Capstone Project: Building a Machine Learning Model, Develop a Flask Web App and Deploy in the Cloud using Heroku

Deploy ML model on the Web App and Train on the Fly

Md Sohel Mahmood

February 4, 2021

To my beloved wife Samia Jannat

Contents

Abstract

Time series data is utilized to predict the future trend. Stock price and weather forecast technique use machine learning algorithms to to develop a sustainable model and see in the future course of the data. This project aims to develop a stock price prediction machine learning model and then deploy. There are three stages for this project. First a machine learning model is created for the time series data extracted from Yahoo Finance. Next, a web app is developed locally using python's Flask library. Finally, it is deployed in Heroku cloud platform to run the application on the cloud.

Keywords: Time Series, Machine Learning, LSTM, Web App, Python, Flask, Cloud Implementation

1 Introduction

We experience Time Series data every day and everywhere. Data like stock price and weather forecast are the prime examples of time stamped data which can be used for ML model deployment. There are other scenarios where people will want to get the financial health of the company or country, measure quarterly metrices, perform analysis on future market and so on where time plays and the base dimension. In all cases, data is a sequence of collected values at a specific time step. A time series can be both univariate and multi-variate. A univariate time series have only one data point per time step whereas multi-variate time series have multiple values of different variables per time step.

There are some other examples with seasonality, trend and noise [Figure 1]. A seasonal data repeats the trend when the season arrives. For example, ice-cream purchase increases every summer. There may be trend also in the data indicating upward or downward movement. If we collect data of ice-cream for over many years, we may see that the people are buying more ice-creams in latest summers than the summers in the 80's or 90's. This may tell us that the series has an upward trend. Again, there may be noise embedded inside the data which may sometimes corrupt the data and make it difficult to extract the actual trend in the series.

Here, the project aims to the following goals: 1. First create a machine learning model for the time series data extracted from Yahoo Finance 2. Develop a local web app using python's Flask library 3. Deploy the final app in Heroku cloud platform to run the application on the cloud

2 Dependencies

- Python 3.6+
- Visualization libraries: Matplotlib
- Libraries for data and array: pandas and numpy
- Machine learning libraries:Tensorflow, Keras, Sciki-Learn

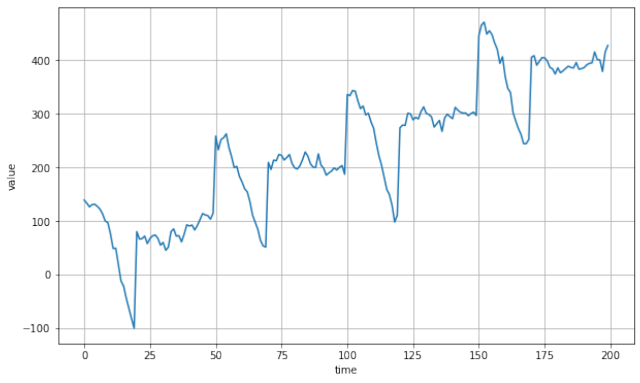


Figure 1: Time series data with trend and seasonality [1]

- Web App library: Flask
- Financial data parsing library: Yfinance

3 Git Repository

To clone the git repository: `git clone https://github.com/mdsohelmahmood/stock-price-predict`

4 Execution of program

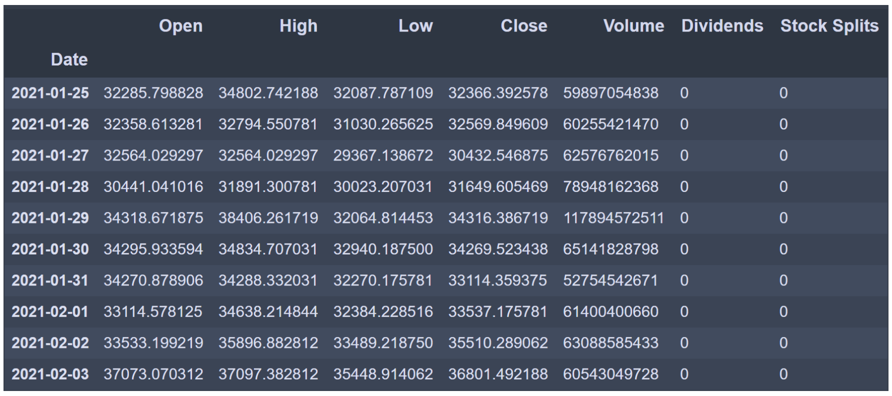
Part 1

First step is to create an ML model for the time series data from historical stock price. Data extracted from Yahoo Finance using YFiance library. We start the project code by importing the necessary libraries.

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
import yfinance as yf
```

Next, the historical data is imported using YFinance. To parse the data, the stock name need to be keyed in. For example, to get the historical price of

Bitcoins in USD, the name should be “BTC-USD” [Figure 2].



	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
2021-01-25	32285.798828	34802.742188	32087.787109	32366.392578	59897054838	0	0
2021-01-26	32358.613281	32794.550781	31030.265625	32569.849609	60255421470	0	0
2021-01-27	32564.029297	32564.029297	29367.138672	30432.546875	62576762015	0	0
2021-01-28	30441.041016	31891.300781	30023.207031	31649.605469	78948162368	0	0
2021-01-29	34318.671875	38406.261719	32064.814453	34316.386719	117894572511	0	0
2021-01-30	34295.933594	34834.707031	32940.187500	34269.523438	65141828798	0	0
2021-01-31	34270.878906	34288.332031	32270.175781	33114.359375	52754542671	0	0
2021-02-01	33114.578125	34638.214844	32384.228516	33537.175781	61400400660	0	0
2021-02-02	33533.199219	35896.882812	33489.218750	35510.289062	63088585433	0	0
2021-02-03	37073.070312	37097.382812	35448.914062	36801.492188	60543049728	0	0

Figure 2: Historical trend of the stock

The next phase is the creation of training and test dataset. Training dataset is the first 80 percent of the total data and the remaining 20 percent will be predicted and will be used as test data.

```
df=hist
d=30
ahead=10
n=int(hist.shape[0]*0.8)
training_set = df.iloc[:n, 1:2].values
test_set = df.iloc[n:, 1:2].values
```

The dataset is scaled and reshaped in the following step.

```
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
X_train = []
y_train = []
for i in range(d, n-ahead):
X_train.append(training_set_scaled[i-d:i, 0])
y_train.append(training_set_scaled[i+ahead, 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))
```

Then the model is defined using several layers of LSTM from tensorflow. Tensorflow is an open source library for machine learning. It can handle sequence information using Recurrent Neural Network (RNN). Tensorflow has several recurrent layer types including SimpleRNN, Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM). This project utilizes LSTM feature.

While going through the RNN, some information may be lost due to data transformations. If the initial data is lost, the subsequent result will have no trace of the first inputs. This is true for human too. If we do not memorize a data, it will be lost from out memory after some time. To fix this issue, RNN can be deployed with cells having Long Short Term Memory (LSTM) that can effectively have information of the previous data. So while using LSTM, we do not need to bother about long term dependencies of the data. LSTM was first introduced in 1997 by Sepp and Jurgen [2] and was developed in later years.

The model is defined as below.

```
model = Sequential()
model.add(LSTM(units = 100, return_sequences = True, input_shape
= (X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units = 100, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(X_train, y_train, epochs = 50, batch_size = 32)
```

When the model runs, it will provide the loss values after each epoch. As the model have more epochs, the loss value will drop. Three LSTM layers are added with 100 units of cells for the first 2 layers and 50 units for the last layer. Dropping 20 percent of theoutput will reduce the amount of overfitting the training data. One unit of dense layer is added at the bottom and the model is then compiled with 'adam' optimizer and 'mean_aquared_error' loss.

The model is saved in the next phase `model.save("BTC-predict.h5")`

Once the model is defined, the test dataset is extracted out from the historical data and from there we need to take the first 30 days as input to the model (of course this number can be defined by the user but here for the sake of understanding, the number 30 is taken to predict the 31st point or the point the user wants).

```
dataset_train = df.iloc[:n, 1:2]
dataset_test = df.iloc[n:, 1:2]
dataset_total = pd.concat((dataset_train, dataset_test), axis
= 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test)
- d:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs))
```

The test data is reshaped afterwards.

```
X_test = []
for i in range(d, inputs.shape[0]):
X_test.append(inputs[i-d:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1],
1))
print(X_test.shape)
```

The final step of part 1 is to predict the test data set and plot along with the actual dataset [Figure 3].

```
predicted_stock_price = model.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
df['Date']=df.index
df=df.reset_index(drop=True)
plt.plot(df.loc[n:, 'Date'],dataset_test.values, color = 'red',
label = 'Real Butcoin Stock Price')
plt.plot(df.loc[n:, 'Date'],predicted_stock_price, color =
'blue', label = 'Predicted Bitcoin Stock Price')
plt.title('Bitcoin Price Prediction')
plt.xlabel('Time')
plt.ylabel('Bitcoin Price')
plt.legend()
plt.xticks(rotation=90)
plt.show()
```

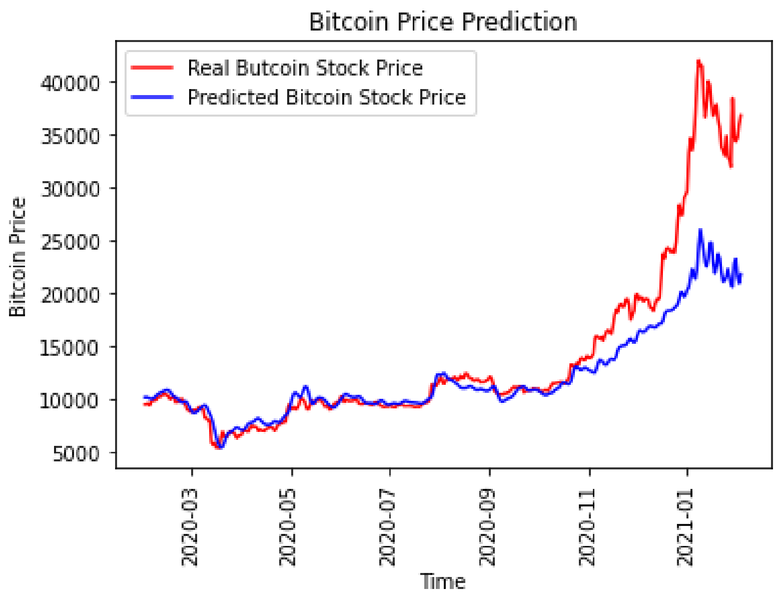
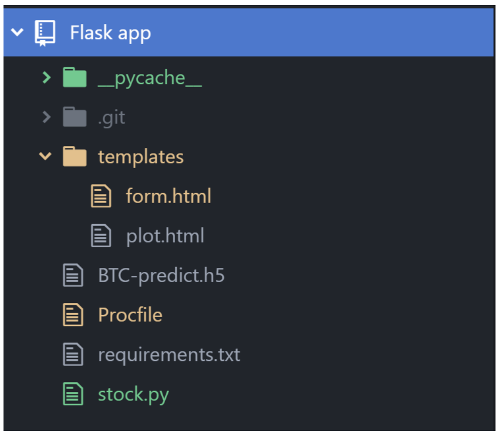


Figure 3: Predicted stock price plotted wth the actual price

Part 2

In Part 2, the local hosting of the web app will be implemented. To use the local server, python's flask library will be utilized in this project because of its ease of deployment. The first step in Part 2 is to consolidate all the project files inside a project folder [Figure 4]. The project folder contains the main project file in .py format and the saved model from Part 1. The html files are saved inside 'templates' folder. That's all we need to run the app in localhost.



```
Flask app
  __pycache__
  .git
  templates
    form.html
    plot.html
  BTC-predict.h5
  Procfile
  requirements.txt
  stock.py
```

Figure 4: Files inside the project folder

Once the project folder is setup, the command line is used to initiate the web app. First a separate python environment is created for flask. The environment is named as 'flask_env' and later it is activated using 'conda'. 'Conda' is a python library and environment manager.

```
conda create -name flask_env
conda activate flask_env
```

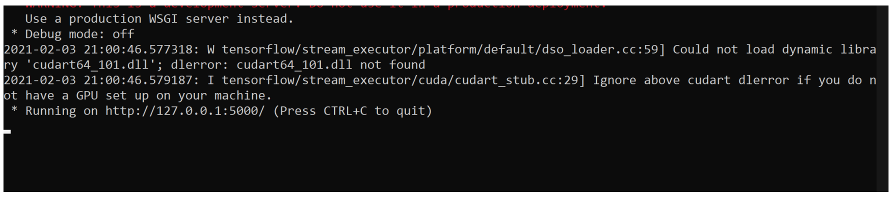
Then the current directory is changed to the project folder using 'cd'. Afterwards the stock.py file set as the running script for the web app by the following command.

```
set FLASK_APP=stock.py
```

Now the html. files need to set for proper user input. I have created two htl files. One name 'form.html' which takes the user input for the machine learning model training and the other one is 'plot.html' which delivers the predicted plot. Those two files are placed inside th templates folder.

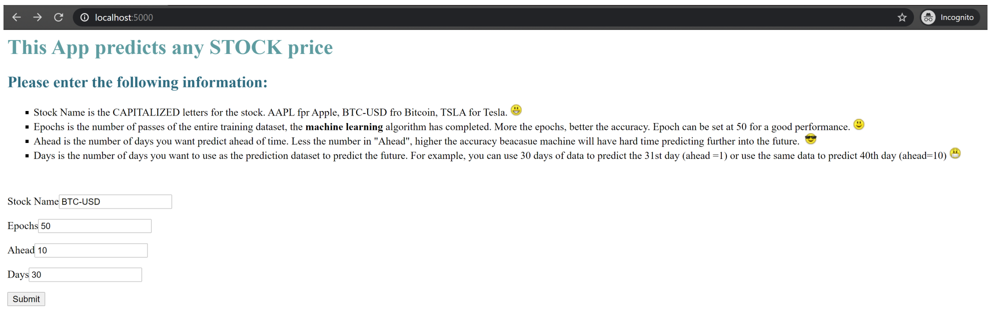
At the final step, 'flask run' command is excuted to deploy the app on local server [Figure 5] .

Figure 6 shows the front-end look of the web app. It shows the necessary input explanations with the textbox below. For this app I am basically trying to predict the future stock trend of any stock. The ML model can be trained differently based on different user input [Figure 7]. So model therefore can be trained on the fly and the user can get instant result based on the inputs.



```
  Use a production WSGI server instead.
 * Debug mode: off
2021-02-03 21:00:46.577318: W tensorflow/stream_executor/platform/default/dso_loader.cc:59] Could not load dynamic libra
ry 'cudart64_101.dll'; dlerror: cudart64_101.dll not found
2021-02-03 21:00:46.579187: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do n
ot have a GPU set up on your machine.
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 5: Flask app running at localhost having address of 127.0.0.1 at port 5000



This App predicts any STOCK price

Please enter the following information:

- Stock Name is the CAPITALIZED letters for the stock. AAPL fpr Apple, BTC-USD fro Bitcoin, TSLA for Tesla.
- Epochs is the number of passes of the entire training dataset, the **machine learning** algorithm has completed. More the epochs, better the accuracy. Epoch can be set at 50 for a good performance.
- Ahead is the number of days you want predict ahead of time. Less the number in "Ahead", higher the accuracy beacasue machine will have hard time predicting further into the future.
- Days is the number of days you want to use as the prediction dataset to predict the future. For example, you can use 30 days of data to predict the 31st day (ahead =1) or use the same data to predict 40th day (ahead=10)

Stock Name BTC-USD

Epochs 50

Ahead 10

Days 30

Submit

Figure 6: Front-end of the web app

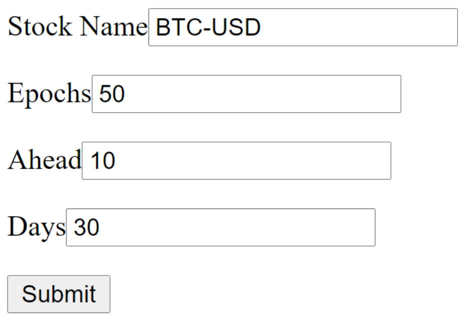
1. Stock Name is the CAPITALIZED letters for the stock. AAPL for Apple, BTC-USD fro Bitcoin, TSLA for Tesla.
2. Epochs is the number of passes of the entire training dataset, the machine learning algorithm has completed. More the epochs, better the accuracy. Epoch can be set at 50 for a good performance.
3. Ahead is the number of days user wants to predict ahead of time. Less the number in "Ahead", higher the accuracy becasue machine will have hard time predicting further into the future.
4. Days is the number of days user wants to use as the prediction dataset to predict the future. For example, one can use 30 days of data to predict the 31st day (ahead =1) or use the same data to predict 40th day (ahead=10).

While the machine is learning on the training data, the command line will show the number of epochs [Figure 8]. Higher number of epochs will take longer time to complete the training.

When completed, the app will be redirected to the plot.html and will deliver the predicted price along with the actual price of the stock [Figure 9].

Part 3

In part 3, I will implement the web app of predicting the stock price on Heroku cloud server. Since the free account in Heroku provides 500 Mb of RAM which is not enough for training the model using tensorflow on the fly. Tensorflow



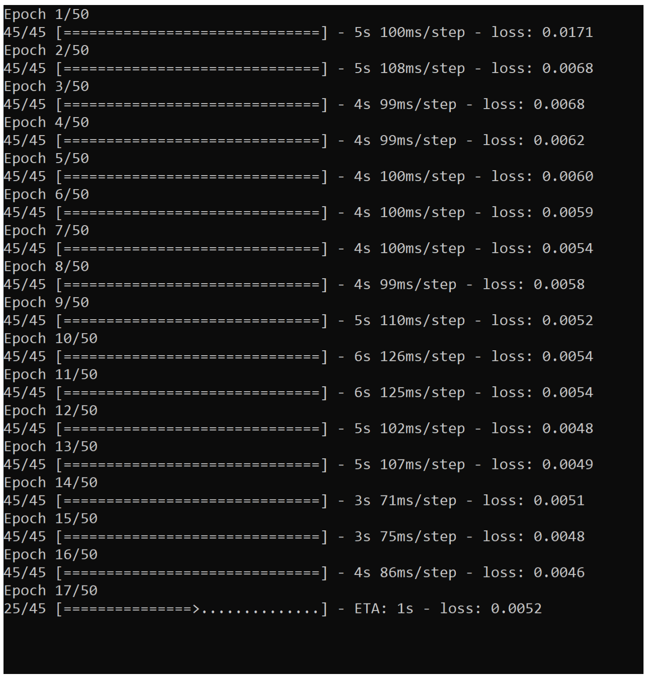
Stock Name BTC-USD

Epochs 50

Ahead 10

Days 30

Figure 7: Sample user input for Bitcoin (BTC-USD)



```
Epoch 1/50
45/45 [==============================] - 5s 100ms/step - loss: 0.0171
Epoch 2/50
45/45 [==============================] - 5s 108ms/step - loss: 0.0068
Epoch 3/50
45/45 [==============================] - 4s 99ms/step - loss: 0.0068
Epoch 4/50
45/45 [==============================] - 4s 99ms/step - loss: 0.0062
Epoch 5/50
45/45 [==============================] - 4s 100ms/step - loss: 0.0060
Epoch 6/50
45/45 [==============================] - 4s 100ms/step - loss: 0.0059
Epoch 7/50
45/45 [==============================] - 4s 100ms/step - loss: 0.0054
Epoch 8/50
45/45 [==============================] - 4s 99ms/step - loss: 0.0058
Epoch 9/50
45/45 [==============================] - 5s 110ms/step - loss: 0.0052
Epoch 10/50
45/45 [==============================] - 6s 126ms/step - loss: 0.0054
Epoch 11/50
45/45 [==============================] - 6s 125ms/step - loss: 0.0054
Epoch 12/50
45/45 [==============================] - 5s 102ms/step - loss: 0.0048
Epoch 13/50
45/45 [==============================] - 5s 107ms/step - loss: 0.0049
Epoch 14/50
45/45 [==============================] - 3s 71ms/step - loss: 0.0051
Epoch 15/50
45/45 [==============================] - 3s 75ms/step - loss: 0.0048
Epoch 16/50
45/45 [==============================] - 4s 86ms/step - loss: 0.0046
Epoch 17/50
25/45 [================>.............] - ETA: 1s - loss: 0.0052
```

Figure 8: Back-end training of the model

itself consumes 300+ Mb. Therefore later, I built and saved the model. Since the model has all the necessary information to predict the test data and no tensorflow is required, it is less than 500 Mb limit and can be easily deployed in Heroku.

There are 2 additional requirements for Heroku to implement the web app.

1. requirements.txt
2. Procfile

The requirements.txt file is a simple text file containing all the required libraries for the app to run. It will look something like Figure 10.

This file is created using the following command.

```
pip freeze > requirements.txt > Procfile
```

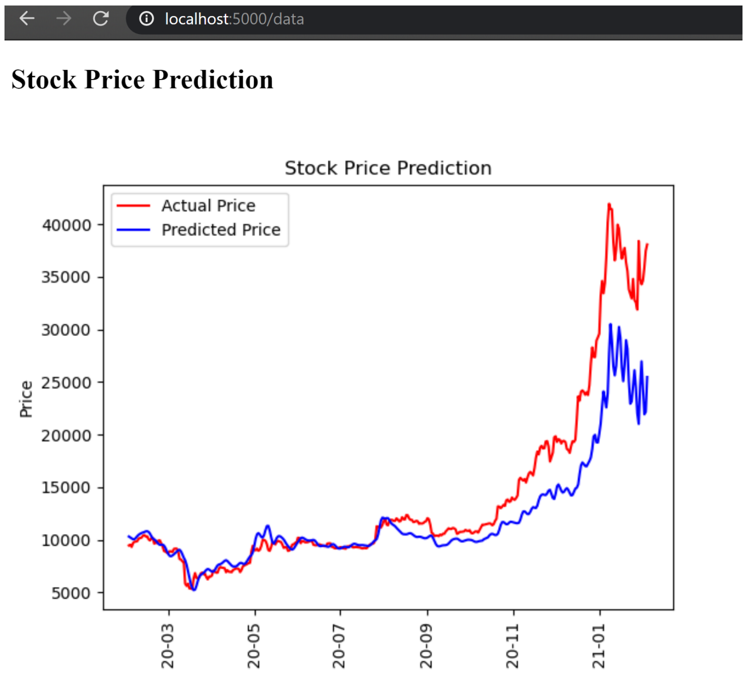


Figure 9: Output of the web app delivering the predicted price of the stock on localhost

Then a Procfile is required which declares the type of the app and the app file to execute. The Procfile is created in command line.

```
echo web:  gunicorn stock:app > Procfile
```

It contains only one line. 'Gunicorn' is a web server gateway interface that implements the app in the cloud. I have used git commands to commit the project files to the server. The command sequence is below.

```
git init
git add .
git commit -am "initial commit"
```

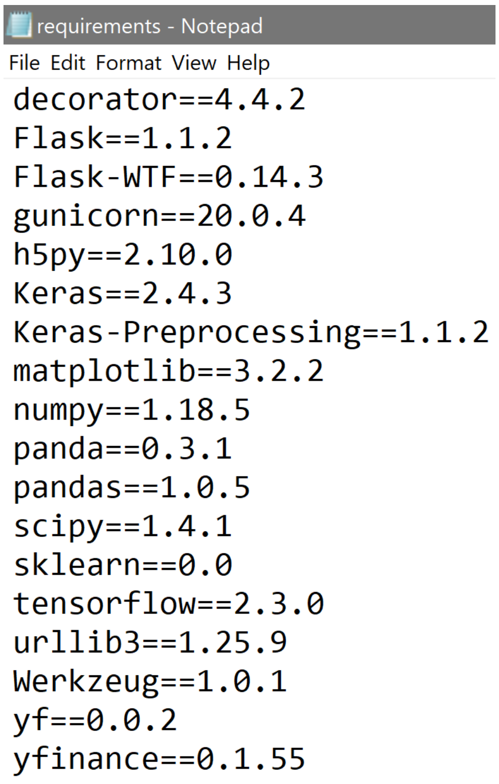
Then Heroku login is required to push the project files.

```
heroku login
heroku create
heroku rename new-name
git push heroku master
```

The command 'Heroku login' will prompt the ser to login in the browser. Once the project is created, the files will be pushed to the master branch subsequently [Figure 11].

When the app is initiated by going to the pointed https link, the same front-end [Figure 12] is displayed but now on the cloud.

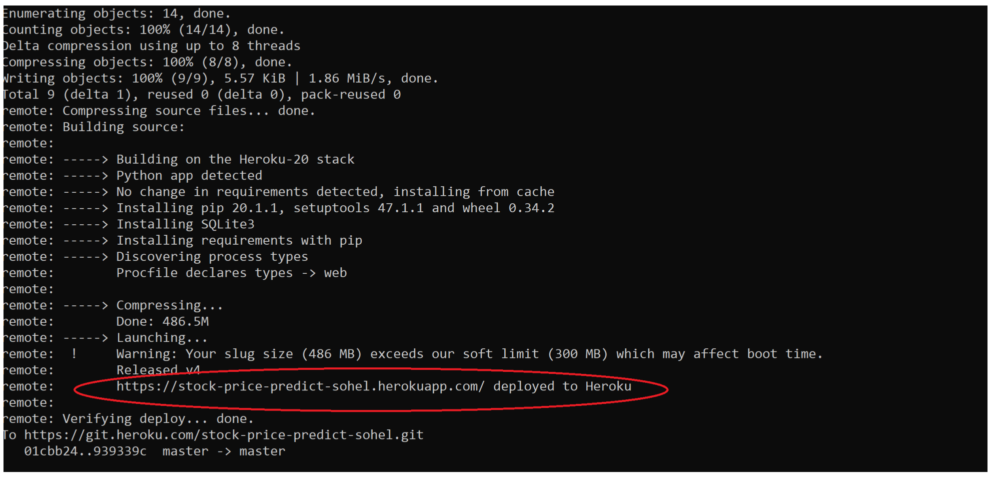
I have proceeded with similar user inputs to train the model on the fly but the limited RAM size could not complete the training. Therefore, I have saved the trained model and then placed the model inside the project folder and


requirements - Notepad

File Edit Format View Help

```
decorator==4.4.2
Flask==1.1.2
Flask-WTF==0.14.3
gunicorn==20.0.4
h5py==2.10.0
Keras==2.4.3
Keras-Preprocessing==1.1.2
matplotlib==3.2.2
numpy==1.18.5
panda==0.3.1
pandas==1.0.5
scipy==1.4.1
sklearn==0.0
tensorflow==2.3.0
urllib3==1.25.9
Werkzeug==1.0.1
yf==0.0.2
yfinance==0.1.55
```

Figure 10: Content of requirements.txt file

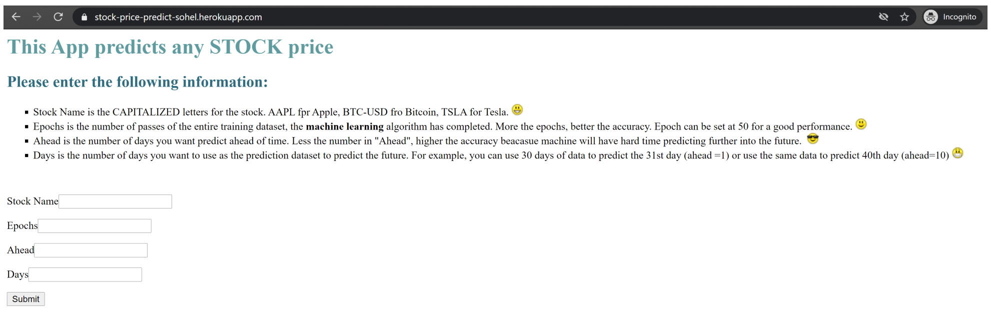


```
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 5.57 KiB | 1.86 MiB/s, done.
Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku-20 stack
remote: -----> Python app detected
remote: -----> No change in requirements detected, installing from cache
remote: -----> Installing pip 20.1.1, setuptools 47.1.1 and wheel 0.34.2
remote: -----> Installing SQLite3
remote: -----> Installing requirements with pip
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 486.5M
remote: -----> Launching...
remote:  !     Warning: Your slug size (486 MB) exceeds our soft limit (300 MB) which may affect boot time.
remote:        Released v4
remote:        https://stock-price-predict-sohel.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/stock-price-predict-sohel.git
   01cbb24..939339c  master -> master
```

Figure 11: Heroku deployment completion

then commit. In this way, I do not need the tensorflow library as well as other ML libraries and my required RAM is greatly reduced below 500 Mb and can be easily deployed on Heroku. The down side of this type of pre-saved model implementation is the model is not trained on the latest data and will provide the predicted output based on the owner's saved model.

To upload the model on the flask app, I have created a separate .py file,



stock-price-predict-sohel.herokuapp.com

This App predicts any STOCK price

Please enter the following information:

- Stock Name is the CAPITALIZED letters for the stock. AAPL fpr Apple, BTC-USD fro Bitcoin, TSLA for Tesla.
- Epochs is the number of passes of the entire training dataset, the **machine learning** algorithm has completed. More the epochs, better the accuracy. Epoch can be set at 50 for a good performance.
- Ahead is the number of days you want predict ahead of time. Less the number in "Ahead", higher the accuracy beacasue machine will have hard time predicting further into the future.
- Days is the number of days you want to use as the prediction dataset to predict the future. For example, you can use 30 days of data to predict the 31st day (ahead =1) or use the same data to predict 40th day (ahead=10)

Stock Name	
Epochs	
Ahead	
Days	

Figure 12: Front-end of the web app on Heroku server

loaded the pre-saved model and test the data in hand. The file is saved as model_stock.py in the same working directory. Only the Procfile is changed and redirected to this new .py file.

```
web:  gunicorn stock_model:app
```

Upon following the similar git command, I ended up pushing the project along with the saved model successfully on the Heroku cloud. The app then shows the predicted price trend of the stock on the cloud [Figure 13].

5 Discussion

The web app that has been developed to predict stock price, works for any stock since the user has the flexibility to input their preferred stock name. The users can also provide how many days ahead in time they want to predict and how many inputs of training samples they want to take to predict one future data point. They also have the flexiility to modify the epochs number so that the loss function can be minimized. The Heroku platform's limited RAM availability hindered the app to train the model on the fly over the cloud but later the pre-saved trained model was uploaded to reduce the RAM usage. The drawback with the model is the user can only get the specific stock prediction which the owner has uploaded. If there is higher allocated memory available, the cloud app will be same as the local server and will provide more flexibility to the users. That being said, the web app has successfully deployed on the local server and on the cloud using the LSTM machine learning algorithm.

6 Acknowledgement

I would like to acknowledge the learning I received from the book [3] by Aurelien Geron and [1] by Laurence Moroney. Also, Udacity for providing me the opportunity to dive deep into the data science realm. The online html editor [4] has also eased the html coding.

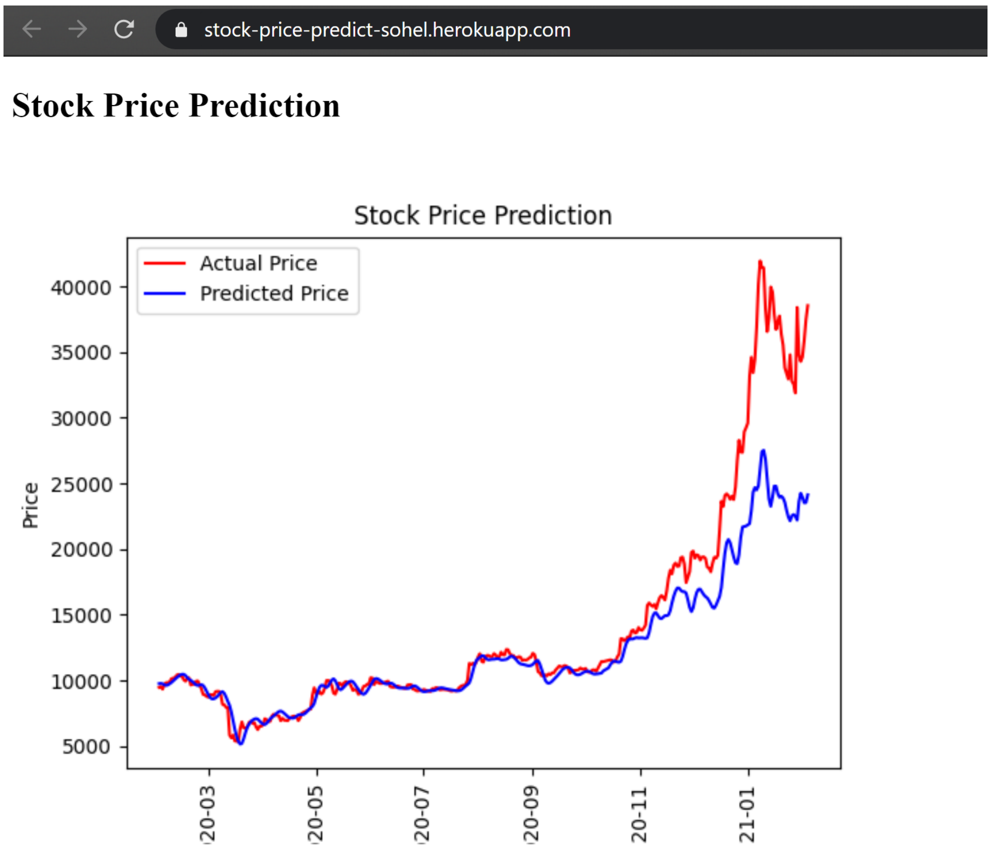


Figure 13: Predicted output price of stock from the web app on Heroku server

7 Conclusion

This work demonstrates the implementation of a stock price prediction model using machine learning libraries. The model seems reliably following the actual price of stocks up to 10 days ahead of time. Three layers of LSTM has been used in the model and dropouts are added to minimize the overfitting of the data. The app is deployed on the local server which provides the flexibility to input any stock's name as well as other variable features that the user can control. The app is then deployed on Heroku cloud platform using the pre-saved model. This web app has shown the capability to get trained on any time series data and successfully predicted the stock price.

References

1. Laurence Moroney, *AI and Machine Learning for Coders, Chapter 9*, (`https://github.com/lmoroney/tfbook/tree/master/chapter9`), O'Reilly Media Inc., Sebastpol, CA 95472

2. Sepp Hochreiter and Jurgen Schmidhuber, *LONG SHORT-TERM MEMORY*, (https://www.bioinf.jku.at/publications/older/2604.pdf), Neural Computation 9(8):1735-1780, 1997.

3. Aurelien Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media Inc., Sebastpol, CA 95472

4. HTML editor,
(https://html-online.com/editor/)