STOCK PRICE PREDICTIONS

ABSTRACT

Stock price prediction has long been a challenging and sought after task in financial markets. With the availability of historical stock data and the advent of new machine learning techniques, there are increasingly new ways to make stock forecasts. This project will look at Long Short-Term Memory Networks (LSTM) and how they are major improvements over Simple Moving Average (SMA) models.

First, we will gather 20 years of data for Amazon's stock price. Then we will calculate 50 day and 200 day moving average data as a baseline to compare to our LSTM models.

We will then build our LSTM models. Root mean squared error will be used as our metric for determining how well our model performs. We will also explore the effect certain hyperparameters have on the performance of our model; more specifically the following 2 hyperparameters: # of epochs and batch size.

The results reveal that LSTM models tend to outperform SMA in capturing day to day trends. LSTM models showcase improved accuracy and adaptability, especially in the presence of abrupt market shifts. However, it is noted that LSTM models can be sensitive to the choice of hyperparameters and require more computational resources compared to the simplicity of SMA.

BUSINESS PROBLEM

Equiwealth Financial Solutions is seeking to enhance their portfolio management strategies by accurately predicting stock prices. They are looking to develop a robust and reliable stock price prediction model that can provide actionable insights for their investment decisions. The company manages a diverse portfolio of stocks for their clients and aims to maximize returns while minimizing risks.

TABLE OF CONTENTS

1. DATA OVERVIEW

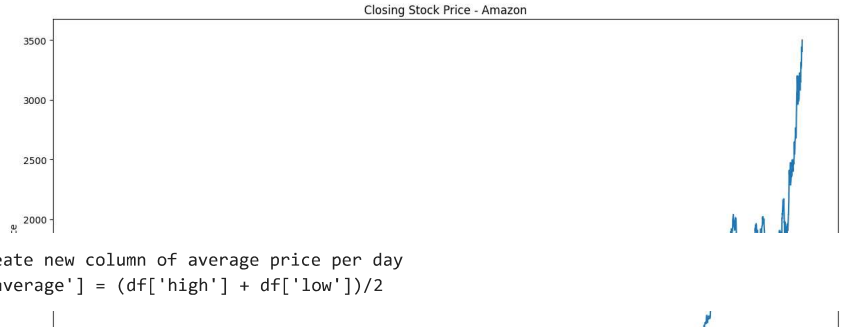
We used a dataset for Amazon stock prices which had 20 years of data from 2000-2020. The data was available on AWS and contained 5200 stock price data points.

DATA LIMITATIONS

- Data only through 2020
- Only for Amazon stock
- Did not include variables such as market sentiment, economic indicators, or global events

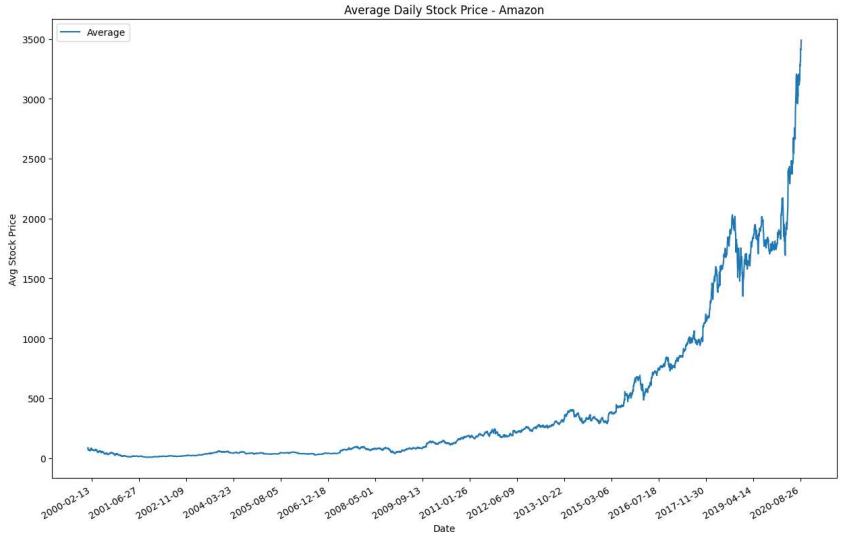
2. EXPLORATORY DATA ANALYSIS

```
# import relevant packages

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
import tensorflow as tf
from keras.layers import Dense, LSTM, Dropout


# Read in Amazon stock prices csv file
df = pd.read_csv('daily_adjusted_AMZN.csv')


# Sort by date
df.sort_values('timestamp', inplace=True)


df.head()


df.info()


# Convert timestamp column to datetime format
df['timestamp'] = pd.to_datetime(df['timestamp'])


# Plot closing price trend

fig, ax = plt.subplots(figsize= (15,10))

ax.plot(df['timestamp'], df['close'])
ax.set_title('Closing Stock Price - Amazon')
ax.set_xlabel('Date')
ax.set_ylabel('Stock Price')
plt.gca().xaxis.set_major_locator(plt.MaxNLocator(20))  # Set the maximum number of ticks
plt.gcf().autofmt_xdate()
```




```
# Create new column of average price per day
df['average'] = (df['high'] + df['low'])/2
```

```
# Plot average price per day
fig, ax = plt.subplots(figsize= (15,10))
ax.plot(df['timestamp'], df['average'], label='Average')
ax.set_title('Average Daily Stock Price - Amazon')
ax.set_xlabel('Date')
ax.set_ylabel('Avg Stock Price')
ax.legend()
plt.gca().xaxis.set_major_locator(plt.MaxNLocator(20))  # Set the maximum number of ticks
plt.gcf().autofmt_xdate()
```

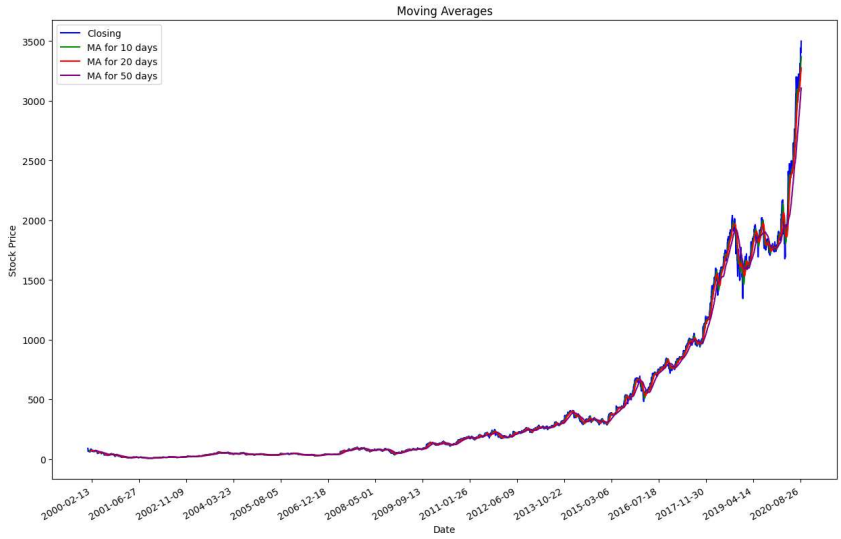


3. MODELING

MOVING AVERAGE MODELING

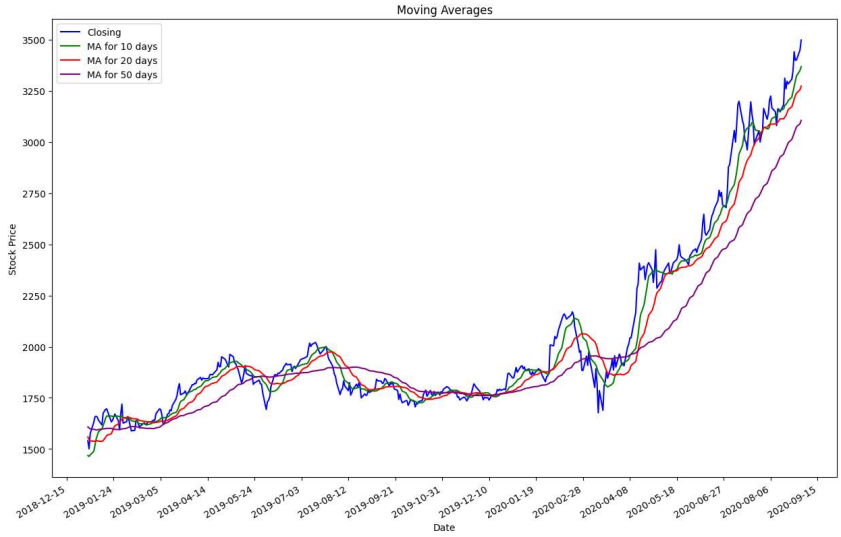
```
# Create new columns of rolling average prices for 10 days, 20 days, and 50 days
ma_day = [10, 20, 50]
```

```
for ma in ma_day:
    column_name = f"MA for {ma} days"
    df[column_name] = df['close'].rolling(ma).mean()


# Plot moving average prices vs actual closing price
fig, ax = plt.subplots(figsize = (15,10))
ax.plot(df['timestamp'], df['close'], label='Closing', color='blue')
ax.plot(df['timestamp'],df['MA for 10 days'], label='MA for 10 days', color='green')
ax.plot(df['timestamp'],df['MA for 20 days'], label='MA for 20 days', color='red')
ax.plot(df['timestamp'],df['MA for 50 days'], label='MA for 50 days', color='purple')
ax.set_title('Moving Averages')
ax.set_ylabel('Stock Price')
ax.set_xlabel('Date')
ax.legend()
plt.gca().xaxis.set_major_locator(plt.MaxNLocator(20))  # Set the maximum number of ticks
plt.gcf().autofmt_xdate()
```




```
# Create new column of just the year
df['year'] = df['timestamp'].dt.year


# Create new dataframe of prices after 2018 to get a closer look at the moving averages vs price
af = df.loc[df['year'] > 2018]


# Plotting the moving averages vs closing price
fig, ax = plt.subplots(figsize = (15,10))
ax.plot(af['timestamp'], af['close'], label='Closing', color='blue')
ax.plot(af['timestamp'],af['MA for 10 days'], label='MA for 10 days', color='green')
ax.plot(af['timestamp'],af['MA for 20 days'], label='MA for 20 days', color='red')
ax.plot(af['timestamp'],af['MA for 50 days'], label='MA for 50 days', color='purple')
ax.set_title('Moving Averages')
```

```
ax.set_ylabel('Stock Price')
ax.set_xlabel('Date')
ax.legend()
plt.gca().xaxis.set_major_locator(plt.MaxNLocator(20))  # Set the maximum number of ticks
plt.gcf().autofmt_xdate()
```




```
# Create a new dataframe of relevant columns for baseline model
df_main = df[['timestamp', 'close', 'average']]


# Create a 50 day rolling average column
df_main['50day'] = df_main['close'].rolling(50).mean()
```

```
<ipython-input-17-efc92c76b392>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  df_main['50day'] = df_main['close'].rolling(50).mean()
```

```
# Create a 200 day rolling average column
df_main['200day'] = df_main['close'].rolling(200).mean()
```

```
<ipython-input-18-9383322f8378>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  df_main['200day'] = df_main['close'].rolling(200).mean()
```

```
# Create a training set of 80% of the data
train_percentage = 0.8
split_index = int(df.shape[0]*train_percentage)


# Create a dataframe of the 20% testing data
df_main_test = df_main.iloc[split_index :, :]


def calculate_rmse(y_true, y_pred):
    """
    Calculate the Root Mean Squared Error (RMSE)
    """
    rmse = np.sqrt(np.mean((y_true - y_pred) ** 2))
    return rmse


def calculate_perf_metrics(var):
    '''
      RMSE of the actual closing price compared rolling average specified by var
    '''
    rmse = calculate_rmse(
        np.array(df_main[split_index:]["close"]),
        np.array(df_main[split_index:][var]),
    )

    return rmse


# Calculating the RMSE of the 50 day rolling average model
rmse_sma_50 = round(calculate_perf_metrics(var='50day'), 2)
print(rmse_sma_50)
```

```
135.39
```

```
# Calculating the RMSE of the 200 day rolling average model
rmse_sma_200 = round(calculate_perf_metrics(var='200day'), 2)
rmse_sma_200
```

```
304.88
```

```
# Creating series variables for date, 50 day, 200 day, close, and average
date = df_main_test['timestamp']
sma_50 = df_main_test['50day']
sma_200 = df_main_test['200day']
close = df_main_test['close']
avg = df_main_test['average']


# Plotting the closing value price vs the rolling average predictions on the test data
date_cent = date[len(date)/2]
y_point_1 = max(close) - 100
y_point_2 = max(close) - 200

fig, ax = plt.subplots(figsize= (17,8))
ax.plot(date, sma_50, label='50 day')
ax.plot(date, sma_200, label='200 day')
ax.plot(date, close, label='Actual close')
ax.text(date_cent, y_point_1, f'50 day: ${rmse_sma_50} off on avg', ha='center', va='center', fontsize=12, color='red')
ax.text(date_cent, y_point_2, f'200 day: ${rmse_sma_200} off on avg', ha='center', va='center', fontsize=12, color='red')
plt.grid(False)
plt.title('Simple Moving Averages')
plt.axis('tight')
plt.ylabel('Stock Price ($)')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7874a1d81bd0>
```

LSTM MODELING

```
# Creating numpy arrays for high price, low price, and average price
high_prices = df.loc[:, 'high'].values
low_prices = df.loc[:, 'low'].values
mid_prices = df.loc[:, 'average'].values


# Reshaping the mid price array
mid_prices = mid_prices.reshape(-1, 1)


# Creating a MinMaxScaler object and scaling the mid prices array
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(mid_prices)


scaled_data


# Creating the train test splits based on the split index variable
train = scaled_data[:split_index]
test = scaled_data[split_index-60:, :]


# Creating the training dataset from the scaled data

x_train = []
y_train = []

for i in range(60, len(train)):
    x_train.append(train[i-60:i])
    y_train.append(train[i, 0])
    if i<= 61:
        print(x_train)
        print(y_train)

# Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)
```

Initial LSTM Model with Batch Size of 1 and 1 Epoch

```
# Creating an initial LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
```

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

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

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

```
4100/4100 [==============================] - 156s 37ms/step - loss: 6.1907e-05
<keras.callbacks.History at 0x7b01f84c77f0>
```

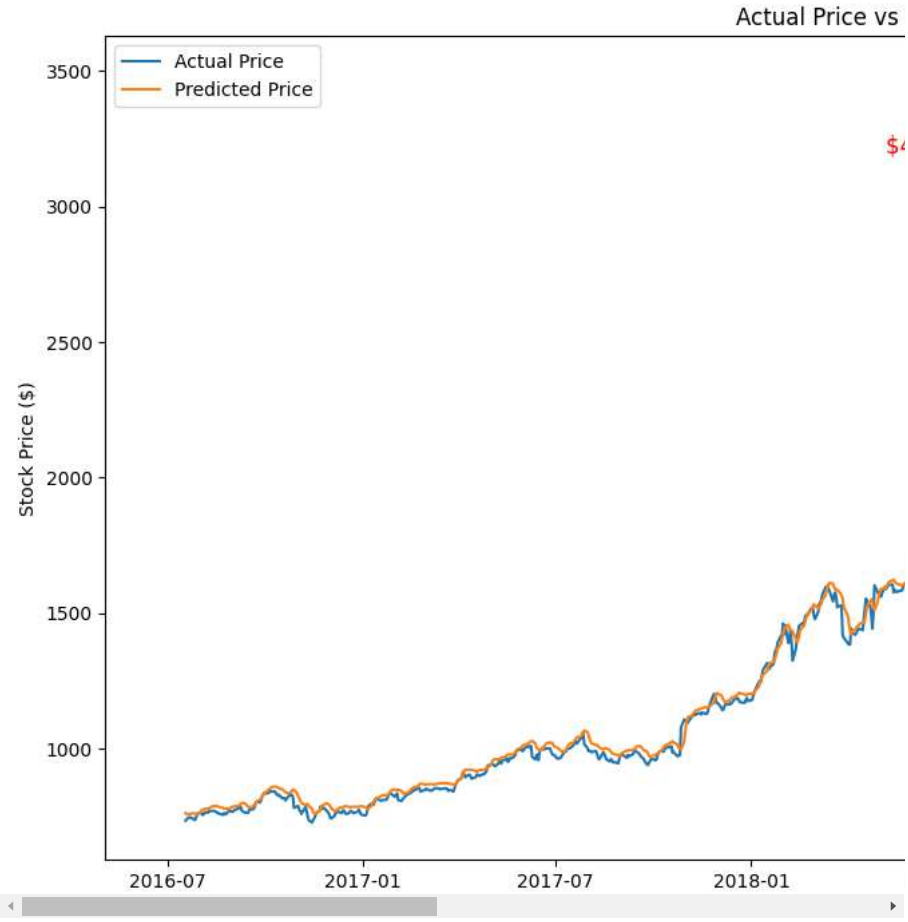
```
# Create the data sets x_test and y_test
x_test = []
y_test = []
y_test = mid_prices[split_index:, :]
for i in range(60, len(test)):
    x_test.append(test[i-60:i, 0])

# Convert the data to a numpy array
x_test = np.array(x_test)
##y_test = np.array(y_test)

# Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))

# Get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

##y_test = y_test.reshape(-1, 1)
y_test_inv = scaler.inverse_transform(y_test)

# Get the root mean squared error (RMSE)
rmse = np.sqrt(np.mean(((predictions - y_test) ** 2)))
mse = mean_squared_error(y_test, predictions, squared=False)
print(rmse, mse)
```

```
33/33 [==============================] - 2s 37ms/step
89.01960606440721 89.01960606440721
```

```
# Creating a function to quickly run LSTM models
def run_LSTM_model(num_epoch, num_neurons_1, num_neurons_2, batch_size):
    ''' Creates a model using these variables:
        num_epoch - number of epochs to fit the model
        num_neurons_1 - number of units for the first level of neural net
        num_neurons_2 - number of units for the second level of neural net
        batch_size - batch size when fitting the model

        Then plots how the model performs
    '''
    mdl = Sequential()
    mdl.add(LSTM(num_neurons_1, return_sequences=True, input_shape=(x_train.shape[1], 1)))
    mdl.add(LSTM(num_neurons_2, return_sequences=False))
    mdl.add(Dense(25))
    mdl.add(Dense(1))

# Compile the model
    mdl.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
    mdl.fit(x_train, y_train, batch_size=batch_size, epochs=num_epoch)
    preds = mdl.predict(x_test)
    preds = scaler.inverse_transform(preds)

    # Get the root mean squared error (RMSE)
    rmse = round(mean_squared_error(y_test, preds, squared=False), 2)

    # Getting the coordinates to place text inside the plot
    date_center = date[len(date)/2]
    y_point = max(preds) - 100

    # Plotting the predicted prices vs the average prices
    fig, ax = plt.subplots(figsize=(17,8))
    ax.plot(date, avg, label='Actual Price')
    ax.plot(date, preds, label='Predicted Price')
    ax.text(date_center, y_point, f'${rmse} off on avg', ha='center', va='center', fontsize=12, color='red')
    plt.grid(False)
```

```
    plt.grid(False)
    plt.title(f'Actual Price vs Predicted with Batch Size of {batch_size}')
    plt.axis('tight')
    plt.ylabel('Stock Price ($)')
    plt.legend()

    return rmse, preds


run_LSTM_model(1, 168, 64, 1)
```

```
 4100/4100 [==============================] - 231s 55ms/step - loss: 5.3076e-05
 33/33 [==============================] - 2s 50ms/step
 47.58
```

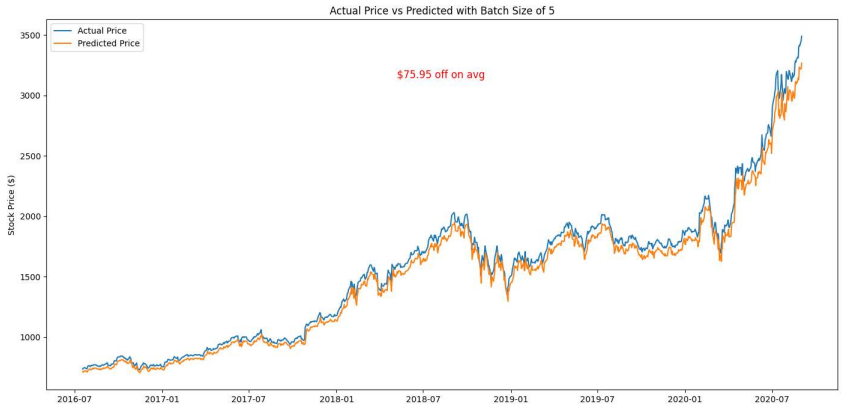


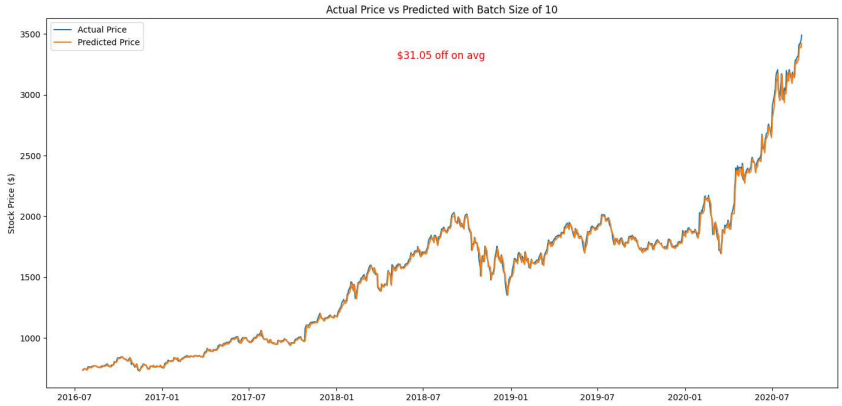
LSTM Models with Hyperparameter Tuning

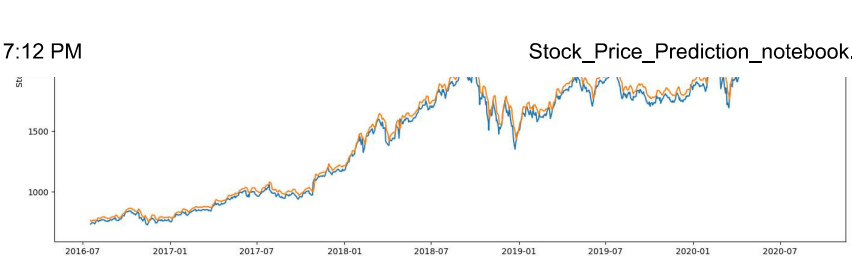
```
# Running LSTM models for the 4 batch sizes
batch_list = [5, 10, 15, 20]

for i in batch_list:
    run_LSTM_model(15, 168, 63, i)
```

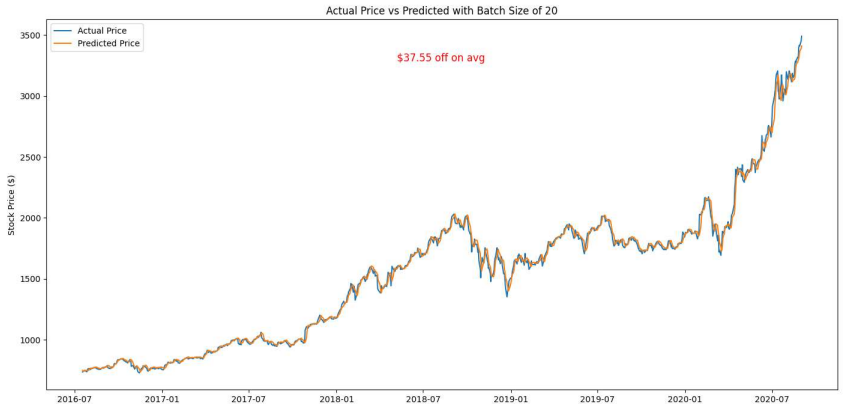
```
Epoch 1/15
820/820 [==============================] - 78s 91ms/step - loss: 5.2131e-05
Epoch 2/15
820/820 [==============================] - 75s 92ms/step - loss: 1.5894e-05
Epoch 3/15
820/820 [==============================] - 76s 93ms/step - loss: 1.5555e-05
Epoch 4/15
820/820 [==============================] - 76s 92ms/step - loss: 9.8530e-06
Epoch 5/15
820/820 [==============================] - 74s 90ms/step - loss: 8.4373e-06
Epoch 6/15
820/820 [==============================] - 76s 92ms/step - loss: 8.4795e-06
Epoch 7/15
820/820 [==============================] - 77s 94ms/step - loss: 7.9267e-06
Epoch 8/15
820/820 [==============================] - 73s 90ms/step - loss: 6.5094e-06
Epoch 9/15
820/820 [==============================] - 76s 92ms/step - loss: 7.2310e-06
Epoch 10/15
820/820 [==============================] - 74s 91ms/step - loss: 4.5989e-06
Epoch 11/15
820/820 [==============================] - 76s 93ms/step - loss: 7.2315e-06
Epoch 12/15
820/820 [==============================] - 74s 90ms/step - loss: 6.0234e-06
Epoch 13/15
820/820 [==============================] - 75s 91ms/step - loss: 4.8625e-06
Epoch 14/15
820/820 [==============================] - 74s 91ms/step - loss: 6.3243e-06
Epoch 15/15
820/820 [==============================] - 77s 94ms/step - loss: 4.3490e-06
33/33 [==============================] - 2s 49ms/step
Epoch 1/15
410/410 [==============================] - 46s 105ms/step - loss: 3.5728e-05
Epoch 2/15
410/410 [==============================] - 44s 108ms/step - loss: 1.1732e-05
Epoch 3/15
410/410 [==============================] - 43s 104ms/step - loss: 1.1536e-05
Epoch 4/15
410/410 [==============================] - 43s 104ms/step - loss: 8.0729e-06
Epoch 5/15
410/410 [==============================] - 45s 109ms/step - loss: 8.9468e-06
Epoch 6/15
410/410 [==============================] - 47s 114ms/step - loss: 8.4344e-06
Epoch 7/15
410/410 [==============================] - 43s 105ms/step - loss: 6.7533e-06
Epoch 8/15
410/410 [==============================] - 43s 104ms/step - loss: 8.4667e-06
Epoch 9/15
410/410 [==============================] - 46s 111ms/step - loss: 4.2227e-06
Epoch 10/15
410/410 [==============================] - 43s 104ms/step - loss: 4.5559e-06
Epoch 11/15
410/410 [==============================] - 42s 104ms/step - loss: 4.7731e-06
Epoch 12/15
410/410 [==============================] - 44s 108ms/step - loss: 3.8324e-06
Epoch 13/15
410/410 [==============================] - 45s 109ms/step - loss: 3.6017e-06
Epoch 14/15
410/410 [==============================] - 43s 104ms/step - loss: 6.1810e-06
Epoch 15/15
410/410 [==============================] - 43s 105ms/step - loss: 4.4133e-06
33/33 [==============================] - 4s 79ms/step
Epoch 1/15
274/274 [==============================] - 41s 133ms/step - loss: 8.5145e-05
Epoch 2/15
274/274 [==============================] - 36s 132ms/step - loss: 1.2695e-05
Epoch 3/15
274/274 [==============================] - 35s 127ms/step - loss: 1.3568e-05
Epoch 4/15
274/274 [==============================] - 39s 143ms/step - loss: 1.0233e-05
Epoch 5/15
274/274 [==============================] - 37s 134ms/step - loss: 7.4066e-06
Epoch 6/15
274/274 [==============================] - 37s 134ms/step - loss: 9.1436e-06
Epoch 7/15
274/274 [==============================] - 37s 136ms/step - loss: 8.7565e-06
Epoch 8/15
274/274 [==============================] - 36s 132ms/step - loss: 7.7038e-06
Epoch 9/15
274/274 [==============================] - 36s 133ms/step - loss: 5.6991e-06
Epoch 10/15
274/274 [==============================] - 37s 136ms/step - loss: 6.2255e-06
Epoch 11/15
274/274 [==============================] - 39s 142ms/step - loss: 6.4226e-06
```

```
Epoch 12/15
274/274 [==============================] - 35s 129ms/step - loss: 5.1096e-06
Epoch 13/15
274/274 [==============================] - 37s 136ms/step - loss: 5.0942e-06
Epoch 14/15
274/274 [==============================] - 36s 132ms/step - loss: 5.3105e-06
Epoch 15/15
274/274 [==============================] - 37s 135ms/step - loss: 7.3411e-06
33/33 [==============================] - 3s 66ms/step
Epoch 1/15
205/205 [==============================] - 29s 126ms/step - loss: 5.8433e-05
Epoch 2/15
205/205 [==============================] - 26s 125ms/step - loss: 1.2377e-05
Epoch 3/15
205/205 [==============================] - 26s 126ms/step - loss: 1.0503e-05
Epoch 4/15
205/205 [==============================] - 26s 125ms/step - loss: 1.1927e-05
Epoch 5/15
205/205 [==============================] - 26s 127ms/step - loss: 9.4027e-06
Epoch 6/15
205/205 [==============================] - 26s 125ms/step - loss: 8.5427e-06
Epoch 7/15
205/205 [==============================] - 28s 137ms/step - loss: 8.9975e-06
Epoch 8/15
205/205 [==============================] - 26s 124ms/step - loss: 7.4151e-06
Epoch 9/15
205/205 [==============================] - 26s 126ms/step - loss: 7.7480e-06
Epoch 10/15
205/205 [==============================] - 28s 138ms/step - loss: 5.0633e-06
Epoch 11/15
205/205 [==============================] - 26s 126ms/step - loss: 5.7714e-06
Epoch 12/15
205/205 [==============================] - 25s 123ms/step - loss: 4.6426e-06
Epoch 13/15
205/205 [==============================] - 26s 126ms/step - loss: 5.0638e-06
Epoch 14/15
205/205 [==============================] - 26s 126ms/step - loss: 4.8629e-06
Epoch 15/15
205/205 [==============================] - 26s 126ms/step - loss: 4.9542e-06
33/33 [==============================] - 3s 53ms/step
```



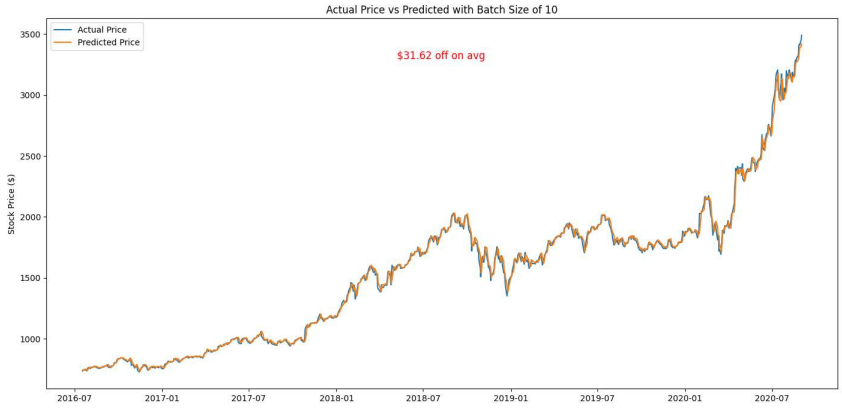



1500
1000
2016-07
2017-01
2017-07
2018-01
2018-07
2019-01
2019-07
2020-01
2020-07


Actual Price vs Predicted with Batch Size of 20
Actual Price
Predicted Price
$37.55 off on avg
Stock Price ($)
3500
3000
2500
2000
1500
1000
2016-07
2017-01
2017-07
2018-01
2018-07
2019-01
2019-07
2020-01
2020-07

```
# Saving the predictions from the best performing model
rm, stock_pred = run_LSTM_model(15, 168, 64, 10)
```

```
Epoch 1/15
410/410 [==============================] - 43s 96ms/step - loss: 5.7382e-05
Epoch 2/15
410/410 [==============================] - 39s 94ms/step - loss: 1.2716e-05
Epoch 3/15
410/410 [==============================] - 41s 100ms/step - loss: 1.0397e-05
Epoch 4/15
410/410 [==============================] - 40s 98ms/step - loss: 1.4486e-05
Epoch 5/15
410/410 [==============================] - 40s 98ms/step - loss: 9.9267e-06
Epoch 6/15
410/410 [==============================] - 39s 95ms/step - loss: 7.5678e-06
Epoch 7/15
410/410 [==============================] - 40s 97ms/step - loss: 6.9499e-06
Epoch 8/15
410/410 [==============================] - 39s 96ms/step - loss: 7.7378e-06
Epoch 9/15
410/410 [==============================] - 39s 96ms/step - loss: 6.7147e-06
Epoch 10/15
410/410 [==============================] - 40s 98ms/step - loss: 5.9893e-06
Epoch 11/15
410/410 [==============================] - 39s 95ms/step - loss: 7.4086e-06
Epoch 12/15
410/410 [==============================] - 39s 95ms/step - loss: 5.5651e-06
Epoch 13/15
410/410 [==============================] - 39s 94ms/step - loss: 5.9950e-06
Epoch 14/15
410/410 [==============================] - 38s 93ms/step - loss: 4.2258e-06
Epoch 15/15
410/410 [==============================] - 39s 96ms/step - loss: 4.1910e-06
33/33 [==============================] - 3s 66ms/step
```



```
# Creating a dataframe of the actual stock prices and predicted stock prices
stocks = pd.DataFrame({'actual_price': y_test[:, 0], 'predicted': stock_pred[:, 0]})


# Creating a new column which calculates the percentage change of the predicted column
stocks['percent_change'] = stocks['predicted'].pct_change() * 100
stocks.head()


''' Simulation which uses predicted percentage change thresholds to buy or sell stocks
    If predicted change is greater than threshold, buy as much stock as possible
    If predicted change is less than lower threshold, sell all the stock
    Use actual price to make calculations to see if prediction will work in real life situation.
    '''
initial_balance = 10000
stocks['base_money'] = initial_balance
stocks['StocksOwned'] = 0
x = 0

for i in range(len(stocks) - 1):
    if stocks['percent_change'][i + 1] > 3 and x == 0:
        # Buy as much stock as possible
        stocks_to_buy = int(stocks['base_money'][i] / stocks['actual_price'][i])
        stocks['StocksOwned'][i + 1] = stocks_to_buy
        stocks['base_money'][i + 1] = stocks['base_money'][i] - stocks['actual_price'][i]*stocks_to_buy
        x = 1
    elif stocks['percent_change'][i + 1] < -3 and stocks['StocksOwned'][i] != 0 and x == 1:
        # Sell all owned stock
        stocks['base_money'][i + 1] = stocks['StocksOwned'][i] * stocks['actual_price'][i + 1]
        stocks['StocksOwned'][i + 1] = 0
        x = 0
    else:
        # No significant movement, do nothing
        stocks['base_money'][i + 1] = stocks['base_money'][i]
        stocks['StocksOwned'][i + 1] = stocks['StocksOwned'][i]


# Check the final amount of stocks owned and the remaining balance
stocks.tail()
```

	actual_price	predicted	percent_change	base_money	StocksOwned
1035	3398.15310	3295.396484	0.013340	2788.065	4
1036	3415.50000	3354.036377	1.779449	2788.065	4
1037	3409.93495	3390.718262	1.093662	2788.065	4
1038	3450.00000	3394.728271	0.118268	2788.065	4
1039	3490.43500	3418.595703	0.703073	2788.065	4

By using the model to make trading decisions, an initial balance of 10000 is now over $16500!

4. CONCLUSION

Based on the the business problem and the data modeling, here are our recommendations for using the stock price predictions:

1. Focus on the trend, not the price: The model is still off on average by about $31, but it is following the overall trend well.
2. Use LSTM model for short term investments - day to day changes.
3. Use SMA as a factor for long term investments - easier to understand and less computationally expensive.