

Given the complexity and volatility of financial markets, predicting stock prices is difficult. The complex interdependencies and non-linear patterns found in stock price time series data are frequently difficult for traditional financial models to represent. Complex connection modeling in sequential data has made machine learning, including deep learning methods like recurrent neural networks (RNNs), popular. Several researchers have attempted the use of machine learning techniques for financial domain issues. Daily buy and sell signals may be achieved with a greater speed and acceptable accuracy by using soft computing techniques like neural networks, without delving into the statistical problems of technical analysis [18]. Among time series forecasts, stock market prediction is typically regarded as one of the most difficult problems [3]. The financial market is a dynamic system that is non-linear, complicated, and evolving [8]. The data utilized to estimate the time series trend of the financial market is noisy, which reduces the accuracy of the prediction. According to some academics, it is difficult to forecast a financial asset's value.

6. Use Case:

The stock prediction system aims to leverage historical market data and advanced machine learning algorithms to forecast future stock prices. By analyzing patterns and trends in stock data, the system provides valuable insights to help investors make informed decisions and optimize their portfolios. The use case involves implementing predictive models that utilize a combination of technical indicators, fundamental analysis, and sentiment analysis to enhance the accuracy of stock price predictions. Real-time data integration and continuous model refinement ensure the system adapts to dynamic market conditions, providing users with up-to-date and reliable forecasts. The stock prediction system serves as a valuable tool for financial analysts, traders, and investors, contributing to effective decision-making in the complex world of financial markets.

7. Architecture

The project's architecture, as described above, consists of a Flask web application that communicates with a Python file that has all the web page routes and training models. When a user first opens the web application and selects a stock to predict, the model is trained and data is predicted through a series of steps that include data exploration, analysis, splitting into feature and target vectors, and finally data scaling. It will then train, assess, and evaluate the near pricing model before predicting the trained data to produce the intended expected output and returning it to the homepage with JavaScript code that includes an Ajax method for plotting the predicted data supplied from Python to the browser.

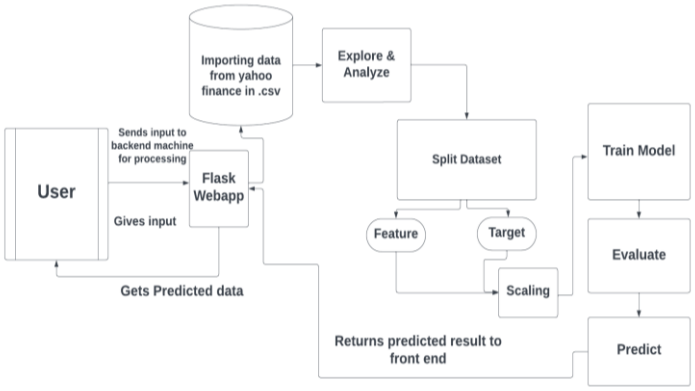


Fig. Application Architecture

8. Data Pipeline

Fig. Data Pipeline for Application

8.1 Data Collection

For data collection, we have used data from Yahoo Finance for ten years of Apple, Microsoft, Amazon, and Walmart. In getting our data ready for the stock market prediction journey, we gathered our tools and loaded up historical stock data. The date became our compass, guiding us through the numbers. Exploratory data analysis revealed the hidden contents of the dataset, and we removed missing values to ensure reliability. Focusing on 'Close' prices, we normalized the data for a common language. Time became a factor as we split our data for training and testing, crafting sequences for our Recurrent Neural Network (RNN) to learn. Trend visualization was the decisive point, setting the stage for our RNN to predict stock prices with confidence.

Why Yahoo Finance?

Yahoo! Finance is a media property that is part of Yahoo!'s network. It provides financial news, data, and commentary including stock quotes, press releases, financial reports, and original content. The application needs the latest stock OHLVC (Open Price, High Price, Low Price, Volume, Close Price) data for each stock. The stock data should be automatically retrieved from the web. For that, Yahoo Finance is a good source for extracting financial data as the format of the data is mostly consistent for this source.

8.2 Data Exploration & analysis

We have used the info() method to provide a comprehensive summary of the Data Frame, including the column names, data types, and the count of non-null values.

```
Stock Name: Microsoft
DataFrame Info for Microsoft:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2516 entries, 2013-11-20 to 2023-11-17
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       2516 non-null   float64
 1   High       2516 non-null   float64
 2   Low        2516 non-null   float64
 3   Close      2516 non-null   float64
 4   Adj Close  2516 non-null   float64
 5   Volume     2516 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 137.6 KB
None
```

Fig. Data frame Information

```
print("Apple Null check:::::",Apple_data.isna().sum())
print("Micosoft Null check:::::",Microsoft_data.isna().sum())
print("Walmart Null check:::::",Walmart_data.isna().sum())
print("Amazon Null check:::::",Amazon_data.isna().sum())
```

```
Apple Null check::::: Open         0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
Micosoft Null check::::: Open         0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
Walmart Null check::::: Open         0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
Amazon Null check::::: Open         0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
```

Fig. Check null values in all four stocks

Using describe() function checked the description of the data in data frame, that contains count, mean, std deviation, minimum value, 25 % percentile, 50 % percentile, 75 % percentile and the max value.

```
Microsoft_data.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	2516.000000	2516.000000	2516.000000	2516.000000	2516.000000	2.516000e+03
mean	146.062401	147.530191	144.552516	146.109285	140.935142	3.026886e+07
std	99.461993	100.531890	98.360917	99.483814	100.291856	1.401040e+07
min	34.730000	35.880001	34.630001	34.980000	29.455135	7.425600e+06
25%	54.490002	55.067500	54.000000	54.535001	48.685607	2.174610e+07
50%	108.250000	109.044998	106.910000	108.149998	102.245506	2.700740e+07
75%	239.434994	242.857498	237.075005	240.375004	236.635010	3.448018e+07
max	373.609985	376.350006	370.179993	376.170013	376.170013	2.025224e+08

Fig. Description of Data of Microsoft Stock

Fig. Plotting graph for all stocks

We've plotted all the 4 data frame outputs in the graphs to see how the data analyzes the volatility of the data.

8.3 Data Preparation

As for the Data preprocessing and preparation process, Initially, we are removing the Volume column from the data frame as it is not required for future prediction because the price of stock is not dependent on volume. In this project our aim is to predict Close price for future based on Historical data, we have only considered Date and Close price from the data frame. We are considering 25% of Data as Test dataset and 75% is considered as Train Dataset

```
Apple Stock Final DataFrame:
                Close
Date
2013-11-20   18.392857
2013-11-21   18.612143
2013-11-22   18.564285
2013-11-25   18.705000
2013-11-26   19.049999
...                ...
2023-11-13  184.800003
2023-11-14  187.440002
2023-11-15  188.009995
2023-11-16  189.710007
2023-11-17  189.690002
```

Fig. Final Data Frame of Apple Stock

```
Apple Stock:
Train Data Size: 1887
Test Data Size: 629

Microsoft Stock:
Train Data Size: 1887
Test Data Size: 629

Walmart Stock:
Train Data Size: 1887
Test Data Size: 629

Amazon Stock:
Train Data Size: 1887
Test Data Size: 629
```

Fig. Train Test Data size for Apple Stock

Normalization: MinMaxScaler is a machine learning technique that scales and transforms features within a range of 0-1, useful in situations with different scales of input features or variables.

```
scaled_train_data['Apple']

array([[0.00682874],
       [0.00958189],
       [0.01316157],
       ...,
       [0.85851168],
       [0.87183854],
       [0.87024253]])
```

Fig. Data after Normalization for Apple Stock

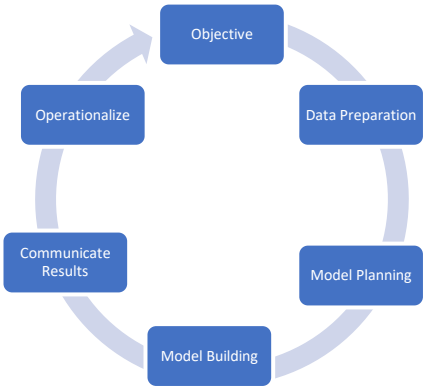


Fig Model Selection Cycle

The first stages involved importing necessary tools and inputting stock data. Using the date as a reference, we analysed the dataset for patterns and trends, prioritizing data cleaning and forecasting future prices using the date and closing prices.

9. Model Planning

Recurrent neural networks (RNNs) are designed for processing sequence data, unlike conventional neural network models that have nodes connecting layers. RNNs, which are named after their connection between hidden layers, use prior data from the hidden layer to calculate the current output. This includes the hidden layer's output from the previous instant. RNNs can handle any length of sequence data, making them an effective alternative to traditional models. Simple RNN consists of three layers: Input, Hidden, and Output. Input receives sequential data, Hidden captures temporal dependencies, and Output generates results based on this data. All layers consist of some characteristics which need to give value : units(number of neurons in the layer) , return sequences(define to get all sequences of output of just last time stamp output sequence), activation(function-(relu, tahn) applied to output of previous layers), input_shape(The tuple, which usually has the form (sequence_length, num_features), represents the dimensions of the input data).

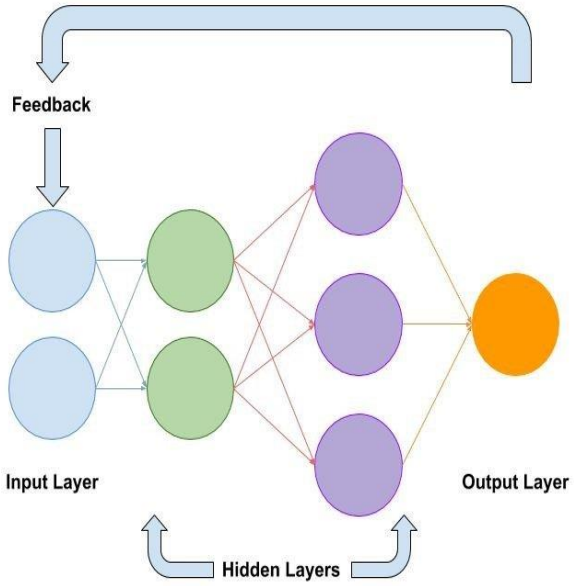


Fig. How Simple RNN Works

The training process involves the interaction between a loss function and a simple recurrent neural network (RNN), with the goal of minimizing the difference between actual and forecasted values through parameter adjustments, with Mean Absolute Error and Mean Squared Error being common regression task loss functions. The Simple RNN is a machine learning model that processes sequential input data, updates hidden states, and generates predictions using a loss function like MSE or MAE. Its parameters are refined, and performance evaluated on a separate validation dataset. For Simple RNN model, we need to have data to understand the pattern and trends of past data to predict future. Here, we are predicting close price for the future and for that we need are considering last 30days price to predict today's price.

X (Feature Matrix)	Y (Target)
Close[0:30]	Close[30]
Close[1:31]	Close[31]
Close[2:32]	Close[32]
And so on	

The table displayed that the first 30 days' Close price is used to predict the 30th day Close price, next 1 to 31 days' Close price to predict 31st day Close price and so on. We are using only one feature 'Close' price from the data set and predict the Close price by understanding the trends from the historical data of 10 years.

10.Model Building

Model building phase included selected model training with Model evaluation and model adjustment. As mentioned earlier, we are going to use Simple RNN model to train our sequential data with train: test ratio of 75:25. Below are the screenshot of Model building as well as training process and evaluation of the model.

```
# initialize the model
model = Sequential()

# Input Layer (input shape is 30X1 (days interval x features))
model.add(SimpleRNN(units=50, return_sequences=True, activation='tanh',
                    input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))

# hidden layers
model.add(SimpleRNN(units=50, return_sequences=True, activation='tanh'))
model.add(Dropout(0.2))

model.add(SimpleRNN(units=50))
model.add(Dropout(0.2))

# adding output layer
model.add(Dense(units=1))
```

Fig. Building Simple RNN Model

```
for stock_name in stock_names:
# Compile the model
  model.compile(optimizer='adam', loss='mean_squared_error')
  models_dict[stock_name] = model

  # Train the model
  history = model.fit(X_train, y_train, epochs=100, batch_size=32, callbacks=[es])
  history_dict[stock_name] = history
```

Fig. Compile and Train Simple RNN Model

Here callbacks = [es], mentioned early stopping. This means even if we have given 100 epochs to train model, when loss will be consisted without any further improvement, model training will be stopped because after consistency of error there is no use to train model again.

```
# Early stopping callback to prevent overfitting
es = EarlyStopping(monitor='loss', patience=5, verbose=1)
```

```
Epoch 45/100
59/59 [==============================] - 4s 70ms/step - loss: 0.0020
Epoch 46/100
59/59 [==============================] - 6s 101ms/step - loss: 0.0020
Epoch 47/100
59/59 [==============================] - 4s 70ms/step - loss: 0.0020
Epoch 48/100
59/59 [==============================] - 4s 68ms/step - loss: 0.0017
Epoch 49/100
59/59 [==============================] - 6s 107ms/step - loss: 0.0062
Epoch 50/100
59/59 [==============================] - 4s 66ms/step - loss: 0.0021
Epoch 51/100
59/59 [==============================] - 4s 67ms/step - loss: 0.0018
Epoch 52/100
59/59 [==============================] - 6s 106ms/step - loss: 0.0017
Epoch 53/100
59/59 [==============================] - 4s 71ms/step - loss: 0.0017
Epoch 53: early stopping
```

Fig. Model training Result with Early stopping

Here, from snippet we can check that, even if epoch value was=100, model train has stopped after 53 epoch for "Apple" stock because loss function value is almost consist of last 3 iteration. Loss is error between the actual close price and predicted close price which is calculated based on mean squared error (MSE), based on below formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE = mean squared error
n = number of data points
Y_i = observed values
$\hat{Y}_i$ = predicted values

Fig. Mean Squared Error

Model evaluation is done by draw the graph of loss value to check whether loss is decreased through the epoch training. Below image shows loss function graph for "Apple" and "Microsoft" stock companies.

```python
# Create subplots for loss visualization
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(12, 8))
axs = axs.flatten()
# Iterate over stock names
for i, stock_name in enumerate(stock_names):
    history = history_dict[stock_name]
    # Plot training loss
    axs[i].plot(history.epoch, history.history["loss"], label='loss')
    axs[i].set_title(f'{stock_name} from Simple RNN Model')
    axs[i].set_xlabel("Epochs")
    axs[i].set_ylabel("Loss")
    axs[i].legend()
# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```

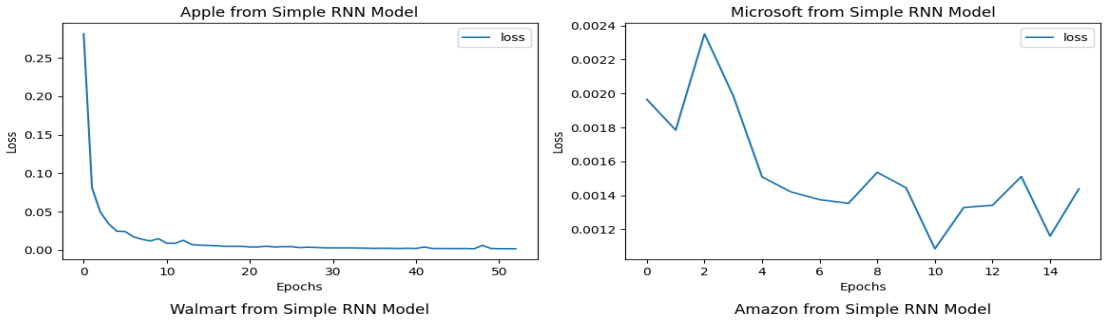
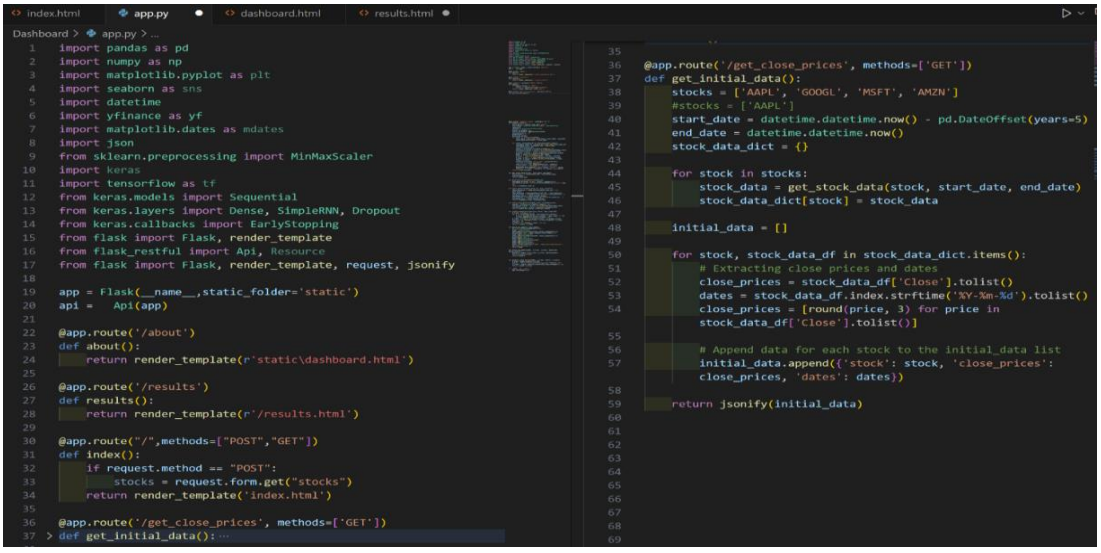


Fig. Loss Function changes during training for Apple and Microsoft

11. Code Screenshots

a) Purpose of each section: The Flask application (app.py) fetches and processes historical stock data for selected stocks, rendering web pages for About and results. It includes a route to provide initial stock data using the Yahoo Finance API.



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import yfinance as yf
import matplotlib.dates as mdates
import json
from sklearn.preprocessing import MinMaxScaler
import keras
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, Dropout
from keras.callbacks import EarlyStopping
from flask import Flask, render_template
from flask_restful import Api, Resource
from flask import Flask, render_template, request, jsonify

app = Flask(__name__,static_folder='static')
api =  Api(app)

@app.route('/about')
def about():
    return render_template(r'static\dashboard.html')

@app.route('/results')
def results():
    return render_template(r'/results.html')

@app.route("/",methods=["POST","GET"])
def index():
    if request.method == "POST":
        stocks = request.form.get("stocks")
    return render_template('index.html')

@app.route('/get_close_prices', methods=['GET'])
def get_initial_data():
    stocks = ['AAPL', 'GOOGL', 'MSFT', 'AMZN']
    #stocks = ['AAPL']
    start_date = datetime.datetime.now() - pd.DateOffset(years=5)
    end_date = datetime.datetime.now()
    stock_data_dict = {}

    for stock in stocks:
        stock_data = get_stock_data(stock, start_date, end_date)
        stock_data_dict[stock] = stock_data

    initial_data = []

    for stock, stock_data_df in stock_data_dict.items():
        # Extracting close prices and dates
        close_prices = stock_data_df['Close'].tolist()
        dates = stock_data_df.index.strftime('%Y-%m-%d').tolist()
        close_prices = [round(price, 3) for price in
        stock_data_df['Close'].tolist()]

        # Append data for each stock to the initial_data list
        initial_data.append({'stock': stock, 'close_prices':
        close_prices, 'dates': dates})

    return jsonify(initial_data)
```

b) Stock Data Processing and Prediction:

This code segment encompasses the essential functions for data processing, sequence creation, and RNN modeling in a Flask application. It efficiently splits data, scales it, builds, and trains an RNN model, and predicts future stock prices, offering a comprehensive solution for stock market forecasting.

```python
@app.route('/predict_stocks', methods=['GET'])
def predict_stocks():
    stock_param = request.args.get('stock')
    start_date = datetime.datetime.now() - pd.DateOffset(years=10)
    end_date = datetime.datetime.now()
    stocks_to_predict =[]
    stocks_to_predict.append(stock_param)
    stock_data_dict = {}
    predictions = {}
    predicted_data=[]
    for stock in stocks_to_predict:
        stock_data = get_stock_data(stock, start_date, end_date)
        stock_data_dict[stock] = stock_data

    for stock, stock_data_df in stock_data_dict.items():
        stockdata_final_df = preprocess_stock_data(stock_data_df)
        stockdata_train, stockdata_test = split_data(stockdata_final_df)
        scaler = MinMaxScaler(feature_range=(0, 1))
        stocktrain_scaled = scaler.fit_transform(stockdata_train)
        stocktest_scaled = scaler.transform(stockdata_test)
        X_train, y_train = create_sequences(stocktrain_scaled)
        model = build_rnn_model(X_train.shape)
        history = train_rnn_model(model, X_train, y_train)
        X_test, y_test = create_sequences(stocktest_scaled)
        y_pred, y_train = predict_stock_prices(model, X_test, scaler, y_train)
        predictions[stock] = {'y_pred': y_pred.tolist(), 'y_train': y_train.tolist()}
        result_json = json.dumps(predictions, indent=2)
        y_train_flat = [value[0] for value in y_train]
        predicted_data.append({'stock':stock,'y_pred': y_pred.tolist(), 'y_train':
        [value[0] for value in y_train]})
    return predicted_data

def get_stock_data(ticker, start_date, end_date):
    stock_data = yf.download(ticker, start=start_date, end=end_date)
    return stock_data

def preprocess_stock_data(stockdata_df):
    stockdata_df.drop('Volume', axis=1, inplace=True)
    stockdata_final_df = pd.DataFrame(stockdata_df['Close'].copy())
    return stockdata_final_df
```

```python
def split_data(stockdata_final_df, test_size=0.3):
    train_datasize = round(len(stockdata_final_df) * (1 -
    test_size))
    test_datasize = len(stockdata_final_df) - train_datasize
    stock_traindf = stockdata_final_df.iloc[:train_datasize, :]
    stockdata_train = stock_traindf.values
    stock_testdf = stockdata_final_df.iloc[train_datasize:, :]
    stockdata_test = stock_testdf.values
    return stockdata_train, stockdata_test

def scale_stock_data(stockdata_train, stockdata_test):
    scaler = MinMaxScaler(feature_range=(0, 1))
    stocktrain_scaled = scaler.fit_transform(stockdata_train)
    stocktest_scaled = scaler.transform(stockdata_test)
    return stocktrain_scaled, stocktest_scaled

def create_sequences(stocktrain_scaled, days_steps=30):
    X_train, y_train = [], []
    for i in range(days_steps, len(stocktrain_scaled)):
        X_train.append(stocktrain_scaled[i - days_steps: i, 0])
        y_train.append(stocktrain_scaled[i, 0])
    X_train, y_train = np.array(X_train), np.array(y_train)
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.
    shape[1], 1))
    y_train = np.reshape(y_train, (-1, 1))
    return X_train, y_train

def build_rnn_model(X_train_shape):
    es = EarlyStopping(monitor='loss')
    model = Sequential()
    model.add(SimpleRNN(units=50, return_sequences=True,
    activation='tanh', input_shape=X_train_shape[1:]))
    model.add(Dropout(0.2))
    model.add(SimpleRNN(units=50, return_sequences=True,
    activation='tanh'))
    model.add(Dropout(0.2))
    model.add(SimpleRNN(units=50))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))
    model.compile(optimizer='adam', loss='mean_squared_error',
    metrics=['accuracy'])
    return model
```

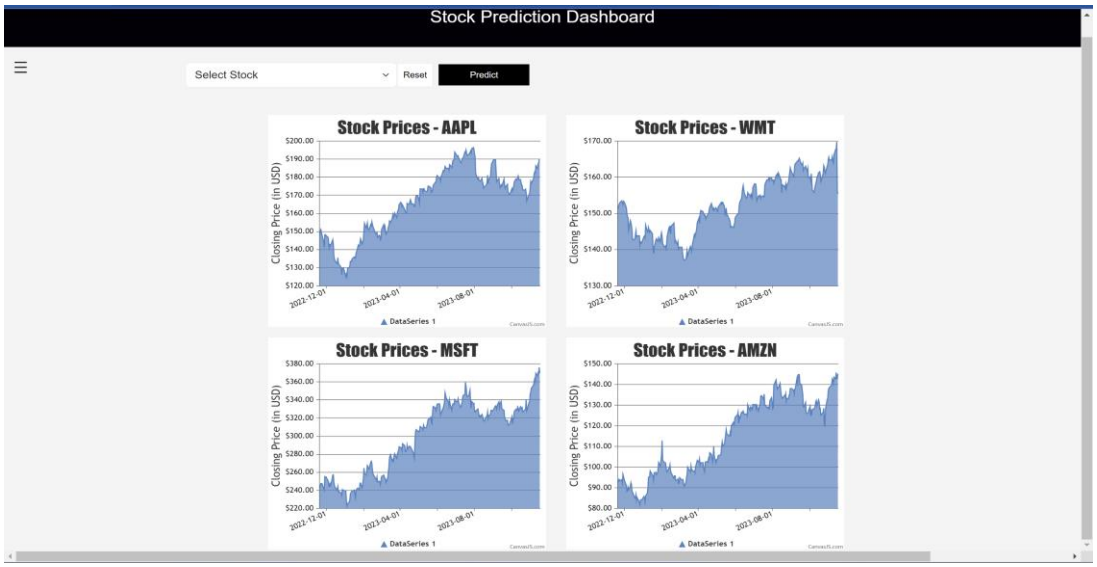
```python
def train_rnn_model(model, X_train, y_train, epochs=10, batch_size=32):
    history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size)
    return history


def predict_stock_prices(model, X_train, scaler, y_train):
    y_pred = model.predict(X_train)
    y_pred = scaler.inverse_transform(y_pred.reshape(1,-1))
    y_train = scaler.inverse_transform(y_train.reshape(-1, 1))
    return y_pred, y_train

if __name__ == "__main__":
    app.run(debug=True)
```

c) RNN Model Training Code:

This code trains an RNN model and predicts stock prices using Keras. The training function (train_rnn_model) trains the model, and the prediction function (predict_stock_prices) produces stock price predictions, ensuring the interpretability of results.

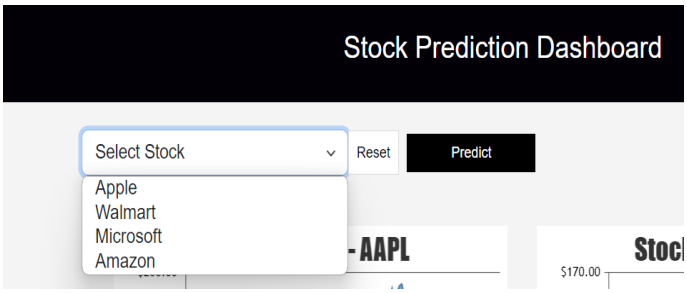
12.Operationalization

Dashboard: It contains four different stock price predictions.

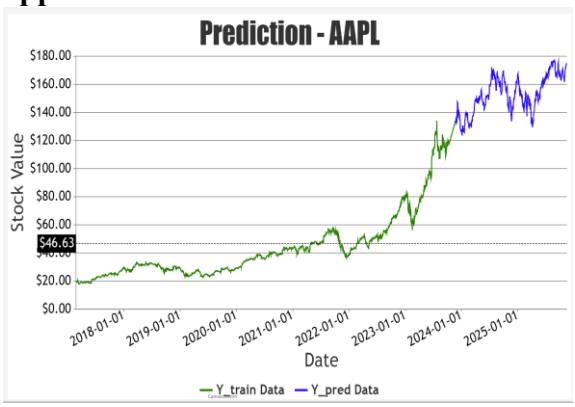


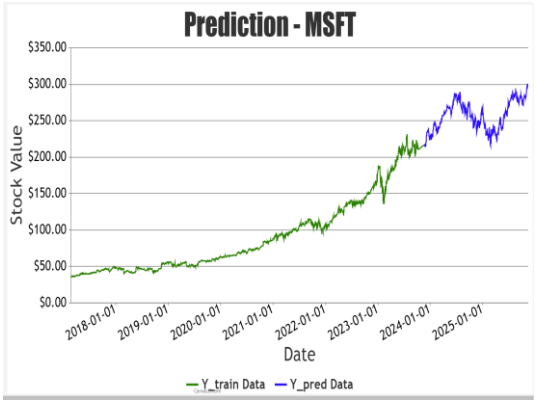
Predictions

Based on selection of stock from dropdown, and click on "Predict" button, it will display below predictions.

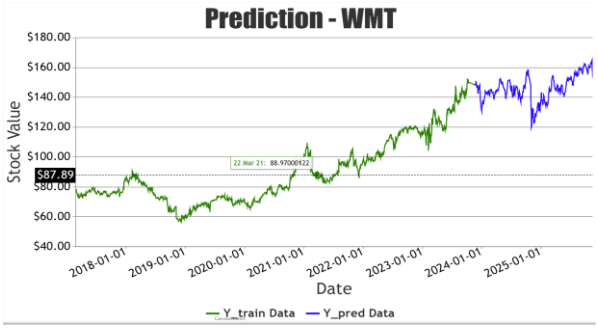


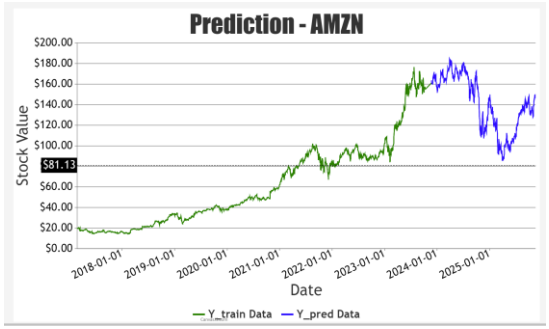
Apple & Microsoft:





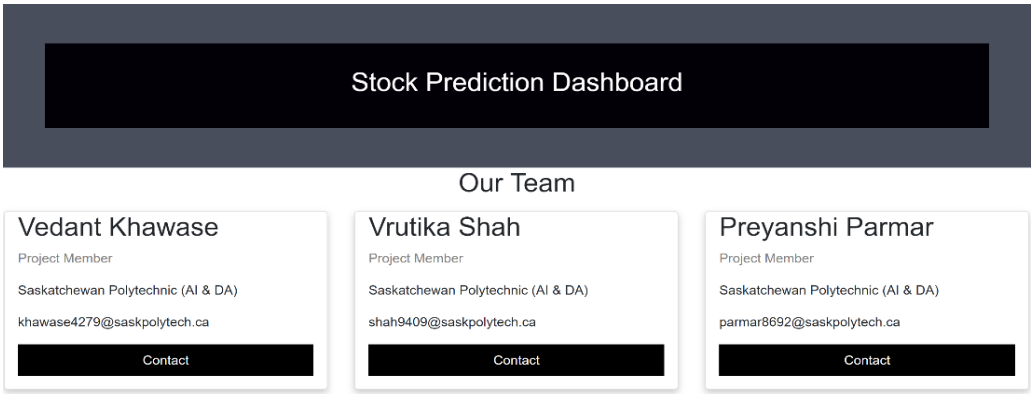
Walmart & Amazon:





This all stocks shows the in the graph line in green is about the historical data and line in blue is about the predicted data.

Contact us Details:



13.Limitations and Future Scopes

The implementation is limited to only four stocks (Amazon, Walmart, Microsoft, and Walmart) by considering only past 10 years historical data. We are only considering Close price for the stocks and user must wait for some time to get prediction because it is taking time. In future, more stock can be included to predict the price and it will also be considered other features such as Open, Low, High parameters for Close price prediction. We will also need to consider other types of factors such as sentiment analysis, financial statements, interest rates, GDP growth and so on. We will also be taking data from many past years like 30 or 40 years to predict the future in better way and at the same time we will also need to work on the model to make to it robust and faster so that it will give such prediction without any waiting time.

14.Conclusion

Analysts argue that estimating the return on the stock market is challenging due to its association with real-time events and randomness. This study aims to demonstrate potential methods like Simple RNN for forecasting and using it as a support system for human decision-making. Although experiments don't guarantee high accuracy, the prediction findings suggest a path in the right direction for the past decade. Combining these findings with human experience information can help make financial judgments.

15.References

1. To download Data from Yahoo Finance - Stock Market Live, quotes, Business & Finance News (no date) Yahoo! Finance. https://finance.yahoo.com/
2. Raval, | BY Param (no date) Stock price prediction using machine learning with source code, ProjectPro. Available at: https://www.projectpro.io/article/stockprice-prediction-using-machine-learning-project/57
3. Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short-term memory. PloS one, 12(7): e0180944, 2017
4. https://medium.com/hackernoon/rnn-or-recurrent-neural-network-for-noobs-a9afbb00e860
5. https://core.ac.uk/download/pdf/286268975.pdf
6. https://www.mdpi.com/2227-7072/11/3/94
7. https://towardsdatascience.com/time-series-forecasting-with-deep-learning-and-attention-mechanism-2d001fc871fc
8. Manish Kumar and M Thenmozhi. Forecasting stock index movement: A comparison of support vector machines and random forest. In Indian institute of capital markets 9th capital markets conference paper, 2006.
9. https://www.researchgate.net/publication/354061072_Comparative_Analysis_of_Recurrent_Neural_Networks_in_Stock_Price_Prediction_for_Different_Frequency_Domains
10. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7517440/
11. https://arxiv.org/ftp/arxiv/papers/2111/2111.01137.pdf
12. https://link.springer.com/article/10.1007/s11280-021-01003-0#Sec15
13. https://www.w3schools.com/js/DEFAULT.asp
14. https:// pythonanywhere.com
15. https://i.stack.imgur.com/QEPAU.png
16. https://www.researchgate.net/profile/Mohamed-Abdelrahman-28/publication/337830532/figure/fig3/AS:834058519064577@1575866455615/A-simple-RNN-architecture.jpg
17. https://i.stack.imgur.com/Xxc9W.png
18. Chi-Jie Lu, Tian-Shyug Lee, and Chih-Chou Chiu. Financial time series forecasting using independent component analysis and support vector regression. Decision Support Systems, 47(2):115–125, 2009