Gold ETF Price Predictor

Using Long-Short Term Memory Networks

CAPSTONE REPORT

Aishwarya S. Acharya

01.08.2020

CONTENTS

DEFINITION

Project Overview

Gold metal is universally deemed valuable. There is strong global market demand for gold. Gold is considered to be an ideal hedge for financial market risk. Gold also has been used to back money because of its limited volatility in price. Gold is particularly in high demand in Asian countries like India where gold is an indicator of opulence.

Gold exchange traded fund represents physical gold in its dematerialized or on paper form. This allows investors to invest in gold online without having to physically gold any gold. This prevents any convenience costs or storage costs and therefore have been gaining popularity. Gold ETFs are more beneficial than in jewelry form as pure gold would have to be mixed with impurities to make hard. Due to the current uncertainty, it may seem like Gold could be a potential option for risk free investing.

Gold ETFs can be termed as open-ended mutual fund schemes, which will invest the investors' money in standard gold bullion of 99.5% purity. Because of its open-ended nature, gold ETFs are traded on stock exchange just like the shares of any company. For many years investment firms have used modeling methodologies to be able to understand the market movements and predict reactions. Algorithms have the power to make a trade happen in a fraction of the time it takes for a human to click a button.

Problem Statement

This project aims in utilizing Deep learning models to predict Gold ETF prices using historical information as a time series. I will analyze the market movements over 5 years and predict the price in the future. We will compare this against a regression model. I will use linear, long-short term memory (LSTM).

The inputs will contain multiple metrics such as adjusted closing price. We will try to predict the future adjusted closing price after training the model.

Metrics

As this is a regression model, we will be using root mean squared error (RMSE) and mean squared error (MSE) to evaluate the model. The RMSE is the square root of the variance of the residuals. Residuals are a measure of how far from the line the data points are. It indicates the absolute fit of the model to the data–how close the observed data points are to the model's predicted values. Whereas R-squared is a relative measure of fit, RMSE is an absolute measure of fit.

$$RMSE = \sqrt{(f-o)^2}$$

Here, f is the forecasted value and o is the actual value.
RMSE is nonnegative and 0 indicates a perfect fir.

We are calculating the final model using R2 score. Based on the maximum R2 value we are iterating through layers and epochs to find the best fit model. R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

R-squared = Explained variation / Total variation

R-squared is always between 0 and 100%:

- 0% indicates that the model explains none of the variability of the response data around its mean.
- 100% indicates that the model explains all the variability of the response data around its mean.

In general, the higher the R-squared, the better the model fits the data. However, there are important conditions for this guideline that I'll talk about both in this post and my next post.

ANALYSIS

Data Exploration

In this project we are using prominent Gold ETFs in: GLD SPDR. However, the model is robust to be able to be used on any ETF as well.

The data used in the project is of Gold SPDR shares from January 1, 2012 to July 20, 2020. The goal is to predict the adjusted closing price after training. The data has been pulled from the Yahoo Finance.

I will be using historical data provided from Yahoo Finance which has 7 columns Date, Open, High, Low, Close, Adjusted close, Volume. The adjusted close takes into consideration dividends, stock splits and new offerings while close is simply the end of the day closing price of the ETF. I will pick a period of 5 years from 2014 to 2019 for training while 2019 to 2020 will be used to validate the data.

Date	High	Low	Open	Close	Volume	Adj Close
2012-01-03	156.300003	154.550003	154.759995	155.919998	13385800	155.919998
2012-01-04	157.380005	155.339996	155.429993	156.710007	11549700	156.710007
2012-01-05	158.029999	155.250000	155.369995	157.779999	11621600	157.779999
2012-01-06	158.630005	156.380005	158.589996	157.199997	9790500	157.199997
2012-01-09	157.589996	156.190002	157.360001	156.500000	8771900	156.500000
2012-01-10	159.470001	158.470001	158.970001	158.639999	8371400	158.639999
2012-01-11	160.050003	158.910004	159.339996	159.669998	7968500	159.669998
2012-01-12	161.619995	159.830002	161.020004	160.380005	8602500	160.380005
2012-01-13	159.589996	158.009995	159.320007	159.259995	8910300	159.259995

Table: The whole data can be found in 'GLD.csv' in the project folder

Note: There were no observable abnormalities in the data, no empty feature and no negative values what may seem incorrect

The mean, standard deviation, minimum, maximum and quartiles of the data are given below

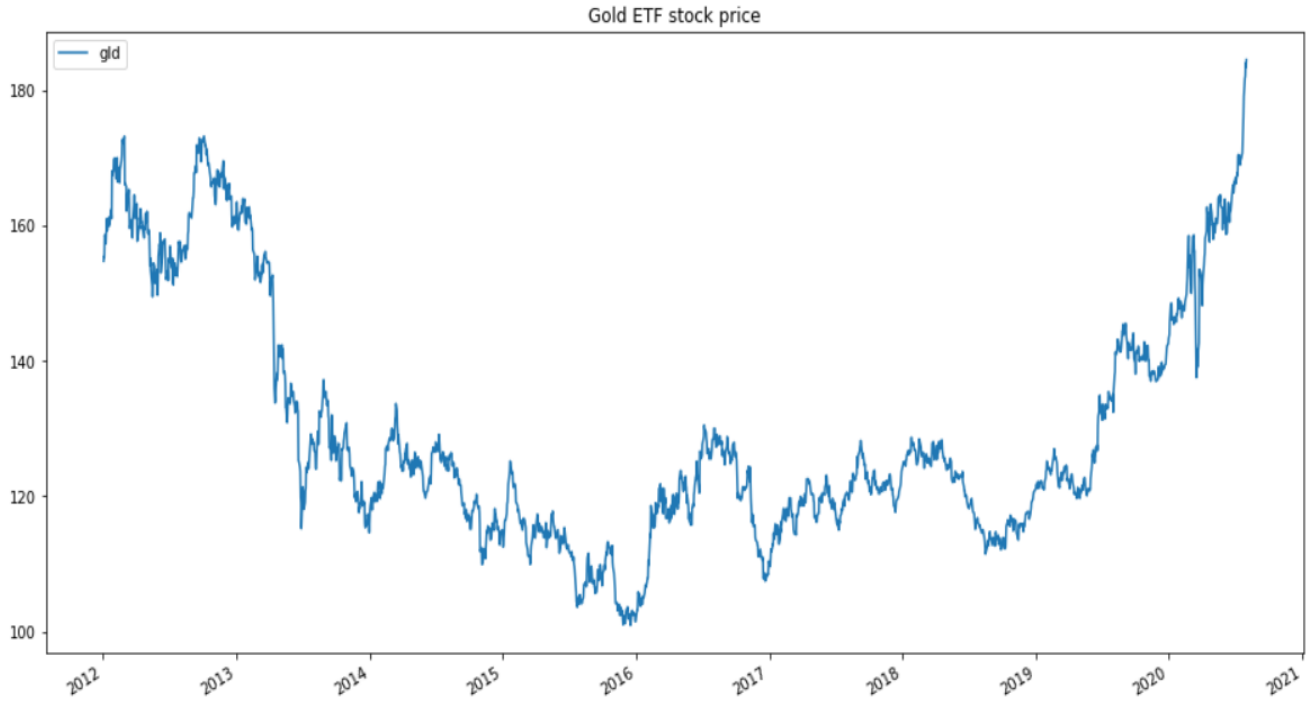
	High	Low	Open	Close	Volume	Adj Close
count	2159.000000	2159.000000	2159.000000	2159.000000	2.159000e+03	2159.000000
mean	130.320139	129.192978	129.763196	129.768731	8.882413e+06	129.768731
std	18.041996	17.774175	17.918821	17.941897	5.255093e+06	17.941897
min	100.989998	100.230003	100.919998	100.500000	1.501600e+06	100.500000
25%	118.009998	117.170002	117.575001	117.580002	5.564500e+06	117.580002
50%	124.089996	123.250000	123.699997	123.650002	7.694800e+06	123.650002
75%	140.635002	139.325005	140.125000	140.129997	1.072710e+07	140.129997
max	186.139999	184.169998	184.509995	185.429993	9.380420e+07	185.429993

We can conclude that stock price high, low, volume are not crucial to this analysis. It does not matter what the maximum or minimum price of the stock is. If the Adjusted closing price is greater than the opening price then there is a profit otherwise there is a loss. This is why we will be focussing on the adjusted closing price for this analysis

	Adj Close
Date	
2020-07-06	167.979996
2020-07-07	169.039993
2020-07-08	170.089996
2020-07-09	169.630005
2020-07-10	169.190002
2020-07-13	169.399994
2020-07-14	170.190002

Exploratory Visualization

To visualize the data we have used matplotlib library. We have plotted Adjusted closing price vs the year to visualize the data in US dollars



X-axis: Trading days over the years
Y-axis: Represents GLD SPDR shares in USD

Algorithms and Techniques

The goal of this project is to study time series data and accurately predict the future closing share price. We will be using 3 models of increasing levels of accuracy to predict the price.

LSTM Model:
A powerful type of neural network designed to handle sequence dependence is called recurrent neural networks. Recurrent neural networks have connections that have loops, adding feedback and memory to the networks over time. This memory allows this type of network to learn and generalize across sequences of inputs rather than individual patterns.

A powerful type of Recurrent Neural Network called the Long Short-Term Memory Network has been shown to be particularly effective when stacked into a deep configuration, achieving state-of-the-art results on a diverse array of problems from language translation to automatic captioning of images and videos.

- Input Parameters:
  - The input parameters include batch size, number of time steps, hidden size
- Neural Network Architecture
  - Number of Layers in the model
  - Number of nodes per layer
- Training Parameters:
  - Training and test split of 80% training and 20%testing data
  - Batch Size
  - Loss function
  - Optimizer Function to minimize error
  - Epochs

We have also used Moving average and Linear Regression network models to compare errors and reach the most efficient model.

Lineal Regression Model:

We have used linear regression model to compare the relative performance and implementation differences of the two model.

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable.

A linear regression line has an equation of the form Y = a + bX, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when x = 0).

The most common method for fitting a regression line is the method of least-squares. This method calculates the best-fitting line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line.

Benchmark

We are using the moving average model as a benchmark model. A moving average term in a time series model is a past error. The moving-average model specifies that the output variable depends linearly on the current and various past values of a stochastic (imperfectly predictable) term.

A simple Moving Average (MA) model looks like this:

$$r_t = c + \theta_1 \epsilon_{t-1} + \epsilon_t$$

Where,

r is a time series variable

c is a constant factor

Theta is a numeric coefficient for the value associated with the first tag

Epsilon are the residuals for the current and previous periods

METHODOLOGY

Data Processing

We have scrapped the SPDR GLD ETF prices from Yahoo finance and saved it in GLD.csv file.

Date	High	Low	Open	Close	Volume	Adj Close
2012-01-03	156.300003	154.550003	154.759995	155.919998	13385800	155.919998
2012-01-04	157.380005	155.339996	155.429993	156.710007	11549700	156.710007
2012-01-05	158.029999	155.250000	155.369995	157.779999	11621600	157.779999
2012-01-06	158.630005	156.380005	158.589996	157.199997	9790500	157.199997
2012-01-09	157.589996	156.190002	157.360001	156.500000	8771900	156.500000

As explained earlier, we will be focussing on the Adjusted closing price in this analysis to predict the future closing price.

Date	Adj Close
2020-07-06	167.979996
2020-07-07	169.039993
2020-07-08	170.089996
2020-07-09	169.630005
2020-07-10	169.190002
2020-07-13	169.399994
2020-07-14	170.190002

Our data seemed to not have any unusual values, negative numbers of blanks. The pre-processing methodologies used are splitting our data in to training and test sets and then scaling the data between 0 and 1. We are splitting the data into 80% training data stored as train_data and 20% resting data stored in test_data.

For purpose of scaling, we use the MinMaxScalar from sklearn. The MinMaxScaler transforms features by scaling each feature to a given range. This range can be set by specifying the feature_range parameter (default at (0,1)). This scaler works better for cases where the distribution is not Gaussian or the standard deviation is very small.

$$x\_scaled = (x\text{-}min(x)) / (max(x) - min(x))$$

```
Scaled training data   Scaled test data
  [[0.75803581],       [[0.79810516]
   [0.76884156],        [0.82680793]
   [0.78347693],        [0.86345285]
   [0.77554365],        [0.88504945]
   [0.76596908],        [0.90622827]
   [0.79524003],        [0.95541316]
   [0.80932838],        [0.97659199]
   [0.81903986],        [0.99582015]
   [0.80372034],        [0.97673125]
   [0.82068116]]        [1.        ]]
```

The data split is training (80%), test (20%) for all 3 models and the shape of the data split is as below:

- x_train (1725, 1)
- y_train (1725,)
- x_test (430, 1)
- y_test (430, 1)

Implementation, Refinement, Evaluation and Visualization

We are following the below steps for to implement our models

- **Step 1**: Split the data in to train and test sets
- **Step 2**: Define model architecture
- **Step 3**: Train and generate predictions
- **Step 4**: Plot prediction and calculate RMSE

Benchmark Model (Moving average model)

Step 1: Create spit training and testing data

```
#Split data 80% training 20% testing

training_size=int(len(data)*0.8)
test_size=len(data)-training_size
train_data,test_data=data[0:training_size],data[training_size:len(data)]
```

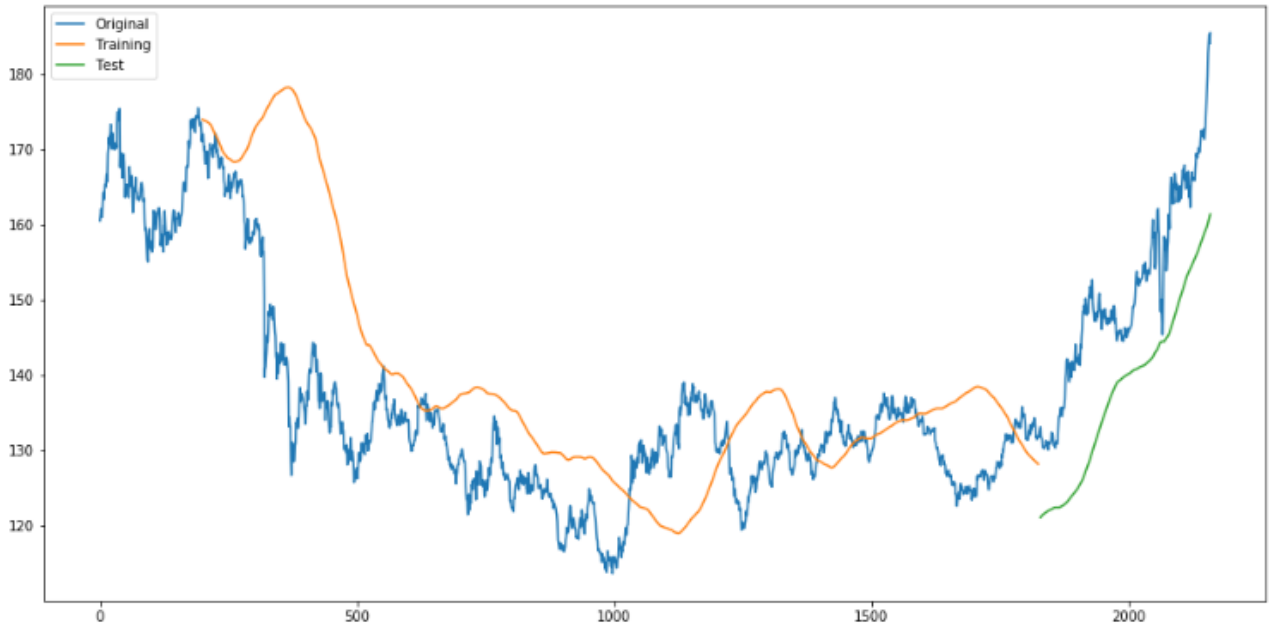
```
#Moving average coefficients
x_train_ma = x_train
y_train_ma = y_train
x_test_ma = x_test
y_test_ma = y_test
```

Step 2&3: Building the model and predicting prices

```
#Training moving average transform
rolling_1 = df_x_train_ma.rolling(window)
train_pred_ma = rolling_1.mean()
```

```
# Testing moving average transform
rolling_2 = df_x_test_ma.rolling(window)
test_pred_ma = rolling_2.mean()
```

Step 4: Plot the prediction and calculate test score



X-axis: Trading days

Y-axis: Represents GLD SPDR shares in USD

Blue line: Actual price

Orange line: Training price

Green line: Testing price

Train RMSE: 9.97543526

Test RMSE: 10.97595171

RESULTS

Linear Regression Model

A linear regression line has an equation of the form Y = a + bX, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when x equals 0).

The most common method for fitting a regression line is the method of least-squares. This method calculates the best-fitting line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line.

Step 1: Create spit training and testing data using the same as mentioned earlier

```
#Regression coefficients
x_train_lr = x_train
y_train_lr = y_train
x_test_lr = x_test
y_test_lr = y_test
```

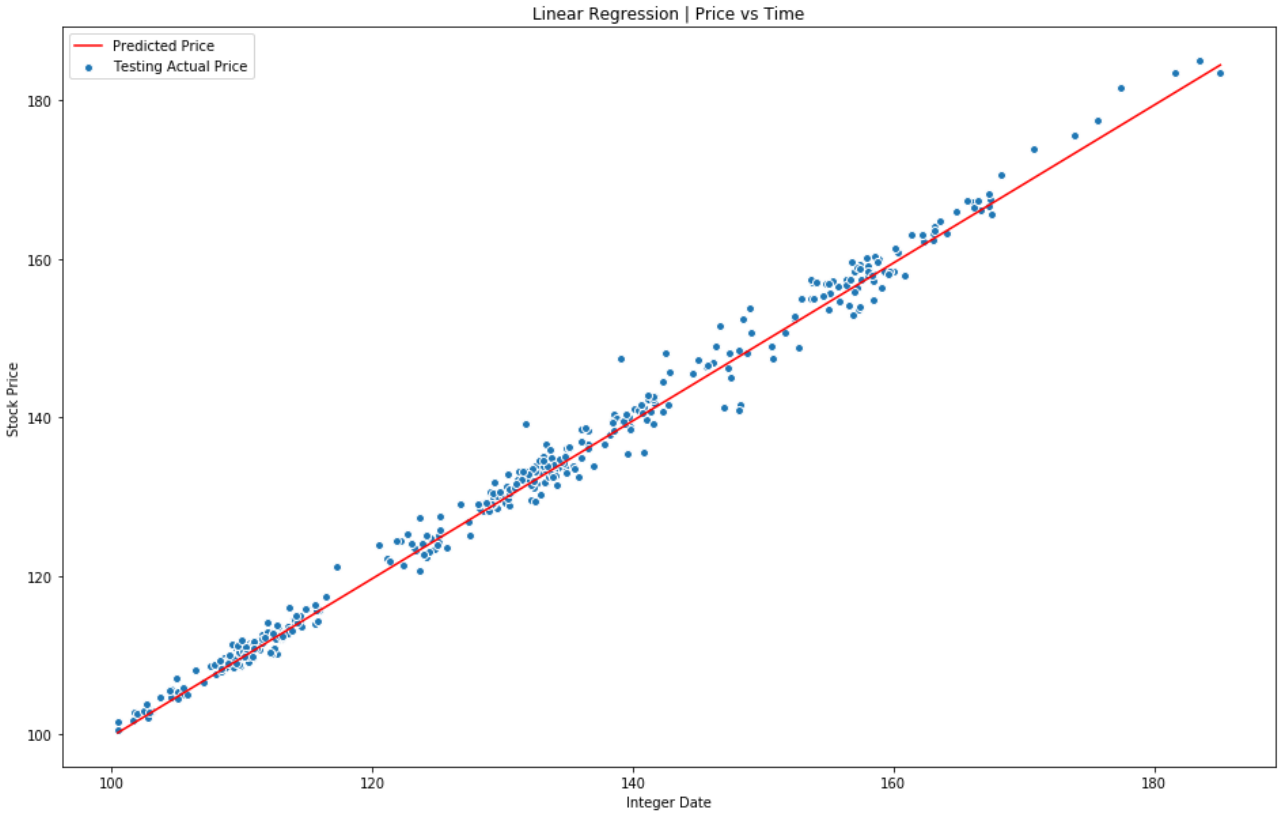
Step 2: Building the model using scikit-learn library

```
#Create linear regression model
modellr = LinearRegression()
modellr.fit(x_train_lr, y_train_lr)
```

Step 3: Training and predicting prices

```
# Generate array with predicted values
train_pred_lr = modellr.predict(x_train_lr)
test_pred_lr = modellr.predict(x_test_lr)
```

Step 4: Plot the prediction and calculate test score



X-axis: Trading days

Y-axis: Represents GLD SPDR shares in USD

Red line: Predicted price

Blue dots: Actual price

Train RMSE: 1.4798217

Train MSE: 2.1898725

Test RMSE: 1.71573706

Test MSE: 2.94375367

Long short term memory (LSTM) Model

We are using tensorflow and keras library to implement LSTM model for the project.

Step 1: Create spit training and testing data using the same as mentioned earlier

```
#LSTM coefficients
x_train_lstm = x_train
y_train_lstm = y_train
x_test_lstm = x_test
y_test_lstm = y_test
```

Step 2: Building the model with 3 layers and one dropout layer. We are using different inputs for the layers and iterating over them to find the best combination of layers.

```
def model_funt(layer):
        model_1=Sequential()
        model_1.add(LSTM(layer[0],return_sequences=True,input_shape=(5,1)))
        model_1.add(Dropout(0.3))
        model_1.add(LSTM(layer[1]))
        model_1.add(Dense(layer[2]))
        model_1.add(Activation("linear"))

        return model_1
```

Step 3: Training and predicting prices: Initially we were implementing this with hardcoded layer inputs in the model but eventually realized that it is much more efficient to automate this process with multiple inputs. We are also iterating over different input epochs in our training process. This was a significant step in improvising the model and help reduce the RMSE very significantly.

We are also plotting the training and validation loss functions to check over fitting.

```
#model.fit(x_train_lstm,y_train_lstm,validation_data=(x_test_lstm,y_test_lstm),epochs=50,batch_size=64,verbose=1)
epochs =[50, 75, 100]

layers = [[4,5,1], [4,4,1], [50,50,1], [4,3,1], [4,6,1], [4,2,1], [4,1,1], [10,10,1], [100,100,1]]
model = 1
final_model = None
max = -100000
for epoch in epochs:
```

```
for epoch in epochs:
    for layer in layers:
        model_1 = model_funt(layer)
        model_1.compile(optimizer='adam', loss='mean_squared_logarithmic_error')
        history = model_1.fit(x_train_lstm, y_train_lstm, validation_data=(x_test_lstm,y_test_lstm),
                              epochs=epoch,batch_size=64,verbose=1)
        train_pred_lstm = model_1.predict(x_train_lstm)
        print (model, " mean is: ", np.mean(train_pred_lstm), " variance is: ", np.var(train_pred_lstm))
        if(max < (r2_score(y_train_lstm, train_pred_lstm))):
            max = r2_score(y_train_lstm, train_pred_lstm)
            final_model = model_1
            final_epoch = epoch
            final_layers = layers

        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('model train vs validation loss')
        plt.title('model train vs validation loss: Loss Function: ' + str(round(history.history['loss'][-1], 6)))
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train', 'validation'], loc='upper right')
        plt.show()
```

```
Epoch 5/50
27/27 [==============================] - 0s 3ms/step - loss: 0.0317 - val_loss: 0.0277
Epoch 6/50
27/27 [==============================] - 0s 3ms/step - loss: 0.0262 - val_loss: 0.0242
Epoch 7/50
27/27 [==============================] - 0s 3ms/step - loss: 0.0237 - val_loss: 0.0227
Epoch 8/50
27/27 [==============================] - 0s 3ms/step - loss: 0.0226 - val_loss: 0.0217
Epoch 9/50
27/27 [==============================] - 0s 3ms/step - loss: 0.0216 - val_loss: 0.0208
Epoch 10/50
27/27 [==============================] - 0s 3ms/step - loss: 0.0200 - val_loss: 0.0197
Epoch 11/50
27/27 [==============================] - 0s 3ms/step - loss: 0.0194 - val_loss: 0.0186
Epoch 12/50
27/27 [==============================] - 0s 3ms/step - loss: 0.0185 - val_loss: 0.0173
```

Over the iterations, over final model is below

```
final_model.summary()
```

```
Model: "sequential_26"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_52 (LSTM)               (None, 5, 100)            40800
_________________________________________________________________
dropout_26 (Dropout)         (None, 5, 100)            0
_________________________________________________________________
lstm_53 (LSTM)               (None, 100)               80400
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 101
_________________________________________________________________
activation_26 (Activation)   (None, 1)                 0
=================================================================
Total params: 121,301
Trainable params: 121,301
Non-trainable params: 0
_________________________________________________________________
```

Step 4: Plot the prediction and calculate test score

We are using the inbuild predict function to make the Predictions of our LSTM model

```
train_pred_lstm=final_model.predict(x_train_lstm)
test_pred_lstm=final_model.predict(x_test_lstm)
```

X-axis: Trading days

Y-axis: Represents GLD SPDR shares in USD

Blue line: Actual price

Orange line: Validation price

Green line: Predicted price

Train RMSE: 1.442452

Train MSE: 2.080667

Test RMSE: 1.646065

Test MSE: 2.709531

Justification

	Benchmark: Moving average	Linear Regression	LSTM
Train RMSE	9.975435	1.479821	1.442452
Test RMSE	10.975951	1.715737	1.646065

The benchmark moving average model clearly has a higher Root Mean Squared Error Linear regression and LSTM model. The linear regression significantly improved upon this error. Finally our improved LSTM model was able to generate an even lower RMSE.

We can conclude that our linear regression and LSTM model have significantly improved the RMSE error on both training and test data sets. There is a small improvement in the LSTM model from the linear regression model as well.

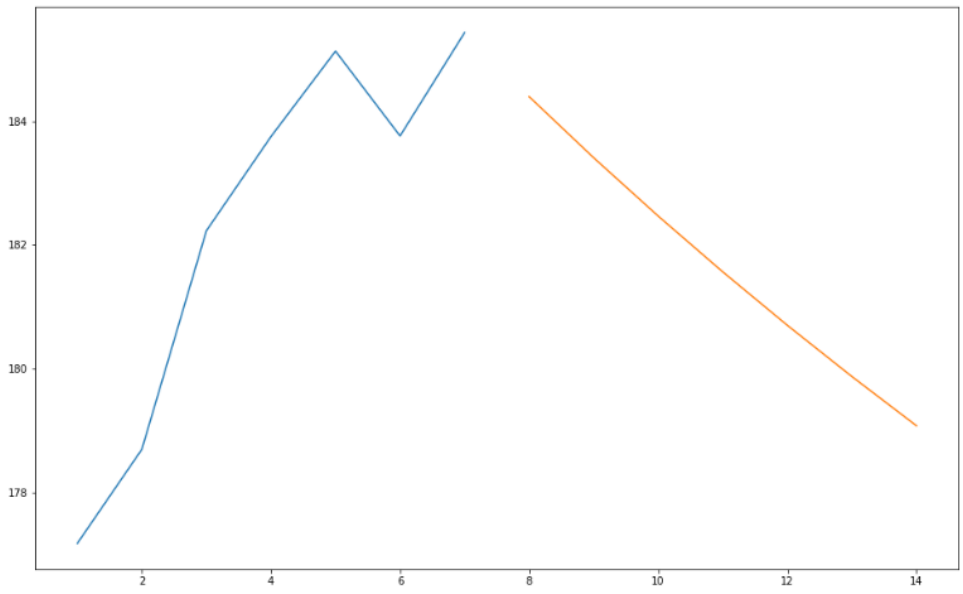
It can be concluded that our models have clearly surpassed the benchmark model and have significantly been able to solve the problem statement.

Predicting the future

Furthermore, we have used this model to predict the future price of the GLS SPDR ETF share. To accommodate seasonality we will be using the past 7 day data to predict the future 7 day price of the share.

```
lst_output=[]
n_steps=1
i=0
while(i<7):

    if(len(temp_input)>1):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = final_model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = final_model.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1
```



X-axis: Trading days

Y-axis: Represents GLD SPDR shares in USD

Blue line: Past price

Orange line: Predicted price