Commodity Price Predictor

By

Vibhav Lal

Udacity Connect Feb 2018

CapStone Project

Table of Contents

[Commodity Price Predictor 3](#_Toc519379912)

[Definition 3](#_Toc519379913)

[Project Overview 3](#_Toc519379914)

[Problem Statement 3](#_Toc519379915)

[Goals 3](#_Toc519379916)

[Metrics 3](#_Toc519379917)

[Analysis 4](#_Toc519379918)

[Data Exploration 4](#_Toc519379919)

[Exploratory Visualization 5](#_Toc519379920)

[Algorithms and Techniques 5](#_Toc519379921)

[Benchmark Model 7](#_Toc519379922)

[Methodology 8](#_Toc519379923)

[Data Preprocessing 8](#_Toc519379924)

[Implementation 10](#_Toc519379925)

[Benchmark model: 10](#_Toc519379926)

[Improved LSTM Model: 12](#_Toc519379927)

[Refinement 16](#_Toc519379928)

[Result 18](#_Toc519379929)

[Model Evaluation and Validation 18](#_Toc519379930)

[Linear Regression Model 18](#_Toc519379931)

[Basic Long-Short Term memory Model 19](#_Toc519379932)

[Improved Long-Short Term Memory Model 20](#_Toc519379933)

[Robustness Check 20](#_Toc519379934)

[Justification 21](#_Toc519379935)

[Conclusion 22](#_Toc519379936)

[Visualization 22](#_Toc519379937)

[Reflection 22](#_Toc519379938)

[Feature Engineering 23](#_Toc519379939)

[Improvement 23](#_Toc519379940)

# Commodity Price Predictor

# Definition

## Project Overview

Commodity is one of the largest traded (by volume) commodity Internationally as well as domestic. Gold spot prices are somewhat fragmented since there are various sources. No standard price, which is why there is no exchange for this product. No set price for any product, production cost differs from producer to producer. Multitude of categories of products/items/sizes/grades within the categories with substantial price differences.

For this project the commodity chosen was gold spot pricing due to the data availability on public sources like Quandl and Fred economic datasets.

The goal of this project is to use machine learning concepts and utilize Deep learning models, Long-Short Term Memory (LSTM) Neural Network algorithm to predict gold spot pricing. Traders and Investors make educated guesses by analyzing data from several sources. LSTM can be built using Keras to learn and predict spot pricing using historical closing prices.

For data with timeframes recurrent neural networks (RNNs) come in handy but recent researches have shown that LSTM, networks are the most popular and useful variants of RNNs.

## Problem Statement

The challenge of this project is to accurately predict the future closing value of a commodity spot price across a given period of time in the future. For this project I will use a ​Long Short Term Memory networks to predict the closing price of the commodity spot pricing using a dataset of past prices.

### Goals

1. Explore commodity spot pricing.
2. Explore feature importance for commodity spot pricing
3. Implement basic model using linear regression
4. Implement LSTM using KERAS library
5. Compare the results and submit the report

## Metrics

For this project measure of performance will be using the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) calculated as the difference between predicted and actual values of the target commodity spot pricing at close price and the delta between the performance of the benchmark model (Linear Regression) and our primary model (Deep Learning).

# Analysis

## Data Exploration

The data in this project is of Gold Spot Pricing from 1987 to 2018. This time series of data points indexed in time order. My goal is to predict the closing price for any given data after training. For ease of reproducibility and reusability, all data was pulled from FRED.

The prediction has to be made for Closing price of the data. The feature data was indexed in daily time order. However, some days were missing the close prices. There was some preprocessing required which is explained in the methodology section.

The dataset is of the follow form:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Date** | **Open** | **Close** | **USD1MTD156N** | **Silver** | **DGASNYH** | **DEXCHUS** |
| 6/22/18 | 1269.70 | 1269.15 | 2.10 | 16.43 | 2.01 | 6.50 |
| 6/21/18 | 1263.70 | 1266.15 | 2.09 | 16.25 | 1.96 | 6.49 |
| 6/20/18 | 1273.25 | 1274.20 | 2.08 | 16.29 | 1.97 | 6.47 |

Table 1. Raw data available in gold-spot-price.csv

Data Definition and sources:

|  |  |  |
| --- | --- | --- |
| **Commodity** | **Identifier** | **Source** |
| Conventional Gasoline Prices: New York Harbor, Regular | DGASNYH |  |
| China / U.S. Foreign Exchange Rate | DEXCHUS |  |
| 1-Month London Interbank Offered Rate (LIBOR), based on U.S. Dollar | USD1MTD156N |  |
| Gold Price: London Fixing | LBMA/GOLD | https://www.quandl.com/data/LBMA/GOLD-Gold-Price-London-Fixing |
| Silver Price: London Fixing | LBMA/SILVER | https://www.quandl.com/data/LBMA/SILVER |

Table 2. Data Source and definition

The mean, standard deviation, maximum and minimum of the data was found to be following:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Open** | **Close** | **USD1MTD156N** | **Silver** | **DGASNYH** | **DEXCHUS** |
| **count** | 8365 | 8365 | 8365 | 8365 | 8365 | 8365 |
| **mean** | 664.76 | 659.43 | 3.52 | 10.61 | 1.21 | 6.53 |
| **std** | 440.15 | 442.21 | 2.79 | 8.17 | 0.85 | 2.04 |
| **min** | 252.90 | 0.00 | 0.00 | 3.55 | 0.00 | 0.00 |
| **25%** | 349.00 | 347.60 | 0.45 | 4.99 | 0.55 | 5.79 |
| **50%** | 403.75 | 402.45 | 3.38 | 6.13 | 0.82 | 6.83 |
| **75%** | 1099.75 | 1093.25 | 5.66 | 16.00 | 1.83 | 8.28 |
| **max** | 1896.50 | 1895.00 | 10.31 | 48.70 | 3.67 | 8.74 |

Table 3. Raw data statistics

For the scope of predicting prices, the open and close spot price is what is relevant. If at the end of the data, we have a higher closing price than the opening price then we have some profits else loss. I will attempt to perform a feature importance study impacting the gold spot pricing in the subsequent sections, however, a full study is outside the scope of this project.

## Exploratory Visualization

To visualize the data using the matplotlib library, gold spot close prices against the number of trading available days as illustrated below.

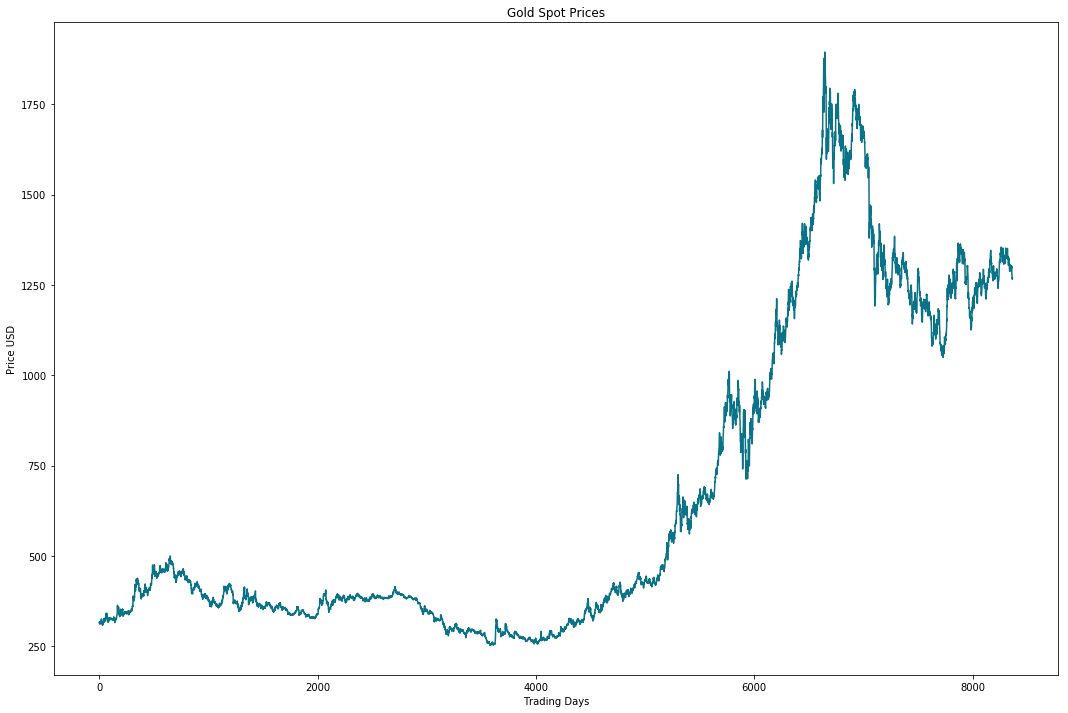


Figure 1. Closing Price (USD) versus Trading Days (1985-05-21 to 2018-06-22)

The target variable (close price) is a continuous variable with several fluctuations over time.

## Algorithms and Techniques

The goal of this project was to study time-series data and explore as many options as possible to accurately predict the commodity spot pricing.

I explored different methods to tackle this problem. Initially linear regression for benchmarking followed by Recurrent Neural Nets (RNN) which are used specifically for sequence and pattern learning. RNNs are networks that allow information to persist due its iterative structure thus memorizes the data patterns. In addition, RNN inhibit vanishing gradient descent which limits usage of this model from learning historical data. On the other hand, ​Long-Short Term Memory Networks (LSTM) are special kind of RNN, capable of learning long-term dependencies.

In addition to adjusting the architecture of the Neural Network, the following full set of parameters can be tuned to optimize the prediction model:

* Input Parameters
  + Preprocessing and Normalization (see Data Preprocessing Section)
* Neural Network Architecture
  + Number of Layers (how many layers of nodes in the model; used 5)
  + Number of Nodes (how many nodes per layer; tested 8, 16, 32, 64, and 128)
* Training Parameters
  + Training / Test Split (roughly 80% and 20% split for benchmarks and LSTM model).
  + Validation Sets (kept constant at 0.05% of training sets).
  + Batch Size (number of steps for a single training step) was set to 100 for LSTM model and 512 for the improved LSTM model.
  + Optimizer Function: used “Adam” throughout.
  + Epochs: 5 for basic model and 20 for improved LSTM.

## Benchmark Model

The primary benchmark model for this project is Linear Regression, the secondary goal for using this model was to understand the comparative performance and implementation nuances between machine learning and deep learning models. Linear Regression models were thoroughly discussed and practiced throughout the Udacity Machine Learning Connect course. Projects P1, P2, P3 served as good examples for understanding linear regression. Based on the knowledge gained the error rate comparison MSE and RMSE utilizing the same dataset as the deep learning models.

Figure below illustrates the predicted results from the Linear Regression model:

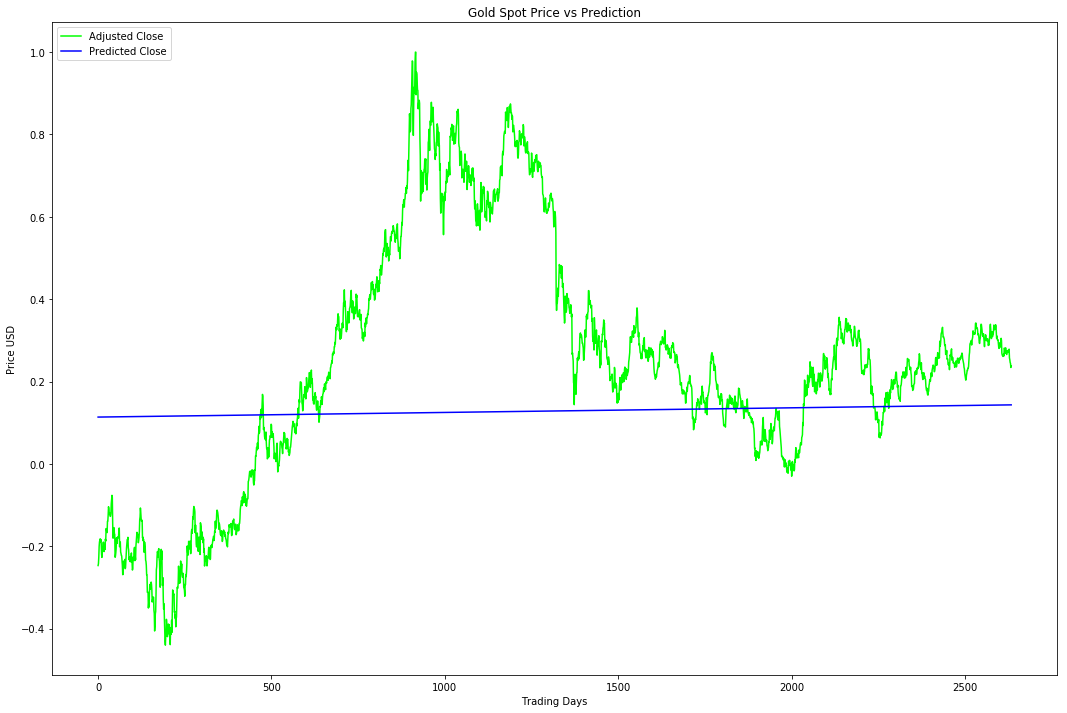


Figure 2. Adjusted Close Price vs Trading Days super-imposed Predicted Close price

Train Score: 0.4152 MSE (0.6443 RMSE)

Test Score: 0.09697600 MSE (0.31140969 RMSE)

# Methodology

## Data Preprocessing

Acquiring and preprocessing the data for this project occurs in following sequence, much of which has been modularized within the notebook and some was implemented into the ​preprocess.py​ file:

* Request the data from the Fred and save it in ​ gold-spot-price-post-trim.csv file in the following format.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Date** | **Open** | **Close** | **USD1MTD156N** | **Silver** | **DGASNYH** | **DEXCHUS** |
| 6/22/18 | 1269.70 | 1269.15 | 2.10 | 16.43 | 2.01 | 6.50 |
| 6/21/18 | 1263.70 | 1266.15 | 2.09 | 16.25 | 1.96 | 6.49 |
| 6/20/18 | 1273.25 | 1274.20 | 2.08 | 16.29 | 1.97 | 6.47 |
| 5/23/85 | 316.40 | 315.40 | 7.19 | 6.20 | 0.42 | 3.22 |
| 5/22/85 | 317.40 | 316.25 | 7.19 | 6.22 | 0.44 | 3.22 |
| 5/21/85 | 316.50 | 314.90 | 7.13 | 6.56 | 0.47 | 3.22 |

Table 4. Preprocessed data available in gold-spot-price.csv

* Fill the missing values using the average of the previous day and next day closing prices. In case previous and next day closing prices were not available, which was the case on December 29, 30, or 31, I used the open price as the close price for that day.
  + Some of the corresponding features such as Gasoline and Forex data had several entries where consecutive day prices were missing. Those entries were dealt in a similar manner.
  + Gasoline:
    - 2/10/06; 2/9/06
    - 5/11/01; 5/10/01
    - 3/27/97; 3/26/97
    - 2/18/94; 2/17/94
  + Forex:
    - 12/24/01
    - 8/28/92; 8/27/92
    - 8/7/92; 8/6/92
    - 6/12/92; 6/11/92
    - 3/20/92; 3/19/92
    - 1/10/92; 1/9/92
    - 12/31/91; 12/30/91; 12/27/91;
    - 12/31/90; 12/24/90
    - 4/20/90; 4/19/90; 4/18/90
* For the prediction analysis we have to remove irrelevant features (Forex, Silver, LIBOR, etc.) from the acquired data. The resulting data is the following structure:

|  |  |  |
| --- | --- | --- |
| **Item** | **Open** | **Close** |
| 0 | 316.5 | 314.9 |
| 1 | 317.4 | 316.25 |
| 2 | 316.4 | 315.4 |
| 3 | 316.35 | 314.7 |
| 4 | 313.9 | 311.25 |

* Normalized the data using ​MinMaxScaler ​ helper function from Scikit-Learn.

|  |  |  |
| --- | --- | --- |
| **Item** | **Open** | **Close** |
| 0 | 0.03870 | 0.03782 |
| 1 | 0.03924 | 0.03864 |
| 2 | 0.03864 | 0.03812 |
| 3 | 0.03860 | 0.03769 |
| 4 | 0.03711 | 0.03559 |

* Stored the normalized data in ​ gold\_spot\_preprocessed\_normalized.csv​ file for future reusability.
* Split the dataset into the training (68.52%) and test (31.48%) datasets for linear regression model. The split was of following shape:

x\_train (5731, 1)

y\_train (5731, 1)

x\_test (2634, 1)

y\_test (2634, 1)

* Split the dataset into the training (80%) and test (20%) datasets for LSTM model. The Split was of following shape:

x\_train (6609, 50, 2)

y\_train (6609,)

x\_test (1646, 50, 2)

y\_test (1646,)

## Implementation

Once the data has been downloaded and preprocessed, the implementation process occurs consistently through all three models as follow:

The flow below illustrates the implementation steps to build, train, and test model. The prediction results can be found the enclosed notebook.

As good practice and to keep the implementation modular the notebook is complimented with helper python functions which are imported and called throughout the notebook.

We will now delve in the code implementation for each of the models.

### Benchmark model

#### Step 1: Load the preprocessed data

Load the data from csv file to the DataFrame

#### Step 2: ​Split into train and test model:

X\_train, X\_test, y\_train, y\_test, label\_range= cd.train\_test\_split\_linear\_regression(commodity)

train\_test\_split\_linear\_regression helper functions defined in commodity\_data.py helps splits the data for linear regression. The function is described below:

def train\_test\_split\_linear\_regression(commodity):

"""

Split the data set into training and testing feature for Linear Regression Model

:param stocks: whole data set containing ['Open','Close'] features

:return: X\_train : training sets of feature

:return: X\_test : test sets of feature

:return: y\_train: training sets of label

:return: y\_test: test sets of label

:return: label\_range: scaled range of label used in predicting price,

"""

# Create numpy arrays for features and targets

feature = []

label = []

# Convert dataframe columns to numpy arrays for scikit learn

for index, row in commodity.iterrows():

# print([np.array(row['Item'])])

feature.append([(row['Item'])])

label.append([(row['Close'])])

# Regularize the feature and target arrays and store min/max of input data for rescaling later

feature\_bounds = [min(feature), max(feature)]

feature\_bounds = [feature\_bounds[0][0], feature\_bounds[1][0]]

label\_bounds = [min(label), max(label)]

label\_bounds = [label\_bounds[0][0], label\_bounds[1][0]]

feature\_scaled, feature\_range = scale\_range(np.array(feature), input\_range=feature\_bounds, target\_range=[-1.0, 1.0])

label\_scaled, label\_range = scale\_range(np.array(label), input\_range=label\_bounds, target\_range=[-1.0, 1.0])

# Define Test/Train Split 80/20

split = .215

split = int(math.floor(len(commodity['Item']) \* split))

# Set up training and test sets

X\_train = feature\_scaled[:-split]

X\_test = feature\_scaled[-split:]

y\_train = label\_scaled[:-split]

y\_test = label\_scaled[-split:]

return X\_train, X\_test, y\_train, y\_test, label\_range

#### Step 3: Using scikit-learn linear\_model library

model = LinearRegressionModel.build\_model(X\_train,y\_train)

Helper function LinearRegressionModel.py build the model for the data.

def build\_model(X, y):

"""

build a linear regression model using sklearn.linear\_model

:param X: Feature dataset

:param y: label dataset

:return: a linear regression model

"""

linear\_mod = linear\_model.LinearRegression() # defining the linear regression model

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

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

linear\_mod.fit(X, y) # fitting the data points in the model

return linear\_mod

#### Step 4: Get prediction on test set

predictions = LinearRegressionModel.predict\_prices(model,X\_test, label\_range)

Helper function within LinearRegressionModel.py makes the prediction for the model

def predict\_prices(model, x, label\_range):

"""

Predict the label for given test sets

:param model: a linear regression model

:param x: testing features

:param label\_range: normalised range of label data

:return: predicted labels for given features

"""

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

predicted\_price = model.predict(x)

predictions\_rescaled, re\_range = cd.scale\_range(predicted\_price, input\_range=[-1.0, 1.0], target\_range=label\_range)

return predictions\_rescaled.flatten()

#### Step 5: Plot the Actual values versus predicted data

Refer to Figure 2

#### Step 6: Measure accuracy of the prediction and calculate the RMS and MSE scores

trainScore = mean\_squared\_error(X\_train, y\_train)

print('Train Score: %.4f MSE (%.4f RMSE)' % (trainScore, math.sqrt(trainScore)))

testScore = mean\_squared\_error(predictions, y\_test)

print('Test Score: %.8f MSE (%.8f RMSE)' % (testScore, math.sqrt(testScore)))

Train Score: 0.4152 MSE (0.6443 RMSE)

Test Score: 0.09697600 MSE (0.31140969 RMSE)

### Improved LSTM Model:

#### Step 1: Build a basic LSTM Model

* Import Keras libraries for smooth implementation of LSTM
* Split train and test data sets
* Build a basic Long-Short Term Memory model
* Train the model
* Make predictions using test data
* Plot the results

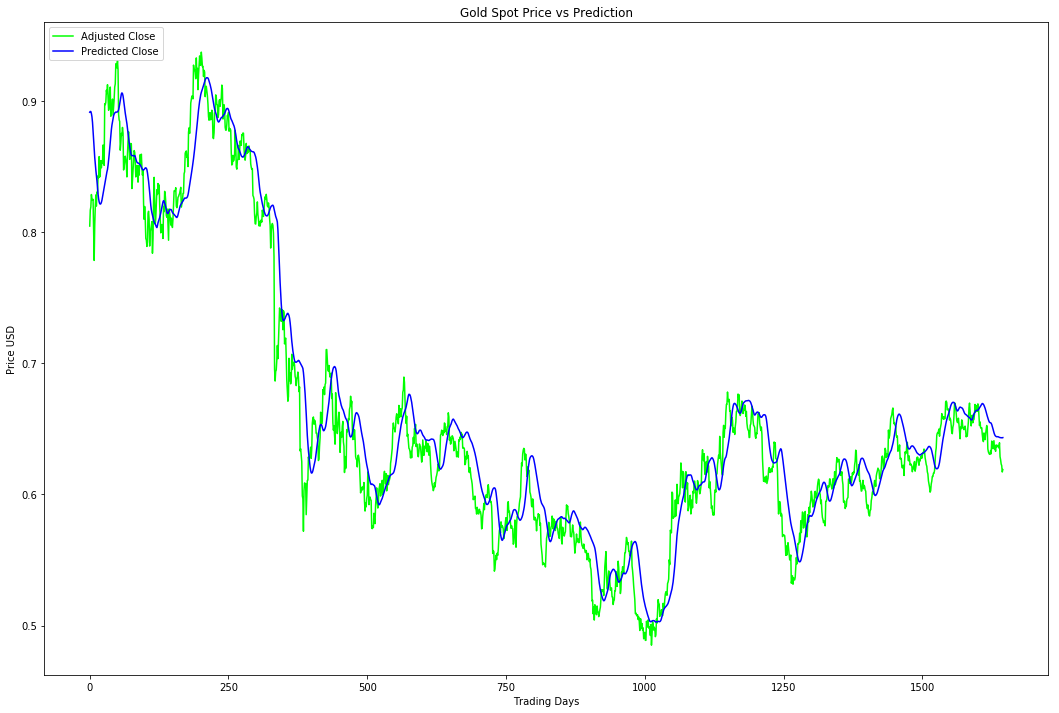


Figure 3. Adjusted Close and Predicted Close Prices for basic LSTM model

#### Step 2: Split into train and test model

The same set of training and testing data is used for improved LSTM as is used with basic LSTM.

#### Step 3: Build an improved LSTM modem

Helper function defined in ​‘lstm.py​’ which builds the improved LSTM model for the project.

# Set up hyperparameters

batch\_size = 512

epochs = 20

# build improved lstm model

model = lstm.build\_improved\_model( X\_train.shape[-1],output\_dim = unroll\_length, return\_sequences=True)

The function uses ​Keras Long Short Term Memory ​ library to implement LSTM model.

* Batch\_size increased to 512.
* Epochs increased from 1 to 20.
* Increased number of nodes in hidden layer to 128 (from 100).
* Added a drop out of 0.2 to all the layers.

def build\_improved\_model(input\_dim, output\_dim, return\_sequences):

"""

Builds an improved Long Short term memory model using keras.layers.recurrent.lstm

:param input\_dim: input dimension of model

:param output\_dim: ouput dimension of model

:param return\_sequences: return sequence for the model

:return: a 3 layered LSTM model

"""

model = Sequential()

model.add(LSTM(

input\_shape=(None, input\_dim),

units=output\_dim,

return\_sequences=return\_sequences))

model.add(Dropout(0.2))

model.add(LSTM(

128,

return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(units=1))

model.add(Activation('linear'))

return model

#### Step 4: Train the model

model.fit(X\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

verbose=2,

validation\_split=0.05

)

Train on 6278 samples, validate on 331 samples

Epoch 1/20

- 13s - loss: 0.0054 - val\_loss: 0.0241

Epoch 2/20

- 13s - loss: 7.2900e-04 - val\_loss: 0.0038

Epoch 3/20

- 12s - loss: 4.0847e-04 - val\_loss: 0.0011

Epoch 4/20

- 11s - loss: 2.9010e-04 - val\_loss: 9.8058e-04

Epoch 5/20

- 12s - loss: 2.8004e-04 - val\_loss: 0.0013

Epoch 6/20

- 12s - loss: 2.7937e-04 - val\_loss: 0.0014

Epoch 7/20

- 10s - loss: 2.4825e-04 - val\_loss: 0.0011

Epoch 8/20

- 12s - loss: 2.6173e-04 - val\_loss: 0.0019

Epoch 9/20

- 12s - loss: 2.4718e-04 - val\_loss: 0.0014

Epoch 10/20

- 11s - loss: 2.4343e-04 - val\_loss: 0.0011

Epoch 11/20

- 10s - loss: 2.3119e-04 - val\_loss: 0.0010

Epoch 12/20

- 10s - loss: 2.3844e-04 - val\_loss: 0.0016

Epoch 13/20

- 12s - loss: 2.4268e-04 - val\_loss: 0.0013

Epoch 14/20

- 14s - loss: 2.3217e-04 - val\_loss: 0.0013

Epoch 15/20

- 12s - loss: 2.4888e-04 - val\_loss: 0.0013

Epoch 16/20

- 13s - loss: 2.2467e-04 - val\_loss: 0.0010

Epoch 17/20

- 12s - loss: 2.2251e-04 - val\_loss: 0.0019

Epoch 18/20

- 9s - loss: 2.2776e-04 - val\_loss: 0.0012

Epoch 19/20

- 10s - loss: 2.3105e-04 - val\_loss: 0.0013

Epoch 20/20

- 9s - loss: 2.0634e-04 - val\_loss: 0.0011

Out[36]: <keras.callbacks.History at 0x1a221f0550>

#### Step 5: Predict the price for the given dataset

# Generate predictions

predictions = model.predict(X\_test, batch\_size=batch\_size)

#### Step 6: Calculate the test score and plot the results of the improved LSTM model

trainScore = model.evaluate(X\_train, y\_train, verbose=0)

print('Train Score: %.8f MSE (%.8f RMSE)' % (trainScore, math.sqrt(trainScore)))

testScore = model.evaluate(X\_test, y\_test, verbose=0)

print('Test Score: %.8f MSE (%.8f RMSE)' % (testScore, math.sqrt(testScore)))

Train Score: 0.00016953 MSE (0.01302042 RMSE)

Test Score: 0.00061908 MSE (0.02488127 RMSE)

## Refinement

For further refinement, provided more time and effort one the aspects of this project where I would fine tune would the LSTM parameters to get better predictions. Thus far I have shown improvement by testing and analyzing each parameter and then selecting the final value for each of them.

The following steps were taken to improve LSTM:

* Increased the number of hidden node from 100 to 128.
* Added Dropout of 0.2 at each layer of LSTM
* Increased batch size from 1 to 512
* Increased epochs from 1 to 20
* Added verbose = 2
* Made prediction with the batch size

The above improved mean squared error, for testing sets, from ​0.00066245 MSE ​ to 0.00061908 MSE. There is a 6.54% improvement.

The predicted plot difference between the basic and improved LSTM are illustrated in the figures below.

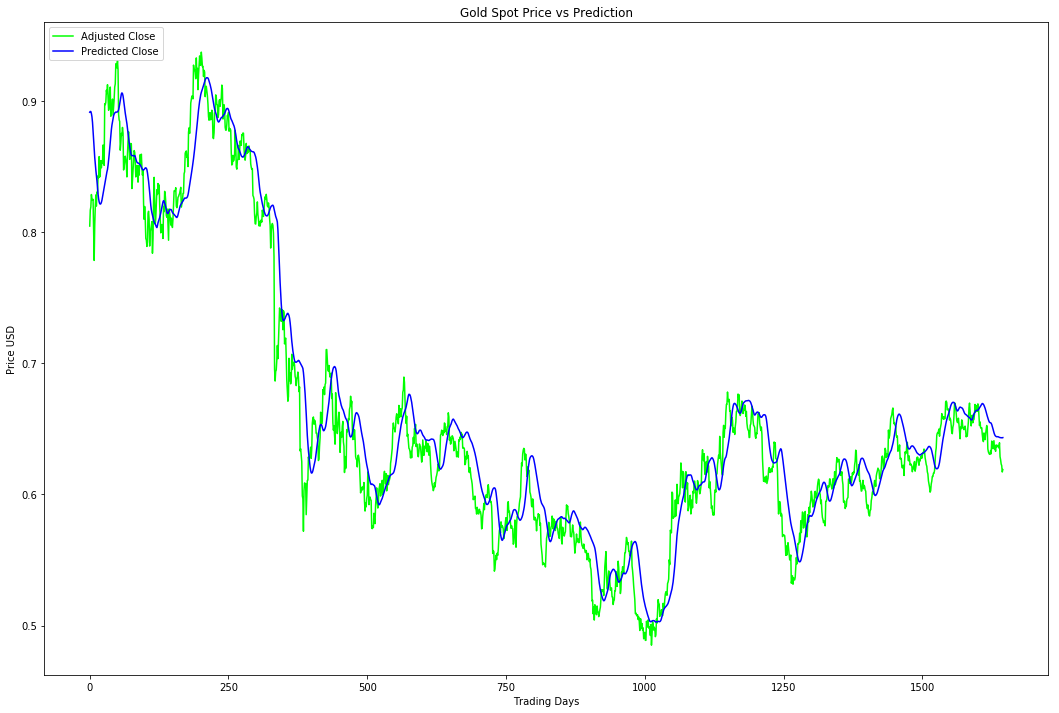


Figure 4. Spot Close and Predicted Close Prices for basic LSTM model

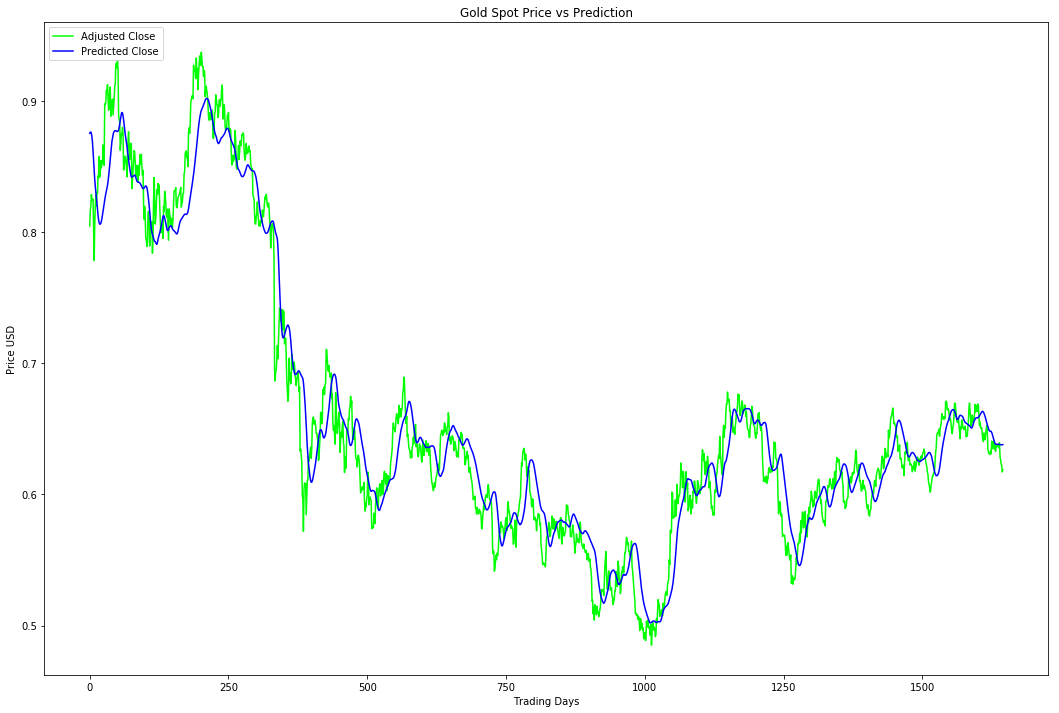


Figure 5. Spot Close and Predicted Close Prices for improved LSTM model

# Result

## Model Evaluation and Validation

There are three models used for prediction in the project. Each of the models were refined and tuned in order to get better predictions and have a reduced mean squared error.

In this section we will review and evaluate the training and test Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) score using the three models

### Linear Regression Model

Train Score: 0.4152 MSE (0.6443 RMSE)

Test Score: 0.09697600 MSE (0.31140969 RMSE)

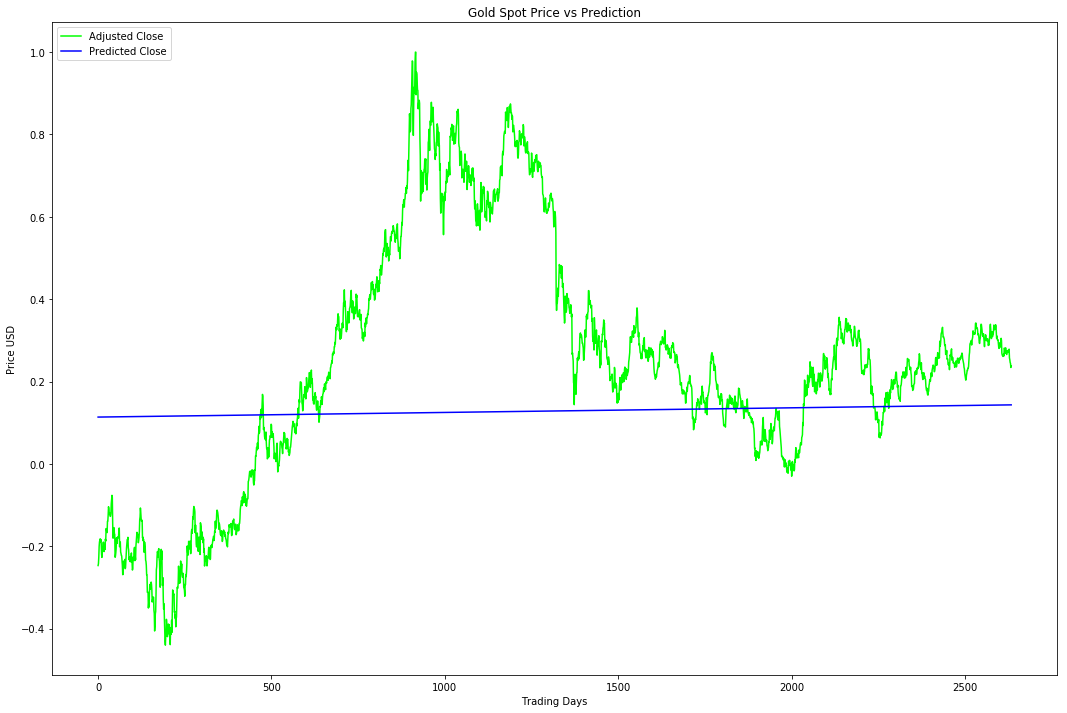


Figure 6. Linear Regression Model evaluation

### Basic Long-Short Term memory Model

Train Score: 0.00017208 MSE (0.01311779 RMSE)

Test Score: 0.00066245 MSE (0.02573813 RMSE)

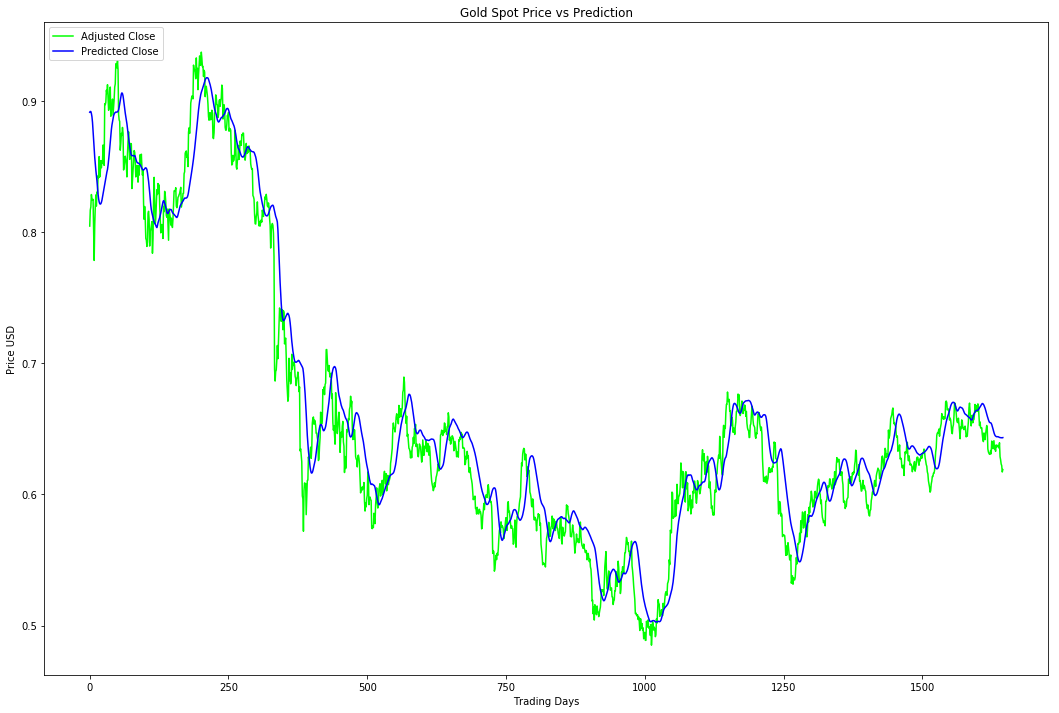


Figure 7. Basic LSTM Model evaluation

### Improved Long-Short Term Memory Model

Train Score: 0.00016953 MSE (0.01302042 RMSE)

Test Score: 0.00061908 MSE (0.02488127 RMSE)

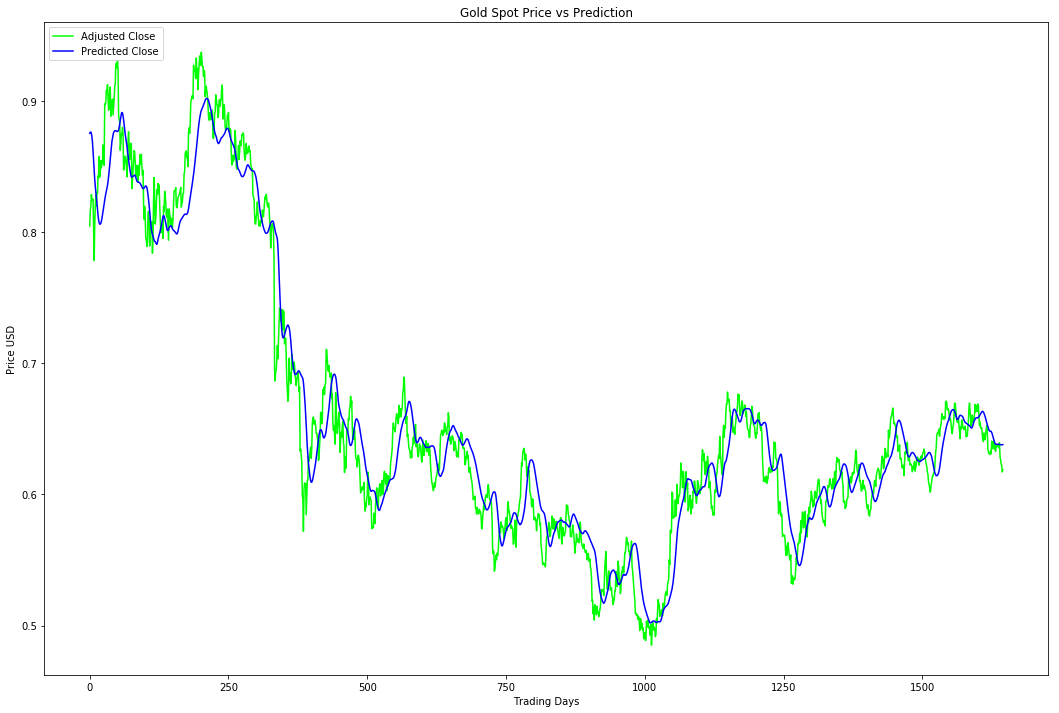


Figure 8. Improved LSTM Model evaluation

### Robustness Check

In order to validate the robustness of the improved LSTM model it will be use new data for the Gold commodity sport pricing. The dataset used here is going to be from June 23, 2018 to July 13, 2018. We can then evaluate the predictions based on the new dataset. The results are as follows:

Test Score: 0.1741 MSE (0.4172 RMSE)

Considering this more recent dataset is newer and have not been trained on. The MSE is still relatively low.

## Justification

Comparing the benchmark model - Linear Regression to the final improved LSTM model, the Mean Squared Error improvement ranges ​ from ​0.09697600 MSE (0.31140969 RMSE) [Linear Regression Model] ​ ​to​ 0.00061908 MSE (0.02488127 RMSE)​ [Improved LSTM]​. This significant decrease in error rate indicates that the refinement and tweaks in the final model have surpassed the basic and benchmark model.

Let us now consider the Average Delta Price between actual and predicted Closing Price values:

Delta Close Price: 0.000619 - RMSE \* Adjusted Close Range

The delta closing pricing a significant low, which indicates the degree of prediction precision.

# Conclusion

## Visualization

One of the most satisfying aspects of working with Python for Data Analysis and Machine Learning is its ability to visualize the data quickly. It provides an ascetic illustration to understand the actual and predicted data. Allow me to give the example of the improved LSTM plot. From the plot we can observe how the prediction data closely tracks the actual data, without actually overfitting the data and with the mean square error approximately 0.0006.

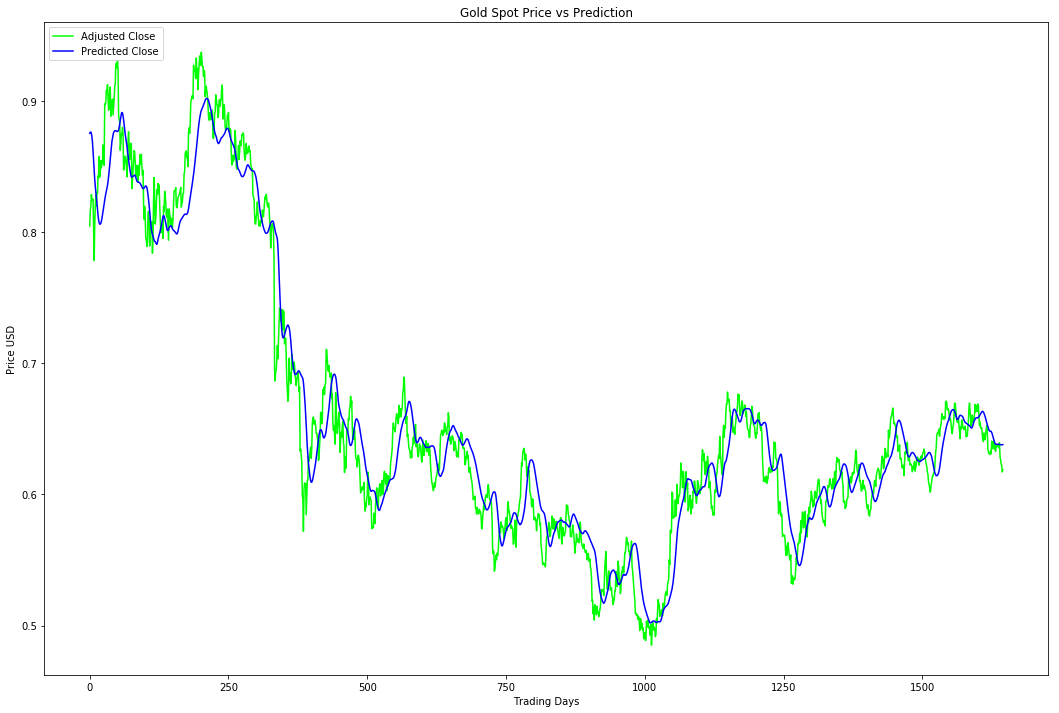


Figure 9. Closely tracked prediction data to actual in the improved LSTM Model

## Reflection

In summary the following are the steps and the process to achieve the desired results.

* Set Up Infrastructure
  + iPython Notebook
  + Incorporate required Libraries (Keras, Tensor flow, Pandas, Matplotlib, Sklearn, Numpy)
  + Git project organization
* Prepare Dataset
  + Incorporate data from several sources
  + Preprocess the data such that there are no missing entries or unusual entries
  + Develop helper functions to make the project design modular
  + Process the requested data into Pandas Dataframe
  + Develop function for normalizing data
  + Dataset used with a 80/20 split on training and test data across all models
* Develop Benchmark Model
  + Set up basic Linear Regression model with Scikit-Learn
  + Calibrate parameters
* Develop Basic LSTM Model
  + Set up basic LSTM model with Keras utilizing parameters from Benchmark Model
* Improve LSTM Model
  + Develop, document, and compare results using additional labels for the LSMT model.
  + Document and Visualize Results
* Plot Actual, Benchmark Predicted Values, and LSTM Predicted Values per time series
* Analyze and describe results for report.
* Investigate the Feature correlation and importance
  + Attempt to find correlation between other commodity prices or factors that could impact the pricing.

The objective for the project was to learn, implement, and successfully apply a new algorithm, in particular, Long-Short Term Memory model, to a real time series data set. The converged improved LSTM model met my expectations for the project and enhanced my knowledge and learning.

The most daunting challenge, which is typical in any Machine Learning project, through this project was to find the most meaningful dataset which enables me to utilize the Machine Learning concepts. Preprocessing the dataset required a fair amount of effort and a great deal of patience.

### Feature Engineering

Originally, I had planned on incorporating feature importance and factors influencing Commodity prices: LIBOR rate, Iron pricing, Coal pricing, Petroleum prices, Construction, Interest rates, Currency fluctuations in to the project. I did attempt to do so as you can see in the Gold\_Feature\_Importance.ipynb notebook. However, in the given time constraints and the complexity tackling continuous pricing variables to find the appropriate correlations, I decided to reduce the scope of the project as described in the proposal and focus on the Gold Spot Price Predictions.

## Improvement

The journey through the Connect program with Udacity has enlighten me in the world of Machine Learning, Deep Learning, and data analysis. I was able to improve my coding skills and most importantly make some meaning of real-life practical data.

Of course, the concepts implemented in this project can be extended towards any commodity. The most prominent improvement would be exploring the feature importance aspect of the project and converge on a solution which can provide some guidance to the consumers of the commodity markets.