

The Rise of Artificial Intelligence in the Stock Market in the 21st Century

Research Paper

Arianne Ghislaine Rull

Siddha Ganju

July 2020

I. Abstract

In this research project, the main focus is to forecast the movement of the stock market. Although the financial market plays a significant role in the economy. By ensuring the success of the stock market, there would be more money circulation. This circulation allows businesses to pay liabilities, raise fundings, and execute operations. Indeed, the common people tend to look at predicting the stock market both tedious and intimidating. As a result, less people are interested in this particular area. Stock analysis is done primarily through technical or fundamental analysis by humans. This method potentially hinders the person from conducting an objective and accurate analysis. In comparison, the traditional stock analysis takes more time compared to automated analysis. Therefore, having an inclusive automated stock analysis powered by machine learning should be echoed to ease the process of trading, produce accurate results and grow a community ranging from different levels of expertise.

The particular methods in this research project are taken to dive into artificial intelligence through tackling a realistic project on stock market prediction. During these four weeks of research under the supervision of Ms. Siddha Ganju, several methods are taken into consideration. In the first week, the project ideation and planning is completed. The targeted project is chosen based on the mentee's interest. Then, the research proposal is completed with two versions written on Google Docs and LaTeX. Next, the actual

experimentation with the result of five trials occurs in the third week. Lastly, the research paper is written using Google Docs.

Indeed, utilizing machine learning to increase the diversity of the people in the stock market and create accurate predictions. The potential is validated by the existing open source projects that seek to create an accessible software for accomplishing the task of forecasting the prediction model. Here is some of the examples of projects that share the same goal:

- [stock-market · GitHub Topics](#)
- [IBM/watson-stock-market-predictor: A IBM Developer code pattern for Watson Studio: forecast the stock market with Python Notebooks, SPSS Modeler, Data Refinery, and other Watson Studio tools.](#)
- [Python Code for stock market predictions with Watson Studio](#)

II. Introduction

There is still a pressing issue of the lack of diversity in the community of people involved in the stock market. This problem could potentially hinder the production of innovative ideas. Specifically in the trading environment, there is an alarming lack of diversity. Maintaining an inclusive workplace environment is necessary to promote advancements through welcoming fresh ideas on the table. Aside from that, the traditional stock analysis is completed by people. Since human decisions are affected by

biases such as personal beliefs, emotions and impulses, this situation could impact the accuracy of their predictions. These cognitive and emotional biases could potentially limit the perspective of the person forecasting the movement of the stocks. Letting these biases could prohibit a person to produce a more accurate result in comparison to an automated analysis.

III. Methods

First and foremost, research utilizes conducting simple experiments as the main method used to accumulate information and connect ideas with each other. In this project, the independent variables are modified to identify its effects to the output of the model. These following variables are the portion of the actual data used for the training model, number of neurons on the first dense layer, number of neurons on the second dense layer, batch size and epoch. These variables affect the forecasted stock price of the model.

Terminologies

- *Dense layer - a property of the model that consist of layers of neurons called a neural network*
- *Neuron - like the neurons on the brain; component or unit that composes a neural network*
- *Neural Network: a model that is designed after the human brain*
- *Batch Size - the overall number of training examples present in a single batch*
- *Epoch - the number of iterations the data is passed forward and backward through the neural network*

IV. Results

Trial #	Percentage of the actual data used for the training model	# of neurons on the first dense layer	# of neurons on the second dense layer	Batch size	Epoch	RMSE
Original	80	25	1	1	1	5.175868645429504
1	80	50	1	1	5	133.71440785725989
2	80	100	1	2	2	5.845912177797393
3	80	20	1	2	2	5.24346892746233
4	90	25	1	1	2	4.586480213681501
5	50	50	1	1	2	8.939634991716115

Original Code

Figure1

```
#Build the LSTM network model
model = Sequential()
# first LSTM layer
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
# second LSTM layer
model.add(LSTM(units=50, return_sequences=False))
# first dense layer
model.add(Dense(units=25))
# second dense layer
model.add(Dense(units=1))

# This block of code aims to compile the model using the following:
# 1: MSE loss function
#   MSE stands for mean squared error
# 2: Adam optimizer

# Now, this compiles the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model using the training data sets. Note, fit is another
# name for train. Batch size is the total number of training examples
# present in a single batch, and epoch is the number of iterations when
# an entire data set is passed forward and backward through the neural
# network.

#Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)

Epoch 1/1
1543/1543 [=====] - 283s 183ms/step - loss: 7.0641e-04
<keras.callbacks.callbacks.History at 0x7f04fd4c79e8>
```

Figure 2

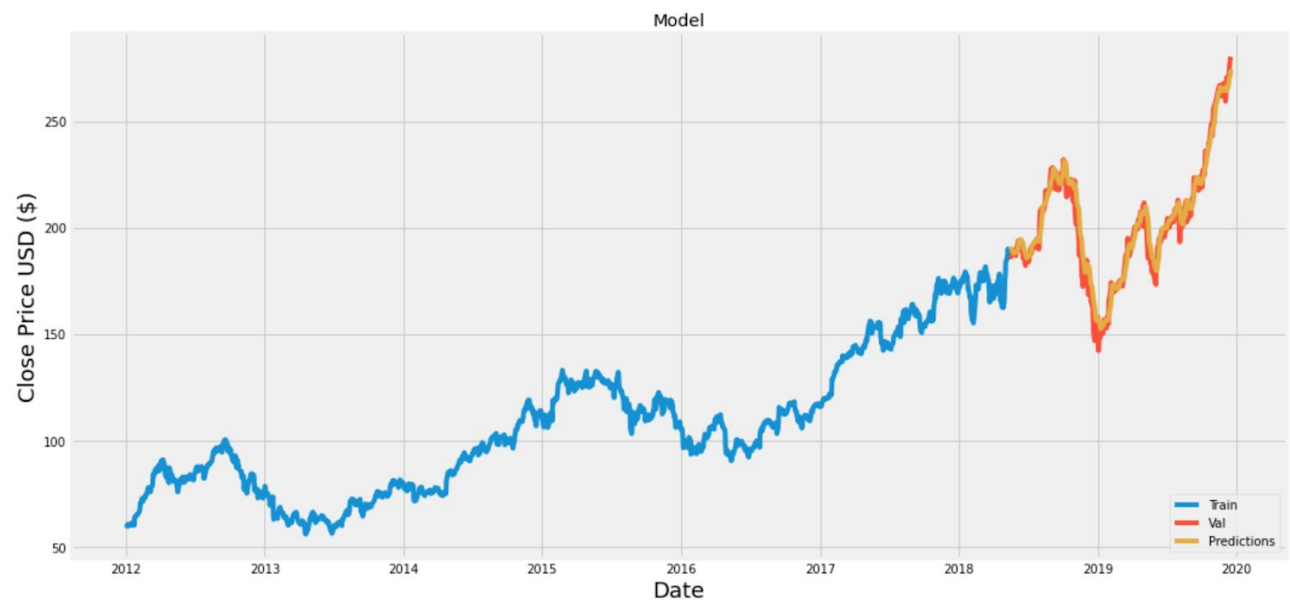



Figure 3



	Close Predictions	
Date		
2018-05-17	186.990005	191.029327
2018-05-18	186.309998	190.401184
2018-05-21	187.630005	189.535690
2018-05-22	187.160004	188.864487
2018-05-23	188.360001	188.307266
...
2019-12-11	270.769989	267.597870
2019-12-12	271.459991	268.928589
2019-12-13	275.149994	270.284882
2019-12-16	279.859985	272.059296
2019-12-17	280.410004	274.535004

400 rows × 2 columns

Trial #1

Figure 4

```
# This part builds the LSTM model with certain properties

# two LSTM layers with 50 neurons
# two Dense layers
##### first dense layer -> 25 neurons
##### second dense layer -> 1 neuron

#Build the LSTM network model
model = Sequential()
# first LSTM layer
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
# second LSTM layer
model.add(LSTM(units=50, return_sequences=False))
# first dense layer

# changed
model.add(Dense(units=50))
# second dense layer
model.add(Dense(units=1))
```

```
[ ] # Train the model using the training data sets. Note, fit is another
    # name for train. Batch size is the total number of training examples
    # present in a single batch, and epoch is the number of iterations when
    # an entire data set is passed forward and backward through the neural
    # network.
```

```
#Train the model
model.fit(x_train, y_train, batch_size=1, epochs=5)
```

```
☐ Epoch 1/5
1543/1543 [=====] - 300s 194ms/step - loss: 6.5353e-04
Epoch 2/5
1543/1543 [=====] - 292s 189ms/step - loss: 2.7299e-04
Epoch 3/5
1543/1543 [=====] - 289s 187ms/step - loss: 2.1020e-04
Epoch 4/5
1543/1543 [=====] - 285s 185ms/step - loss: 1.8933e-04
Epoch 5/5
1543/1543 [=====] - 283s 183ms/step - loss: 1.4930e-04
<keras.callbacks.callbacks.History at 0x7fa6e08346a0>
```

Figure 5

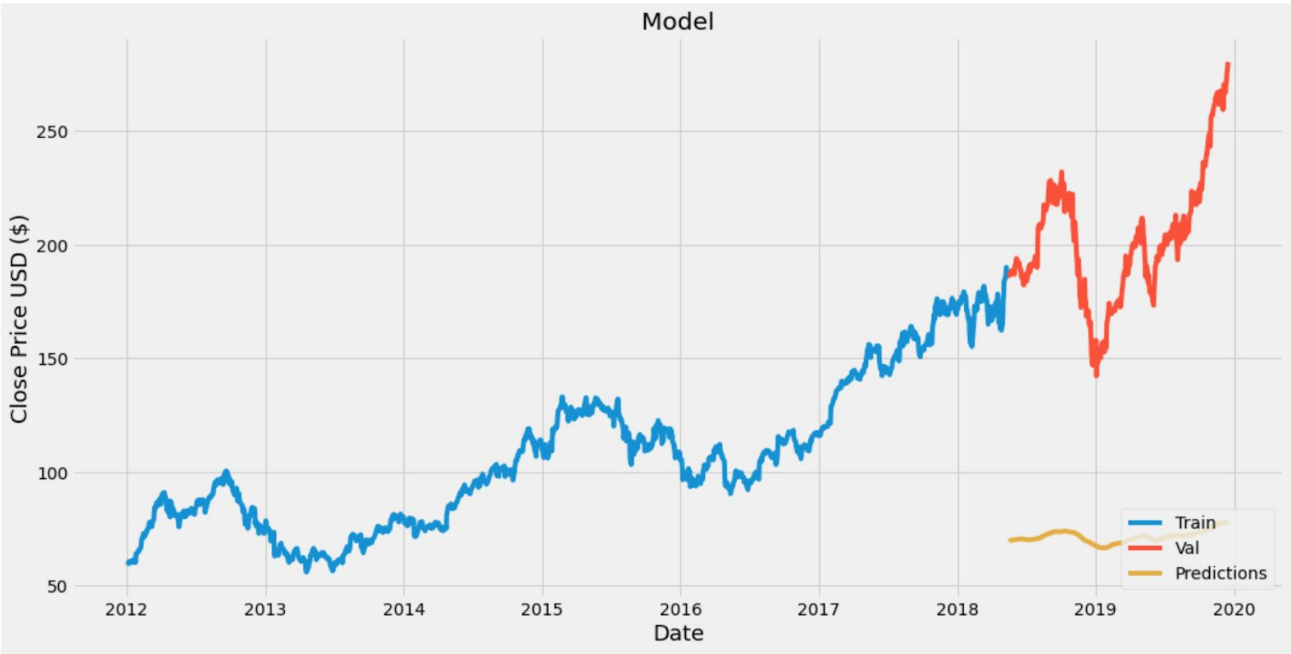


Figure 6

```
[ ] # Show the actual prices and predicted prices  
valid
```



Close Predictions		
Date		
2018-05-17	186.990005	69.376816
2018-05-18	186.309998	69.497498
2018-05-21	187.630005	69.600113
2018-05-22	187.160004	69.689789
2018-05-23	188.360001	69.766403
...
2019-12-11	270.769989	77.685478
2019-12-12	271.459991	77.723526
2019-12-13	275.149994	77.769928
2019-12-16	279.859985	77.830101
2019-12-17	280.410004	77.912033

400 rows × 2 columns

Trial #2

Figure 7

```
[ ] #Build the LSTM network model
    model = Sequential()
    # first LSTM layer
    model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
    # second LSTM layer
    model.add(LSTM(units=50, return_sequences=False))
    # first dense layer
    model.add(Dense(units=100))
    # second dense layer
    model.add(Dense(units=1))

[ ] # This block of code aims to compile the model using the following:
    # 1: MSE loss function
    #    MSE stands for mean squared error
    # 2: Adam optimizer

    # Now, this compiles the model
    model.compile(optimizer='adam', loss='mean_squared_error')

[ ] # Train the model using the training data sets. Note, fit is another
    # name for train. Batch size is the total number of training examples
    # present in a single batch, and epoch is the number of iterations when
    # an entire data set is passed forward and backward through the neural
    # network.

    #Train the model
    model.fit(x_train, y_train, batch_size=2, epochs=2)

☞ Epoch 1/2
   1543/1543 [=====] - 159s 103ms/step - loss: 9.6153e-04
   Epoch 2/2
   1543/1543 [=====] - 157s 102ms/step - loss: 3.8197e-04
   <keras.callbacks.callbacks.History at 0x7f904c373668>
```

Figure 8

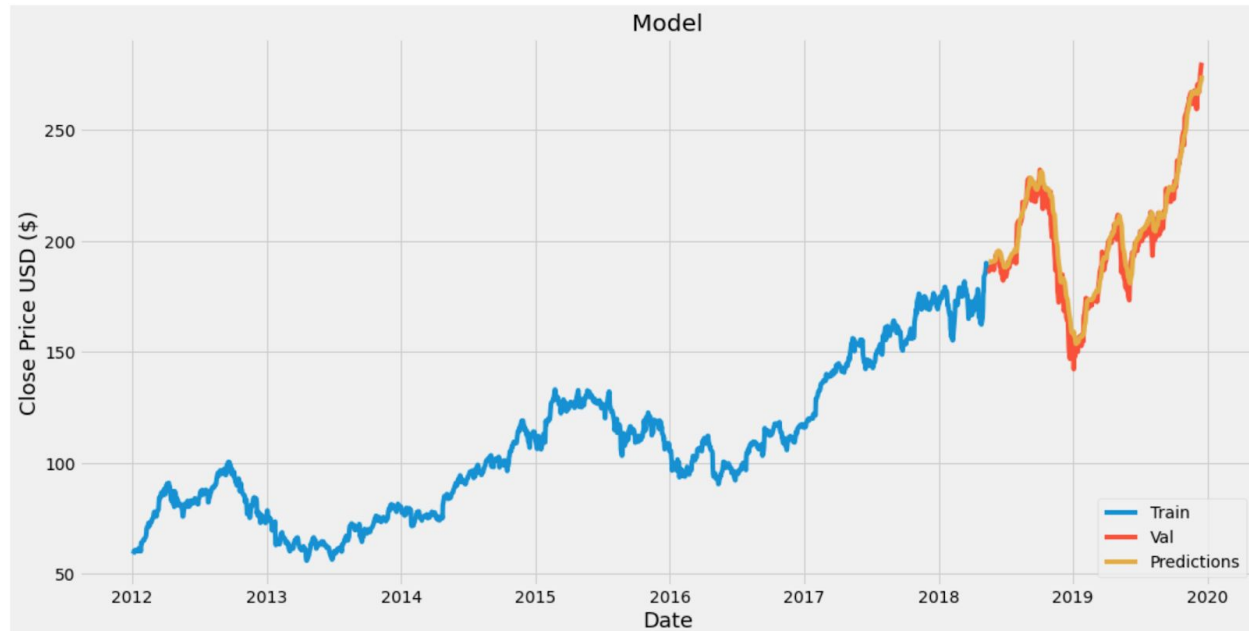


Figure 9

```
# Show the actual prices and predicted prices
valid
```

Close Predictions		
Date		
2018-05-17	186.990005	190.864334
2018-05-18	186.309998	190.848923
2018-05-21	187.630005	190.599823
2018-05-22	187.160004	190.471115
2018-05-23	188.360001	190.350082
...
2019-12-11	270.769989	268.957184
2019-12-12	271.459991	269.985382
2019-12-13	275.149994	271.049927
2019-12-16	279.859985	272.517242
2019-12-17	280.410004	274.622375

400 rows x 2 columns

Trial #3

Figure 10

```
[ ] #Build the LSTM network model
    model = Sequential()
    # first LSTM layer
    model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
    # second LSTM layer
    model.add(LSTM(units=50, return_sequences=False))
    # first dense layer
    model.add(Dense(units=20))
    # second dense layer
    model.add(Dense(units=1))

[ ] # This block of code aims to compile the model using the following:
    # 1: MSE loss function
    #    MSE stands for mean squared error
    # 2: Adam optimizer

    # Now, this compiles the model
    model.compile(optimizer='adam', loss='mean_squared_error')

[ ] # Train the model using the training data sets. Note, fit is another
    # name for train. Batch size is the total number of training examples
    # present in a single batch, and epoch is the number of iterations when
    # an entire data set is passed forward and backward through the neural
    # network.

    #Train the model
    model.fit(x_train, y_train, batch_size=2, epochs=2)

↳ Epoch 1/2
   1543/1543 [=====] - 145s 94ms/step - loss: 0.0011
Epoch 2/2
   1543/1543 [=====] - 141s 92ms/step - loss: 3.5766e-04
<keras.callbacks.callbacks.History at 0x7f1aa0cd34a8>
```

Figure 11

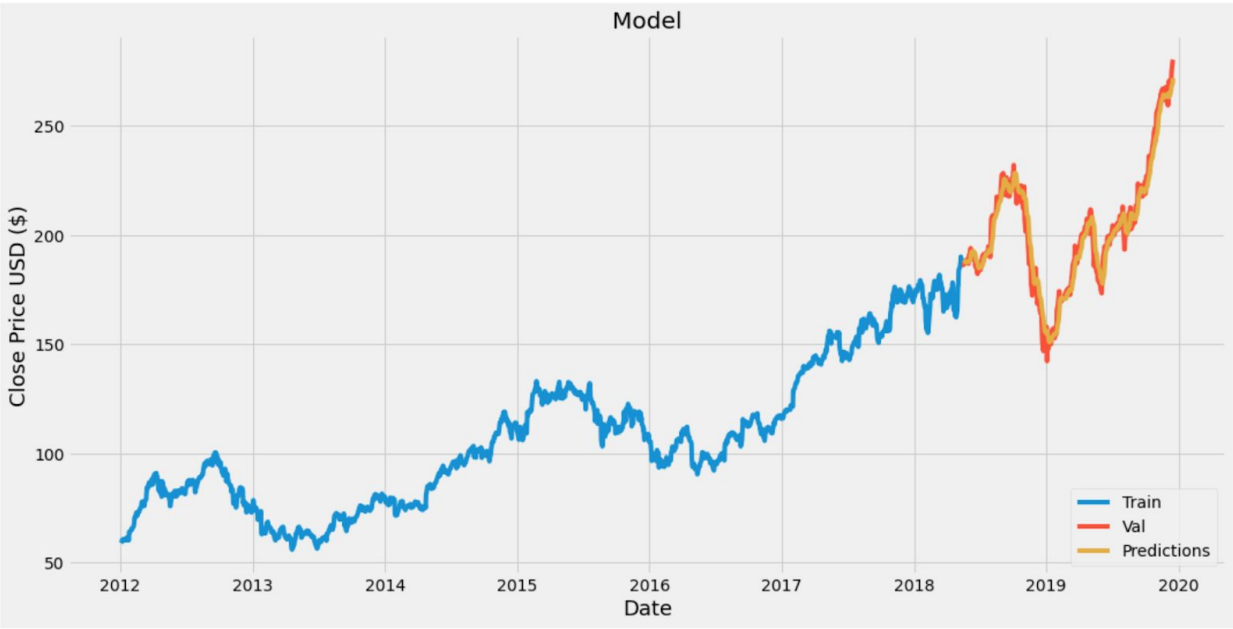


Figure 12

```
[ ] # Show the actual prices and predicted prices
valid
```

	Close	Predictions
Date		
2018-05-17	186.990005	188.605148
2018-05-18	186.309998	188.450409
2018-05-21	187.630005	188.043137
2018-05-22	187.160004	187.731354
2018-05-23	188.360001	187.452469
...
2019-12-11	270.769989	265.878326
2019-12-12	271.459991	266.997406
2019-12-13	275.149994	268.162109
2019-12-16	279.859985	269.715179
2019-12-17	280.410004	271.904755

400 rows x 2 columns

Trial #4

Figure 12

```
[ ] # Changed this

# This block of code aims to curate a different data frame specifically for the closing price.
# Then, it is converted to an array and a variable is created to store the length of the
# training data set. This training dataset contains about 100 percent of the actual data.

# This part creates a different dataframe with a "Close" column only
data = df.filter(['Close'])
# This part converts the dataframe to a numpy array
dataset = data.values
# This part aims to get / compute the number of rows that should be used to train the model on
training_data_len = math.ceil( len(dataset) *.9)

[ ] # Train the model using the training data sets. Note, fit is another
# name for train. Batch size is the total number of training examples
# present in a single batch, and epoch is the number of iterations when
# an entire data set is passed forward and backward through the neural
# network.

#Train the model
model.fit(x_train, y_train, batch_size=1, epochs=2)

↳ Epoch 1/2
1743/1743 [=====] - 505s 290ms/step - loss: 9.8049e-04
Epoch 2/2
1743/1743 [=====] - 502s 288ms/step - loss: 4.3167e-04
<keras.callbacks.callbacks.History at 0x7f60800ac630>

[ ] # This part builds the LSTM model with certain properties

# two LSTM layers with 50 neurons
# two Dense layers
#### first dense layer -> 25 neurons
#### second dense layer -> 1 neuron

#Build the LSTM network model
model = Sequential()
# first LSTM layer
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
# second LSTM layer
model.add(LSTM(units=50, return_sequences=False))
# first dense layer
model.add(Dense(units=25))
# second dense layer
model.add(Dense(units=1))
```

Figure 13

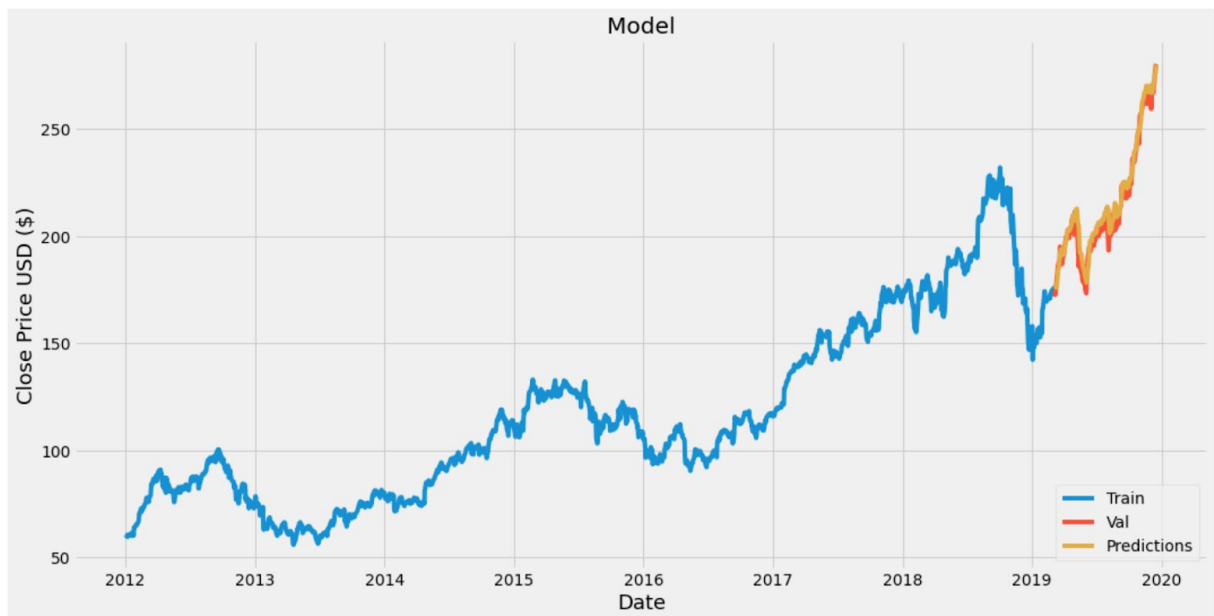


Figure 14

```
[ ] # Show the actual prices and predicted prices
    valid
```



	Close	Predictions
Date		
2019-03-06	174.520004	177.422897
2019-03-07	172.500000	177.365112
2019-03-08	172.910004	176.597610
2019-03-11	178.899994	175.881439
2019-03-12	180.910004	177.027863
...
2019-12-11	270.769989	272.112305
2019-12-12	271.459991	273.526337
2019-12-13	275.149994	274.819153
2019-12-16	279.859985	276.840790
2019-12-17	280.410004	279.959595

200 rows × 2 columns

Trial #5

Figure 15

```
[ ] # This block of code aims to curate a different data frame specifically for the closing price.
    # Then, it is converted to an array and a variable is created to store the length of the
    # training data set. This training dataset contains about 80 percent of the actual data.

    # This part creates a different dataframe with a "Close" column only
    data = df.filter(['Close'])
    # This part converts the dataframe to a numpy array
    dataset = data.values
    # This part aims to get / compute the number of rows that should be used to train the model on
    training_data_len = math.ceil( len(dataset) *.50)
```

```
[ ]
    #Build the LSTM network model
    model = Sequential()
    # first LSTM layer
    model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
    # second LSTM layer
    model.add(LSTM(units=50, return_sequences=False))
    # first dense layer
    model.add(Dense(units=50))
    # second dense layer
    model.add(Dense(units=1))
```

```
[ ] # This block of code aims to compile the model using the following:
    # 1: MSE loss function
    #    MSE stands for mean squared error
    # 2: Adam optimizer

    # Now, this compiles the model
    model.compile(optimizer='adam', loss='mean_squared_error')
```

```
[ ] # Train the model using the training data sets. Note, fit is another
    # name for train. Batch size is the total number of training examples
    # present in a single batch, and epoch is the number of iterations when
    # an entire data set is passed forward and backward through the neural
    # network.

    #Train the model
    model.fit(x_train, y_train, batch_size=1, epochs=2)
```

```
↳ Epoch 1/2
942/942 [=====] - 180s 191ms/step - loss: 6.4344e-04
Epoch 2/2
942/942 [=====] - 174s 185ms/step - loss: 2.7326e-04
<keras.callbacks.callbacks.History at 0x7f9dc00e5ac8>
```


Figure 16

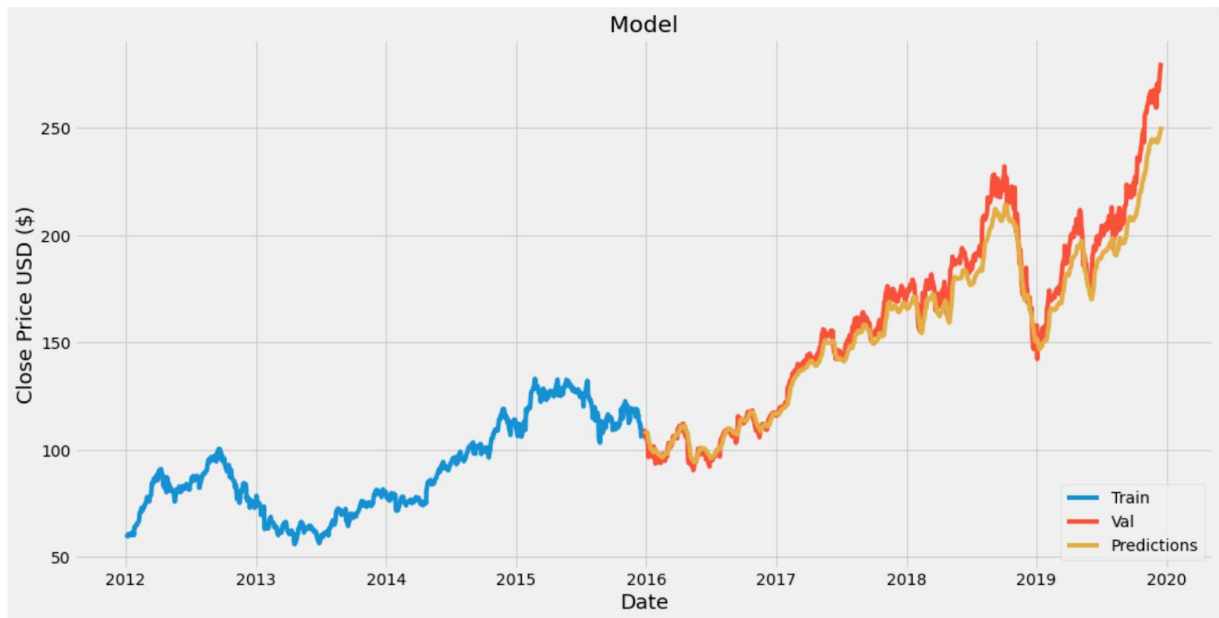


Figure 17

```
[ ] # Show the actual prices and predicted prices
valid
```



Close Predictions

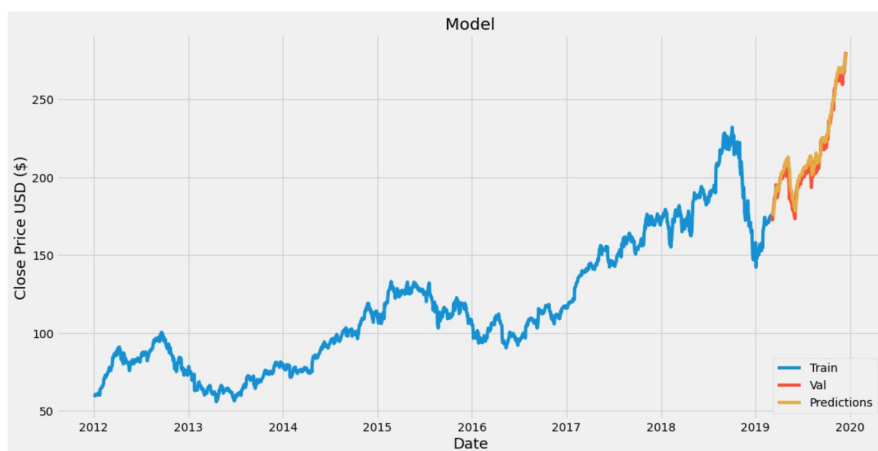
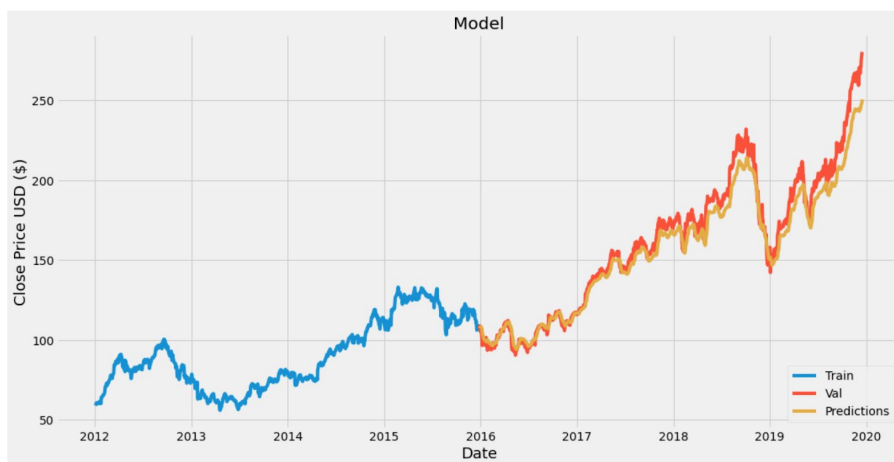
Date

2015-12-28	106.820000	108.595230
2015-12-29	108.739998	108.338715
2015-12-30	107.320000	108.368774
2015-12-31	105.260002	108.344315
2016-01-04	105.349998	108.027596
...
2019-12-11	270.769989	245.561218
2019-12-12	271.459991	246.483032
2019-12-13	275.149994	247.431549
2019-12-16	279.859985	248.753143
2019-12-17	280.410004	250.655472

1001 rows × 2 columns

V. Discussion

The independent variables in this model affect the outcome. Firstly, the portion that is used to determine the percentage of the actual data affects the values on the dataset. For instance, a training model that utilizes 90 percent of the actual data is more reliable in comparison to a training model that only uses 50 percent. Here's a side-by-side figure of the difference between two training models. The first picture is a model that uses 50 percent while the other one uses 90 percent. Since the second model leaves more room for training, it is able to produce more accurate results in comparison to the first model.

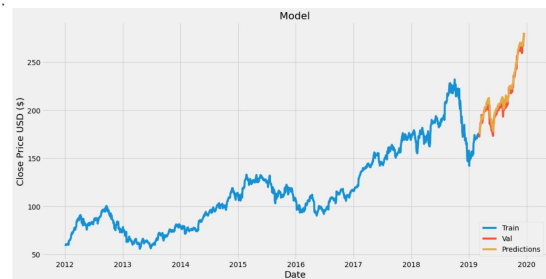
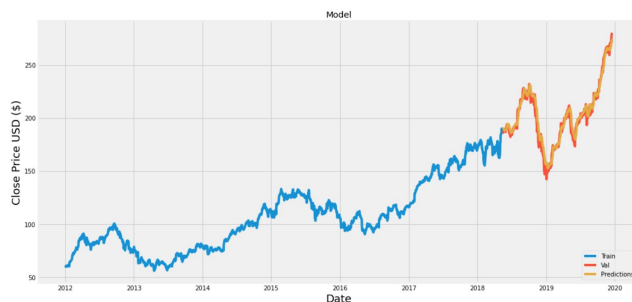


Next, the first dense layer of neurons is modifiable while the second one is constant.

Whenever the number of the neurons on the second layer is increased, the code breaks down. Henceforth, it is expected to stay the same. In addition, the batch size is the number of training samples in a single batch. For this variable, it is modifiable as well.

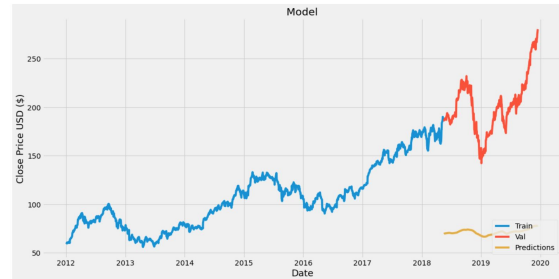
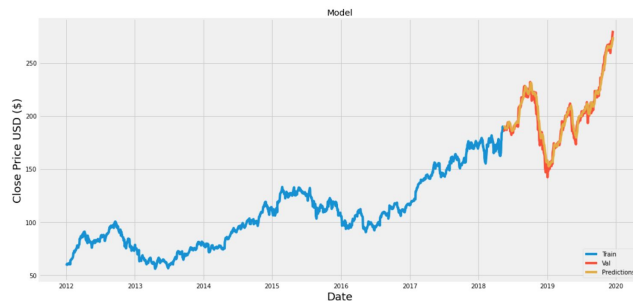
Lastly, the epoch represents the number of iterations the data is passed through the network. The higher the epoch is, the more iterations there are. In addition, a higher epoch affects the duration of training the model as well. Henceforth, these are the variables that determine the accuracy of the model.

Trial #	Percentage of the actual data used for the training model	# of neurons on the first dense layer	# of neurons on the second dense layer	Batch size	Epoch	RMSE
Original	80	25	1	1	1	5.175868645429504
1	80	50	1	1	5	133.71440785725989
4	90	25	1	1	2	4.586480213681501



After five trials, the most accurate model is the fourth trial. The percentage of the actual data used for the training model and the epoch both plays a factor on the stock predictions since the rest of the independent variables remain the same. Having a higher

percentage of data and epoch both allows the model to become more accurate although it takes a longer period of time dedicated for training.



In contrast, the first model is the least accurate one. This model is an example of overfitting. According to Investopedia, overfitting is the “modeling error that occurs when a function is too closely fit to a limited set of data points”. The reasoning behind this occurrence is it matches the blue trend on less data.

References

Randerson112358. (2020, April 21). Stock Price Prediction Using Python & Machine

Learning. Retrieved July 23, 2020, from

<https://medium.com/@randerson112358/stock-price-prediction-using-python-machine-learning-e82a039ac2bb>

W. Kenton, “How Overfitting Works,” Investopedia, 29-Jan-2020. [Online]. Available:

<https://www.investopedia.com/terms/o/overfitting.asp>. [Accessed: 26-Jul-2020].