PSTP: PEPPERONI STOCK TREND PREDICTOR

A-Level Computer Science Project

Yan Taborov (U6f)

Analysis:

Identification of Problem:

In order to come up with an idea for my A-Level Computer Science Project, I brainstormed ideas that are of my interest and can be applied to solve modern-world problems. After researching for hours, I eventually came up with an idea of a Stock Market Trend Predicting programme using Python and Machine Learning. Although I never studied finances and markets at school, these topics have been of my interest for the last couple of years and I have spent some time researching them on my own.

The problem that many investors and traders face today is that in some particular cases it becomes extremely difficult to predict the share price due to market manipulations, monopolistic shareholders and insider trading. This programme should serve for investors as a tool in order to acquire better understanding of the possible financial outcomes when managing stocks.

Dialogue with Client:

Client: John Johnson – Investment Banker

Note:

Y = interviewer

J = client

Transcript:

Y: "Is there any system currently that helps you in predicting the stock markets?"

J: "Right now, we are using some statistical models and basic tools of trend analysis, but, of course, we will need a sophisticated solution that is mouldable according to the market conditions.

Y: "Please give some background about your specific requirements and objectives from this software."

J: "Our biggest concern is, obviously, to do better—to have a tool capable of analysing historical data and providing us with predictions as accurate as possible, so that we can make better decisions based on this prediction.

Y: "Do you consider some signs or measures that count while forming your trading strategies?"

J: "We need to be able to analyse software, usually meaning a substantial corpus of financial data, including historical trends in price movement, volumes of trading, and selected economic indicators which might bear directly on the investment product. The software should apply machine learning algorithms in its integration to recognize the pattern and trend of every data being input. It should also have a user-friendly interface, such that the data presented is understood by every level of company staff in a short time.

Identification of Users:

As such, experienced investors need to be catered to, and that is what I would like to design in a way that even newcomers would find it easy to use. It has got to clearly communicate the user interface that gives any trader, whether experienced or a newbie, the ability to use it.

Setting Limits:

The developed project will be able to cater to the needs of financial institutions, trading firms, and individual investors, spread all over the world, to provide desired solutions for them in a customised format. The main objective is that the data or information related to users and financials should be safe. The system shall provide exact predictions in the applicable time. Ideally, each prediction should take no more than a couple of seconds, but exceptions could exist for more complex scenarios. Efficient data management practices will be employed, optimising memory usage through data normalisation.

Existing Systems:

The presentation of prediction results will be designed for easy interpretation. In developing a stock market trend predictor in Python, I scanned the tools which are available on the market today. XYZ Trend Analytics uses computer algorithms for the prognosis of behaviour, while AB Investments makes trends in the stock market using mathematical patterns. PQR Market Insights analyses news and sentiment around stocks to aid in making predictions. Other competitors include companies such as BlackRock Aladdin, which has capitalization running into trillions and is famous for predictive capabilities. However, these solutions come with limitations, including limited access and costs. On the other end, my project democratically allows easy access to accurate predictions through the use of Python and Machine Learning while at the same time having an easily navigable interface. The custom solutions I offer are amicable to a larger user group and bridge the gap between experienced investors and novices. This emphasis on ease of access and usability is what sets my project apart from current competitors, presenting an opportunity to articulate not only to a bigger audience interested in the ease of using data but also in making investment decisions.

Data Security:

Security aspect will not be assessed as no personal or private information is stored.

Non-Functional Requirements:

1. Performance.
    a. The system should provide predictions for stock market prices in real-time or near real-time, ensuring timely decision-making for investors.
    b. The system should be capable of handling a substantial number of concurrent users without significant degradation in performance.

    c. The prediction process should be completed within a reasonable time frame, typically a few seconds per prediction.
2. Accuracy.
    a. The predictive model should strive for a high level of accuracy in forecasting stock prices to enhance the reliability of its insights.
    b. The system should be able to adapt and fine-tune its algorithms to improve prediction accuracy over time based on user feedback and performance analysis.
3. Security.
    a. Measures should be in place to inform the user when the online stock exchange markets experience troubles.
4. Usability:
    a. The user interface should be intuitive and user-friendly, catering to both experienced investors and those with limited technical expertise.
    b. Clear and concise documentation should be provided to guide users through system functionalities and usage.
5. Maintainability:
    a. The codebase should be well-structured and modular, making it easy for developers to maintain and enhance the system in the future.
    b. Regular updates and improvements should be feasible without disrupting the overall system functionality.
6. Compatibility:
    a. The system should be compatible with various web browsers and operating systems to provide a seamless experience to a diverse user base.
7. Resource Utilisation:
    a. The system should manage computing resources efficiently to avoid excessive usage and optimize energy consumption.

Functional requirements:

1. Input Data Handling:
    a. The system should be capable of ingesting historical stock market data from the reliable sources.
    b. Data should be cleaned and pre-processed to remove anomalies and inconsistencies.
2. Prediction Model Implementation:
    a. The system should incorporate machine learning algorithms to build and train a prediction model based on historical stock data.
    b. The prediction model should accept relevant input features for each stock and generate price forecasts as output.
3. Prediction Display:
    a. The predictions should be presented in a clear and comprehensible manner, possibly in the form of graphs or charts.
4. User Interaction:
    a. Users should be able to select specific stocks for which they want to receive predictions.
    b. Users should be able to select more than one stocks for which they want to receive predictions.
5. Prediction Accuracy Analysis:

    a. The system should provide metrics to assess the accuracy of its predictions, such as mean squared error or percentage error.
6. Data Export and Reporting:
    a. The system should allow users to export prediction results and related data for further analysis.
    b. Users should be able to generate reports summarizing prediction accuracy and insights.
7. Error Handling:
    a. The system should provide meaningful error messages to users in case of input errors or system failures.
8. User-Friendly Interface:
    a. The user interface should be intuitive, allowing users to easily navigate and interact with the prediction functionalities.

Documented design

The goal of this project is to develop a predictive model for forecasting stock market trends using machine learning techniques. For data manipulation, model generation, and user interface development, I will make use of the Python programming language and related libraries.

The Name

I used the name Pepperoni Stock Trend Predictor for no specific reason.

The Problem

The goal of this project is to develop a predictive model for forecasting stock market trends using machine learning techniques. For data manipulation, model generation, and user interface development, I will make use of the Python programming language and related libraries.

The Plan

After doing my research, I found out that a long short-term memory neural network (LSTM) for times series forecasting is one of the best solutions for predicting stock values.

LSTM

LSTMs (long short-term memory) are a special type of RNN (recurrent neural network) that can maintain information through many time steps and do not have the gradient vanishing problem. They were designed to surmount the short-term memory dilemma that is standard to RNNs; hence, they are very popular for tasks such as predicting stock prices. The vanishing gradient problem, as seen in a recurrent neural network during backpropagation, when it comes to updating the weights of a neural network using gradients, is that the gradient decreases as it travels backwards in time up to a point where it doesn't contribute to learning.

Recurrent neural networks can suffer from short-term memory due to the first layers receiving minimal gradient updates and subsequently not learning. This can cause the network to forget information in longer sequences.

To address this issue, LSTM's and GRU's were developed, which utilise internal gates to regulate the flow of data.

LSTMs and GRUs are used extensively in state-of-the-art recurrent neural network outcomes, as they allow for the learning of which data in a series should be kept or discarded. This enables the conveyance of relevant information down long chains of sequences to generate predictions. They are used in voice recognition, voice synthesis, text production, and even video captioning. The text adheres to metrics and units, and is free from grammatical errors, spelling mistakes, and punctuation errors. The content of the improved text is as close as possible to the source text, with no additional aspects added.

When vectors pass through a neural network, they undergo several modifications as a result of various math operations. Consider a value that is continually multiplied by, say, 3. Some values can inflate and become astronomical, making other values appear small.

In the heart of LSTMs lie the cell state and its many gates. The cell state acts like a sort of transportation highway for relative information down the sequence chain. We can think of it as the memory of the network. Cell state carries information throughout the processing of the sequence. Information can travel across many time steps to affect the state at a later time step, hence degrading the effectiveness of gradient-based learning through time with respect to time steps that are separated from the influencing time steps by more than one or two intervening time steps. The gates allow the information to be added or removed within the cell state. The gates are several neural networks deciding what information about the cell state will be allowed to pass through. Gates can learn which information to retain and which to discard during training.

In an LSTM cell, three separate gates regulate information flow: a forget gate, an input gate, and an output gate.

Forget gate.

This gate will decide what information is unimportant and should be thrown away, and what should be passed on. For this reason, that a sigmoid function is capable of processing the information of the previous hidden state and the current input.

Input Gate

The input gate updates the cell state by adding together the products of the previous concealed state and the current input, using a sigmoid function. This decides the values to change by converting to be in between 0 and 1; the numbers 0 and 1 tell the importance. To help regulate this network, we'll also send the hidden state and current input through a tanh function. Squishing values between -1 and 1, the sigmoid value is multiplied by the tanh value. The information to be retained from the tanh result is determined by the sigmoid output.

The Cell State

The Forget vector is then multiplied pointwise with the cell state. The model should now have enough data to calculate the status of each cell. This has the potential to erase values in the cell state if it is multiplied by values close to zero. The input gate output is used to finally pointwise add, updating the cell state with new values which the neural network finds important. The result of this is our new cell state.

Output Gate

It is what determines the next hidden state, the hidden state containing data from the previous inputs. It is also used to make predictions. First, we apply the sigmoid function to gate the combination of the former concealed state and the current input. Then, the new adjusted cell state is passed into the tanh function. To get which information to have in the hidden state, the output tanh is multiplied by the tanh output to get a hidden state. The sum of both the modified cell state and the hidden state is then passed forward to the next step.

The forget gate finds what is significant to keep from the earlier steps, and the input gate determines what data from the current phase should be added. The output gate states the next state of the hidden state.

Data Gathering and Pre-processing

When choosing a library for web scraping, I need to consider factors such as data availability, ease of integration, customisation options and data variety. There are numerous tools I could consider for retrieving financial data for the stock markets. I have done my research on it and selected a few with the reasons why I could utilize them for my project.

The first one that I came across with is Alpha Vantage.

Alpha Vantage could be a suitable choice for my stock market trend predictor due to several key features and capabilities that align with my project's requirements. However, Alpha Vantage imposes usage limits on their free tier, including the number of API requests you can make within a time frame. More advanced data manipulation may require a premium subscription, which is not a great option for me.

I also came across with a tool called Quandl.

Quandl gives a great deal of financial and economic datasets beyond just stock prices. This includes data on equities, futures, options, indices, economic indicators, and interest rates. However, while browsing the internet, I have also found some disadvantages of using such model: Free access provides limited data and functionality, while more comprehensive datasets and advanced features, which I might need to use, will require a paid subscription.

My next stop was Tiingo.

I did my research and the Tiingo was a suitable option for me, until I encountered the price of the subscription. As my project will not have any commercial use, I have decided to use the software options which are not hidden behind a paywall.

I was hopeless, until I came up with an idea to talk to my friend who is a back-end developer with a lot of experience. He suggested me to use Python pandas data reader to scrape the market data directly from Yahoo Finance. It was the best solution for me because it is an open-source library and has great functionality.

The pandas_datareader library will be used to obtain historical stock data from trusted sources, such as Yahoo Finance.

I also realised the need of data cleaning and pre-processing while constructing my Stock Market Trend Predictor. This critical step is fine-tuning the data to verify its reliability as well as precision. Data becomes more consistent and representative of actual market trends when inconsistencies and missing values are removed. This preparation protects against inaccurate predictions caused by noisy or inadequate data. I can produce more relevant results by using data cleaning and preliminary processing.

Moving averages (100- and 200-day averages) will be computed as well.

Moving averages are crucial analytical tools, smoothing out fluctuations in price data across defined time intervals.

Visualisation

The closing prices and moving averages of selected equities will be displayed using graphs and/or charts.

Matplotlib will be used to provide clear visualisations that aid in the comprehension of historical trends. Matplotlib is primarily a 2D plotting library, it was chosen because it ensures accessibility by providing images in PDF, PGF, and PNG formats.

Splitting and Scaling Data

The dataset will be divided into training and testing subsets.

A scaler will be used because the values need to be between 0 and 1, because only the data between zero and one can be passed to the model.

Data Preparation

A series of the past 100 days' data will be used as input (x_train) for each prediction, while the value of the 101st day is used as output (y_train).

To generate these sequences, a for loop will iterate through the data.

Architecture Model

I decided to choose Keras framework to generate a sequential model for a couple of reasons. Keras provides a high-level abstraction that would shield me from the complexities of low-level implementation details. This abstraction will allow me to focus on designing and training my neural network model without getting deep down in technical complications. Keras also offers a modular structure that makes it easy to construct complex neural network architectures by stacking and configuring different layers.

To record temporal associations in data, LSTM (Long Short-Term Memory) layers are used, because LSTM layers are designed to capture long-term dependencies in sequences, which is crucial for analysing stock market data. Financial trends and patterns often span across multiple days, and LSTM's memory cells enable the network to retain relevant information over extended periods.

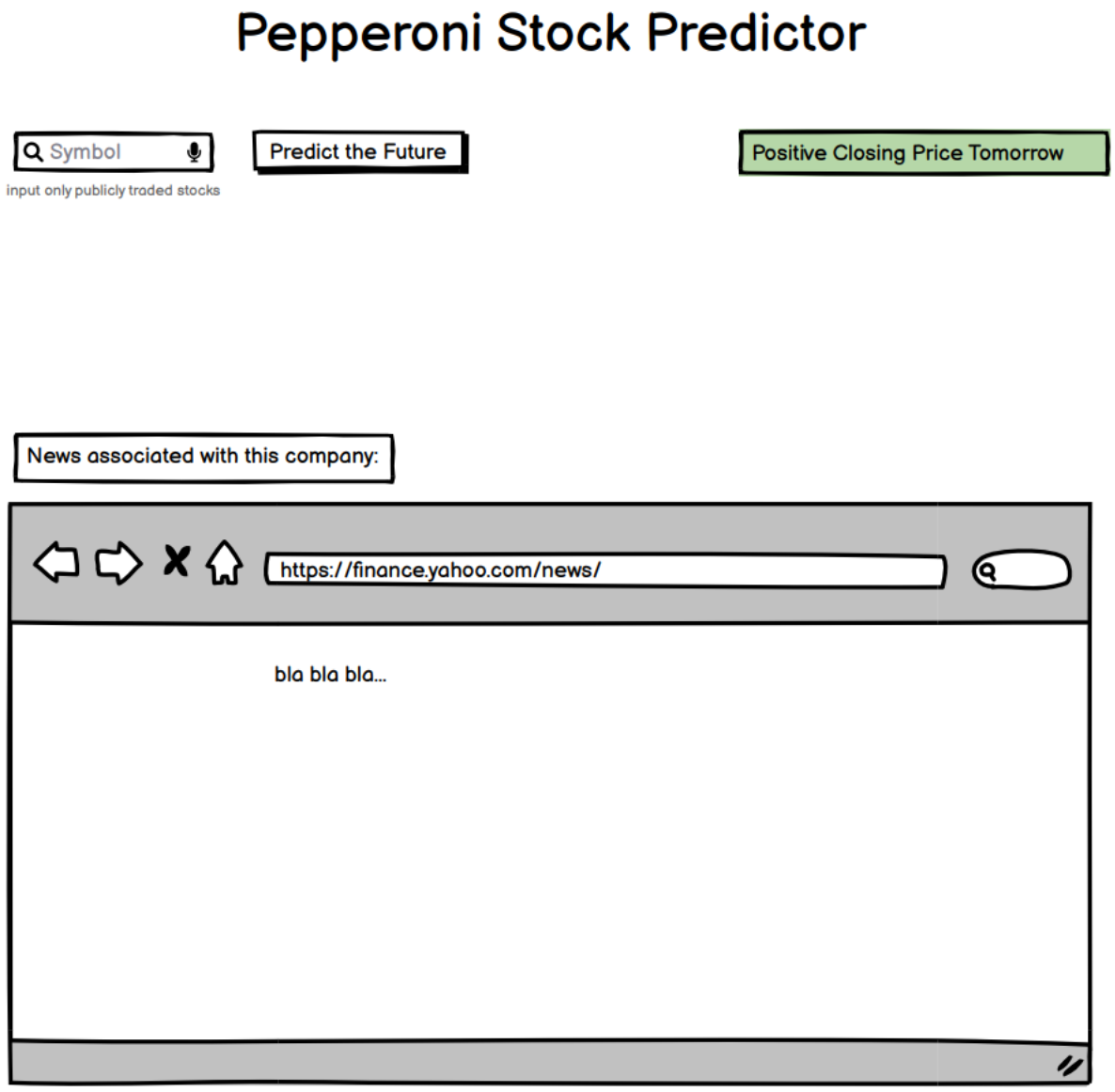
Dense and Dropout layers will be added to the model to increase its capacity and prevent overfitting.

Model Collection and Training

The Adam optimizer will be used because this optimizer adjusts the learning rate for each parameter during training. This allows it to converge faster and find optimal solutions in scenarios where the landscape of the loss function is varied or changes over time. The Mean Squared Error (MSE) loss function is used to construct the model because the stock prices are continuous variables, and predicting their precise values is essential.

User Interface Design

Balsamiq mock-up:



User interface design was not the hardest part, but not the easiest neither. The reason being is that there is a great deal of services which provide user interface for python programmes and all of them have their own pros and cons. So, I needed to compare all of them and chose the most suitable one.

The first one that I could think of and the only one platform that I knew at that moment was Tkinter, suggested by my computer science teacher, Dr. Wild. So, I decided to ask my friend what alternatives were there on the market.

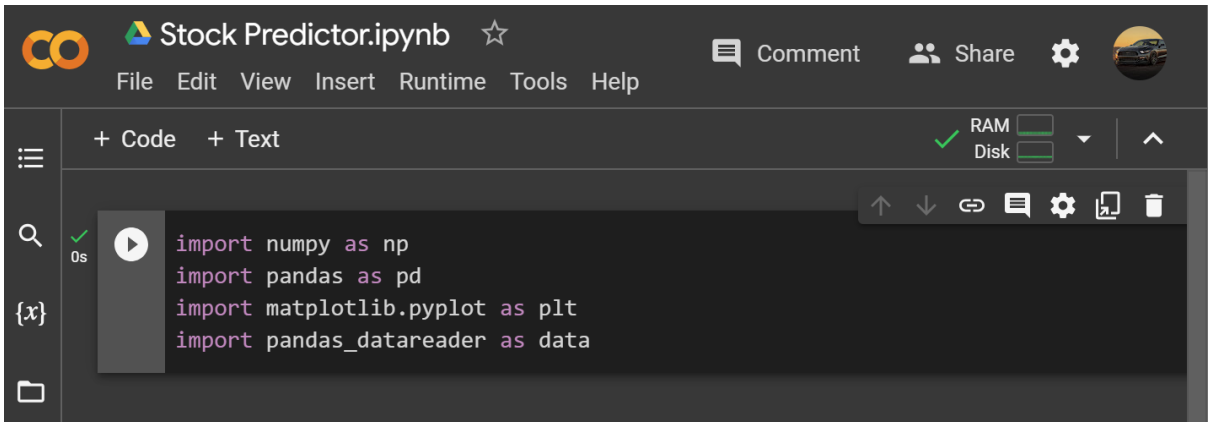
The first one he suggested was Dash by Plotly. It is a web application framework that enables developers to create web-based data visualizations in Python. The next one I came across with is Streamlit. The reason why I have chosen it for my project is because of its simplicity and speed. This is particularly advantageous for my project as it allows me to focus more on the machine learning model and data analysis rather than spending excessive time on interface development. I would just need to download streamlit module to use it.

Technical Solution

A little bit of preface

When it came time to creating my own LSTM model for the first time, I chose to use the Keras framework (because I was already quite familiar with it). At that point, I realised that the hardest and most challenging part was understanding how to prepare and transform my input data to match the expectations of the Keras model input and how to transform my training and validation labels to match the output of the network for validation and testing. Additionally, I have discovered when dealing with LSTMs that there are numerous network-specific characteristics that I must comprehend and take care of in order to begin working with it.

For coding my programme, I will be using Colab Tool from Google because at the moment of writing the code I do not have my own computer with me.



Importing basic libraries.

I have chosen pandas_datareader because I want to scrap the data from Yahoo Finance website.

```
start = '2010-01-01'
end = '2015-01-01'

df = data.DataReader('AAPL', 'yahoo', start, end)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-947aca9c0b7f> in <cell line: 4>()
      2 end = '2015-01-01'
      3
----> 4 df = data.DataReader('AAPL', 'yahoo', start, end)

                                  3 frames
/usr/local/lib/python3.10/dist-packages/pandas_datareader/yahoo/daily.py in
_read_one_data(self, url, params)
    151         try:
    152             j = json.loads(re.search(ptrn, resp.text, re.DOTALL).group(1))
--> 153             data = j["context"]["dispatcher"]["stores"]["HistoricalPriceStore"]
    154         except KeyError:
    155             msg = "No data fetched for symbol {} using {}"

TypeError: string indices must be integers

SEARCH STACK OVERFLOW
```

I faced an error “string indices must be integers”, and after looking for a solution, I decided to override “pandas_datareader”.

```
from pandas_datareader import data as pdr
import yfinance as yf

yf.pdr_override()

start ='2000-01-01'
end ='2023-07-07'

df = pdr.get_data_yahoo('BLK', start, end)
df.head()
```

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

Date	Open	High	Low	Close	Adj Close	Volume
2000-01-03	17.4375	17.500	16.625	16.625	10.540839	64600
2000-01-04	16.6250	16.625	15.625	15.875	10.065311	81900
2000-01-05	15.7500	15.750	15.000	15.750	9.986056	38100
2000-01-06	15.7500	16.250	15.750	16.000	10.144572	41200
2000-01-07	15.8750	16.250	15.750	15.875	10.065311	34700

0s completed at 19:38

I have decided to choose BlackRock share price as this company has already produced a stock prediction algorithm. I refined starting and ending points. I should take bigger timeframe for my dataset, because it will improve the accuracy of the model.

```
df.tail()
```

Date	Open	High	Low	Close	Adj Close	Volume
2023-06-29	686.700012	690.909973	681.979980	688.210022	688.210022	524700
2023-06-30	695.119995	695.979980	690.450012	691.140015	691.140015	921900
2023-07-03	690.000000	702.159973	689.739990	693.580017	693.580017	376900
2023-07-05	691.669983	700.630005	685.869995	692.830017	692.830017	681400
2023-07-06	685.000000	686.500000	674.799988	680.630005	680.630005	675300

We can see that our dataset is ending 06/07/2023.

```
df = df.reset_index()
df.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2000-01-03	17.4375	17.500	16.625	16.625	10.540837	64600
1	2000-01-04	16.6250	16.625	15.625	15.875	10.065311	81900
2	2000-01-05	15.7500	15.750	15.000	15.750	9.986058	38100
3	2000-01-06	15.7500	16.250	15.750	16.000	10.144566	41200
4	2000-01-07	15.8750	16.250	15.750	15.875	10.065311	34700

I reset the index as I do not need the dates in the front because they are not useful for me.

```
df = df.drop(['Date', 'Adj Close'], axis = 1)
df.head()
```

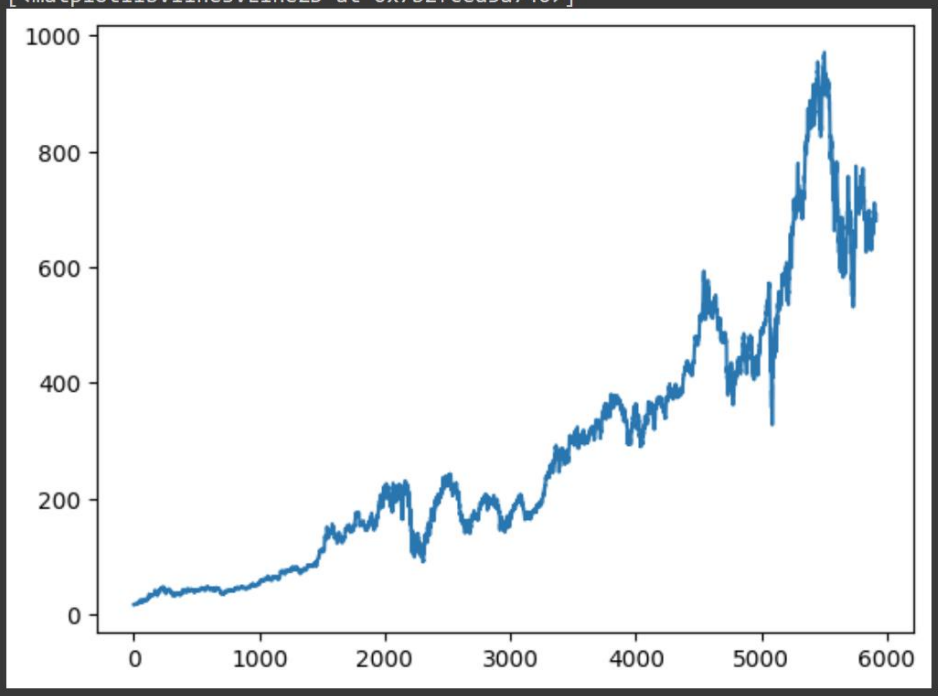
	Open	High	Low	Close	Volume
0	17.4375	17.500	16.625	16.625	64600
1	16.6250	16.625	15.625	15.875	81900
2	15.7500	15.750	15.000	15.750	38100
3	15.7500	16.250	15.750	16.000	41200
4	15.8750	16.250	15.750	15.875	34700

+ Code + Text

I decided to drop the “Date” and “Adj Close” columns, because they are useless for my task. I have also run “axis = 1”, which specifies the axis along which the data is computed. In our case, Axis 1 will act on all the columns in each row.

```
plt.plot(df.Close)
```

```
[<matplotlib.lines.Line2D at 0x7b2fcea3a740>]
```



We can now see a simple graph showing the closing price of the stock.

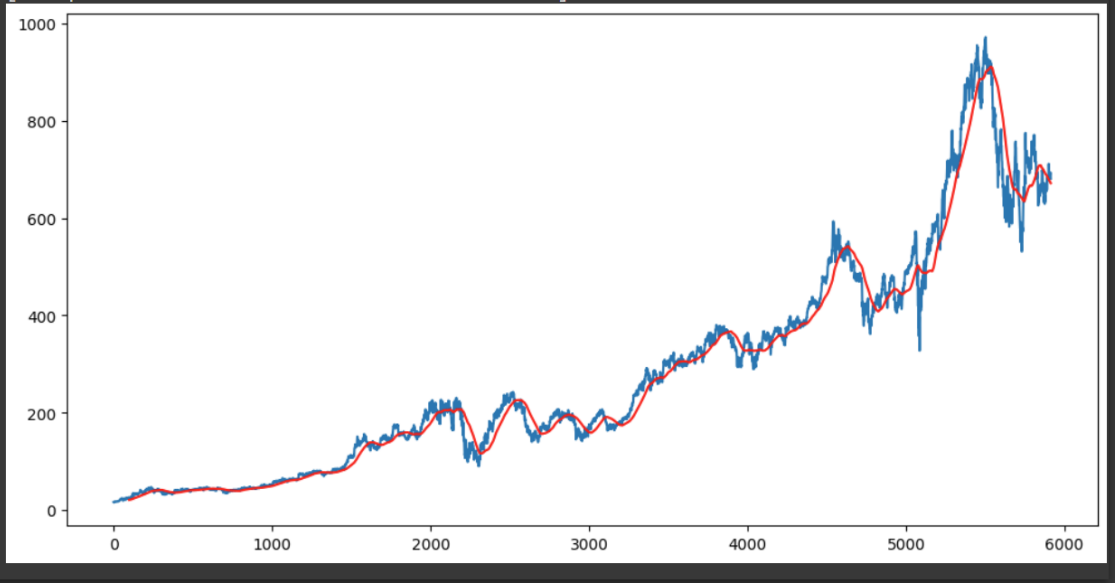
```
ma100 = df.Close.rolling(100).mean()
ma100
```

```
0             NaN
1             NaN
2             NaN
3             NaN
4             NaN
           ...
5909    673.252300
5910    672.739901
5911    672.267501
5912    671.897201
5913    671.543801
Name: Close, Length: 5914, dtype: float64
```

Now I have created the 100-days moving averages (Moving averages (MA) are used in technical analysis and are designed to smoothen price fluctuations by producing a continuously updated mean value).

```
plt.figure(figsize = (12,6))
plt.plot(df.Close)
plt.plot(ma100, 'r')
```

```
[<matplotlib.lines.Line2D at 0x7c817b1e7b80>]
```



I have now plotted the moving average on my closing graph (in a red colour so it is more visible).

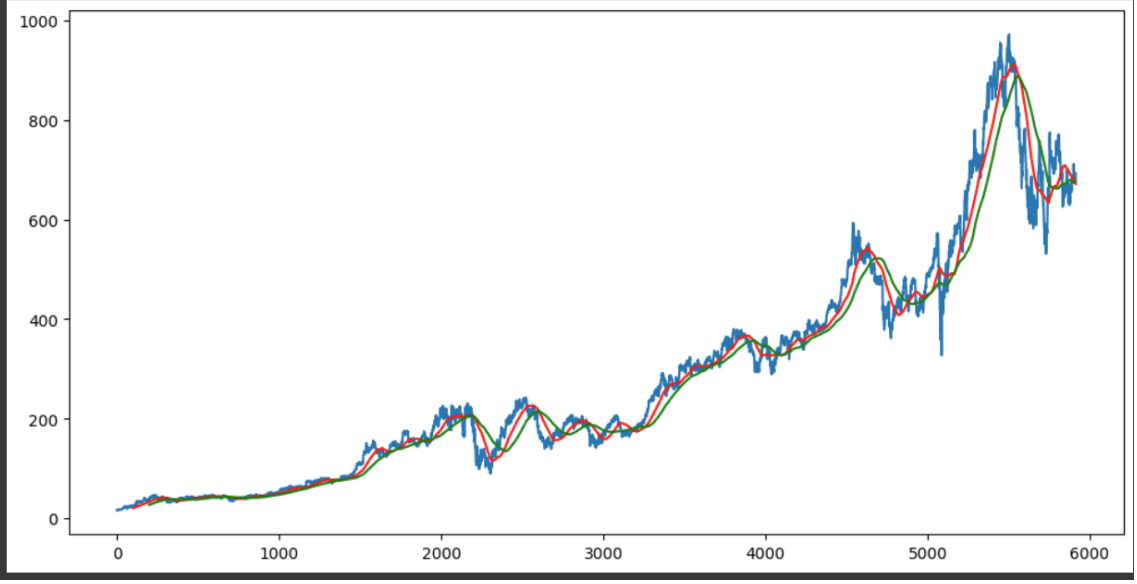
```
ma200 = df.Close.rolling(200).mean()
ma200
```

```
0             NaN
1             NaN
2             NaN
3             NaN
4             NaN
           ...
5909    675.226851
5910    675.458001
5911    675.708001
5912    676.010451
5913    676.279901
Name: Close, Length: 5914, dtype: float64
```

I have done a similar procedure to define the 200-days moving average. The values “NaN” which means none are present for the first 200 days but starting from the 201 index they have a real value.

```
plt.figure(figsize = (12,6))
plt.plot(df.Close)
plt.plot(ma100, 'r')
plt.plot(ma200, 'g')
```

```
[<matplotlib.lines.Line2D at 0x7c81770dd810>]
```



I have created a chart including the Closing price, 100MA (red colour) and 200MA (green colour).

Training and testing

I have been plotting simple visualisations. Now I will move on to training and testing.

```
df.shape
```

```
(5914, 5)
```

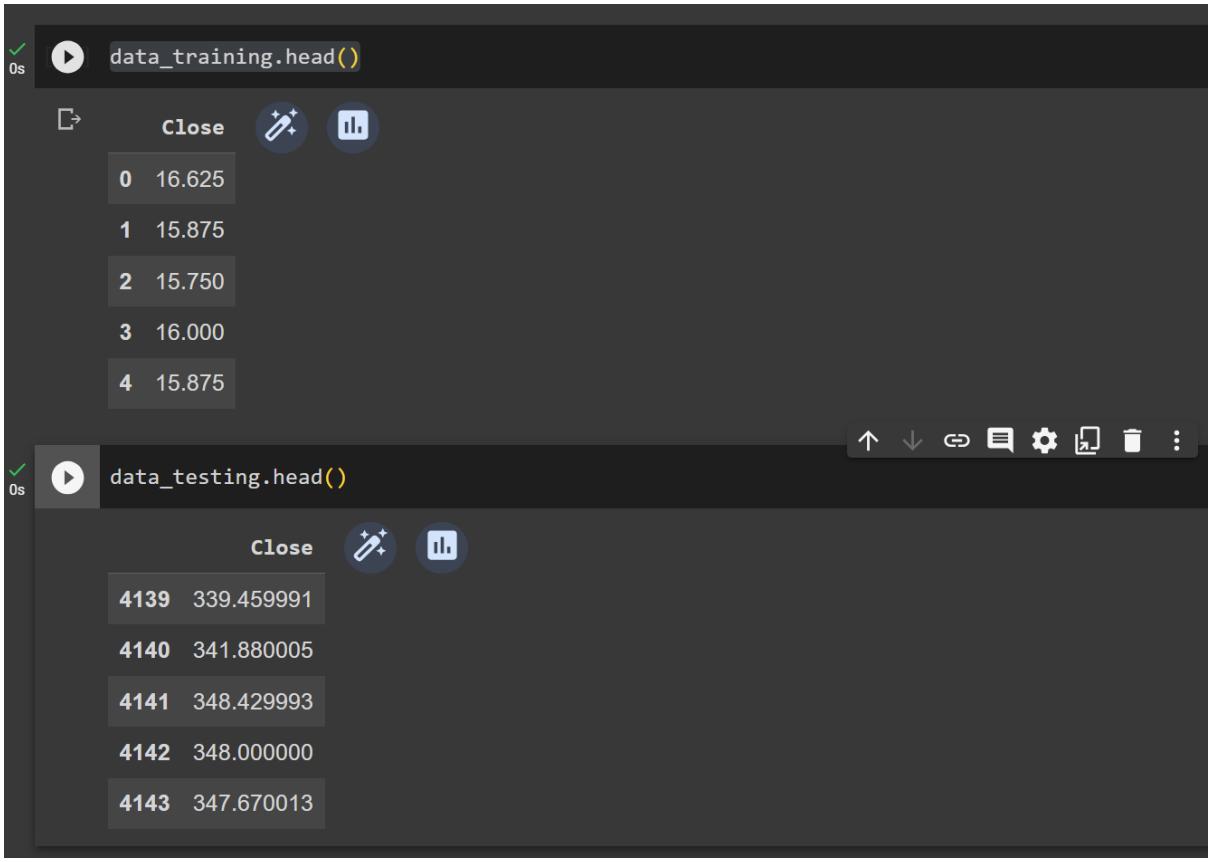
I have run a simple prompt to understand how many rows and columns I have to deal with. I have 5914 rows and 5 columns of data. But my main data is the Closing column so I will only leave the Closing column and will remove everything else.

Splitting data into training and testing

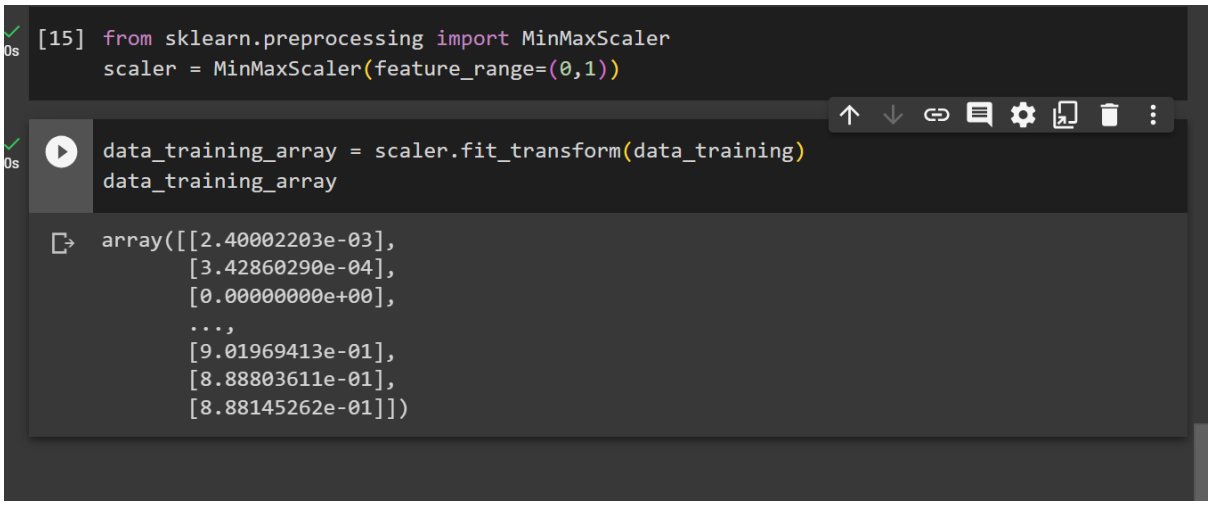
```
data_training = pd.DataFrame(df['Close'][0:int(len(df)*0.7)])
data_testing = pd.DataFrame(df['Close'][int(len(df)*0.7):int(len(df))])

print(data_training.shape)
print(data_testing.shape)
```

```
(4139, 1)
(1775, 1)
```

I have split the data and created dataframes for training and testing. 70% of the whole dataset for training and 30% for testing. I can also see that I have 4139 rows of data for training and 1775 for testing.



I can check that everything works fine and that the training dataframe starts from zero index, while data for testing starts from 4139 index.



I have moved on to scaling down the data, because I cannot provide the data that I have currently to the prediction model. I scaled down the data between 0 and 1. In order to do that I have to import min&max scaler from SKLearn Preprocessing.

I also converted the data into an array.

```
[17] data_training_array.shape

(4139, 1)
```

```
x_train = []
y_train = []

for i in range(100, data_training_array.shape[0]):
  x_train.append(data_training_array[i-100:i])
  y_train.append(data_training_array[i, 0])


x_train, y_train = np.array(x_train), np.array(y_train)
```

I have divided the data into x_train and y_train.

Let's say we take 10 days of data, and we want to predict the stock price for the 11th day. Let's say the value for the first day is £80. Next day it increases to £83, then it decreases to £71, then £76, £81 and so on.

First 10 days: 80, 83, 71, 76, 81, 80, 87, 74, 73, 76 → 79 (11th day value (y_train))

We should also know that the value for the 11th day is dependent on the previous 10 days.

Next: 83, 71, 76, 81, 80, 87, 74, 73, 76, 79 → 82 (12th day value (y_train))

I joined the y_train value and removed the first value of the x_train.

We can say that the first x days is the x_train, and the value of x+1 day is the y_train.

In this example I have taken a 10-steps approach. However, for my model I will be using 100-steps system. Which means that I will have 100 days in my x_train, and 101st day in my y_train.

I have then defined a "for loop", with 100-days steps. Because "i" is starting from 100, I have used "data_training_array[i-100:i]", so x_train will append from zero.

I then converted the x_train and y_train into NumPy arrays, so I can supply the data into my model.

Creating the machine learning model.

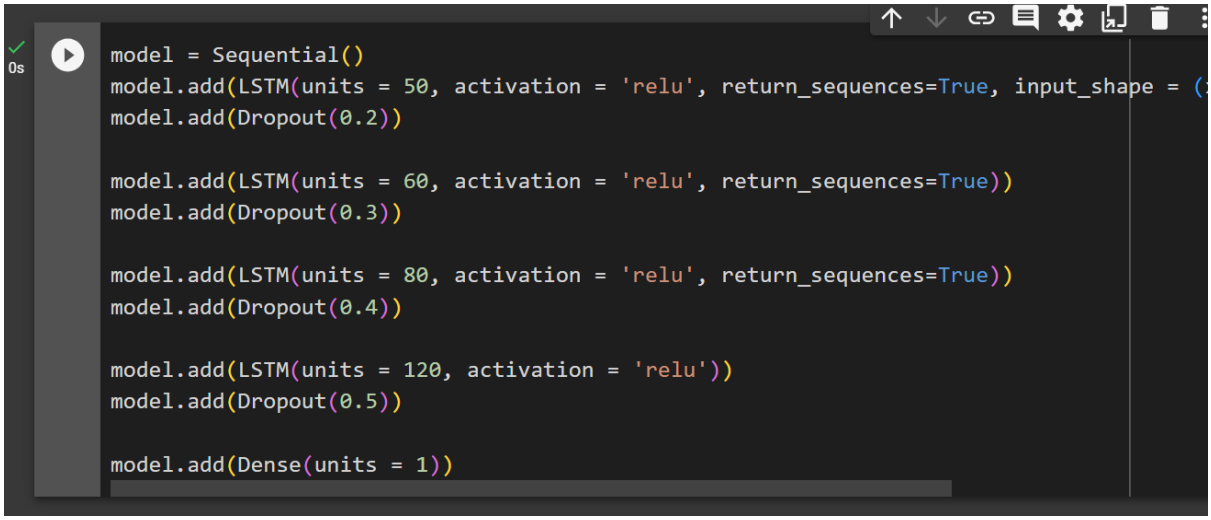
```
Machine learning model

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

Dense layer is the regular deeply connected neural network layer.

The dropout layer uses a method where random neurons are excluded from learning. They are arbitrarily "dropped out".

LSTM stands for Long Short-Term Memory.

Sequential groups a linear stack of layers into a model.



```
model = Sequential()
model.add(LSTM(units = 50, activation = 'relu', return_sequences=True, input_shape = (
model.add(Dropout(0.2))

model.add(LSTM(units = 60, activation = 'relu', return_sequences=True))
model.add(Dropout(0.3))

model.add(LSTM(units = 80, activation = 'relu', return_sequences=True))
model.add(Dropout(0.4))

model.add(LSTM(units = 120, activation = 'relu'))
model.add(Dropout(0.5))

model.add(Dense(units = 1))
```

I have created a sequential model, then added layers.

In Long Short-Term Memory (LSTM) layers, the “units” parameter defines the dimensionality of the output space, simply it refers to the number of LSTM cells to create. Each LSTM cell maintains a cell state and produces an output. An "LSTM with 50 neurons" or an "LSTM with 50 units" means that the dimension of the output vector is 50.

The Rectified Linear Activation Function, or ReLU, is a linear function that outputs the input if it is positive, and zero otherwise.

When return_sequences=True, LSTM produces the hidden state and cell state for every timestep in the input data, thus preserving the temporal relationships between the input timesteps. This is particularly useful when stacking LSTM layers, where the second LSTM layer requires the sequence output from the first layer.

The dropout layer takes the rate as an argument. It represents the fraction of the input units to drop. For example, when I set the rate to 0.3, it means that 30% of the neurons in this layer will be randomly dropped in each epoch.

Then I added the “Dense” layer to connect all the previous layers.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 dropout (Dropout)           (None, 100, 50)           0

 lstm_1 (LSTM)               (None, 100, 60)           26640

 dropout_1 (Dropout)         (None, 100, 60)           0

 lstm_2 (LSTM)               (None, 100, 80)           45120

 dropout_2 (Dropout)         (None, 100, 80)           0

 lstm_3 (LSTM)               (None, 120)               96480

 dropout_3 (Dropout)         (None, 120)               0

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 178,761
Trainable params: 178,761
Non-trainable params: 0
_________________________________________________________________
```

This prompt has shown me the summary of my model.

```
model.compile(optimizer='adam', loss = 'mean_squared_error')
model.fit(x_train, y_train, epochs = 50)
```

```
Epoch 1/50
127/127 [==============================] - 50s 337ms/step - loss: 0.0221
Epoch 2/50
127/127 [==============================] - 43s 339ms/step - loss: 0.0068
Epoch 3/50
127/127 [==============================] - 45s 352ms/step - loss: 0.0056
Epoch 4/50
127/127 [==============================] - 44s 348ms/step - loss: 0.0059
Epoch 5/50
127/127 [==============================] - 41s 325ms/step - loss: 0.0049
Epoch 6/50
127/127 [==============================] - 41s 324ms/step - loss: 0.0043
Epoch 7/50
127/127 [==============================] - 43s 337ms/step - loss: 0.0043
Epoch 8/50
127/127 [==============================] - 42s 333ms/step - loss: 0.0038
Epoch 9/50
127/127 [==============================] - 43s 337ms/step - loss: 0.0039
Epoch 10/50
127/127 [==============================] - 41s 324ms/step - loss: 0.0036
Epoch 11/50
127/127 [==============================] - 43s 336ms/step - loss: 0.0031
Epoch 12/50
 48/127 [==========>...................] - ETA: 27s - loss: 0.0032
```

I have compiled the model.

Adam optimization helps improve the accuracy of neural networks by adjusting learnable parameters.

The Mean Squared Error measures how close a regression line is to a set of data points. It was not new to me because we came across it previously while studying statistics.

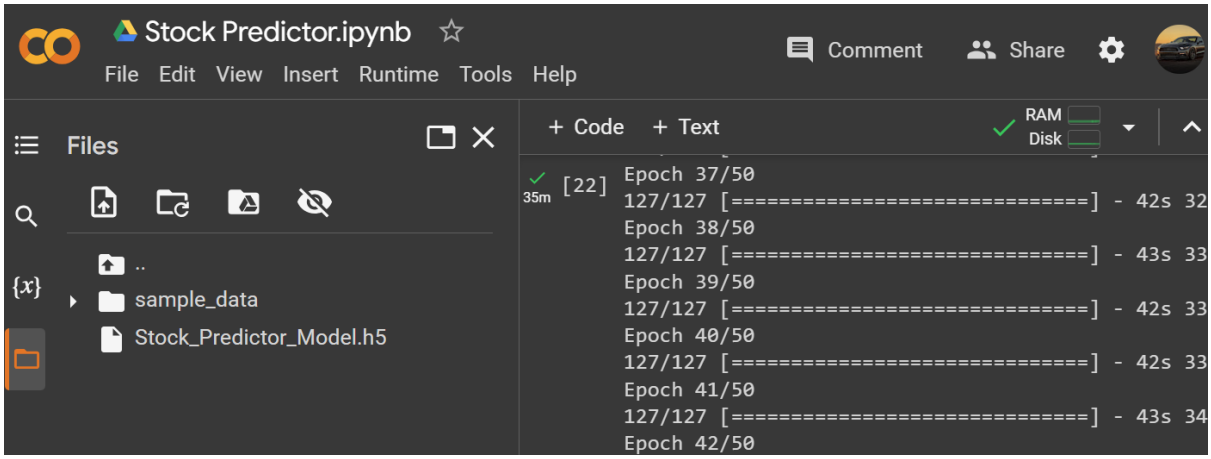
An epoch is a complete iteration through the entire training dataset in one cycle for training the model. I have chosen 50 epochs to have a better accuracy.

I started running the model at 10:53.

```
Epoch 36/50
127/127 [==============================] - 43s 339ms/step - loss: 0.0022
Epoch 37/50
127/127 [==============================] - 42s 329ms/step - loss: 0.0022
Epoch 38/50
127/127 [==============================] - 43s 338ms/step - loss: 0.0022
Epoch 39/50
127/127 [==============================] - 42s 331ms/step - loss: 0.0022
Epoch 40/50
127/127 [==============================] - 42s 330ms/step - loss: 0.0022
Epoch 41/50
127/127 [==============================] - 43s 341ms/step - loss: 0.0021
Epoch 42/50
127/127 [==============================] - 42s 329ms/step - loss: 0.0022
Epoch 43/50
127/127 [==============================] - 41s 323ms/step - loss: 0.0021
Epoch 44/50
127/127 [==============================] - 42s 334ms/step - loss: 0.0021
Epoch 45/50
127/127 [==============================] - 43s 336ms/step - loss: 0.0021
Epoch 46/50
127/127 [==============================] - 43s 337ms/step - loss: 0.0022
Epoch 47/50
127/127 [==============================] - 41s 326ms/step - loss: 0.0019
Epoch 48/50
127/127 [==============================] - 43s 338ms/step - loss: 0.0021
Epoch 49/50
127/127 [==============================] - 42s 326ms/step - loss: 0.0022
Epoch 50/50
127/127 [==============================] - 41s 325ms/step - loss: 0.0020
<keras.callbacks.History at 0x79b3daf7d720>
```

Finished at about 11:23. Total running time - about 30 minutes.

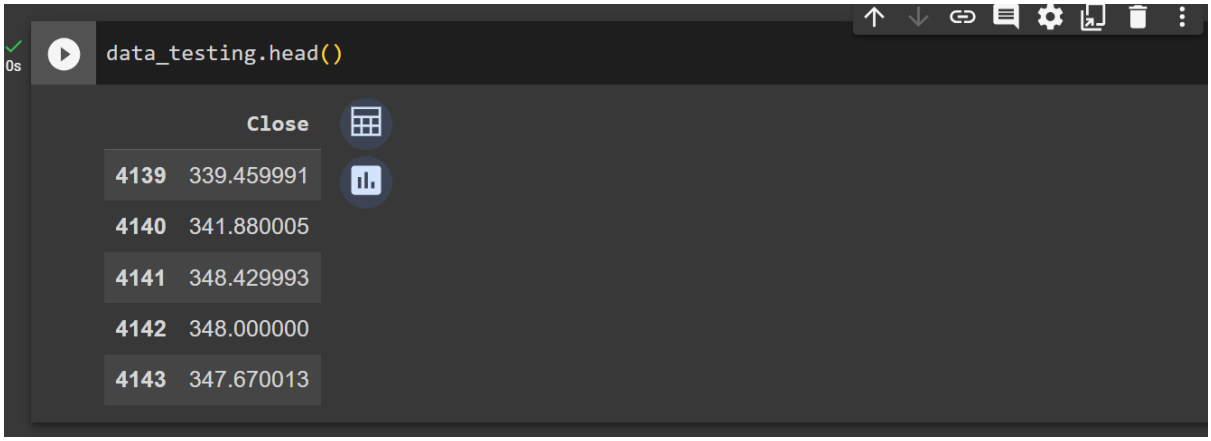
```
model.save('Stock_Predictor_Model.h5')
```



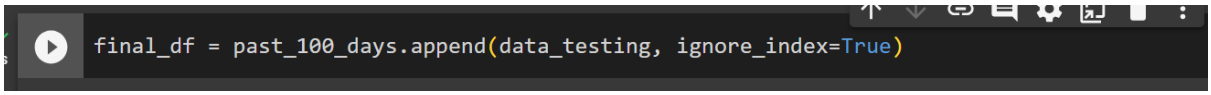
I saved the model so I can use it for my web application backed up by StreamLit.

Preparing the data for the predictions

Now I can move on to predicting the values. In order to do that I will use my testing data.



I need to consider that to predict the value for 4139 index, I need the information from the previous 100 days. In order to get those values, I have to fetch those 100 days from the training data.



Appended the previous 100 days data.

I have to scale down the data again using Min&Max Scaler.

```
input_data = scaler.fit_transform(final_df)
input_data
```

```
array([[0.        ],
       [0.013993  ],
       [0.01818794],
       ...,
       [0.59236989],
       [0.59126982],
       [0.5733752 ]])
```

I can now see that all of the data is scaled down between 0 and 1.

```
input_data.shape
```

```
(1875, 1)
```

I can see that I have 1875 rows of data from 'Close' column.

```
x_test = []
y_test = []

for i in range(100, input_data.shape[0]):
  x_test.append(input_data[i-100:i])
  y_test.append(input_data[i, 0])
```

I have previously defined x_train and y_train. Now I defined the x_test and y_test.

```
x_test, y_test = np.array(x_test), np.array(y_test)
print(x_test.shape)
print(y_test.shape)
```

```
(1775, 100, 1)
(1775,)
```

I converted the data into NumPy array.

Making predictions.

```
y_predicted = model.predict(x_test)
```

```
56/56 [==============================] - 6s 104ms/step
```

Completed the prediction.

```
y_predicted.shape
```

```
(1775, 1)
```

```
y_test
```

```
array([0.07295714, 0.07650675, 0.08611407, ..., 0.59236989, 0.59126982,
       0.5733752 ])
```

Outputted the “y_test”.

```
y_predicted
```

```
array([[0.13432147],
       [0.13350224],
       [0.1323097 ],
       ...,
       [0.54443204],
       [0.54536533],
       [0.54737526]], dtype=float32)
```

Outputted the “y_predicted”.

```
scaler.scale_
```

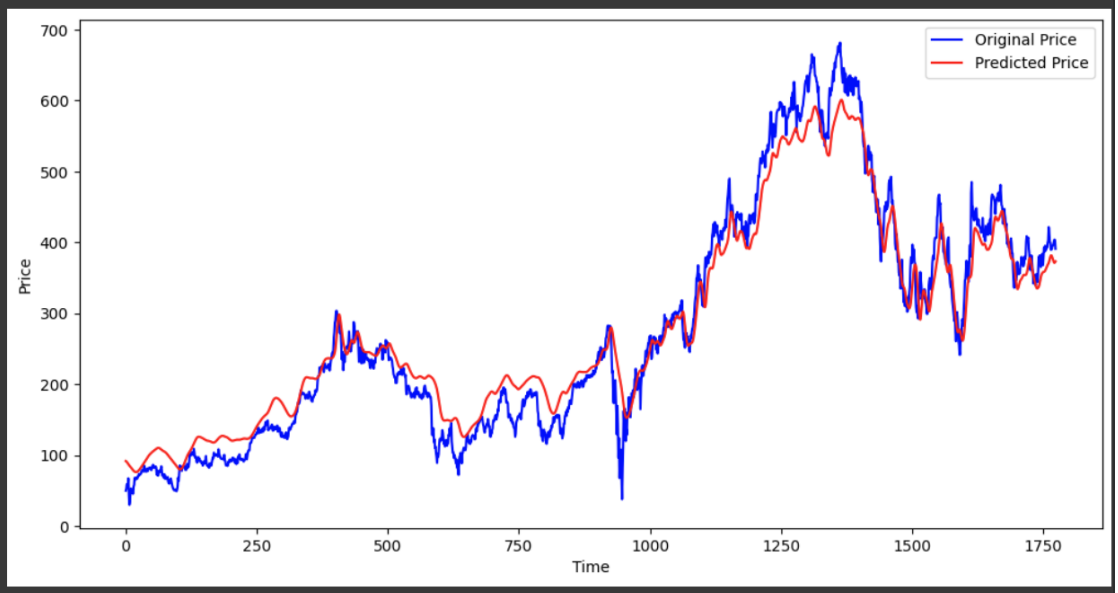
```
array([0.00146677])
```

I have checked the factor by which I scaled down my data.

```
scale_factor = 1/0.00146677
y_predicted = y_predicted * scale_factor
y_test = y_test * scale_factor
```

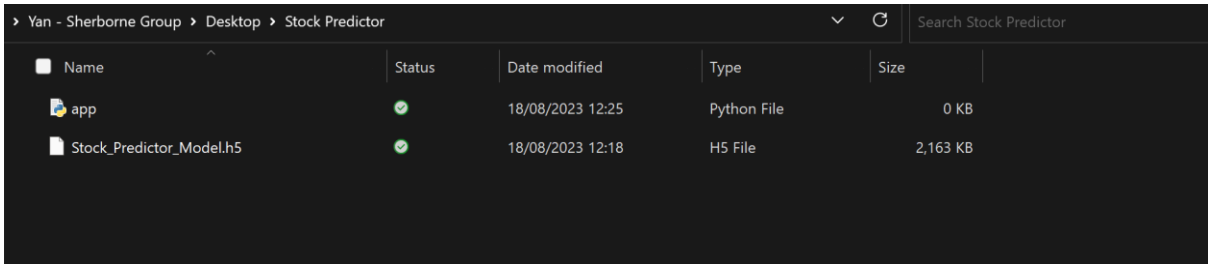
I scaled up my data.

```
plt.figure(figsize=(12,6))
plt.plot(y_test, 'b', label = 'Original Price')
plt.plot(y_predicted, 'r', label = 'Predicted Price')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```



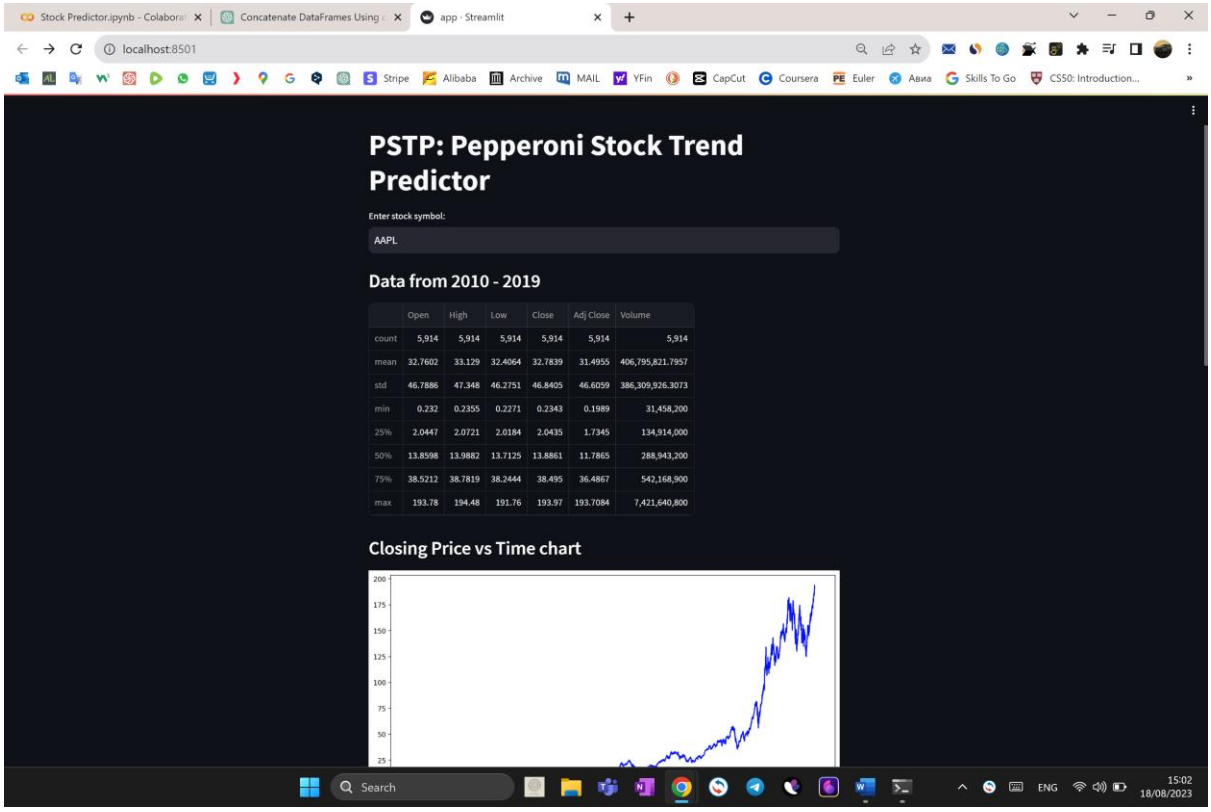
I plotted the original and predicted graph. We can see that the precision of the trend prediction can be considered relatively good.

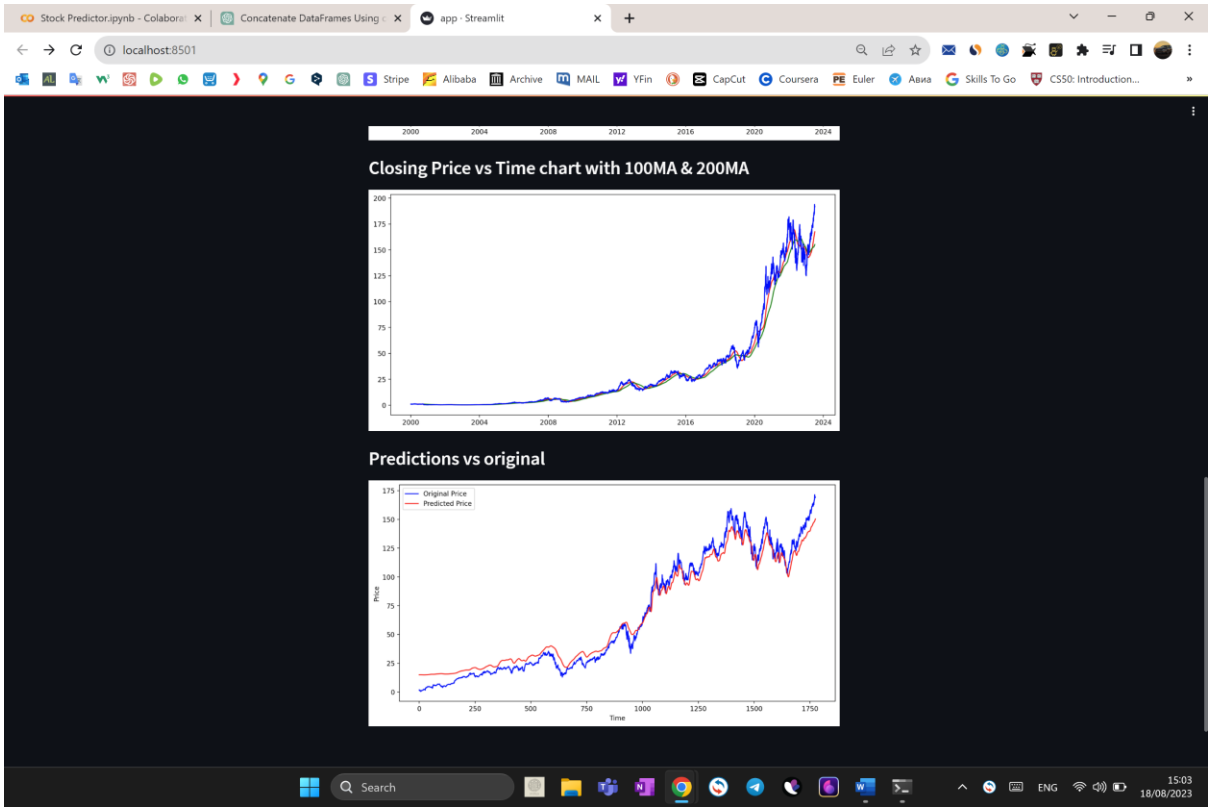
Creating a StreamLit web application.



I have moved back to my computer, and I need to copy the code from Google Colab into the python file.

Final results




Stock Predictor.ipynb - Colabora
Concatenate DataFrames Using
app · Streamlit
localhost:8501
Stripe
Alibaba
Archive
MAIL
YFin
CapCut
Coursera
Euler
Авиа
Skills To Go
CS50: Introduction...
2000
2004
2008
2012
2016
2020
2024
Closing Price vs Time chart with 100MA & 200MA
Predictions vs original
Original Price
Predicted Price
Price
Time
Search
ENG
15:03
18/08/2023

Changes made

I have made several changes to my code to replace both NumPy and pandas with pure Python code for data operations.

1. Fetching Data:

I replaced pandas-based data retrieval with yfinance.

2. Describing Data:

Removed data statistics as there is no equivalent pure Python functionality to replicate pandas' describe method.

3. Splitting Data into Training and Testing:

Replaced pandas DataFrame operations with native Python list operations for slicing.

4. Scaling Function:

I replaced the use of scikit-learn with a custom Python function for data scaling.

5. Testing:

Replaced the use of pandas DataFrames and NumPy arrays with native Python lists for data manipulation within the testing section.

6. Predictions:

Replaced the use of NumPy arrays with native Python lists for data manipulation in the predictions section.

7. Scaling Back to Original Values:

Replaced the use of NumPy arrays with native Python lists for data manipulation when scaling the predicted and original values back to their original scale.

Changes made to graphs

I decided to replace graph plotting algorithms and implement my own algorithms to plot the graphs.

I made use of python turtle and csv libraries.

Plotting graph with turtle

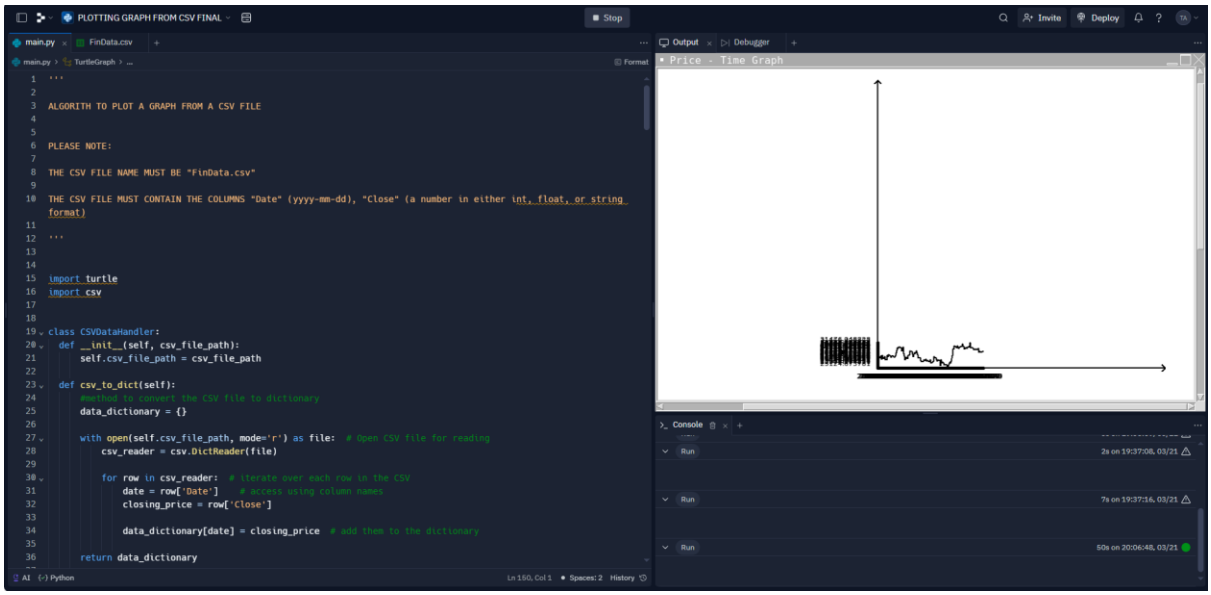
Class CSVDataHandler represents the class of objects responsible for importing and converting financial data from a CSV file into a dictionary, where the keys will be dates and the values represent the closing prices.

```
class TurtleGraph:
    # a class to represent and plot a graph using turtle graphics
    def __init__(self, screen_size_x=1000, screen_size_y=1000, title="Price - Time Graph"):
        # initialise the graph with a screen size and a title
        self.screen_size_x = screen_size_x
        self.screen_size_y = screen_size_y
        self.screen = self.initialise_screen()
        self.turtle = self.initialise_turtle()
        self.screen.title(title)

    def initialise_screen(self):
        # initialise the screen
        screen = turtle.Screen()
        screen.screensize(self.screen_size_x, self.screen_size_y)
        return screen

    def initialise_turtle(self):
        t = turtle.Turtle()
        t.speed(0) # fastest drawing speed
        t.width(2) # line width
        return t

    def find_axis_lenghts(self):
      # the subrouine name tells itself what it does
      return [self.screen_size_x * 5 / 11, self.screen_size_y * 5 / 11]

    def draw_axis(self):
        # Calculate axis lengths and positions
        axis_length_x, axis_length_y = self.find_axis_lenghts()
        arrow_size = 0.005  # Relative size of the arrow head

        # Draw X and Y axes with arrows
        self.draw_line_with_arrows(0, 0, axis_length_x, 0, arrow_size)
        self.draw_line_with_arrows(0, 0, 0, axis_length_y, arrow_size, vertical=True)
```

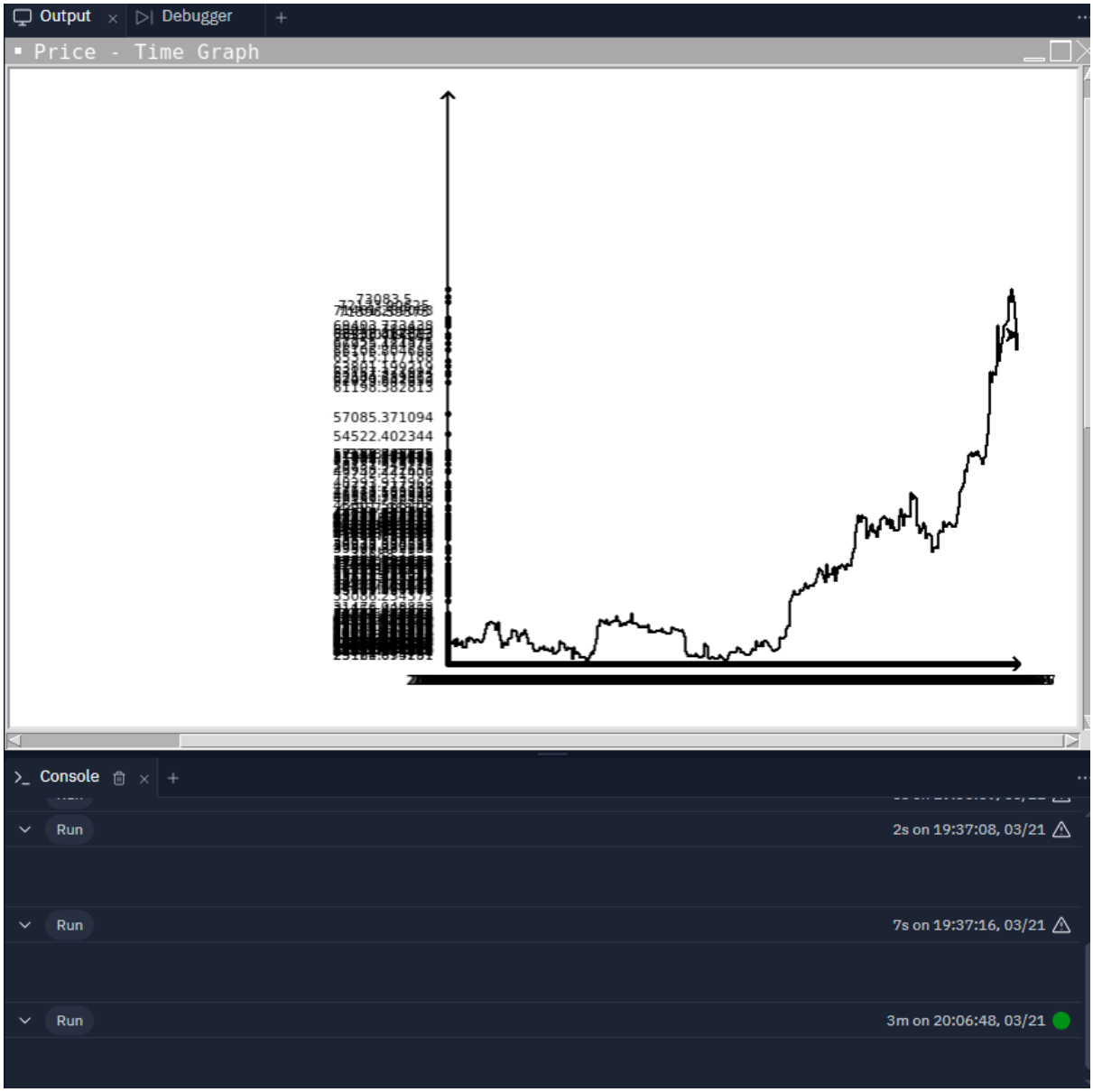
TurtleGraph class: initialisation creates a graphical window of the given size with the given title. The class provides functionalities of initialising the turtle, calculating the axis lengths and drawing axes with arrowheads.

It finds the relative distance of the different data points belonging to the x axis (dates) and rescales the y-axis data (closing prices) with respect to the minimum and maximum values. I used python dictionaries to store the data from csv files.

The program then instantiates a TurtleGraph, draws the axes, reads from the financial data on a CSV file named “FinData.csv” using the CSVDataHandler, then plots this read data on the graph.



After the program finished execution:



Rotating the x-axis labels

The labels on the x and y axis were not clear, therefore I needed to make them rotated at a right angle to the axis.

Defining the labels_x_axis class:

The programme allows drawing text in any direction, which standard turtle text does not support.

It contains two methods:

write_tilted_text: rotates the text to a specified angle when writing it on the screen.

tilted_text: implementation to display text at an exact angle and position

It scales the position of the text using `xscale' and `yscale' and uses mapping from a provided point to `text_rotating_point'.

Monkey Patching:

Monkey patching is a mechanism for adding, modifying, or extending behavior of libraries or classes at runtime. It was implemented on the write method to write tilted text.

Demonstration Using Turtle Graphics:

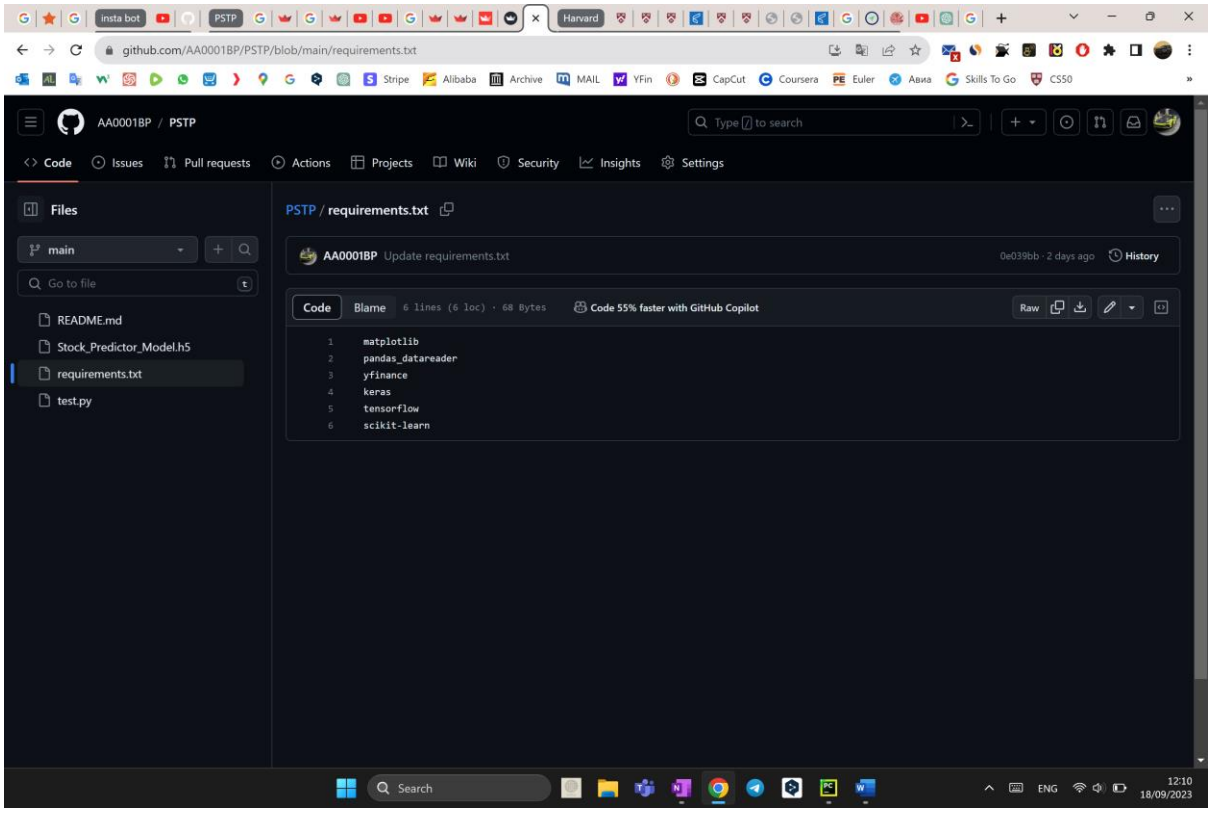
An instance of turtle.Turtle() is created, and its speed, color, and the background color of the screen are set.

The turtle's heading is set to 270 degrees (pointing directly down), and it moves forward by 100 units.

Then, it writes with the set font, alignment, and angle to 'abc'. The turtle then moves backward by 100 units, returning to its starting position.

Server Requirements

“requirements.txt” is a txt file uploaded on a server that contains a list of packages and libraries needed to be installed before running the program.

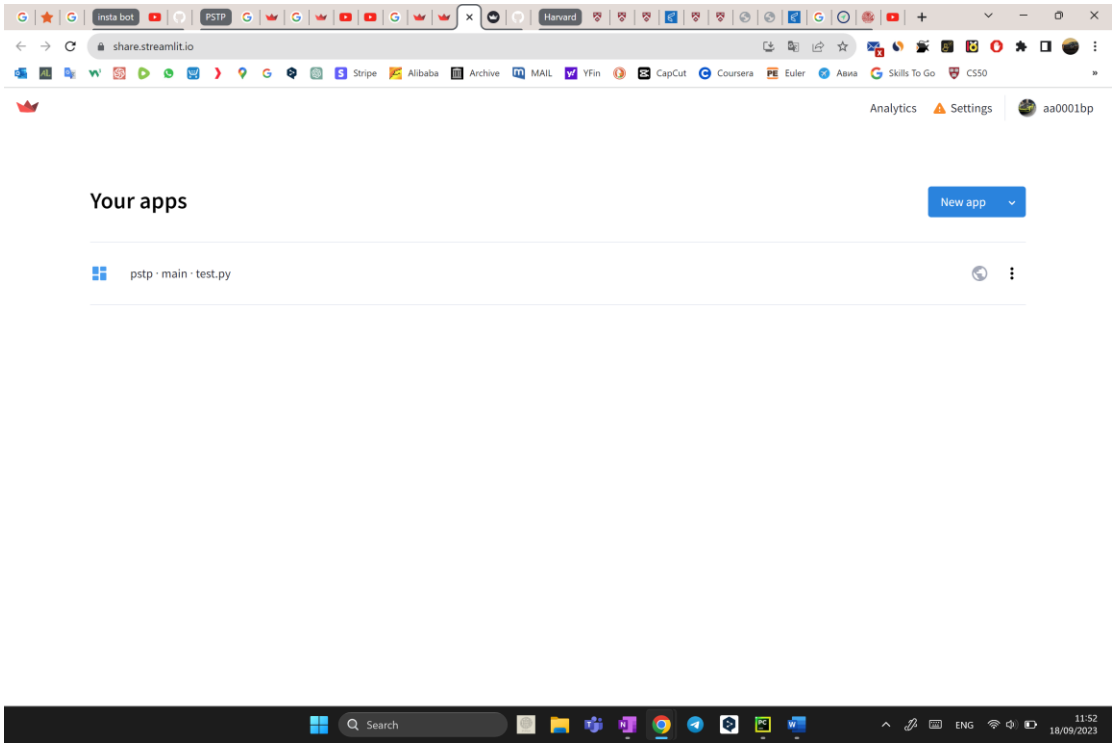


Uploading on a server

Before uploading on a server, I needed to run the terminal using special prompts in order to start the application. This way the app would only be accessible on the home network.

Streamlit, the tool that I used to create the web application, also provides free cloud hosting with custom domain name.

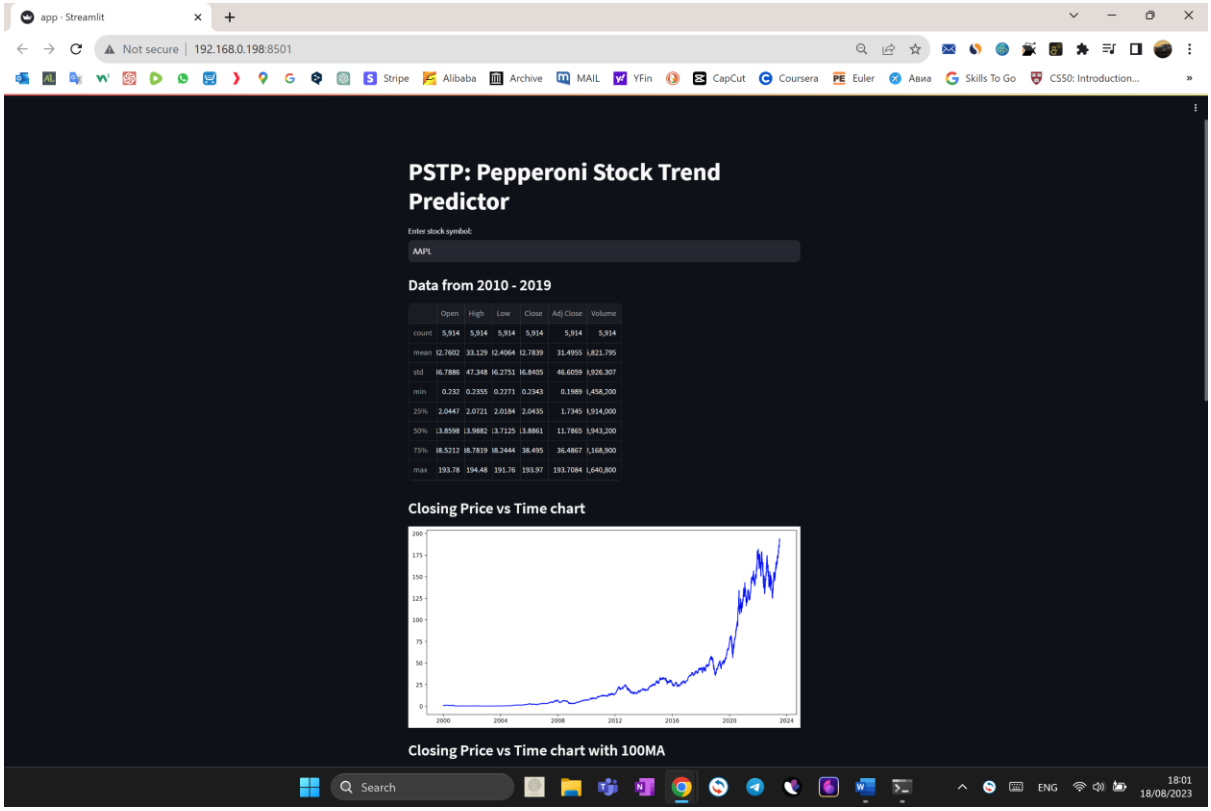
So, I created a repository on GitHub, uploaded my code there with “requirements.txt” (to install all the extra modules). Then I registered on Streamlit Cloud and connected it with my GitHub repository.

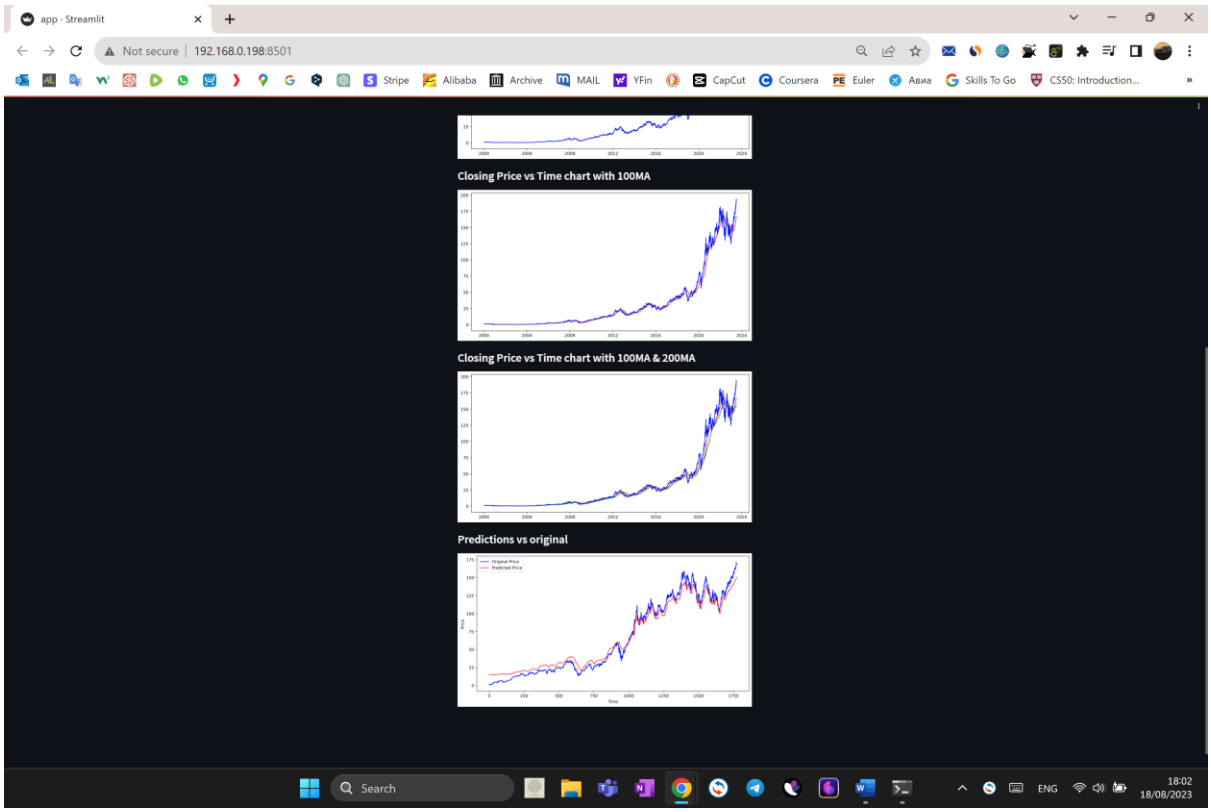


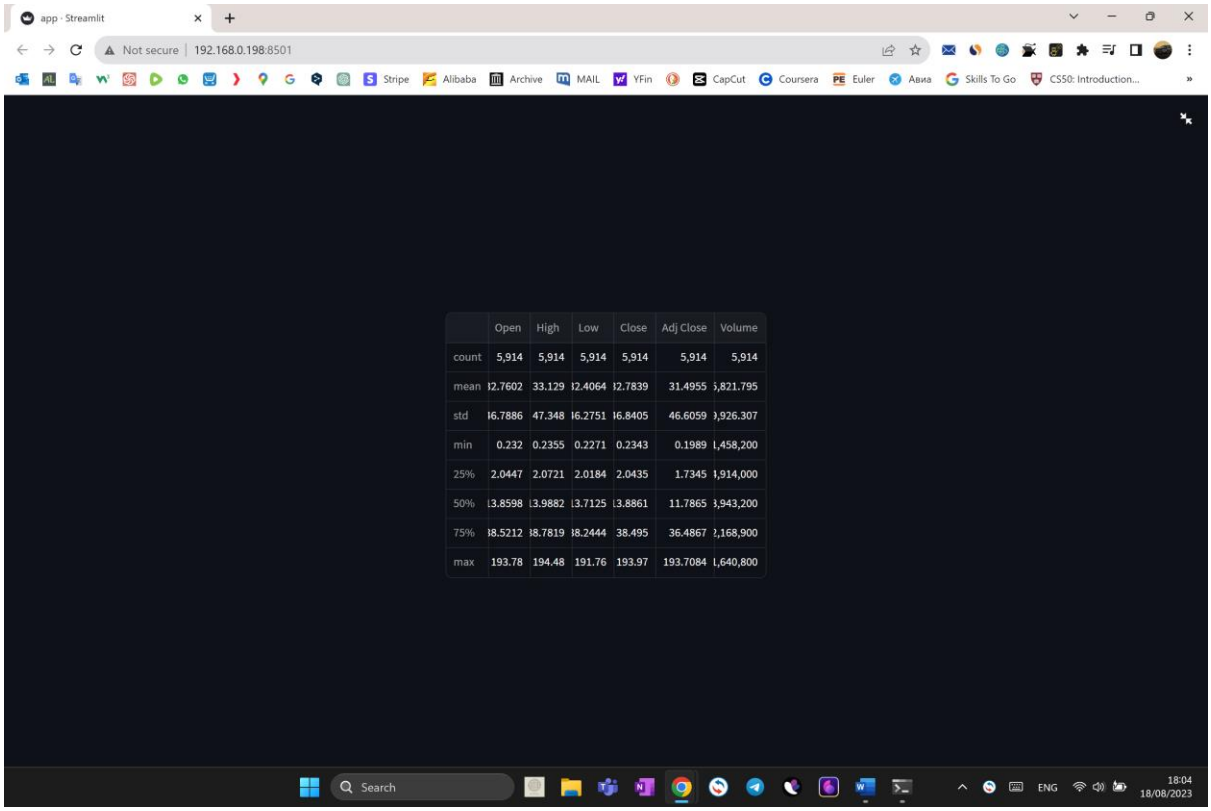
Testing

Input

Real stocks





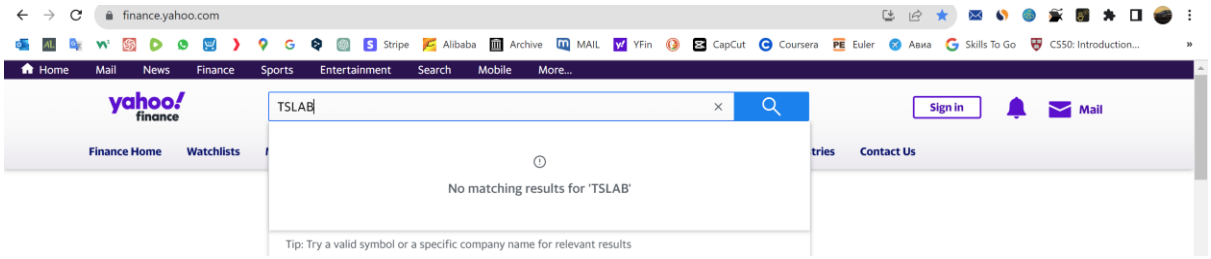


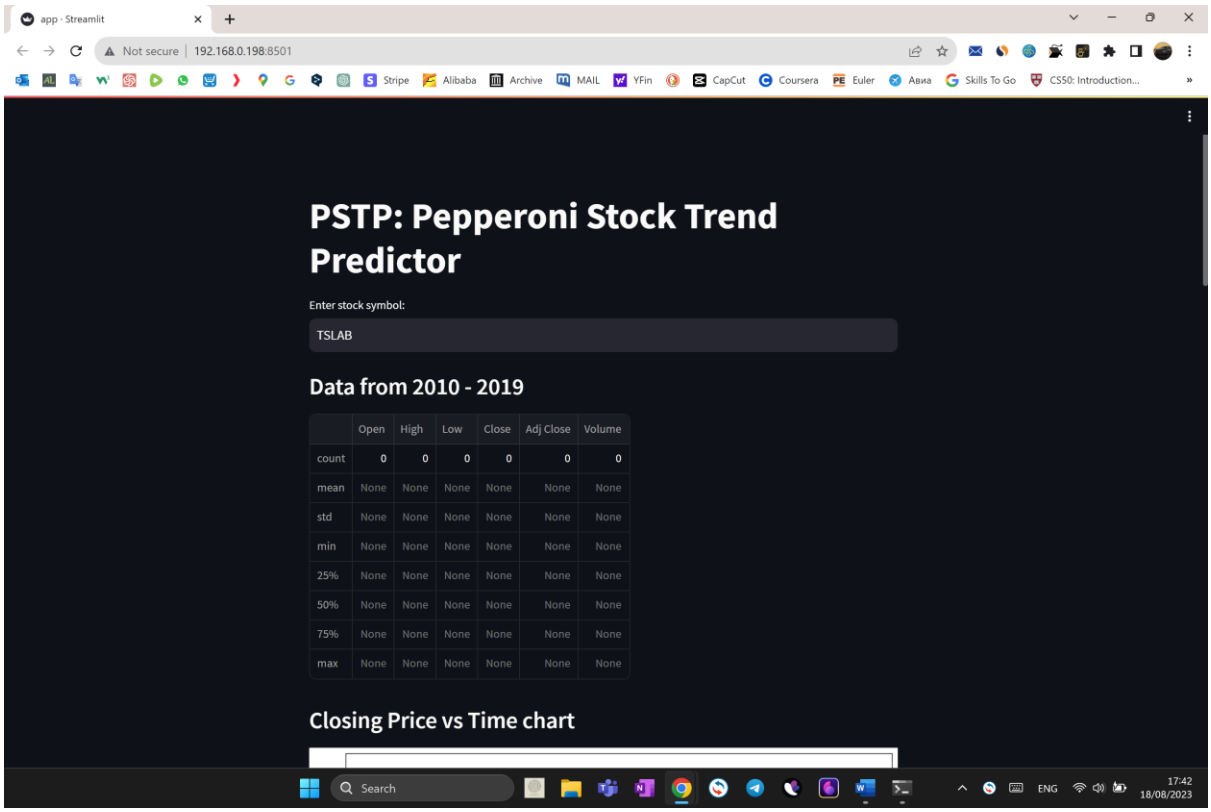
	Open	High	Low	Close	Adj Close	Volume
count	5,914	5,914	5,914	5,914	5,914	5,914
mean	12.7602	33.129	12.4064	12.7839	31.4955	3,821.795
std	16.7886	47.348	16.2751	16.8405	46.6059	9,926.307
min	0.232	0.2355	0.2271	0.2343	0.1989	1,458,200
25%	2.0447	2.0721	2.0184	2.0435	1.7345	1,914,000
50%	13.8598	13.9882	13.7125	13.8861	11.7865	3,943,200
75%	18.5212	18.7819	18.2444	38.495	36.4867	2,168,900
max	193.78	194.48	191.76	193.97	193.7084	1,640,800

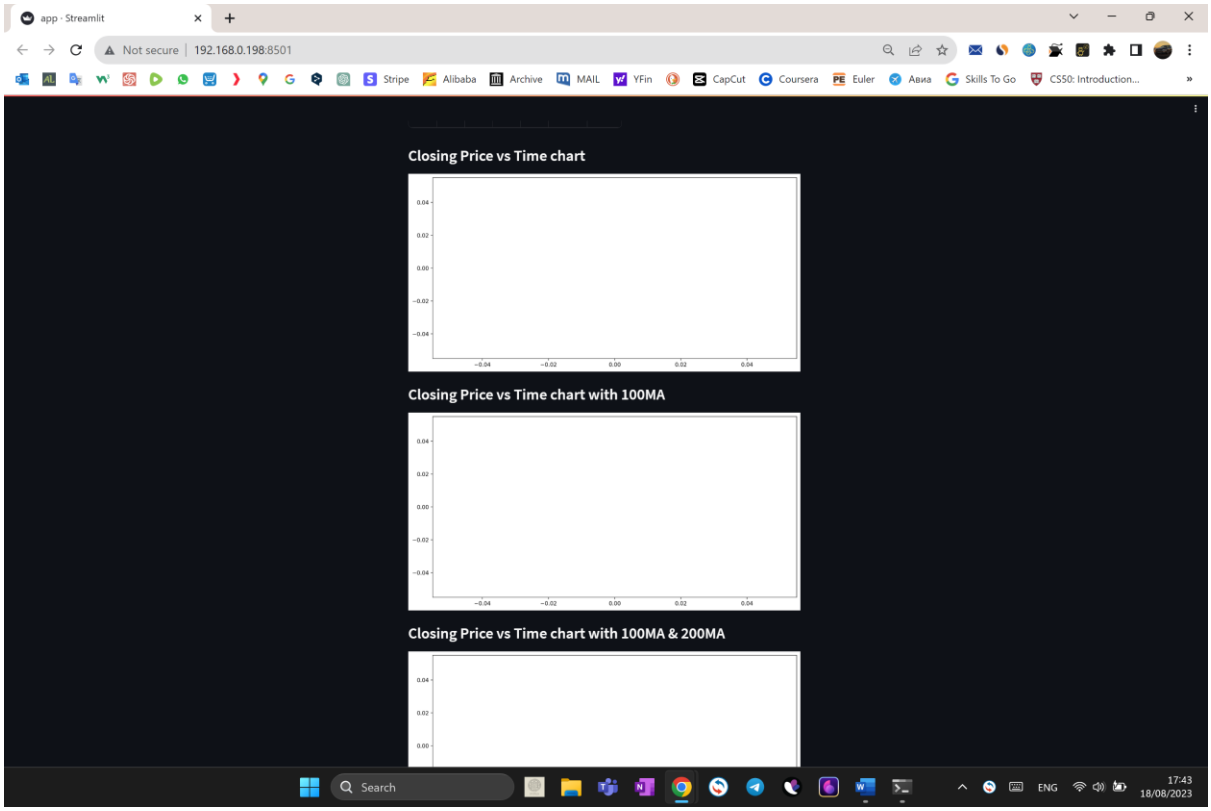
The programme works really well showing all the data that is needed. Furthermore, it is possible to scale up and download the table and the graphs.

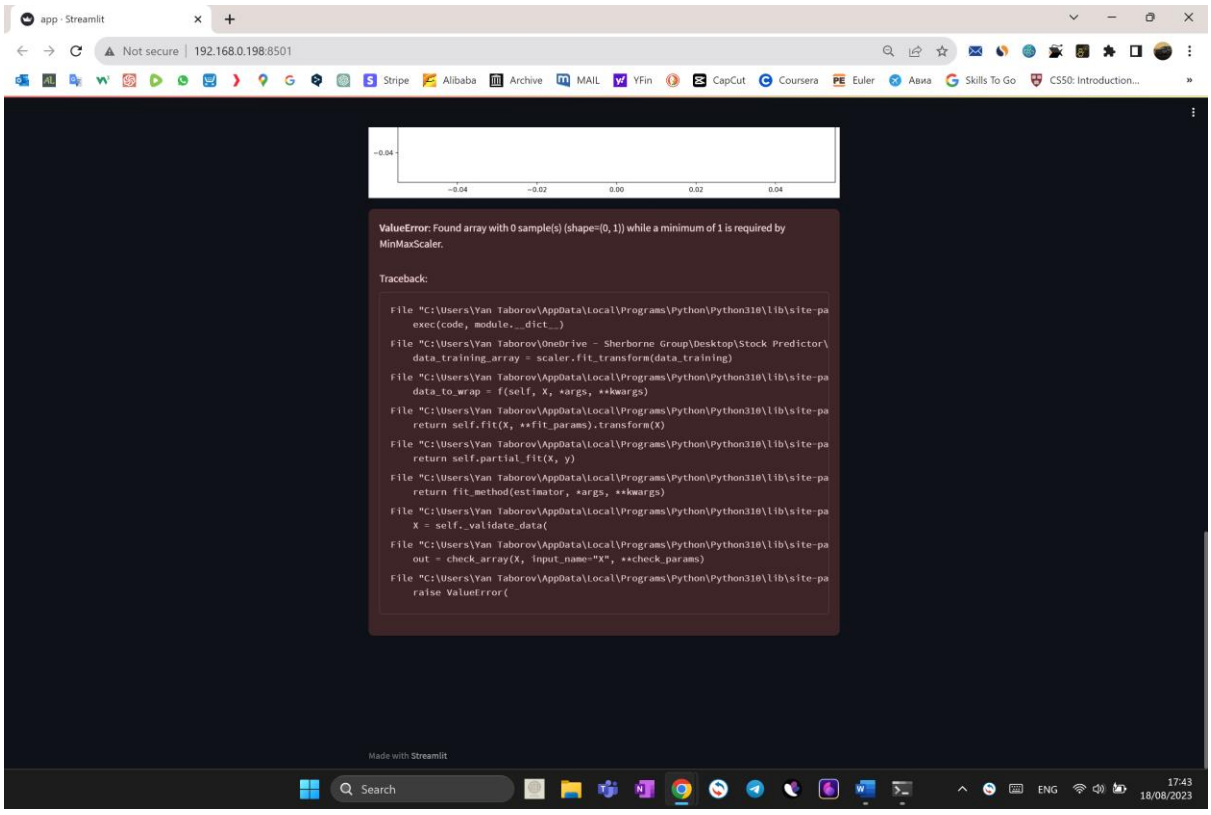
Non-existing stock

The next thing that was tested is the input of the stock symbol that does not exist. I went onto Yahoo Finance to make sure that the stock symbol is actually not real. I have chosen “TSLAB”.





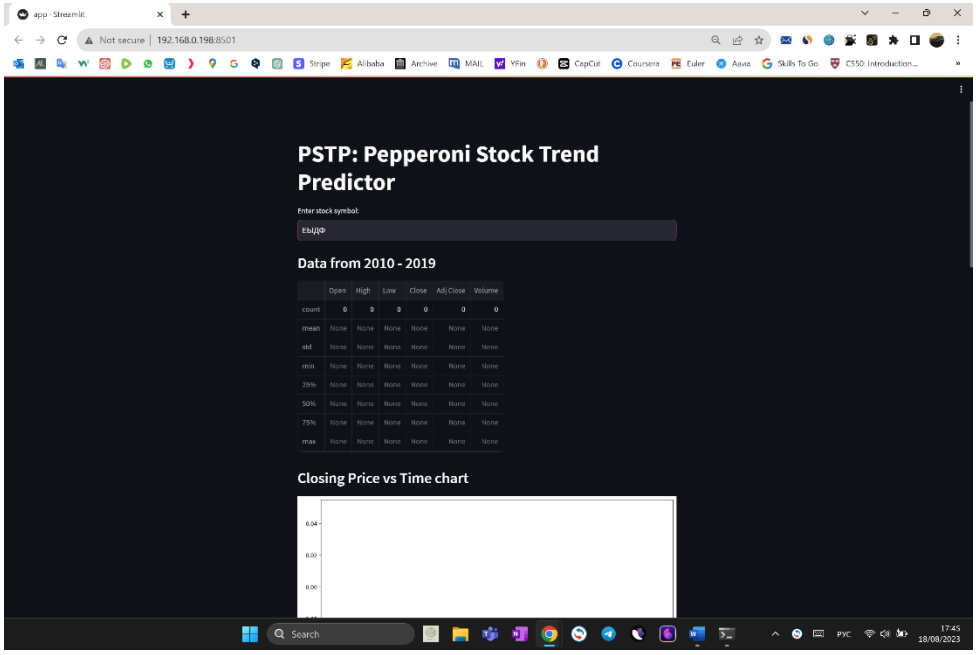




As we can see, no data is present and I received an error *"ValueError: Found array with 0 sample(s) (shape=(0, 1)) while a minimum of 1 is required by MinMaxScaler",* which means that no data was passed to the scaler.

Cyrillic input

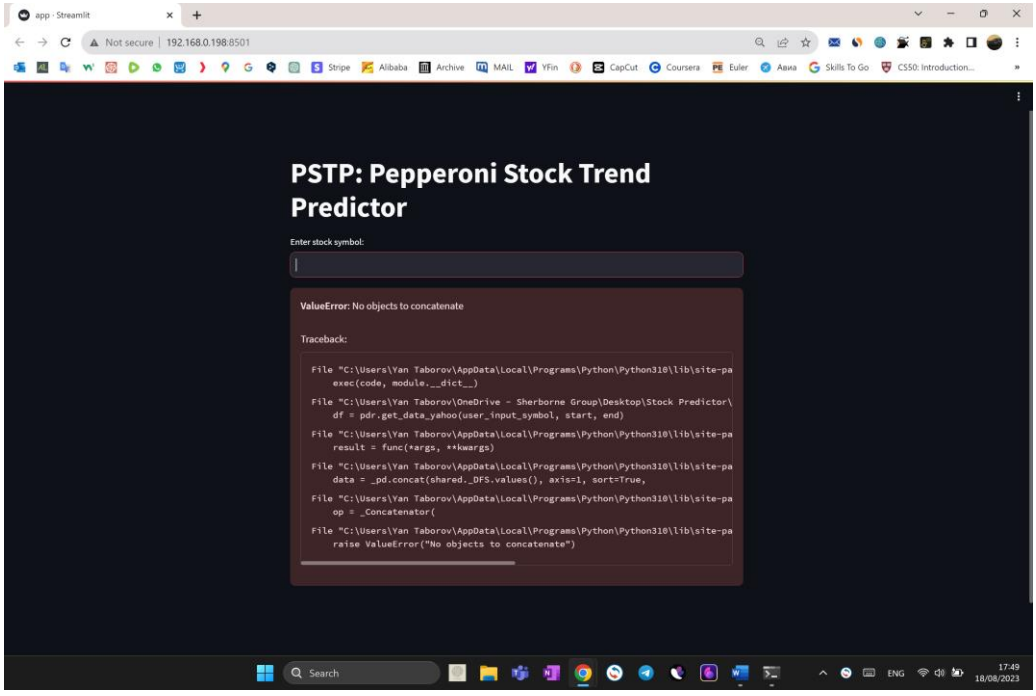
Then I decided to test it using a Cyrillic-letters input.



I received no data and the same error.

Blank input

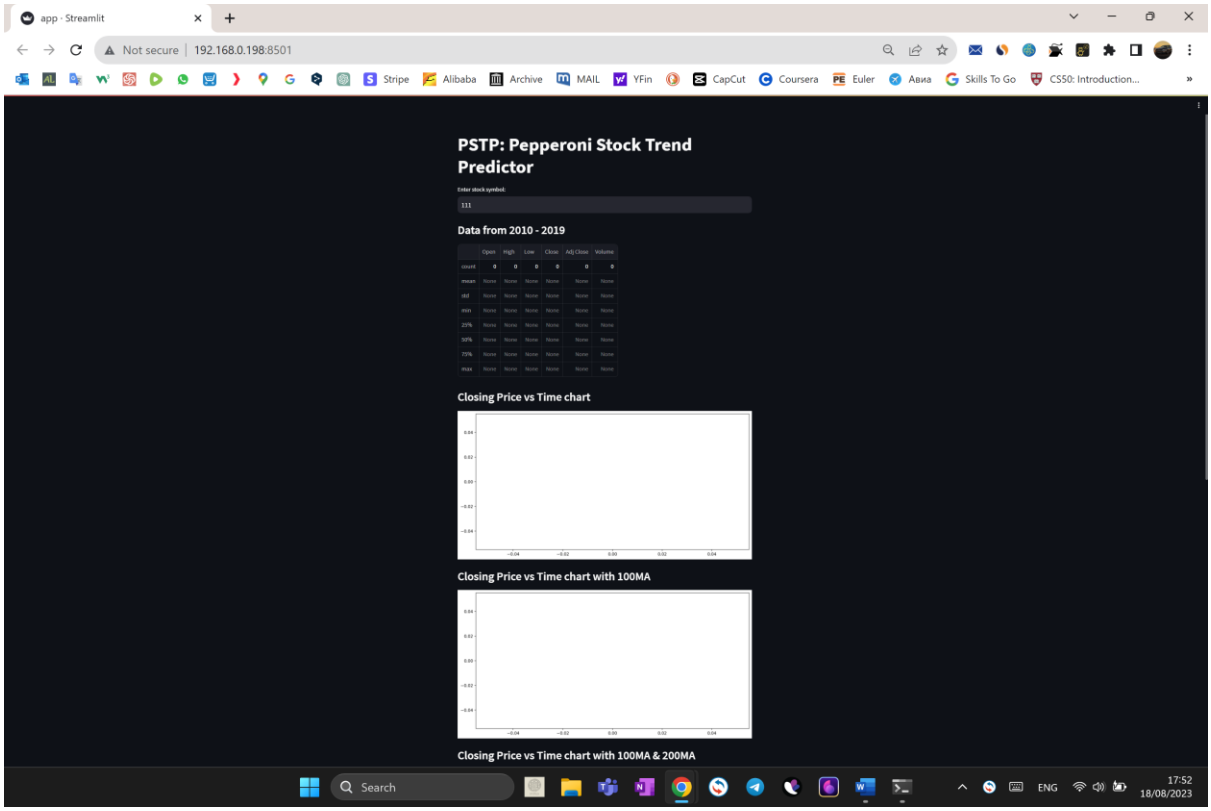
I inputted nothing and pressed “enter”.



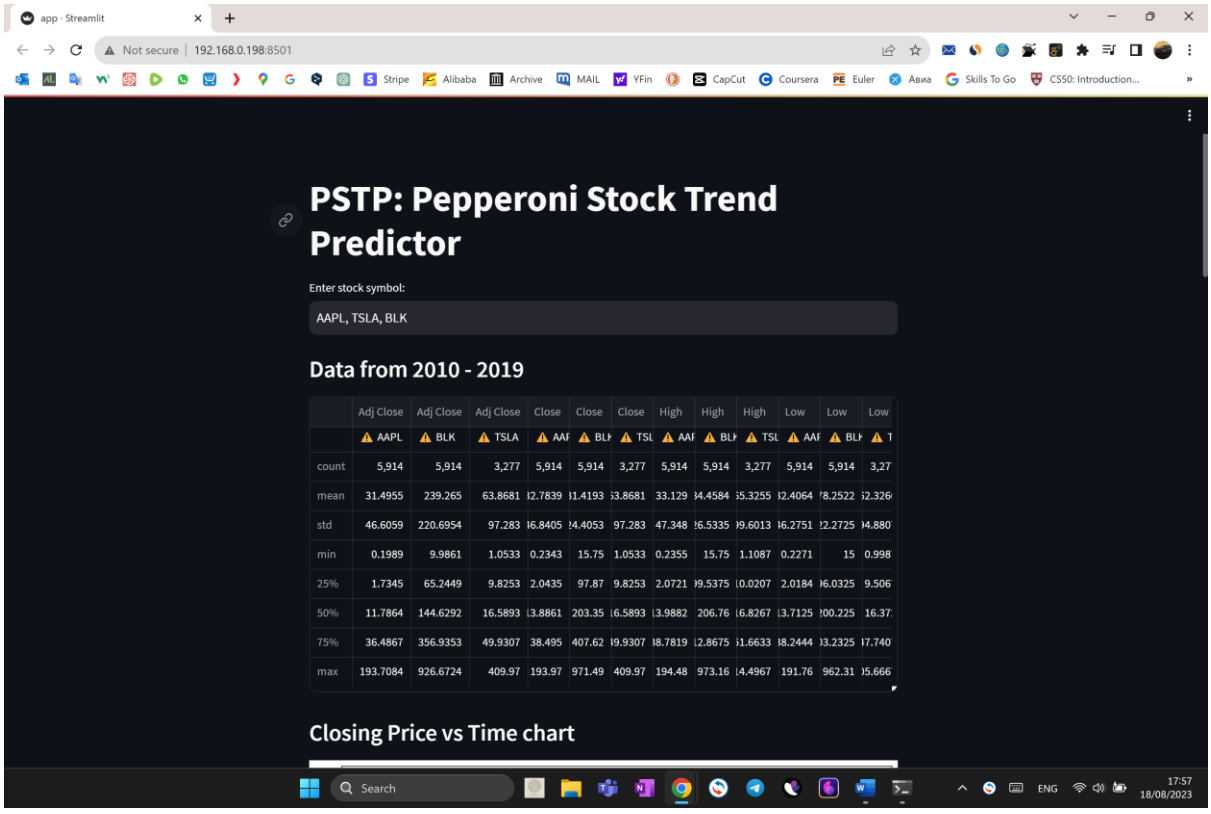
I received “ValueError: No objects to concatenate” which means that no objects were passed to the programme.

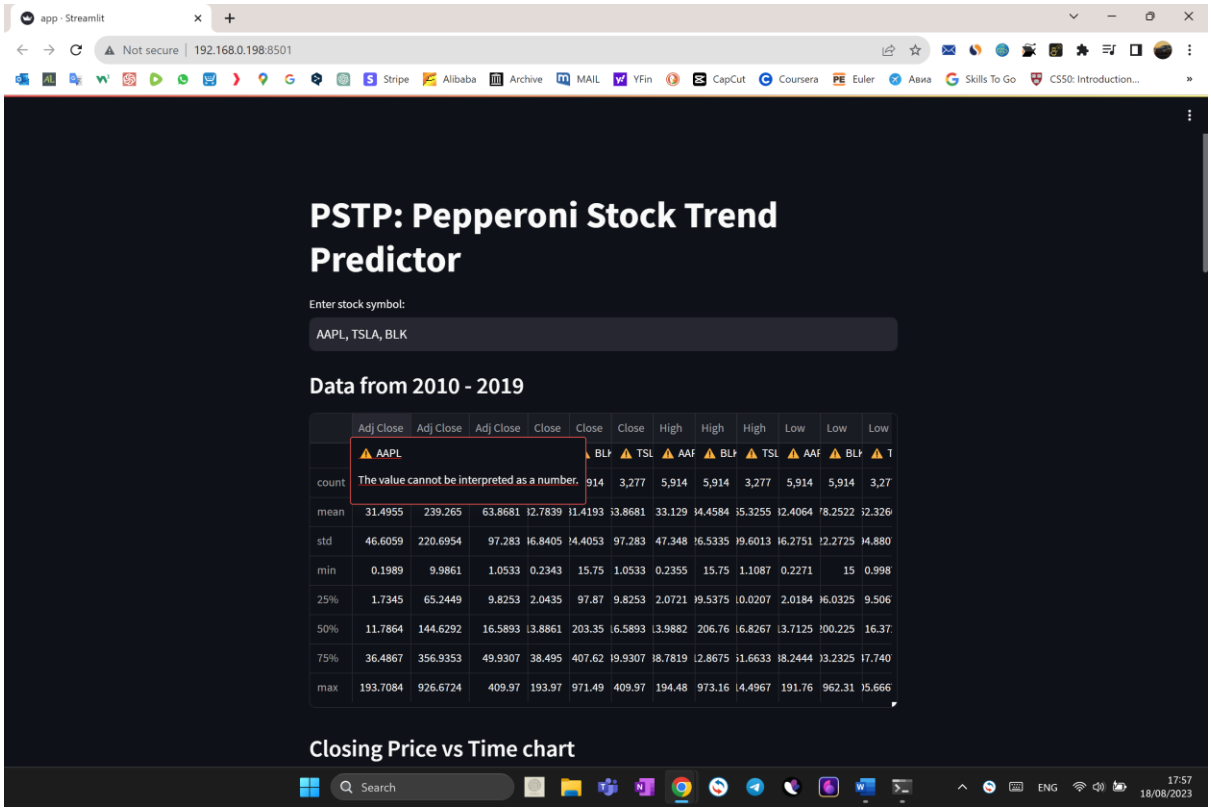
Only numbers input

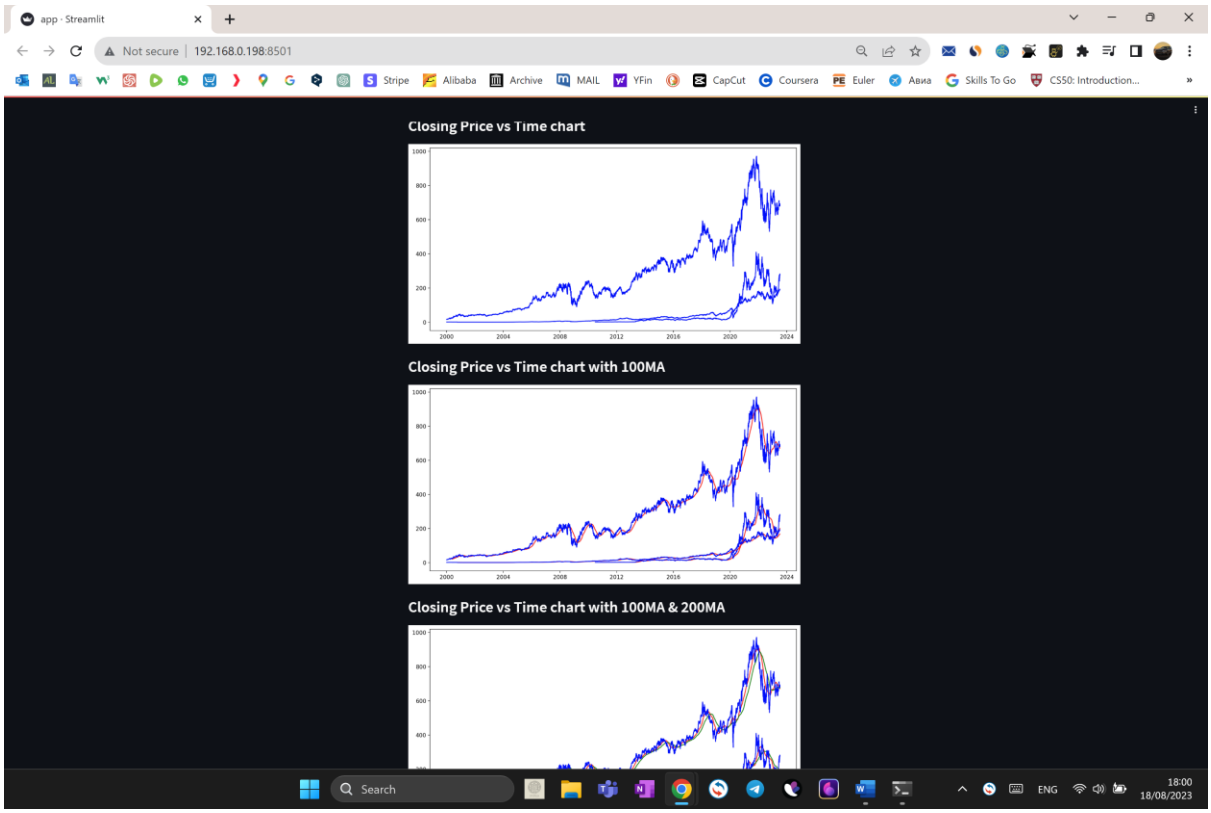
I have passed “111” to the model and received no data.



Input consisting of multiple companies







The programme works well when inputting multiple companies at the same time which makes it really useful to compare them. I received a warning that the values in the first row cannot be interpreted as numbers, which is absolutely fine, because they are the names of the stocks of the companies.

Real-time testing for a maximum number of concurrent users

Preparation

In order to test the performance of the web-application, I needed a large number of clients using my web-application at the same time. I did not have an opportunity to have hundreds of real users, therefore I used automatization, as I have experience implementing it from my previous projects.

I used an anti-detect browser called Dolphin Anty, which is often used for web-application testing.

Note:

An anti-detect browser is used for testing and lets users create multiple browsing environments by manufacturing different digital fingerprints for each of the profiles the user makes.

One of the most important features of an anti-detect browser is it allows all browsing environments to have different IP addresses. In order to use it, I needed to purchase proxies for my bot-customers. I purchased 10 proxies for 50 clients (one IP address for each five clients), which I loaded into four different computers which I borrowed from my friends and family, so that computers can easily cope with the workload. I could also run all the testing process on a virtual server, but in this case, I decided to use real computers.

Note:

Proxy functions similarly to VPN (virtual private network) and can be used in anti-detect browsers.

Browser Automation:

Browser profiles based on Anty engine can be launched with DevTools Protocol enabled.

This means that once the profile is started, you can connect to it via the port generated at startup and get the ability to automate the browser using tools such as Python Selenium.

Start a profile via API

Request URL

To start a browser profile via API, I need to send a GET request to
http://localhost:3001/v1.0/browser_profiles/PROFILE_ID/start?automation=1

where PROFILE_ID is the ID of the browser profile; automation=1 - mandatory parameter.

Local API works only when Dolphin Anty is running on 3001 port.

API response

If the profile was launched successfully, the response will come with the following structure:

```
{
  "success": true,
  "automation": {
    "port": 50568,
    "wsEndpoint": "/devtools/browser/c71c1a9d-f07c-4dd9-84a9-53a4c6df9969"
  }
}
```


I wrote a simple python script, which utilises Requests and Selenium libraries, connects to the ports of the profiles and makes the process of running the testing software automated. It allows me to start testing with one button with no need to configurate all the bot-clients.

Results:

The automated testing using 50 different clients has shown no significant degradation in the performance of the website. The website functioned well with no delays.

Testing with more than half a hundred different users is not viable in this case, as the web-application would be run on different servers specifically for each company which uses the product and, at the current stage, no more than 50 members of staff are expected to use the product simultaneously. This would be done to minimise the chances of data leaks and other security reasons. In addition, if the server would be down, only one company would be affected.

Further Improvements:

After reviewing my project, I have decided to implement some further improvements for a better functionality and user experience of the programme.

Dates

I have added input fields for the starting date and for the end date. This way the user can define what specific dates they want to look at.

PSTP: Pepperoni Stock Trend Predictor

To restart - press R

Enter stock symbol:

AAPL

Enter starting date in a format "yyyy-mm-dd":

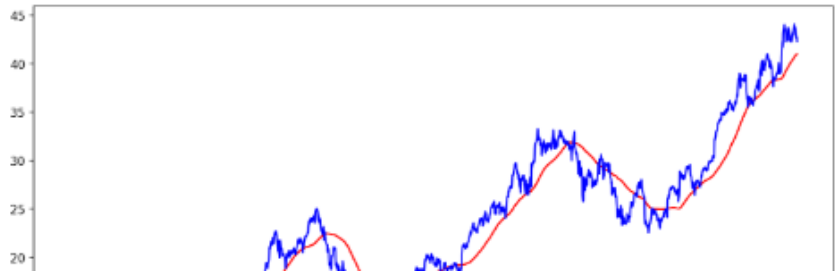
2010-01-01

Enter end date in a format "yyyy-mm-dd":

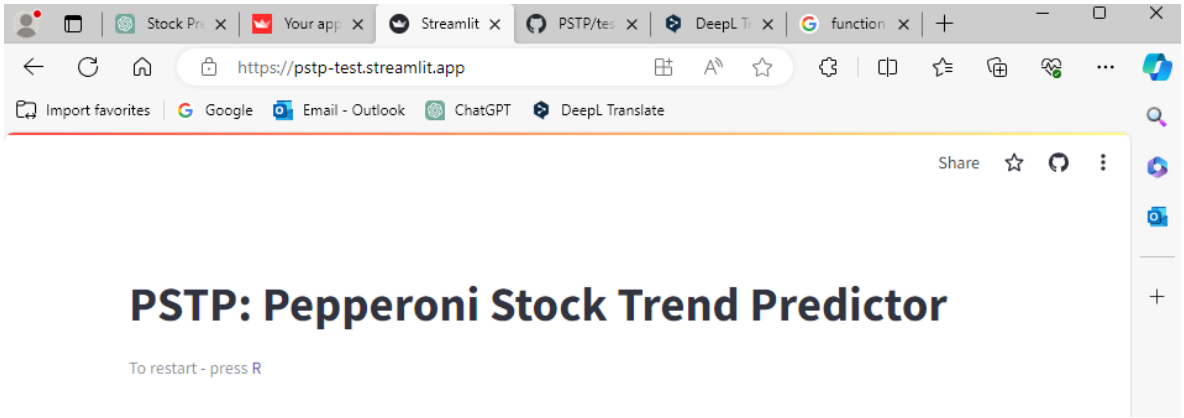
2018-01-01

Closing Price in Dollars vs Time chart

Closing Price vs Time chart with 100MA



Customised domain name and “Refresh” button (“To restart – press R”):



News aggregator

Stock markets are sometimes unpredictable due to the external circumstances. News related to the company would be beneficial for the customer to make informed decisions.

Therefore, I have developed a simple news aggregator built with the Streamlit library and the News API.

Breakdown of the code:

1. Import necessary libraries:

```
#--- NEWS AGGREGATOR ---#

from newsapi import NewsApiClient
import streamlit as st
```

2. Set up the News API client with an API key:

```
api = NewsApiClient(api_key='9efcc41031a041c49a6b9f72df29194d')
```

3. Created a Streamlit app with a header and a text input for the user to enter the name of a company:

```
st.header("News related to the company")
user_input_stock = st.text_input('Enter the name of the company: ')
```

4. Use a selectbox to allow the user to choose the number of articles they want to see:

```
number_of_articles = st.selectbox("How many articles would you like to see", (3, 5, 10, 20, 50), label_visibility="visible")
```

5. Make a request to the News API to get articles related to the specified company:

```
response_data = api.get_everything(qintitle=user_input_stock, sources="the-wall-street-journal,
```

```
fortune, australian-financial-review, bloomberg, business-insider, business-insider-uk,

financial-post", sort_by="popularity", language="en", page_size=number_of_articles)
```

6. Iterate through the articles in the API response and display each article's title, source name, publication date, and a button to open the article:

```
for article in response_data['articles']:
    st.subheader(article['title'])
    st.caption(article['source']['name'] + "  •  " + article['publishedAt'][:10])
    st.markdown(f'''
<a href={article['url']}><button style="background-color:#FF33E9;">Open the article</button></a>
''',
unsafe_allow_html=True)
```

7. Add a caption with a reminder about the nature of the data and its potential inaccuracies:

> Pepperoni Stock Trend Predictor would like to remind you that the data contained on this website and via API might not necessarily be real-time nor accurate. All prices may differ from the actual market price, meaning prices are indicative and not appropriate for trading purposes. Therefore, Pepperoni Stock Trend Predictor does not bear any responsibility for any trading losses user might incur as a result of using this data. Pepperoni Stock Trend Predictor or anyone involved with it will not accept any liability for loss or damage as a result of reliance on the information contained within this website. Please be fully informed regarding the risks and costs associated with trading the financial markets. Pepperoni Stock Trend Predictor does not give any warranties (including, without limitation, as to merchantability or fitness for a particular purpose or use). Please note that the historical returns summaries provided on this website are based on past prices and are not a guarantee or indication of future returns. It is important to understand that past performance is not necessarily indicative of future results, and that investing carries inherent risks. It is important to carefully consider your own financial situation and risk tolerance before making any investment decisions.

Note: The disclaimer in the caption provides information about the data and warns users about the potential risks associated with trading based on the information provided by the aggregator. The styling of the "Open the article" button is defined using HTML and inline CSS within the `st.markdown` function. The `unsafe_allow_html=True` parameter is used to allow the rendering of HTML in the Streamlit app.

Evaluation

Conclusion and reflection

In this project I created a Stock Trend Prediction Web Application in Python. I have used Streamlit, an open-source Python library.

The Stock Market Trend Predictor project aims to research the technology of stock market trends prediction using machine learning techniques. The project involves data gathering, pre-processing, model building, user interface design, and result visualisation.

My project begins by identifying the problem of unpredictable stock prices caused by market manipulations, monopolistic shareholders and insider trading. The project envisions a user-friendly system that caters to both experienced investors and newcomers, providing insights and predictions to help with investment decisions.

To address this challenge, the project employs the following components:

Data Gathering: Historical stock market data is collected from reliable sources. Various data sources are considered. The final choice is pandas_datareader due to its open-source nature and functionality.

Data Pre-processing: The collected data is cleaned and pre-processed to remove anomalies and inconsistencies. Moving averages (100-day and 200-day) are computed to identify trends and patterns in stock prices.

Model Building: A Long Short-Term Memory (LSTM) neural network model is chosen due to its ability to capture long-term sequences. Keras is used to build the model and the Adam optimizer with Mean Squared Error (MSE) loss function is employed for training.

User Interface: The user interface is developed using Streamlit. Streamlit allows for easy interaction with the prediction model and provides visualizations of stock trends and predictions.

Testing and Evaluation: The developed model and user interface are tested extensively with real and simulated data. Various scenarios, such as non-existing stocks, Cyrillic input, and multiple companies, are tested to ensure the system's robustness and accuracy.

Requirements

The system should provide predictions for stock market prices in real-time or near real-time, ensuring timely decision-making for investors.	Requirement met.
The system should be capable of handling a substantial number of concurrent users without significant degradation in performance.	Requirement met.
The prediction process should be completed within a reasonable time frame, typically a few seconds per prediction.	Requirement met.
The predictive model should strive for a high level of accuracy in forecasting stock prices to enhance the reliability of its insights.	Cannot be measured.

The system should be able to adapt and fine-tune its algorithms to improve prediction accuracy over time based on user feedback and performance analysis.	Requirement is not met. See “Future Enhancements”.
Measures should be in place to inform the user when the online stock exchange markets experience troubles.	Requirement is not met. See “Future Enhancements”
The user interface should be intuitive and user-friendly, catering to both experienced investors and those with limited technical expertise.	The programme is very easy to make use of. Requirement met.
Clear and concise documentation should be provided to guide users through system functionalities and usage.	No documentation is needed.
The codebase should be well-structured and modular, making it easy for developers to maintain and enhance the system in the future.	Requirement met.
Regular updates and improvements should be feasible without disrupting the overall system functionality.	Requirement met, because a file with the new version of the programme can be uploaded to the server, assuring no or minimal (up to a couple of seconds) disruptions.
The system should be compatible with different web browsers and operating systems to provide a seamless experience to a diverse user base.	Requirement met, the web application can be used on a computer as well as on a smartphone.
The system should be developed to manage computing resources efficiently to avoid excessive usage and optimize energy consumption.	See paragraph “Future Enhancements”
The system should be capable of ingesting historical stock market data from the reliable sources.	Requirement met.
Data should be cleaned and pre-processed to remove anomalies and inconsistencies.	Requirement met.
The system should incorporate machine learning algorithms to build and train a prediction model based on historical stock data.	Requirement met.
The prediction model should accept relevant input features for each stock and generate price forecasts as output.	Requirement met, however the model cannot generate the exact price in a current version.
The predictions should be presented in a clear and comprehensible manner, possibly in the form of graphs or charts.	Requirement met.
Users should be able to select specific stocks for which they want to receive predictions.	Requirement met.
Users should be able to select more than one stocks for which they want to receive predictions.	Requirement met.
The system should provide metrics to assess the accuracy of its predictions, such as mean squared error or percentage error.	Requirement met, however, the mean squared error is not accessible to the user yet.

The system should allow users to export prediction results and related data for further analysis.	Requirement met.
The user interface should be intuitive, allowing users to easily navigate and interact with the prediction functionalities.	Requirement met.
The system should provide meaningful error messages to users in case of input errors or system failures.	See paragraph “Future Enhancements”

Future enhancements:

1. Errors should be removed from the programme.
2. A Machine Learning model can be developed to extract important information about the companies from the news articles.
3. User login system can be added so that the customers can add the stock to their “favourites page”. The data collected from the user login system can be then used in marketing purposes.
4. Data encryption can be added to ensure that nobody has information which stock is the most popular one and no geolocation/IP addresses of the users can be traced.
5. The system should have the capability to send notifications to users when significant price changes are predicted for selected stocks.
6. The system can be improved to manage computing resources efficiently to avoid excessive usage and optimize energy consumption.
7. The system should track its performance metrics, such as prediction processing time and server load, for monitoring and optimization purposes.
8. The system can be improved so it will be able to adapt and fine-tune its algorithms to improve prediction accuracy over time based on user feedback and performance analysis.
9. The system should implement regular data backups and recovery mechanisms to prevent data loss in case of system failures.

Appendix