Group Members - Jash Popat, Manu Jain, Nishtha Agarwaal
CSIT334 - Statistics and Optimisation for Computing
Course Professor - Prof. V.K. Jayraman
Faculty Mentor - Prof. Prajish Prasad
April 23, 2023

Semester Project Report

1. Introduction

The stock market, like many other asset markets, is a complex system that is influenced by a multitude of factors like economic trends, global events, and company performance. Predicting stock prices has been a challenging task for as long as the stock market has existed and has been the subject of extensive research in the field of finance and data science.

With great interest in this field, we wished to try out hand at predicting stock prices using historical data and thus, devised a method of doing so by comparing the performance of company shares using the linear regression and recurrent neural network (RNN) models.

The project involved multiple steps, like collecting and preprocessing the data, developing and training the models, and evaluating their performance using appropriate metrics. By analyzing the results, we gained insights into the strengths and weaknesses of each model (including several others that didn't match up to our expectations) and determined which one is better suited for predicting stock prices over a period of days.

2. Literature Survey

Stock price prediction has been a subject of great interest in the field of finance and data science for many years. With an increasing number of models being proposed to predict the behavior of the stock market every year, it can be seen that interest for this subject is simply bound to grow more with each passing year.

The most popular methods used over the years include Support Vector Machines (SVM), TIme Series Analysis, Random Forests and many others. Out of these we chose two that we believed fit the best with our ideologies and understanding. These were Linear Regression and Recurrent Neural Networks (RNN). CNN was also planned to be included, but due to accuracy issues, was removed in the end.
Linear regression models have been widely used in finance for stock price prediction. A study by Brown and Warner (1985) showed that linear regression models can be used to

predict stock returns with reasonable accuracy. In another study, Lee and Park (2013) developed a linear regression model to predict stock prices using technical indicators such as moving averages, relative strength index, and Bollinger Bands. The results showed that the model performed well in predicting stock prices for a short period.

On the other hand, RNN models have gained popularity in recent years for their ability to model sequential data. RNN models have been used to predict stock prices by taking into account the temporal dependencies in the data. In a study by Ding et al. (2015), an RNN model was used to predict the stock price movement based on historical data. The results showed that the RNN model outperformed traditional linear regression models in predicting the stock price movement.

It has been seen that we are not the first to compare these two methods, as several studies in the past have compared the performance of linear regression and RNN models for stock price prediction. In a study by Zhang et al. (2018), the authors compared the performance of linear regression, SVM, and RNN models for stock price prediction. The results showed that the RNN model outperformed the other models in predicting stock prices (a reason as to why we chose RNN ourselves).

3. Problem Statement and Aim

The purpose of this project is to compare the performance of linear regression and RNN models for stock price prediction over a period of days using the closing price as the dependent variable. The models will be judged on their ability to predict the closing price of a stock .

The project aims to achieve the following goals:

- Collect historical stock price data, including open, high, low, and close prices, for a specified time period.
- Preprocess the data by cleaning and normalizing it to ensure it is suitable for modeling.
- Divide the data into training and test sets to evaluate model performance.
- Develop a linear regression model to predict the closing price of a stock for a specified number of days into the future.
- Develop an RNN model to predict the closing price of a stock for a specified number of days into the future.
- Train both models using the training data and evaluate their performance using the test data.
- Use appropriate evaluation metrics, such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE), to compare the performance of the two models.

- Analyze the results to determine which model performs better for predicting stock prices.

4. **Data set details**

   For our data, we used the yfinance module in Python. It is a third-party library that allows you to download financial data from Yahoo Finance. Here are some details about the data sets available through the yfinance module:

   - Historical price data - opening price, closing price, high and low prices, and trading volume
   - Dividend and stock split data
   - Company information
   - Financial statements
   - Analyst recommendations

5. **Our Roles**

a. **Manu**

My role for this project was creating a Recurrent Neural Network (RNN) model as a team member for this project. Convolutional Neural Networks (CNN) were the first technique we looked into for this assignment. But after careful study, we found that the Mean Squared Error (MSE) values and the graphical representations connected to the CNN model were unsatisfactory, which resulted in the decision to give up on this strategy. We chose to concentrate on creating an RNN model instead. I used the 'yfinance' module to get the necessary data to speed up this process. The data was then cleaned, organized, and transformed as part of a thorough preprocessing procedure to verify that it was compatible with the RNN model. Additionally, I decided to determine the model's best hyperparameters on my own. To do this, I used a trial-and-error method, rigorously evaluating the effects of different combinations of the model's hyperparameters on its performance. We were able to find the ideal collection of hyperparameters through an iterative method that produced the greatest results for our RNN model.

b. Nishtha

My role was preparing the data as part of this function to ensure that it was clean and organized in a way that could be used for modeling. This included data cleansing, normalization, feature engineering, and data splitting. After preprocessing the data, we used yfinance to collect and alter financial data, and then I utilized linear regression techniques to develop a predictive model. This entailed choosing relevant features, applying the model to training data, and assessing its performance on test data use Mean Squared Error and R-squared metric.

c. Jash

My role was to help prepare the data to be used in the Linear Regression model along with creating the model itself. To prepare the data, I had to clean the model for any irregularities and split the data into training and test set. Many of the algorithms and codes generated for cleaning the data were done by me, after which I also helped in creating the Linear Regression model. Before all the models were finalized, I had also researched and aided in the creation of the CNN model, which was sadly cut out due to the multitude of issues we faced with it. The disproving of the model due to the high MSE was done in part due to the many tests I ran on the model.

6. Linear Regression

Linear regression is a statistical technique commonly used to model the relationship between a dependent variable and one or more independent variables. In the context of stock prediction, linear regression can be used to model the relationship between the price of a stock (the dependent variable) and one or more independent variables, such as the number of days, economic indicators, or company financial data.

Assumption:

The linearity assumption of linear regression states that the relationship between the dependent variable and independent variables is linear. This means that the relationship can be described by a straight line, which can be represented by a mathematical equation.

Challenge:

One potential challenge of using linear regression for stock prediction is dealing with missing or inconsistent data. If there are missing values or inconsistencies in the data, this can affect the accuracy of the model and lead to biased or unreliable predictions. This challenge is tackled with the help of preprocessing.

I. Preprocessing

For the linear regression model, we preprocessed the data by dropping any rows with missing values. We then created a new column called 'Days' that represents the number of days since the start_date. The 'Days' column is then reshaped into a numpy array and assigned to X, while the 'Close' column is assigned to y.

```
data.dropna(inplace = True)
data['Days'] = (data.index - pd.to_datetime(start_date)).days
X = data['Days'].values.reshape(-1, 1)
y = data['Close'].values
```

II. Modelling

Before we started making the model, we decided to use Days(X) as the independent variable and the stock closing price(Y) as the dependent variable.
Once preprocessing and feature selection is done, we used the following steps to build the model, evaluate it, and visualize it:

- Split data into training and testing sets
- Train the regression model using the train data
- Make predictions on the test data
- Evaluate the model using MSE and R-squared metrics
- Visualize the results using a scatter plot graph

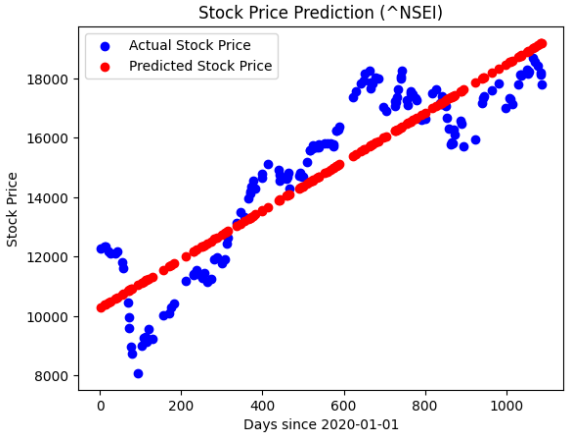
III. Code

```
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

ticker = "GOOG"
start_date = "2020-01-01"
end_date = "2023-01-01"

data = yf.download(ticker, start=start_date, end=end_date)
print(data)

# Prepare data for the model
data.dropna(inplace = True)
data['Days'] = (data.index - pd.to_datetime(start_date)).days
X = data['Days'].values.reshape(-1, 1)
y = data['Close'].values

# Split the data into training and testing sets
```

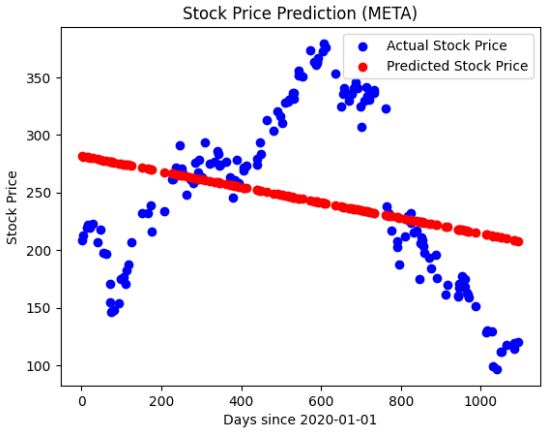
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

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

# Make predictions on the test set
y_pred = regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print("Mean squared error:", mse)

# Predict the stock price for a specific day
future_day = 1000
predicted_price = regressor.predict([[future_day]])
print(f"Predicted stock price for day {future_day}: ${predicted_price[0]:.2f}")

from sklearn.metrics import r2_score

# Calculate the R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)

import matplotlib.pyplot as plt
plt.scatter(X_test, y_test, color='blue', label='Actual Stock Price')
plt.scatter(X_test, y_pred, color='red', label='Predicted Stock Price')
plt.xlabel('Days since ' + start_date)
plt.ylabel('Stock Price')
plt.title('Stock Price Prediction (' + ticker + ')')
plt.legend()
plt.show()
```

IV. Results

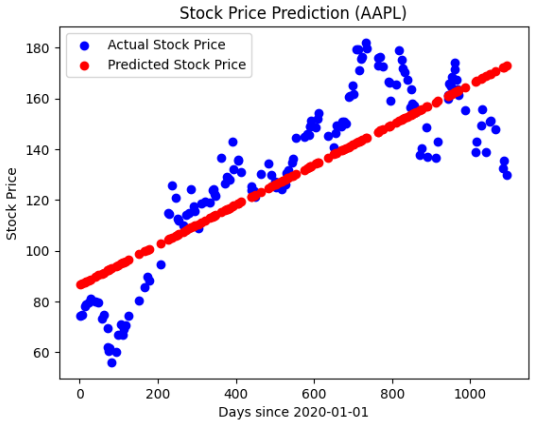
A. NSEI



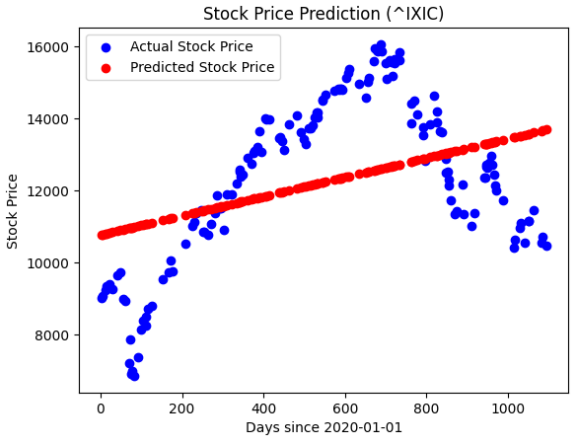
B. Meta



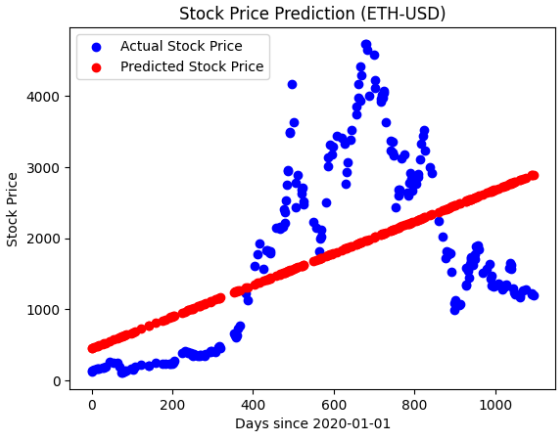
C. Apple



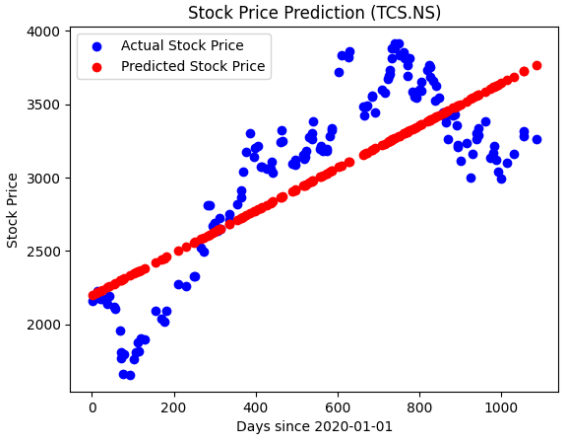
D. NASDAQ (POINTS/ not price)



E. Ethereum - USD



F. Tata Consultancy



7. RNN Using LSTM

Recurrent Neural Networks, or RNNs, are a subset of neural networks created specifically to process sequential data. RNNs are able to keep an internal state or memory, in contrast to conventional feedforward neural networks, which enables them to handle sequential data such as time series, audio, and text.

RNNs' fundamental characteristic is their architecture, which includes loops that let them transmit information from one step of a sequence to the next. As a result, they can handle inputs of different lengths and carry out tasks like sentiment analysis, speech recognition, and language translation that require a grasp of the context.

The Long Short-Term Memory (LSTM) network, one of the most widely used RNN variants, was created to address the vanishing gradient issue that can arise when training deep networks. LSTMs are particularly useful for jobs involving long-term dependencies because they employ gating mechanisms to selectively retain or forget information from the past.

I. Preprocessing

Preprocessing for our project entails a number of processes that get the data ready for the LSTM model's training and testing. An description of each preprocessing stage is provided below:

A. Data loading: Using the yfinance library, we download historical stock price information for the selected company. From the start date until the end date, the data is retrieved.

```
data = yf.download(company, start=start, end=end)
```

B. Scaling: The MinMaxScaler from the sklearn.preprocessing module is used to scale the data to a range of 0 to 1. This process is crucial since it speeds up model convergence and enhances training.

```
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(data['Close'].values.reshape(-1,1))
```

C. Preparing training data: Using the prediction_days variable, we build input and output sequences for the model. The output sequence consists of the closing price that comes after the input sequence. The input sequence is made up of the prediction_days number of consecutive closing prices.

```
x_train = []
y_train = []
for x in range(prediciton_days, len(scaled_data)):
    x_train.append(scaled_data[x-prediciton_days: x, 0])
```

```
    y_train.append(scaled_data[x, 0])
```

D. Reshaping the training data: We modify the training data to conform to the model's LSTM layer's specifications for input shape. Samples, time steps, and features should be the format for the input shape. We only have one feature in this situation, and that is the closing price.

```
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

E. Preparing test data: Using the yfinance library, we obtain the test data for the given company. From the test start date until the test end date, the data is retrieved.

```
test_data = yf.download(company, start=test_start, end=test_end)
```

F. Making model inputs: Using the same prediction_days variable, we make input sequences for the model. prediction_days number of consecutive closing prices from the pooled dataset make up the input sequence.

```
model_inputs = total_dataset[len(total_dataset) - len(test_data) - prediciton_days:].values
#last predicition days + len(test_data) data point from the total dataset
model_inputs = model_inputs.reshape(-1, 1)
model_inputs = scaler.transform(model_inputs)
```

The data is supplied into the LSTM model for training and prediction after preprocessing. After that, the model's performance is assessed using metrics like mean squared error (MSE) and root mean square error (RMSE), and the outcomes are plotted for visualization.

II. **Modelling**

A. Model construction: Keras is used to build a sequential model. By giving a list of layer instances, we can quickly generate the sequential model, which is a linear stack of layers.

```
model = Sequential()
```

B. Adding Layers: Three LSTM layers are added to the model. With 50 units each, the first two LSTM layers pass on the whole sequence of hidden states to the following layer. There are 50 units in the third LSTM layer, which does not return sequences.

```
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(LSTM(units=50, return_sequences=True))
model.add(LSTM(units=50))
```

C. Adding a Dense layer: To obtain the forecasted stock price, we add a Dense layer with one unit. The input is connected to every neuron in the output layer by the dense layer, which is a fully connected layer.

```
model.add(Dense(units=1))  # Prediction of the next closing price.
```

D. Assembling the model: The mean squared error (MSE) is used as the loss function during model construction together with the Adam optimizer. The model's weights are updated by the optimizer, and the loss function calculates the discrepancy between expected and actual results.

```
model.compile(optimizer='adam', loss='mean_squared_error')
```

E. Model training: Using 25 epochs and a batch size of 32, we train the model using the training datasets (x_train and y_train). A batch size is the amount of samples the model analyzes all at once before updating the weights, whereas an epoch is a complete iteration through the training dataset.

```
model.fit(x_train, y_train, epochs=35, batch_size=32)
```

F. Finding best hyperparameter using Grid Search CV
A method for determining the ideal set of hyperparameters for a machine learning model is called grid search. It operates by thoroughly testing each possible set of hyperparameter values in order to identify the one that yields the greatest model performance. An LSTM model's hyperparameters, such as the quantity of neurons, the learning rate, and the dropout rate, among others, can be optimized via grid search. Grid search, especially for deep learning models like LSTMs, was time-consuming. It requires repeatedly training and testing the model with different combinations of hyperparameters, which can take a long time, especially when dealing with big datasets or complicated models. And for this reason we went for a trial-and-error method. We found that epochs of 25 worked best and batch size of 32 was also working well.

III. Code

```
import numpy as np
import pandas as pd
import yfinance as yf
import math
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.layers.normalization.batch_normalization_v1 import BatchNormalization
```

```
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout, LSTM
import datetime as dt

company = 'AAPL'
start = dt.datetime(2012, 1, 1)
end = dt.datetime(2020, 1, 1)

data = yf.download(company, start=start, end=end)

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(data['Close'].values.reshape(-1,1))

prediciton_days = 300

x_train = []
y_train = []

# basically this part is taking in 121 data. It will learn from 120 data to predict the 121st
data (y_train).

for x in range(prediciton_days, len(scaled_data)):
    x_train.append(scaled_data[x-prediciton_days: x, 0])
    y_train.append(scaled_data[x, 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))

model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(LSTM(units=50, return_sequences=True))
model.add(LSTM(units=50))
model.add(Dense(units=1))  # Prediction of the next closing price.
```

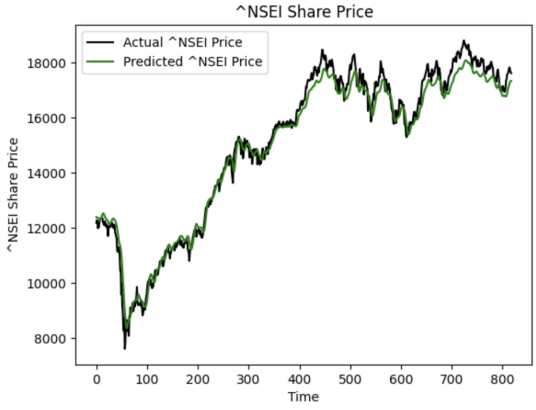
```
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs=35, batch_size=32)

test_start = dt.datetime(2021,1,1)      # important
test_end = dt.datetime.now()

test_data = yf.download(company, start=test_start, end=test_end)
actual_price = test_data['Close'].values

total_dataset = pd.concat((data['Close'], test_data['Close']), axis = 0)
model_inputs = total_dataset[len(total_dataset) - len(test_data) - prediciton_days:].values
#last predicition days + len(test_data) data point from the total dataset
model_inputs = model_inputs.reshape(-1, 1)
model_inputs = scaler.transform(model_inputs)

x_test = []
y_test = []
for x in range(prediciton_days, len(model_inputs)):
  x_test.append(model_inputs[x-prediciton_days:x, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

y_test = total_dataset[len(total_dataset) - len(test_data):].values
y_test_norm = scaler.transform(y_test.reshape(-1, 1)) #actual price

predicted_price_norm = model.predict(x_test)
predicted_price = scaler.inverse_transform(predicted_price_norm) #predicted actual price

mse = mean_squared_error(y_test_norm, predicted_price_norm)
print(f"Mean Squared Error: {mse}")
rmse = math.sqrt(mse)
print(f"Root Mean Squared Error: {rmse}")
```

```
plt.plot(actual_price, color = "black", label=f"Actual {company} Price")
plt.plot(predicted_price, color = "green", label=f"Predicted {company} Price")
plt.title(f"{company} Share Price")
plt.xlabel('Time')
plt.ylabel(f"{company} Share Price")
plt.legend()
plt.show()

real_data = [model_inputs[len(model_inputs) + 1 - prediciton_days:len(model_inputs+1),
0]]
real_data = np.array(real_data)
real_data = np.reshape(real_data, (real_data.shape[0], real_data.shape[1], 1))

prediction = model.predict(real_data)
prediction = scaler.inverse_transform(prediction)
print(f"Prediction: {prediction}") #price of company the next day
```

IV. Results

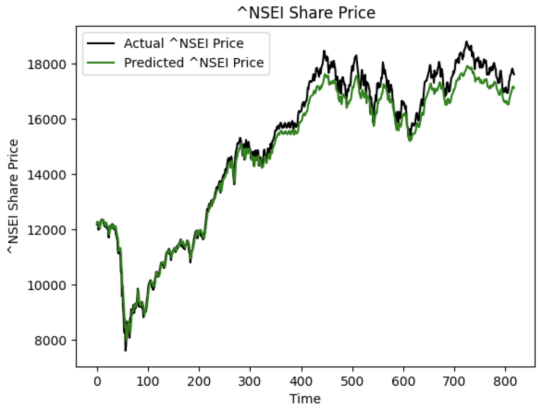
Using 3 different Epoch Values and comparing their MSE values. We tried using the NIFTY 50 as input for comparison and testing but the NIFTY 50 data is in points and not price.

1. NSEI
   A. 25 Epochs



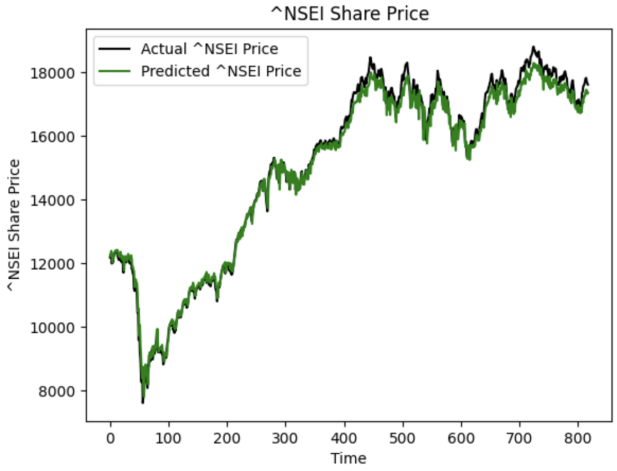

```
26/26 [==============================] - 1s 6ms/step
Mean Squared Error: 0.0025225774119540228
Root Mean Squared Error: 0.05022526666881942
```

B. 50 Epochs




```
Mean Squared Error: 0.0033478867650066626
Root Mean Squared Error: 0.057860926064198646
```

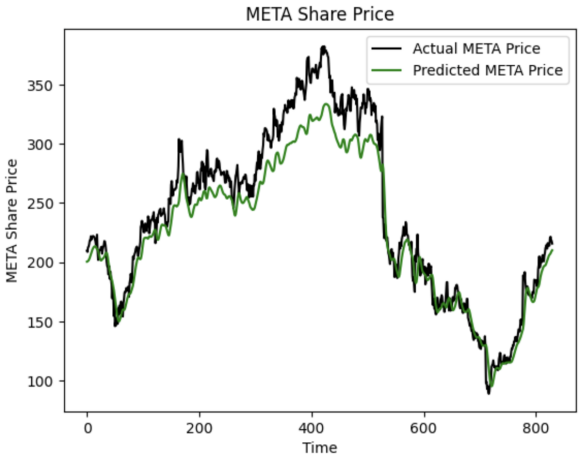
C. 75 Epochs




```
26/26 [==============================] - 1s 5ms/step
Mean Squared Error: 0.0013124470016785512
Root Mean Squared Error: 0.03622771041176286
```

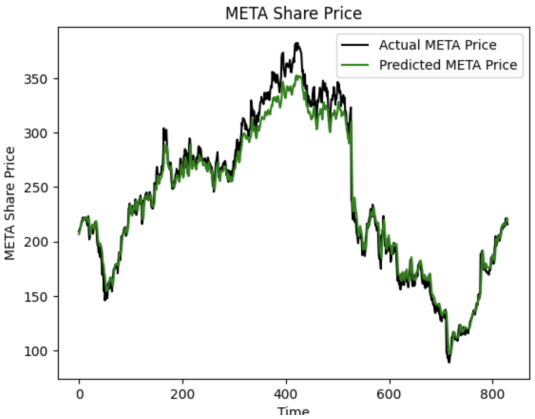
2. Meta

A. 25 Epochs



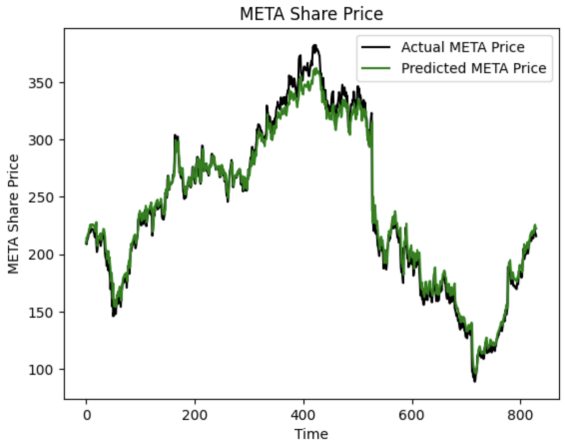

```
26/26 [==============================] - 1s 6ms/step
Mean Squared Error: 0.011542780407076528
Root Mean Squared Error: 0.1074373324644489
```

B. 50 Epochs




```
26/26 [==============================] - 1s 4ms/step
Mean Squared Error: 0.0033698534808805683
Root Mean Squared Error: 0.05805043911014428
```

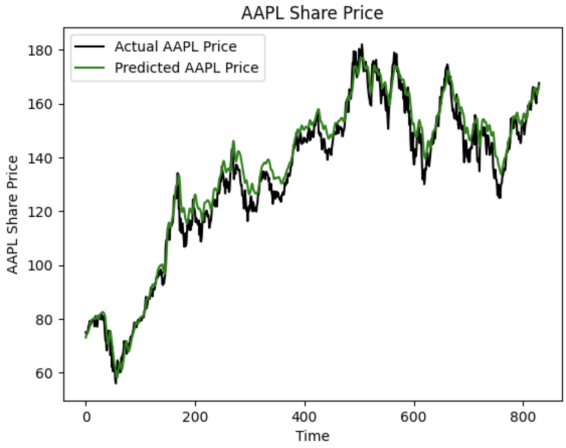
C. 75 Epochs




```
Mean Squared Error: 0.0021276028141056593
Root Mean Squared Error: 0.04612594512967359
```

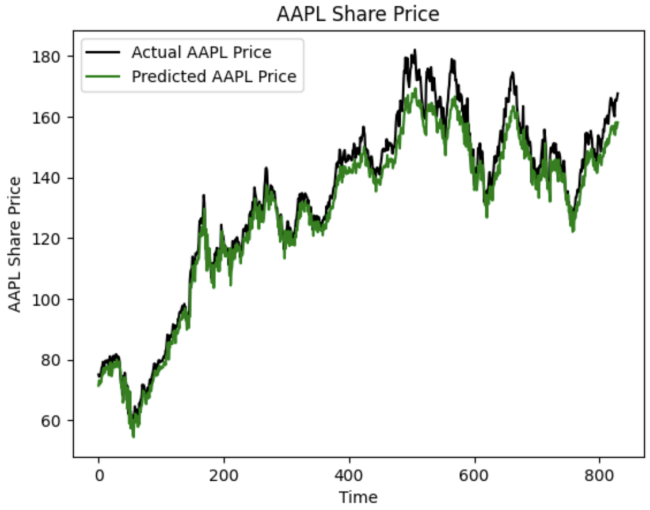
3. Apple

A. 25 Epochs



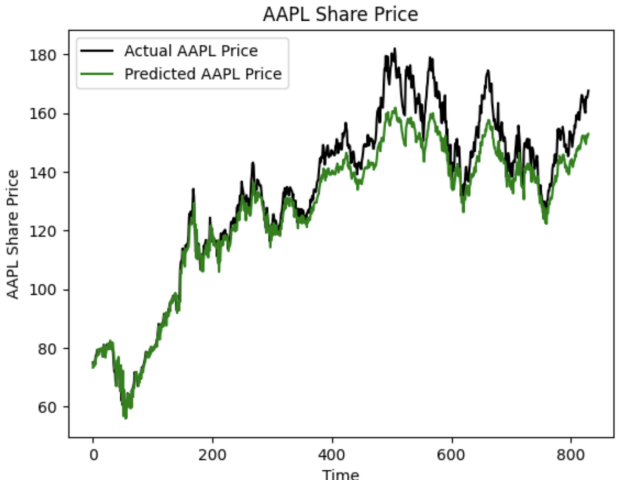

```
26/26 [==============================] - 1s 6ms/step
Mean Squared Error: 0.008115420494040997
Root Mean Squared Error: 0.09008562867650421
```

B. 50 Epochs




```
26/26 [==============================] - 1s 6ms/step
Mean Squared Error: 0.009399935916127023
Root Mean Squared Error: 0.09695326666042266
```

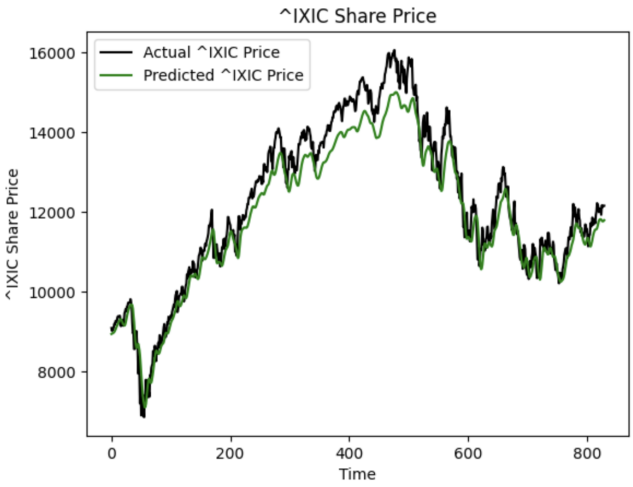
C. 75 Epochs




```
26/26 [==============================] - 1s 6ms/step
Mean Squared Error: 0.01814930150757313
Root Mean Squared Error: 0.13471934347959513
```

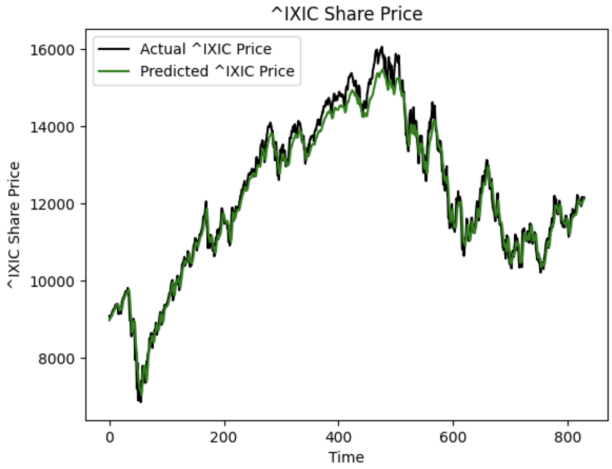
4. Nasdaq (POINTS/ not price)

A. 25 Epochs



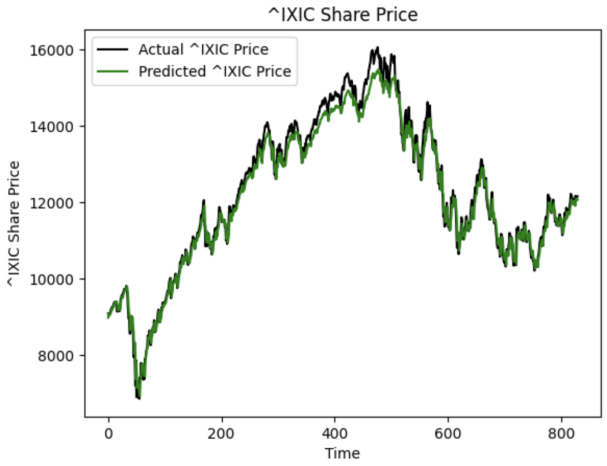

```
26/26 [==============================] - 1s 6ms/step
Mean Squared Error: 0.006600273412959821
Root Mean Squared Error: 0.08124206676937645
```

B. 50 Epochs




```
26/26 [==============================] - 1s 6ms/step
Mean Squared Error: 0.0019419060430372257
Root Mean Squared Error: 0.04406706301805494
```

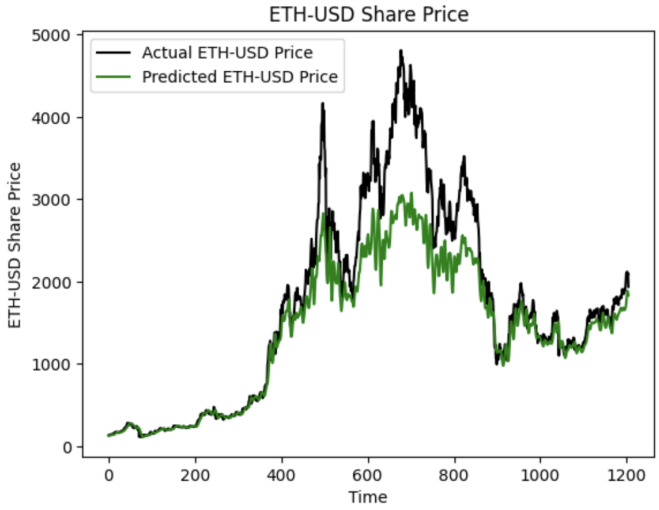
C. 75 Epochs




```
Mean Squared Error: 0.0018212571069578 79
Root Mean Squared Error: 0.042676188992901876
```

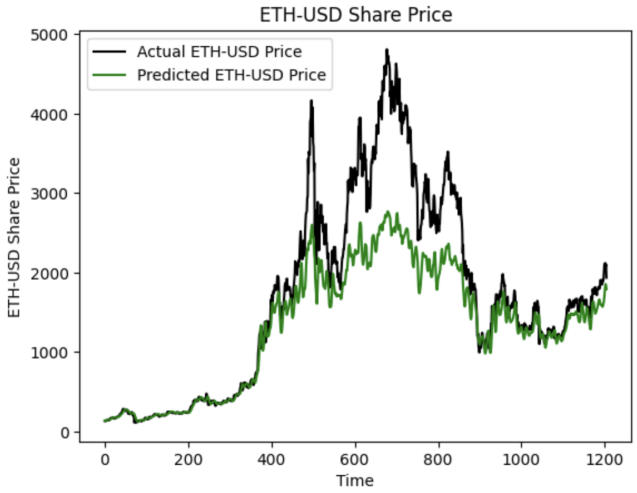
5. Ethereum - USD

A. 25 Epochs



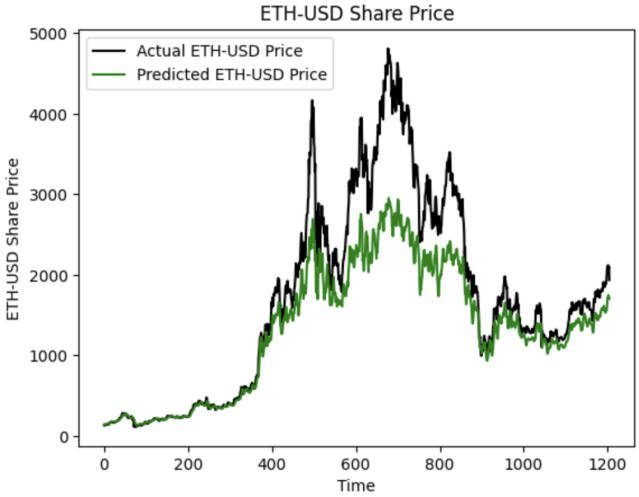

```
38/38 [==============================] - 2s 8ms/step
Mean Squared Error: 0.1520974616673925
Root Mean Squared Error: 0.38999674571384885
```

B. 50 Epochs




```
38/38 [==============================] - 1s 7ms/step
Mean Squared Error: 0.21971739939789223
Root Mean Squared Error: 0.46874022592251696
```

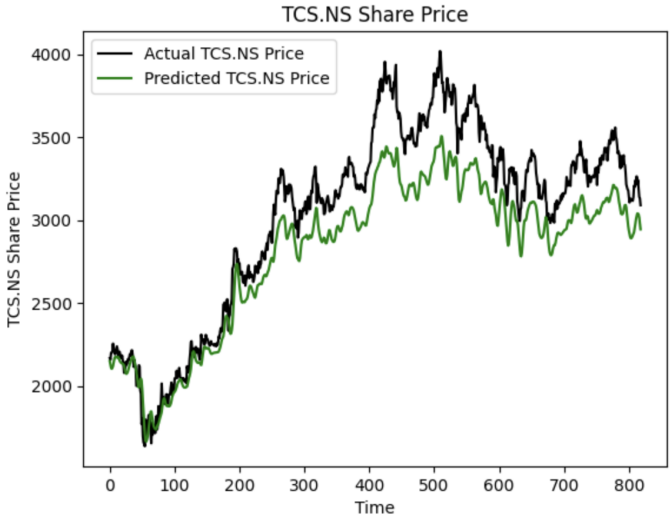
C. 75 Epochs




```
38/38 [==============================] - 1s 4ms/step
Mean Squared Error: 0.1991866474860711
Root Mean Squared Error: 0.4463033133263421
```

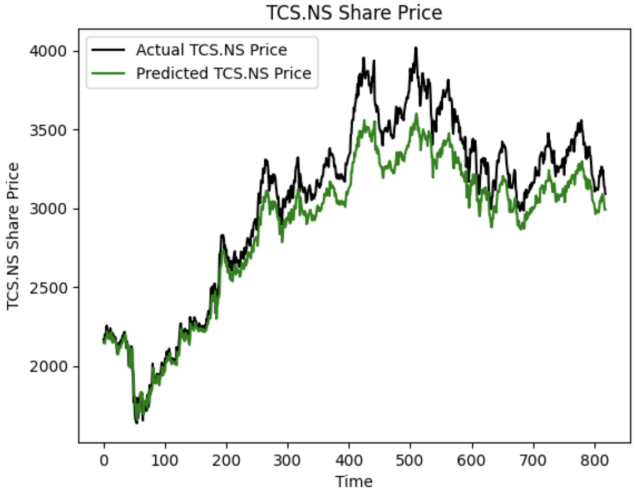
6. Tata Consultancy

A. 25 Epochs



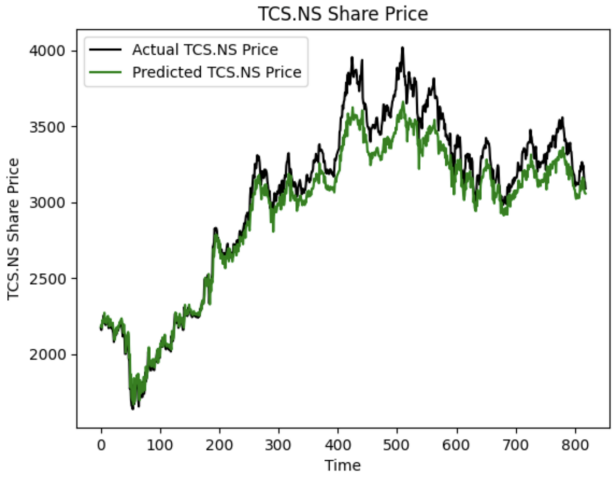

```
26/26 [==============================] - 1s 5ms/step
Mean Squared Error: 0.021352340927674986
Root Mean Squared Error: 0.14612440223205359
```

B. 50 Epochs




```
26/26 [==============================] - 1s 6ms/step
Mean Squared Error: 0.011975415078387713
Root Mean Squared Error: 0.1094322396663237
```

C. 75 Epochs




```
26/26 [==============================] - 1s 6ms/step
Mean Squared Error: 0.0069986711721658525
Root Mean Squared Error: 0.08365806101127286
```

Some other Nifty 50 Companies:

Other companies with **75 epochs** (scroll to see the graphs) - https://colab.research.google.com/drive/1HdVQUJSH1RXk_n1FH3kj3149WyUi0j5C?usp=sharing

Other companies with **50 epochs** (scroll to see the graphs) - https://colab.research.google.com/drive/1XAdyXZS89xCDsP-54De5Nrx613g6RycP?usp=sharing

Other companies with **25 epochs** (scroll to see the graphs) - https://colab.research.google.com/drive/1rPxzlne48fehB1slXdJsbwRplcWCyFGF?usp=sharing

8. Inferences

A. Linear Regression

Based on the results, a linear regression model has some limitations when it comes to accurately predicting stock prices based on the number of days and closing price. From our understanding, we thought these could be the possible reasons:

- Volatility: Stock prices can be highly volatile and subject to sudden changes based on a variety of factors, including news events, market sentiment, and investor behavior. These factors can be difficult to predict, making it challenging to accurately forecast future prices.
- Non-linearity: While the relationship between the number of days and closing price may be linear over short time periods, it may not hold over longer periods or during periods of high volatility. In these cases, a linear regression model may not be able to capture the full complexity of the relationship between the variables.
- Confounding variables: Other factors can also influence stock prices, including macroeconomic conditions, interest rates, company financials, and industry trends. These factors may be difficult to control for in a linear regression model, which could lead to inaccurate predictions.

B. Recurrent Neural Network

Our findings show that the MSE consistently decreases from 25 to 75 epochs as the epoch values rise. This suggests that the model performs better when the number of epochs is increased. With larger epoch values, the performance advantage is less noticeable yet. This shows that there might be a ceiling above which raising epoch values no longer results in appreciable performance gains.

Our research shows that raising the epoch values can enhance a machine learning model's performance, as seen by a drop in MSE. This improvement is considerable up to a certain point, but after that, overfitting causes the model's performance to stagnate or even decline. Looking at the companies, if you see that when the EPOCHs fluctuate for cryptocurrency, the MSE value increases meaning the modeling is performing worse. This may be because crypto currencies are highly fluctuating and are not regulated. Whereas some companies have good performance with lesser epochs values for example APPLE has better performance with lower epochs whereas Tata Consultancy has better performance with high EPOCHS. As we can see in the graphs of Tata Consultancy, the predicted price comes closer and closer to the original price.

This emphasizes how crucial it is to choose the right number of epochs in order to strike a balance between model complexity and generalization. Further investigation is required to establish the ideal number of epochs for various models and datasets, as well as to investigate further methods for reducing the dangers of overfitting.

9. Conclusion

As seen with the above tests and graphs that were used to compare the performance of linear regression and RNN models, stock prices were predicted rather well using the closing price as the dependent variable. Both models were trained and evaluated using historical stock price data, and their performance was measured using appropriate evaluation metrics such as MSE and RSM.

Our analysis showed that the RNN model outperformed the linear regression model in predicting stock prices for a specified number of days into the future. The RNN model was able to capture the complex patterns and long-term dependencies in the data, which made it more accurate in predicting the future closing prices of stocks.

On the other hand, linear regression is a simpler model that assumes a linear relationship between the dependent and independent variables. While it is a useful tool for predicting stock prices, its performance may be limited by the linear assumption and the inability to capture non-linear patterns in the data.

In conclusion, the choice of model depends on much more than just a few factors, including the type of data, the complexity of the problem, and the accuracy required. While both models have their own strengths and weaknesses, our analysis shows that RNNs can be more accurate in predicting stock prices, especially for long-term predictions.

10. References

Zhang, B. L., Song, J. M., & Zhou, Y. M. (2021). Comparison of artificial neural network and support vector machine models for predicting stock prices using technical indicators. IEEE Xplore. https://ieeexplore.ieee.org/document/9848887/

Lee, J., & Park, S. (2013). A new approach to predicting stock price direction using support vector machines and recurrent neural networks. Applied Soft Computing, 13(5), 2605-2614.

EasyChair. (2022). An empirical study on predictive maintenance using machine learning for mechanical systems [Online]. Easychair.org. https://easychair.org/publications/preprint/rD6d

yfinance. (n.d.). PyPI. Retrieved April 23, 2023, from https://pypi.org/project/yfinance/

National Stock Exchange of India Limited. (n.d.). from https://www.nseindia.com/