Forecasting a Stock Price - IBM Advanced Data Science Capstone Project.

by Wany Fourreau.

Forecasting a Stock Price using Linear Models such as LinearRegression, K-Nearest Neighbor(KNN), Support Vector Machine(SVM) and using RNN-LSTM as a Time Series Forecasting Model.

Disclaimer: The reader is not encouraged to invest in the stock market or attempt to trade based upon anything that is said or any models that the author has discussed in the paper. The equity market is a risky place and randomness is the best guarantee. This is simply an attempt to understand Advanced Data Science Modeling Process.

Introduction

A short understanding of my interpretation of market psyche.

Why should we try to predict a stock price?
In an attempt to minimize risk and maximize profit investors and traders alike want to have an edge. Within that philosophy predictive models are created to serve as forecasting indicators which are assumed to provide that edge. In the financial world or to be more precise, at a trading firm, the term often used is "technical analysis" for predicting short term moves, for example in a stock, an index, a commodity or others. The term "Fundamental analysis" is used as well but it is usually applied to understanding the health of a company and for long term investments. Technical analysis is often used as short-term indicators to give the trader an edge. In my personal view even when a model does not work a trader wants to have one to follow because it makes the transaction appear less random. It also serves as a tool to keep emotions out of the trade. After all trading is an emotional affair but do pretend I did not tell you that since this is a scientific project for a data science class.

The Modeling Aspect - The Edge

Predictive models not always but are often applied as short-term indicators. Many models are available in Machine Learning which can serve as stock market forecasting tools. What we must understand is no models can take the place of good governance over money management. Here however in my Advanced Data Science Capstone Project I would like to see how a few Machine Learning predictive models work in comparison to a stock's actual price performance. My goal is to submit a Linear Regression model, a K-Nearest Neighbor model, a Support Vector Machine model and an RNN-LSTM Time Series Forecasting model using the company Amazon.com stock's historical price data.

An overall understanding about the models I have selected.
I will make the answers as short and as simple as possible. If you desire deeper input please go to wikipedia.com or a data science/machine learning website like Towardsdatasience.com or machinelearningmastery.com

What is a Linear Regression Model?
To give you a simple answer. Linear regression makes an attempt at modeling the relationship between two variables by fitting a linear equation to observed data where one variable is considered to be an explanatory variable and the other is considered to be a dependent variable. It is the most basic and commonly used type of predictive analysis.

What is a K-Nearest Neighbor (KNN) Model?
Medium.com put it in the simplest term possible. KNN is a supervised learning classification and regression algorithm that uses nearby points in order to generate a prediction. It is said to be one of the most basic yet essential classification algorithms in Machine Learning.

What is a Support Vector Machine(SVM) model?
Similar to the KNN model, SVM is a supervised machine learning model that can be used for classification and regression. SVM is effective in high dimensional spaces. It is effective in cases where the number of dimensions is greater than the number of samples. It is memory efficient and is versatile.

What is a RNN-LSTM Time Series Forecasting model?
RNN stands for recurrent neural network. LSTM stands for long short-term memory network. Since time series prediction problems are difficult types of predictive modeling problems, recurrent neural networks are used in deep learning to train very large architectures successfully.

Files Organization

- **Forecasting a Stock Move Steps_1_to_5.inpg -** The file contains steps 1 through step 5. It shows the viewer where the Source of data is. How it is Extracted, Transformed and Loaded. What feature engineering took place before modeling is created and compiled**.**
- **Forecasting a Stock Move Linear Models.inpg -** The file focuses on the modeling process of the Linear Models. It provides the viewer with steps 1 to 5 in case the viewer escapes that file then it continues to model building, evaluation, performance and observation.
- **Forecasting a Stock Move with LSTM Time Series For.inpg -** The interesting thing with this deep learning model is that you can actually use raw data. Whereas with the linear model certain modifications with the date had to be made, here that does not apply. In model building, training and evaluation access to tensor flow for backend and keras for the compiling is needed.
- **Forecasting a Stock Move with LSTM Time Series For copy1.inpg** - **Can I improve the Performance of the RNN- LSTM Model ? -** This file is where I focus on trying to improve the LSTM model by making changes on how it is compiled to train. The major change is doubling the number of epochs from 50 to 100.

- **Forecasting a Stock Move with LSTM Time Series For copy2.inpg** - Since I am recommending the RNN-LSTM model here I am using a new set of recent data to test the model again. I used that new data as well in the file above which is **Forecasting a Stock Move with LSTM Time Series For copy1.inpg**. I want to see how the new test set compares from the original model to the performance it shows in the improved model.
- **Model Evaluation Metrics.pdf -** This file shows the evaluation metrics for the four models.
- **Model Deployment.pdf -** Deploying my preferred model
- **Data Product.pdf** - An overall summarization of the project.
- **The ADD** - Work documentation.
- **Video Presentation**

Advanced Data Science Capstone Project Process Model.

Initial Data Exploration

The user case I would like to implement is "Forecasting a Stock Price using Linear Models and an RNN LSTM Time Series Forecasting Model "

I will be using the historical price data of Amazon.com which goes by the ticker AMZN.

Work Documentation or Contents:

(Scroll this document)

AdvanceDataScienceCapstone_Forecasting a Stock Price.Introduction<pages 1-3>
AdvanceDataScienceCapstone_Forecasting a Stock Price .data.exp.<page 3>
AdvanceDataScienceCapstone_Forecasting a Stock Price.etl.<pages 4-6>
AdvanceDataScienceCapstone_Forecasting a Stock Price.feature_eng.<pages 6-9>
AdvanceDataScienceCapstone_Forecasting a Stock Price.model_def.<pages 9-11>
AdvanceDataScienceCapstone_Forecasting a StockPrice.model_train.<LinearRegression, KNN, SVM, RNN LSTM Time Series Forecasting><pages 11-15>
AdvanceDataScienceCapstone_Forecasting a Stock Price.model_evaluate.<pages 15-20>
AdvanceDataScienceCapstone_Forecasting a Stock Price.model_deployment.<pages 20-22>

Data is scraped using the yahoo.finance API

```
# Working with AMAZON.COM
#Total data set starts '2009-04-01' ends '2021-03-31'(3020,7)
#Train data set starts '2009-04-01' ends '2018-03-31'(1812,7)
#Test data set starts '2018-04-01' ends '2021-03-31'(755,7)
#validation data is set at 20 percent (453,7)
```

Extract, Transform, Load (ETL)
Historical data price on Amazon.com starting date 2009-04-01, ending date 2021-03-31

Load Dataset:

	Open	**High**	**Low**	**Close**	**Volume**	**Dividends**	**Stock Splits**
Date							
2009-04-01	73.019997	75.089996	71.709999	73.500000	7041400	0	0
2009-04-02	73.629997	77.239998	73.440002	76.339996	11066900	0	0
2009-04-03	76.419998	78.320000	75.500000	78.169998	5809900	0	0
2009-04-06	77.260002	78.360001	76.000000	77.989998	5751300	0	0
2009-04-07	76.970001	77.080002	74.879997	75.510002	5748800	0	0

	Open	**High**	**Low**	**Close**	**Volume**	**Dividends**	**Stock Splits**
Date							
2021-03-24	3151.040039	3160.310059	3085.149902	3087.070068	2959000	0	0
2021-03-25	3072.989990	3109.780029	3037.139893	3046.260010	3563500	0	0
2021-03-26	3044.060059	3056.659912	2996.000000	3052.030029	3306700	0	0
2021-03-29	3055.439941	3091.250000	3028.449951	3075.729980	2746000	0	0
2021-03-30	3070.010010	3073.000000	3034.000000	3055.290039	2337600	0	0

(3020, 7)

```
Open            float64
High            float64
Low             float64
Close           float64
Volume            int64
Dividends         int64
Stock Splits      int64
dtype: object
```

Data Wrangling

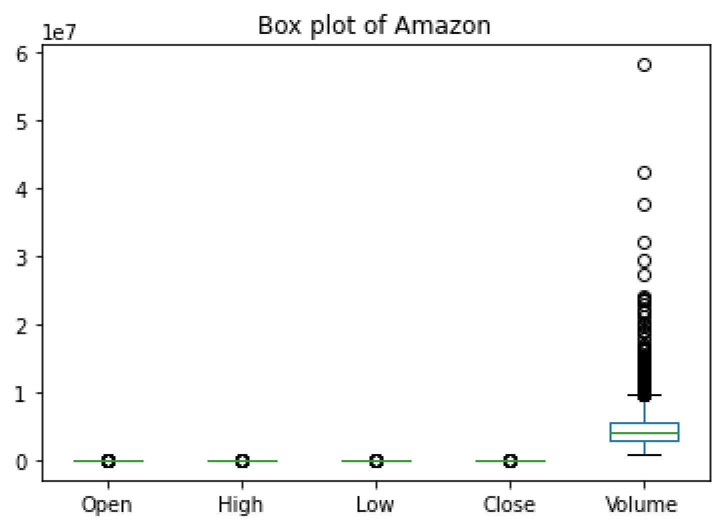
Data Cleaning:

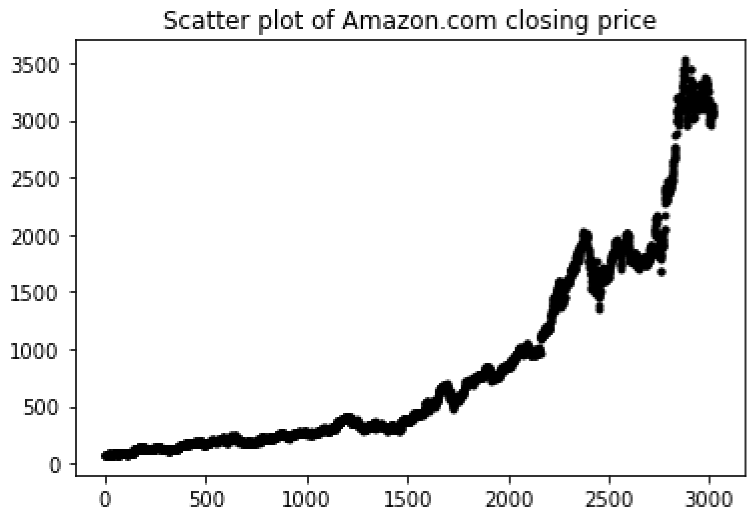
	Date	Open	High	Low	Close	Volume
0	2009-04-01	73.019997	75.089996	71.709999	73.500000	7041400
1	2009-04-02	73.629997	77.239998	73.440002	76.339996	11066900
2	2009-04-03	76.419998	78.320000	75.500000	78.169998	5809900
3	2009-04-06	77.260002	78.360001	76.000000	77.989998	5751300
4	2009-04-07	76.970001	77.080002	74.879997	75.510002	5748800

Data Describe

	Open	High	Low	Close	Volume
count	3020.000000	3020.000000	3020.000000	3020.000000	3.020000e+03
mean	857.271172	866.373522	846.929987	856.993414	4.764370e+06
std	869.753037	879.527268	858.165551	868.878627	3.089278e+06
min	73.019997	75.089996	71.709999	73.500000	8.813000e+05
25%	217.407497	219.577503	214.312500	217.450001	2.917250e+06
50%	398.070007	399.850006	393.130005	397.600006	4.008150e+06
75%	1488.180023	1521.279999	1475.910004	1495.932556	5.657050e+06
max	3547.000000	3552.250000	3486.689941	3531.449951	5.830580e+07

Feature Engineering - Exploratory Data Analysis



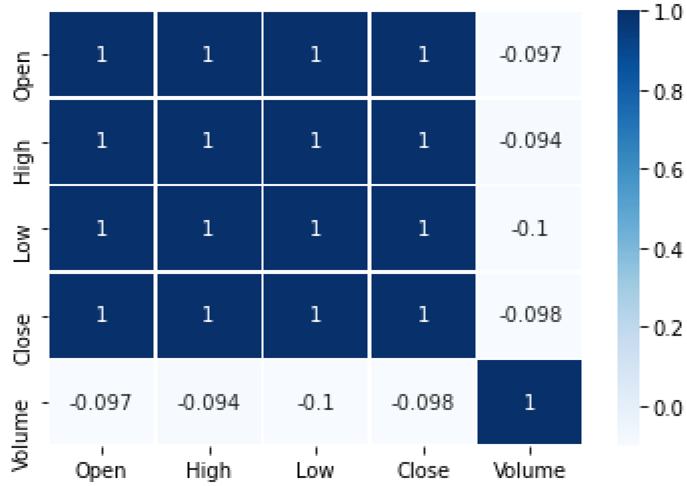


Measuring Correlation:
The range of value for Pearson Correlation is between -1 to 1. It is a popular way of measuring for correlation. If two features are positively correlated then they are directly proportional and if they are negatively correlated then they are inversely proportional.

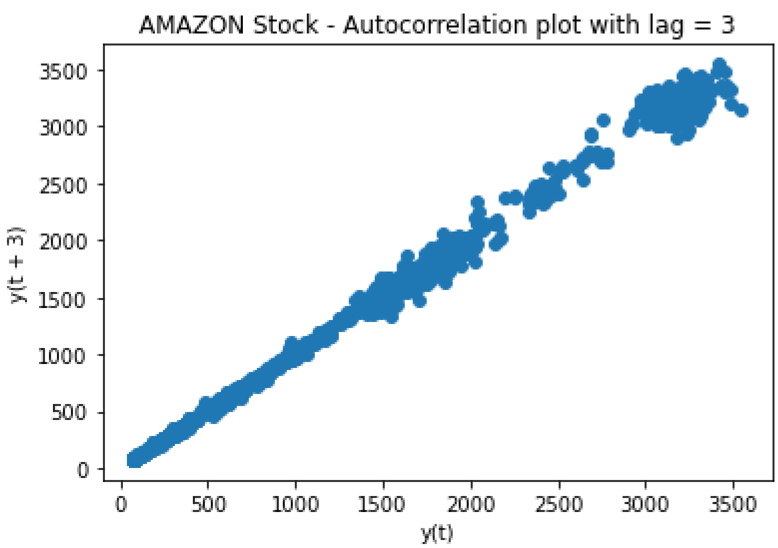
	Open	**High**	**Low**	**Close**	**Volume**
Open	1.000000	0.999892	0.999852	0.999742	-0.097273
High	0.999892	1.000000	0.999819	0.999871	-0.094226
Low	0.999852	0.999819	1.000000	0.999882	-0.101377
Close	0.999742	0.999871	0.999882	1.000000	-0.097886
Volume	-0.097273	-0.094226	-0.101377	-0.097886	1.000000

Let us use the Pearson Correlation Map to better understand feature correlation which is high.

<matplotlib.axes._subplots.AxesSubplot at 0x7f9cf6b01dd0>

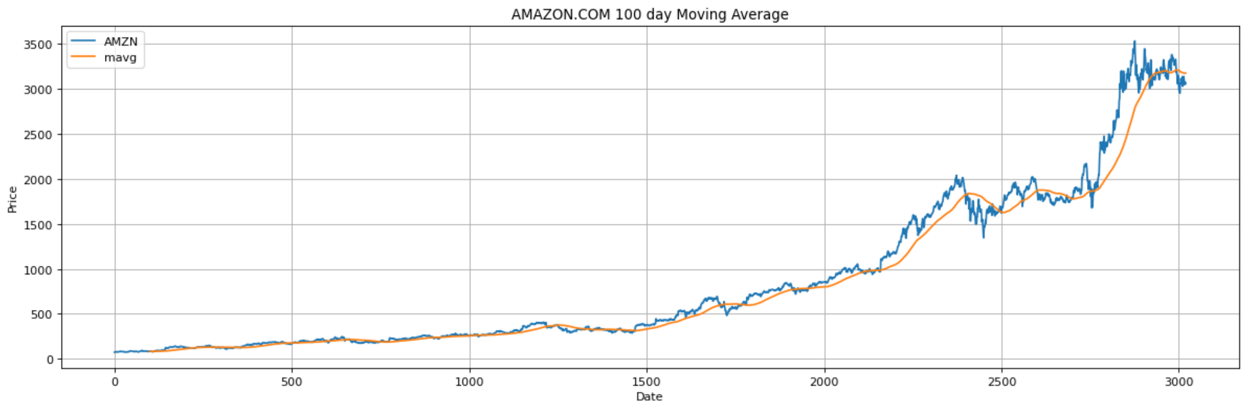


Plot Cross-correlation



The Rolling Mean (Moving Average)

Moving average helps us smooth out data that has a lot of fluctuation and helps us see the long term trend better.

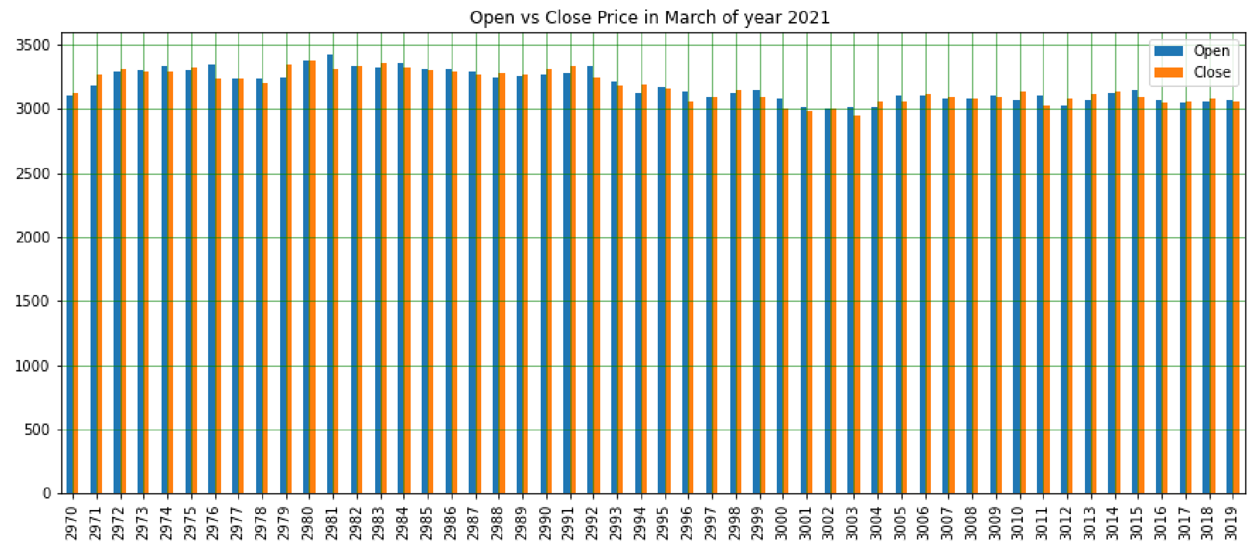


Model Definition

Reshape the Data for Linear Models: LinearRegression, K-Nearest Neighbor(KNN), Support Vector Machine(SVM)

	Date	Open	High	Low	Close
0	2009-04-01	73.019997	75.089996	71.709999	73.500000
1	2009-04-02	73.629997	77.239998	73.440002	76.339996
2	2009-04-03	76.419998	78.320000	75.500000	78.169998
3	2009-04-06	77.260002	78.360001	76.000000	77.989998
4	2009-04-07	76.970001	77.080002	74.879997	75.510002

Below we will visualize the open price vs the close price.
The Open price is the first quote at 9.30am when the equity market is officially opened. The Close price is the last quote at 4pm when the market is officially closed. There is pre-market trading which takes place before the open and after-market trading which takes place after 4pm however those quotes are not reflected here. The historical market data we are focusing on is based on official market ticks.



To avoid errors the Date format will be changed to separate Year, Month, Day

	Day	Month	Year	High	Open	Low	Close
0	1	4	2009	75.089996	73.019997	71.709999	73.500000
1	2	4	2009	77.239998	73.629997	73.440002	76.339996
2	3	4	2009	78.320000	76.419998	75.500000	78.169998
3	6	4	2009	78.360001	77.260002	76.000000	77.989998
4	7	4	2009	77.080002	76.970001	74.879997	75.510002

	Day	Month	Year	High	Open	Low	Close
3015	24	3	2021	3160.310059	3151.040039	3085.149902	3087.070068
3016	25	3	2021	3109.780029	3072.989990	3037.139893	3046.260010
3017	26	3	2021	3056.659912	3044.060059	2996.000000	3052.030029
3018	29	3	2021	3091.250000	3055.439941	3028.449951	3075.729980

3019	30	3	2021	3073.000000	3070.010010	3034.000000	3055.290039

Splitting the data into training set and test set
(3020, 6)
(3020,)

(2265, 6)
(2265,)
(755, 6)
(755,)

Creating and Training the Linear Models
LinearRegression, KNN-K Nearest Neighbor, SVM-Support Vector Machine

```
# Linear Regression Model Training and Testing
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_predict=lr_model.predict(X_test)

# SVM Model Training and Testing
svm_regressor = SVR(kernel='linear')
svm_model=svm_regressor.fit(X_train, y_train)
y_svm_predict=svm_model.predict(X_test

# KNN Model Training and Testing
knn_regressor=KNeighborsRegressor(n_neighbors=4)
knn_model=knn_regressor.fit(X_train, y_train)
y_knn_predict=knn_model.predict(X_test)
```

 # Define the Lstm model
(Recurrent Neural Network Long short term Memory Time Series Forecasting)

```
lstm_model = Sequential()
lstm_model.add(LSTM(units = 200, return_sequences = True, input_shape = (X_train.shape[1], 1)))
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(units = 50))
lstm_model.add(Dropout(0.2))
```

lstm_model.add(Dense(1))

lstm_model.compile(loss='mean_squared_error', optimizer='adam')
lstm_model.summary()

 # Model Training
 # Compile
history = lstm_model.fit(X_train, y_train, epochs=50, validation_split=0.20, batch_size=32, shuffle=False)

From the training data set I have written an automatic set up for a 20 percent validation split so later I could visualize how the losses fit over the epochs.

Output:

Model: "sequential"

```
Layer (type)             Output Shape            Param #
=================================================================
lstm (LSTM)               (None, 50, 200)          161600

dropout (Dropout)          (None, 50, 200)          0

lstm_1 (LSTM)             (None, 50)               50200

dropout_1 (Dropout)         (None, 50)               0

dense (Dense)             (None, 1)                51
=================================================================
Total params: 211,851
Trainable params: 211,851
Non-trainable params: 0

Epoch 1/50
56/56 [==============================] - 14s 243ms/step - loss: 6.3986e-04 - val_loss: 0.0058
Epoch 2/50
56/56 [==============================] - 12s 218ms/step - loss: 0.0022 - val_loss: 0.0301
Epoch 3/50
56/56 [==============================] - 12s 217ms/step - loss: 0.0021 - val_loss: 0.0115
Epoch 4/50
56/56 [==============================] - 12s 215ms/step - loss: 0.0014 - val_loss: 0.0130
Epoch 5/50
56/56 [==============================] - 12s 216ms/step - loss: 0.0018 - val_loss: 0.0170
Epoch 6/50
56/56 [==============================] - 12s 215ms/step - loss: 0.0018 - val_loss: 0.0167
```

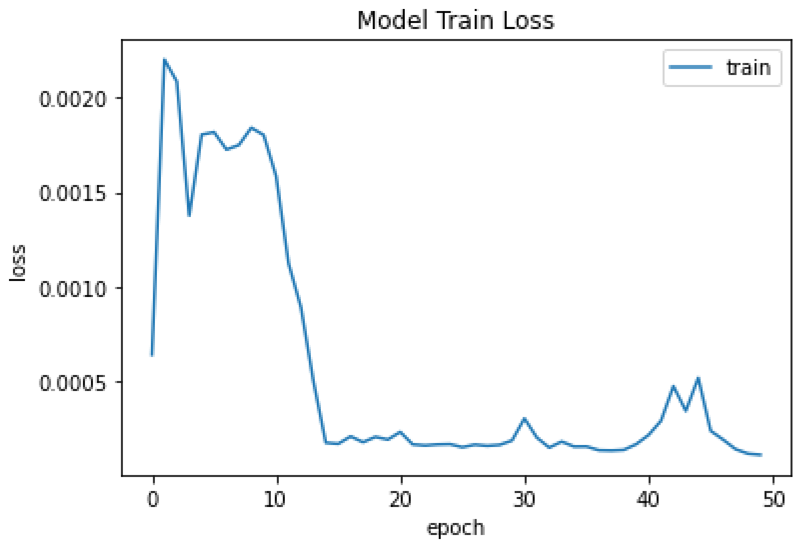
Epoch 7/50
56/56 [==============================] - 12s 219ms/step - loss: 0.0017 - val_loss: 0.0141
Epoch 8/50
56/56 [==============================] - 12s 216ms/step - loss: 0.0017 - val_loss: 0.0106
Epoch 9/50
56/56 [==============================] - 12s 219ms/step - loss: 0.0018 - val_loss: 0.0110
Epoch 10/50
56/56 [==============================] - 12s 215ms/step - loss: 0.0018 - val_loss: 0.0062
Epoch 11/50
56/56 [==============================] - 12s 217ms/step - loss: 0.0016 - val_loss: 0.0063
Epoch 12/50
56/56 [==============================] - 12s 217ms/step - loss: 0.0011 - val_loss: 0.0046
Epoch 13/50
56/56 [==============================] - 12s 215ms/step - loss: 8.9055e-04 - val_loss: 0.0028
Epoch 14/50
56/56 [==============================] - 12s 218ms/step - loss: 5.0147e-04 - val_loss: 0.0032
Epoch 15/50
56/56 [==============================] - 12s 215ms/step - loss: 1.7450e-04 - val_loss: 0.0033
Epoch 16/50
56/56 [==============================] - 12s 218ms/step - loss: 1.6880e-04 - val_loss: 0.0027
Epoch 17/50
56/56 [==============================] - 12s 219ms/step - loss: 2.0823e-04 - val_loss: 0.0034
Epoch 18/50
56/56 [==============================] - 12s 219ms/step - loss: 1.7727e-04 - val_loss: 0.0030
Epoch 19/50
56/56 [==============================] - 12s 217ms/step - loss: 2.0566e-04 - val_loss: 0.0027
Epoch 20/50
56/56 [==============================] - 12s 216ms/step - loss: 1.9138e-04 - val_loss: 0.0038
Epoch 21/50
56/56 [==============================] - 12s 218ms/step - loss: 2.3153e-04 - val_loss: 0.0017
Epoch 22/50
56/56 [==============================] - 12s 217ms/step - loss: 1.6557e-04 - val_loss: 0.0025
Epoch 23/50
56/56 [==============================] - 12s 217ms/step - loss: 1.6025e-04 - val_loss: 0.0026
Epoch 24/50

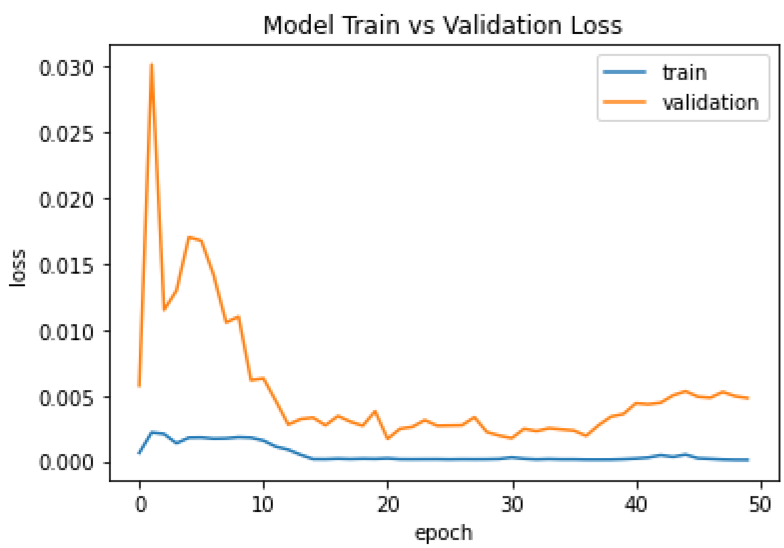
```
56/56 [==============================] - 12s 214ms/step - loss: 1.6543e-04 - val_loss:
0.0031
Epoch 25/50
56/56 [==============================] - 12s 216ms/step - loss: 1.6716e-04 - val_loss:
0.0027
Epoch 26/50
56/56 [==============================] - 12s 215ms/step - loss: 1.5083e-04 - val_loss:
0.0027
Epoch 27/50
56/56 [==============================] - 12s 218ms/step - loss: 1.6433e-04 - val_loss:
0.0027
Epoch 28/50
56/56 [==============================] - 12s 217ms/step - loss: 1.5830e-04 - val_loss:
0.0034
Epoch 29/50
56/56 [==============================] - 12s 217ms/step - loss: 1.6313e-04 - val_loss:
0.0022
Epoch 30/50
56/56 [==============================] - 12s 217ms/step - loss: 1.8641e-04 - val_loss:
0.0020
Epoch 31/50
56/56 [==============================] - 12s 216ms/step - loss: 3.0296e-04 - val_loss:
0.0018
Epoch 32/50
56/56 [==============================] - 12s 217ms/step - loss: 2.0194e-04 - val_loss:
0.0025
Epoch 33/50
56/56 [==============================] - 12s 216ms/step - loss: 1.4881e-04 - val_loss:
0.0023
Epoch 34/50
56/56 [==============================] - 12s 214ms/step - loss: 1.8064e-04 - val_loss:
0.0025
Epoch 35/50
56/56 [==============================] - 12s 217ms/step - loss: 1.5418e-04 - val_loss:
0.0024
Epoch 36/50
56/56 [==============================] - 12s 214ms/step - loss: 1.5467e-04 - val_loss:
0.0023
Epoch 37/50
56/56 [==============================] - 12s 216ms/step - loss: 1.3484e-04 - val_loss:
0.0019
Epoch 38/50
56/56 [==============================] - 12s 215ms/step - loss: 1.3321e-04 - val_loss:
0.0028
Epoch 39/50
```

56/56 [==============================] - 12s 214ms/step - loss: 1.3662e-04 - val_loss: 0.0034
Epoch 40/50
56/56 [==============================] - 12s 215ms/step - loss: 1.6670e-04 - val_loss: 0.0036
Epoch 41/50
56/56 [==============================] - 12s 214ms/step - loss: 2.1558e-04 - val_loss: 0.0044
Epoch 42/50
56/56 [==============================] - 12s 219ms/step - loss: 2.9044e-04 - val_loss: 0.0043
Epoch 43/50
56/56 [==============================] - 12s 213ms/step - loss: 4.7389e-04 - val_loss: 0.0045
Epoch 44/50
56/56 [==============================] - 12s 216ms/step - loss: 3.4266e-04 - val_loss: 0.0050
Epoch 45/50
56/56 [==============================] - 12s 215ms/step - loss: 5.1796e-04 - val_loss: 0.0053
Epoch 46/50
56/56 [==============================] - 12s 216ms/step - loss: 2.3772e-04 - val_loss: 0.0049
Epoch 47/50
56/56 [==============================] - 12s 217ms/step - loss: 1.9114e-04 - val_loss: 0.0048
Epoch 48/50
56/56 [==============================] - 12s 218ms/step - loss: 1.4078e-04 - val_loss: 0.0053
Epoch 49/50
56/56 [==============================] - 12s 216ms/step - loss: 1.1688e-04 - val_loss: 0.0050
Epoch 50/50
56/56 [==============================] - 12s 215ms/step - loss: 1.1060e-04 - val_loss: 0.0048

<tensorflow.python.keras.callbacks.History at 0x7f6902c07390>

Post-training, plot the loss as:
Evaluation of how the model has trained over the epochs





It appears that the model has trained ok but there are a lot of ripples in the validation set. By the 40th epoch the validation and training loss over the epoch have reduced. However as I continued to rerunned the file to continue the work, the model has been taking longer to train and the loss over the epochs have been widening as you can see on the chart above.

 # Define the Lstm model for improved performance
(Trying to fit the train & validation loss over epoch. Increased epochs from 50 to 100)
From the training data set I have written an automatic set up for a 20 percent validation split so later I could visualize how the losses fit over the epochs. After having done further research I increased the number of epochs to 100 from 50 hoping for a better fit. Look at the plots below.

Model: "sequential"

```
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
lstm (LSTM)                 (None, 50, 200)           161600
_________________________________________________________________
dropout (Dropout)           (None, 50, 200)           0
_________________________________________________________________
lstm_1 (LSTM)               (None, 50)                50200
_________________________________________________________________
dropout_1 (Dropout)         (None, 50)                0
_________________________________________________________________
dense (Dense)               (None, 1)                 51
=================================================================
Total params: 211,851
Trainable params: 211,851
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
56/56 [==============================] - 15s 272ms/step - loss: 6.6543e-04 - val_loss: 0.0065
Epoch 2/100
56/56 [==============================] - 13s 236ms/step - loss: 0.0014 - val_loss: 0.0034
Epoch 3/100
56/56 [==============================] - 13s 240ms/step - loss: 0.0014 - val_loss: 0.0039
Epoch 4/100
56/56 [==============================] - 13s 226ms/step - loss: 6.7753e-04 - val_loss: 4.1746e-04
Epoch 5/100
56/56 [==============================] - 13s 224ms/step - loss: 3.4188e-04 - val_loss: 7.9842e-04
Epoch 6/100
56/56 [==============================] - 12s 219ms/step - loss: 3.1885e-04 - val_loss: 9.1225e-04
Epoch 7/100
56/56 [==============================] - 12s 218ms/step - loss: 3.4647e-04 - val_loss: 0.0038
Epoch 8/100
56/56 [==============================] - 13s 227ms/step - loss: 4.5662e-04 - val_loss: 0.0018
Epoch 9/100
```

```
56/56 [==============================] - 13s 232ms/step - loss: 3.2245e-04 - val_loss: 3.1135e-04
Epoch 10/100
56/56 [==============================] - 13s 235ms/step - loss: 3.0910e-04 - val_loss: 3.4119e-04
Epoch 11/100
56/56 [==============================] - 13s 233ms/step - loss: 7.3178e-04 - val_loss: 0.0160
Epoch 12/100
56/56 [==============================] - 13s 238ms/step - loss: 0.0015 - val_loss: 0.0166
Epoch 13/100
56/56 [==============================] - 14s 244ms/step - loss: 0.0017 - val_loss: 0.0104
Epoch 14/100
56/56 [==============================] - 14s 241ms/step - loss: 0.0017 - val_loss: 0.0145
Epoch 15/100
56/56 [==============================] - 13s 240ms/step - loss: 0.0018 - val_loss: 0.0220
Epoch 16/100
56/56 [==============================] - 14s 244ms/step - loss: 0.0017 - val_loss: 0.0113
Epoch 17/100
56/56 [==============================] - 13s 241ms/step - loss: 0.0019 - val_loss: 0.0108
Epoch 18/100
56/56 [==============================] - 14s 247ms/step - loss: 0.0016 - val_loss: 0.0050
Epoch 19/100
56/56 [==============================] - 14s 244ms/step - loss: 0.0014 - val_loss: 0.0070
Epoch 20/100
56/56 [==============================] - 14s 246ms/step - loss: 9.2826e-04 - val_loss: 0.0032
Epoch 21/100
56/56 [==============================] - 14s 251ms/step - loss: 7.2293e-04 - val_loss: 0.0018
Epoch 22/100
56/56 [==============================] - 13s 238ms/step - loss: 3.6631e-04 - val_loss: 0.0024
Epoch 23/100
56/56 [==============================] - 13s 238ms/step - loss: 1.7970e-04 - val_loss: 0.0018
Epoch 24/100
56/56 [==============================] - 13s 231ms/step - loss: 1.9414e-04 - val_loss: 0.0027
Epoch 25/100
56/56 [==============================] - 13s 235ms/step - loss: 2.2744e-04 - val_loss: 0.0023
Epoch 26/100
```

56/56 [==============================] - 13s 239ms/step - loss: 1.7127e-04 - val_loss: 0.0026
Epoch 27/100
56/56 [==============================] - 13s 240ms/step - loss: 1.5967e-04 - val_loss: 0.0026
Epoch 28/100
56/56 [==============================] - 13s 239ms/step - loss: 1.4843e-04 - val_loss: 0.0015
Epoch 29/100
56/56 [==============================] - 13s 239ms/step - loss: 1.6384e-04 - val_loss: 0.0017
Epoch 30/100
56/56 [==============================] - 13s 238ms/step - loss: 1.6595e-04 - val_loss: 0.0015
Epoch 31/100
56/56 [==============================] - 13s 234ms/step - loss: 1.5042e-04 - val_loss: 0.0022
Epoch 32/100
56/56 [==============================] - 13s 238ms/step - loss: 1.3558e-04 - val_loss: 0.0018
Epoch 33/100
56/56 [==============================] - 14s 244ms/step - loss: 1.8223e-04 - val_loss: 0.0026
Epoch 34/100
56/56 [==============================] - 14s 243ms/step - loss: 1.6080e-04 - val_loss: 0.0012
Epoch 35/100
56/56 [==============================] - 13s 240ms/step - loss: 1.5341e-04 - val_loss: 0.0030
Epoch 36/100
56/56 [==============================] - 14s 250ms/step - loss: 1.5784e-04 - val_loss: 0.0018
Epoch 37/100
56/56 [==============================] - 14s 243ms/step - loss: 1.4216e-04 - val_loss: 0.0018
Epoch 38/100
56/56 [==============================] - 13s 238ms/step - loss: 1.4342e-04 - val_loss: 0.0017
Epoch 39/100
56/56 [==============================] - 13s 240ms/step - loss: 1.5275e-04 - val_loss: 0.0026
Epoch 40/100

56/56 [==============================] - 14s 242ms/step - loss: 1.2694e-04 - val_loss: 0.0020
Epoch 41/100
56/56 [==============================] - 13s 240ms/step - loss: 1.3435e-04 - val_loss: 0.0022
Epoch 42/100
56/56 [==============================] - 14s 242ms/step - loss: 1.2509e-04 - val_loss: 0.0023
Epoch 43/100
56/56 [==============================] - 13s 241ms/step - loss: 1.1778e-04 - val_loss: 0.0012
Epoch 44/100
56/56 [==============================] - 14s 243ms/step - loss: 1.4833e-04 - val_loss: 0.0010
Epoch 45/100
56/56 [==============================] - 13s 240ms/step - loss: 1.9291e-04 - val_loss: 0.0012
Epoch 46/100
56/56 [==============================] - 14s 250ms/step - loss: 1.4631e-04 - val_loss: 0.0013
Epoch 47/100
56/56 [==============================] - 14s 242ms/step - loss: 1.4827e-04 - val_loss: 5.4244e-04
Epoch 48/100
56/56 [==============================] - 14s 242ms/step - loss: 2.2923e-04 - val_loss: 2.5059e-04
Epoch 49/100
56/56 [==============================] - 13s 239ms/step - loss: 3.4161e-04 - val_loss: 2.2227e-04
Epoch 50/100
56/56 [==============================] - 14s 243ms/step - loss: 3.7520e-04 - val_loss: 9.0896e-04
Epoch 51/100
56/56 [==============================] - 14s 243ms/step - loss: 3.6465e-04 - val_loss: 8.7105e-04
Epoch 52/100
56/56 [==============================] - 13s 239ms/step - loss: 2.9146e-04 - val_loss: 3.5326e-04
Epoch 53/100
56/56 [==============================] - 14s 245ms/step - loss: 2.7333e-04 - val_loss: 5.4463e-04
Epoch 54/100

```
56/56 [==============================] - 14s 242ms/step - loss: 4.1298e-04 - val_loss: 0.0013
Epoch 55/100
56/56 [==============================] - 13s 240ms/step - loss: 3.2201e-04 - val_loss: 0.0011
Epoch 56/100
56/56 [==============================] - 13s 240ms/step - loss: 3.7207e-04 - val_loss: 0.0014
Epoch 57/100
56/56 [==============================] - 13s 241ms/step - loss: 3.4661e-04 - val_loss: 0.0013
Epoch 58/100
56/56 [==============================] - 13s 235ms/step - loss: 3.1107e-04 - val_loss: 7.8487e-04
Epoch 59/100
56/56 [==============================] - 14s 241ms/step - loss: 2.6821e-04 - val_loss: 0.0013
Epoch 60/100
56/56 [==============================] - 13s 233ms/step - loss: 2.3595e-04 - val_loss: 0.0012
Epoch 61/100
56/56 [==============================] - 14s 241ms/step - loss: 1.7650e-04 - val_loss: 0.0010
Epoch 62/100
56/56 [==============================] - 13s 241ms/step - loss: 1.3822e-04 - val_loss: 0.0011
Epoch 63/100
56/56 [==============================] - 13s 233ms/step - loss: 1.1097e-04 - val_loss: 0.0013
Epoch 64/100
56/56 [==============================] - 13s 228ms/step - loss: 1.2696e-04 - val_loss: 9.3943e-04
Epoch 65/100
56/56 [==============================] - 13s 234ms/step - loss: 1.1926e-04 - val_loss: 5.3474e-04
Epoch 66/100
56/56 [==============================] - 13s 224ms/step - loss: 2.0304e-04 - val_loss: 3.8525e-04
Epoch 67/100
56/56 [==============================] - 13s 230ms/step - loss: 2.7919e-04 - val_loss: 4.9860e-04
Epoch 68/100
```

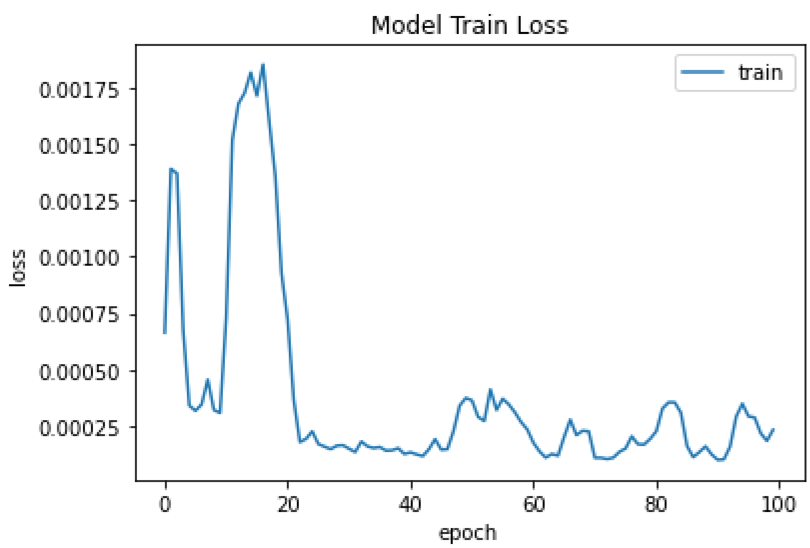
56/56 [==============================] - 13s 238ms/step - loss: 2.1053e-04 - val_loss: 1.8525e-04
Epoch 69/100
56/56 [==============================] - 13s 233ms/step - loss: 2.3021e-04 - val_loss: 2.2364e-04
Epoch 70/100
56/56 [==============================] - 13s 235ms/step - loss: 2.2723e-04 - val_loss: 2.1468e-04
Epoch 71/100
56/56 [==============================] - 14s 242ms/step - loss: 1.0991e-04 - val_loss: 2.9275e-04
Epoch 72/100
56/56 [==============================] - 13s 240ms/step - loss: 1.0986e-04 - val_loss: 3.4520e-04
Epoch 73/100
56/56 [==============================] - 13s 240ms/step - loss: 1.0447e-04 - val_loss: 1.9678e-04
Epoch 74/100
56/56 [==============================] - 13s 237ms/step - loss: 1.1067e-04 - val_loss: 1.5764e-04
Epoch 75/100
56/56 [==============================] - 14s 245ms/step - loss: 1.3668e-04 - val_loss: 1.5405e-04
Epoch 76/100
56/56 [==============================] - 14s 245ms/step - loss: 1.5150e-04 - val_loss: 2.1639e-04
Epoch 77/100
56/56 [==============================] - 13s 240ms/step - loss: 2.0564e-04 - val_loss: 3.1736e-04
Epoch 78/100
56/56 [==============================] - 13s 241ms/step - loss: 1.7058e-04 - val_loss: 2.0294e-04
Epoch 79/100
56/56 [==============================] - 13s 240ms/step - loss: 1.6880e-04 - val_loss: 2.5969e-04
Epoch 80/100
56/56 [==============================] - 13s 240ms/step - loss: 1.9406e-04 - val_loss: 3.9011e-04
Epoch 81/100
56/56 [==============================] - 14s 243ms/step - loss: 2.2786e-04 - val_loss: 7.0434e-04
Epoch 82/100

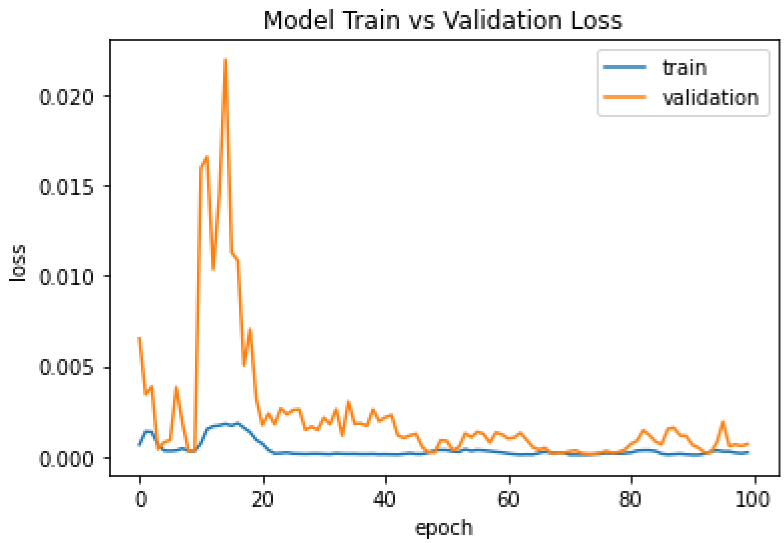
56/56 [==============================] - 14s 245ms/step - loss: 3.2872e-04 - val_loss: 8.8095e-04
Epoch 83/100
56/56 [==============================] - 14s 253ms/step - loss: 3.5599e-04 - val_loss: 0.0015
Epoch 84/100
56/56 [==============================] - 13s 240ms/step - loss: 3.5587e-04 - val_loss: 0.0012
Epoch 85/100
56/56 [==============================] - 14s 242ms/step - loss: 3.1006e-04 - val_loss: 8.4004e-04
Epoch 86/100
56/56 [==============================] - 14s 244ms/step - loss: 1.6231e-04 - val_loss: 6.7507e-04
Epoch 87/100
56/56 [==============================] - 14s 243ms/step - loss: 1.1309e-04 - val_loss: 0.0015
Epoch 88/100
56/56 [==============================] - 13s 238ms/step - loss: 1.3533e-04 - val_loss: 0.0016
Epoch 89/100
56/56 [==============================] - 14s 242ms/step - loss: 1.6094e-04 - val_loss: 0.0012
Epoch 90/100
56/56 [==============================] - 14s 242ms/step - loss: 1.2523e-04 - val_loss: 0.0011
Epoch 91/100
56/56 [==============================] - 14s 242ms/step - loss: 1.0077e-04 - val_loss: 6.6090e-04
Epoch 92/100
56/56 [==============================] - 14s 243ms/step - loss: 1.0301e-04 - val_loss: 5.0688e-04
Epoch 93/100
56/56 [==============================] - 14s 242ms/step - loss: 1.5979e-04 - val_loss: 2.2428e-04
Epoch 94/100
56/56 [==============================] - 14s 245ms/step - loss: 2.9299e-04 - val_loss: 2.3039e-04
Epoch 95/100
56/56 [==============================] - 14s 244ms/step - loss: 3.5006e-04 - val_loss: 7.9333e-04
Epoch 96/100

56/56 [==============================] - 13s 236ms/step - loss: 2.9381e-04 - val_loss: 0.0019
Epoch 97/100
56/56 [==============================] - 14s 246ms/step - loss: 2.8894e-04 - val_loss: 5.9175e-04
Epoch 98/100
56/56 [==============================] - 13s 227ms/step - loss: 2.1834e-04 - val_loss: 6.7060e-04
Epoch 99/100
56/56 [==============================] - 13s 229ms/step - loss: 1.8522e-04 - val_loss: 6.0122e-04
Epoch 100/100
56/56 [==============================] - 13s 229ms/step - loss: 2.3421e-04 - val_loss: 6.9808e-04

Post-training, plot the loss as:

Evaluation of how the model has trained over the epochs. Now epoch equal to 100. Have the changes made a difference?

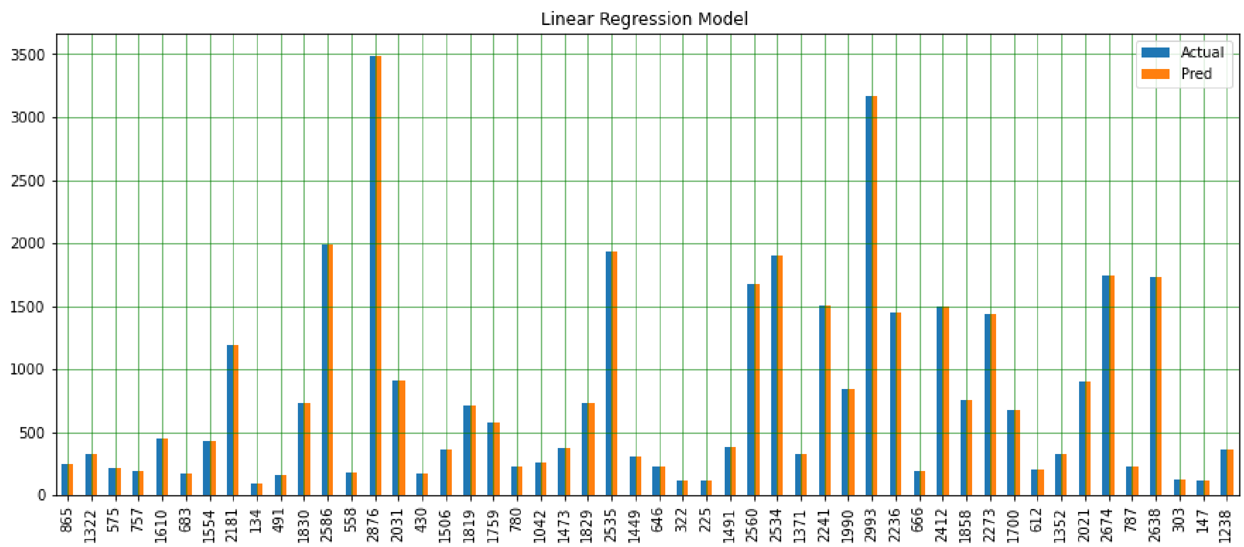




Yes the model has a better fit over the epoch. But can the model perform well against the test data set?

Model Evaluation & Visualizing the Results for the Linear Models
Here we built and evaluated the Linear Models. They are LinearRegression model, Support Vector Machine Model, K-Nearest Neighbor Model.
Cross-validation is used to measure accuracy on the Linear Regression Model and the KNN Model.

Model-1 Linear Regression



Linear Model root mean square error 2.4024554165180363e-13

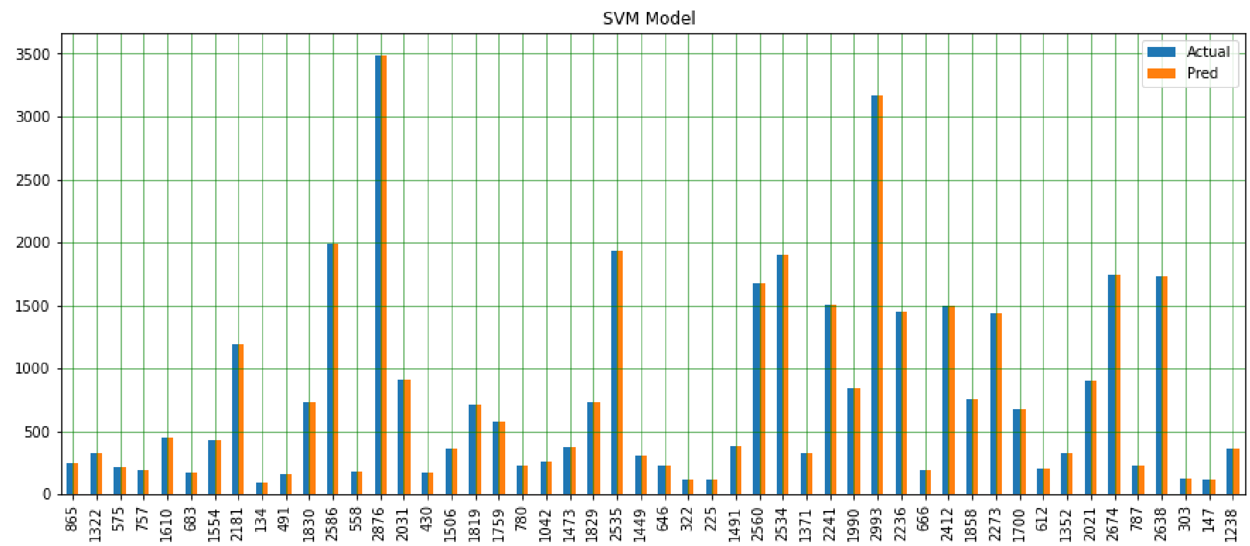
Linear Model R2 score 1.0

Linear Model Mean Absolute Error 1.938511161746562e-13

Linear Cross-Validation
Cross-Validation is a technique where the model is evaluated on a dataset on which it is not trained. It can be the test data or another dataset.

KFold(n_splits=25, random_state=None, shuffle=False)
Accuracy: 99.99998687063808

Model-2 Support Vector Machine(SVM)

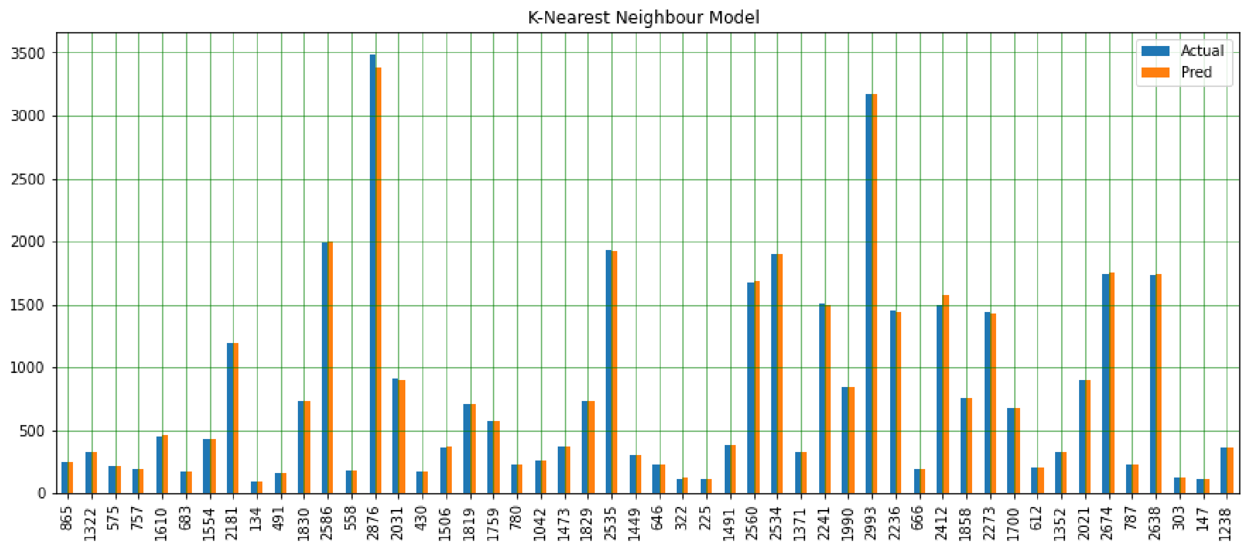


SVM Model root mean square error 0.04632123250742317

SVM Model R2 score 0.9999999971194898

SVM Model Mean Absolute Error 0.04208701205917427

Model-3 K-Nearest Neighbour(KNN)



KNN Model root mean square error 8.055662406275989

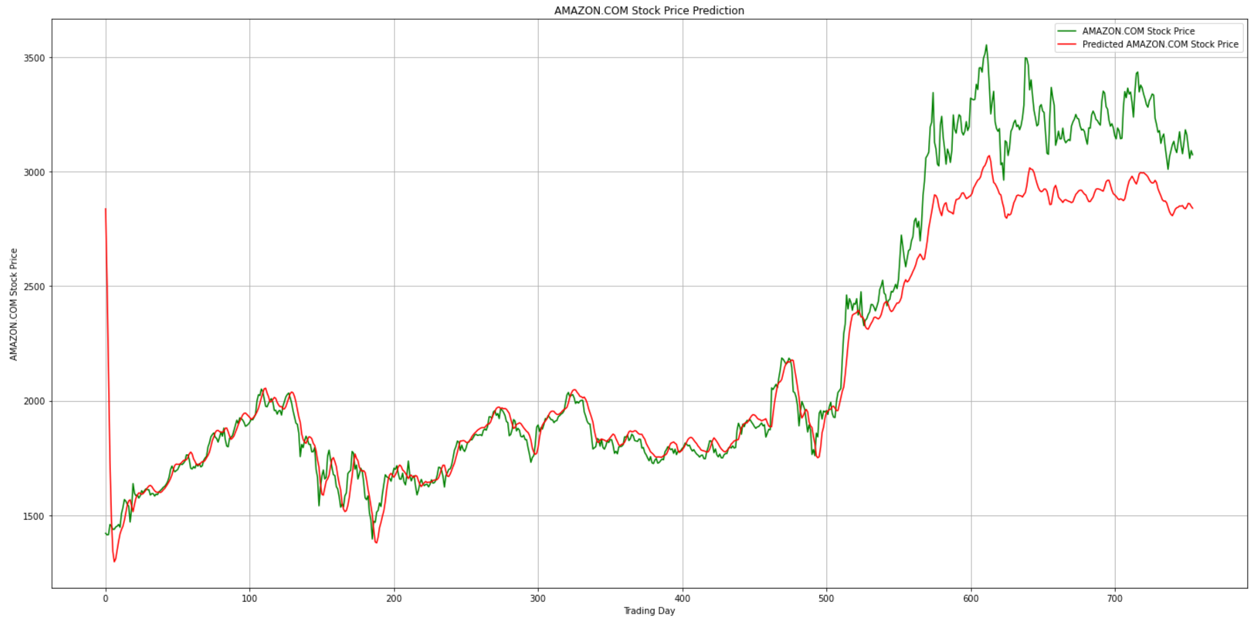
KNN Model R2 score 0.999128812353679

KNN Model Mean Absolute Error 3.361844693114426

KNN Cross-Validation
KFold(n_splits=25, random_state=None, shuffle=False)
Accuracy: 99.97465262168085

Model Evaluation & Visualizing the Results for the RNN-Lstm Model

Visualizing the results for the original model



Model Performance
LSTM Model root mean square error 186.0728774117609

LSTM Model R2 score 0.911753314248435

LSTM Model Mean Absolute Error 116.32627308826572

Model Evaluation & Visualizing the Results for the RNN-Lstm Model Improved

Visualizing the results for the improved model

Looking at the visualization above the model's performance has improved.

Model Performance
LSTM Model root mean square error 104.88462191329695

LSTM Model R2 score 0.9719614280456601

LSTM Model Mean Absolute Error 68.07768667865273

Reporting Performance of Model Evaluation for all 4 Models.

Model evaluation is a critical task in data science. This is one of the few measures business stakeholders are interested in. Model performance heavily influences business impact of a data science project. I have selected the following metrics RMSE, MAE, R2 because in my view they are best suited for the predictive models that I have worked with in this project.

<u>Linear Models and RNN-LSTM Time Series Forecasting Model Metrics.</u>

RMSE (Root Mean Square Error)
Linear Model root mean square error 2.4024554165180363e-13
SVM Model root mean square error 0.04632123250742317
KNN Model root mean square error 8.055662406275989
LSTM Model root mean square error 186.0728774117609
LSTM Model (Improved) root mean square error 104.88462191329695

MAE (Mean Absolute Error)
Linear Model Mean Absolute Error 1.938511161746562e-13
SVM Model Mean Absolute Error 0.04208701205917427
KNN Model Mean Absolute Error 3.361844693114426
LSTM Model Mean Absolute Error 116.32627308826572
LSTM Model(Improved) Mean Absolute Error 68.07768667865273

R2 or R-Squared Error
Linear Model R2 score 1.0
SVM Model R2 score 0.9999999971194898
KNN Model R2 score 0.9999128812353679
LSTM Model R2 score 0.911753314248435
LSTM Model(Improved) R2 score 0.9719614280456601

Model redeployment.

<u>Testing a New Never Seen Before Test Data on The RNN-LSTM-Time Series Forecasting Model and see the Result.</u>

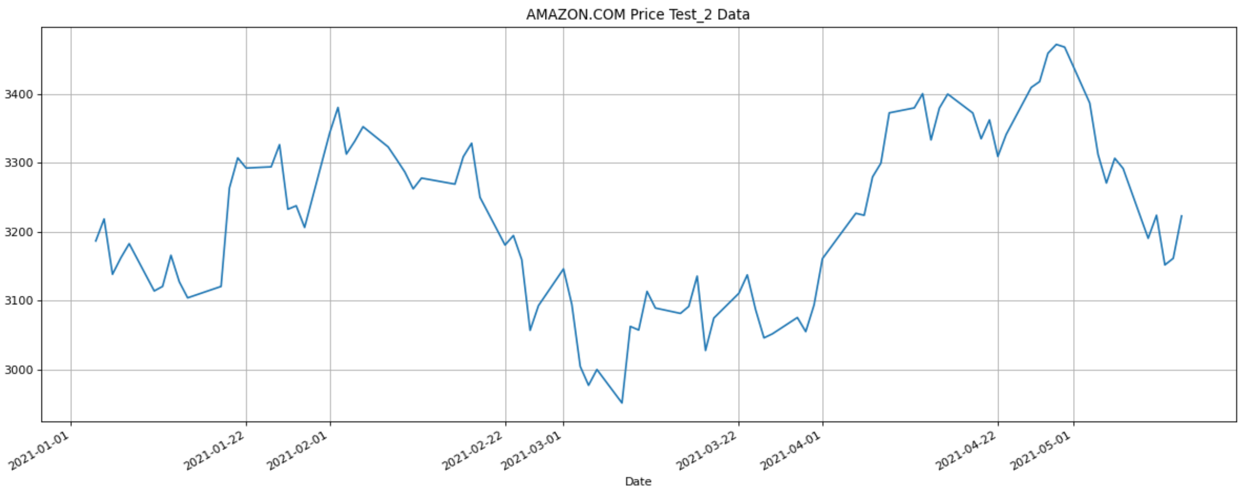
Model Deployment Test New

Here I would like to test an independent new set of data. Let us see how the model fairs. AMZN data from January 02, 2021 to May 15, 2021.

Prepare Test-2 data

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
2021-01-04	3270.00000	3272.000000	3144.020020	3186.629883	4411400	0	0
2021-01-05	3166.01001	3223.379883	3165.060059	3218.510010	2655500	0	0
2021-01-06	3146.47998	3197.510010	3131.159912	3138.379883	4394800	0	0
2021-01-07	3157.00000	3208.540039	3155.000000	3162.159912	3514500	0	0
2021-01-08	3180.00000	3190.639893	3142.199951	3182.699951	3537700	0	0

(92, 7)



(92, 5)

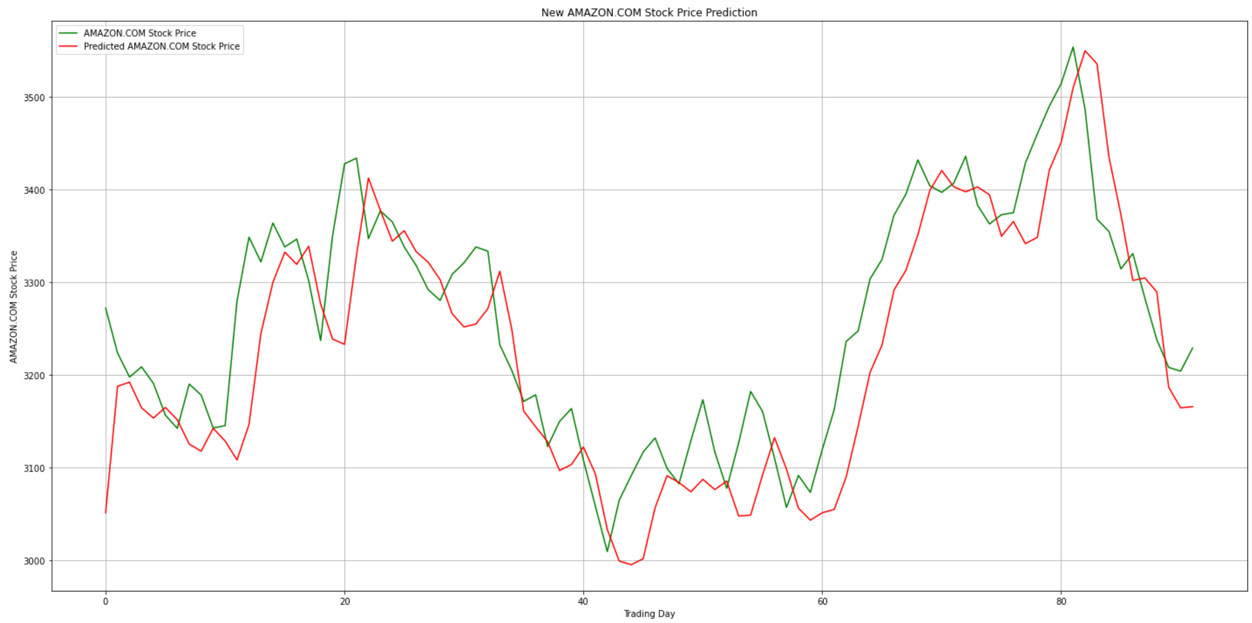
Reshaping the data:
(92, 50, 1)

Making Prediction Using New Test Data

Visualizing The New Result Based on New Test Data - Original Model

Did I improve model performance?

Based on the visualization and the performance metrics Having a correctly fitted training model gave better results. See below.



Conclusion

I am recommending the RNN-LSTM model as one to deploy. Although it can be difficult to achieve precise predictions by training a model that is not under-fitted or over-fitted, in my view this is a very impressive unsupervised model.

References.

My research materials come from some of the following resources: Towardsdatasience.com, medium.com, machinelearningmastery.com, tensorflow.org, analyticsvidhya.com, guru99.com,kdnuggets.com, dev.to, analyticsindiamag.com, towardsai.net, data-flair.training.

The End