# 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. Implement basic model using linear regression
3. Implement LSTM using KERAS library
4. Compare the results and submit the report
5. Apply the concepts and techniques learned in the Udacity Machine Learning NanoDegree

## 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:

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

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. Feature importance and relevant of other factors impacting the gold spot pricing will be outside the scope of the implementation.

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

## Exploratory Visualization

To visualize the data I used matplotlib library plotting the close prices against the number of trading available days as illustrated below.

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 method to tackle this problem through 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 1 for LSTM model and 512 for the improved LSTM model.
  + Optimizer Function: used “Adam” throughout
  + Epochs: 1 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 this I used error rate comparison MSE and RMSE utilizing the same dataset as the deep learning models.

Following is the predicted results that i got from my benchmark model:

Train Score: 0.1852 MSE (0.4303 RMSE)

Test Score: 0.08133781 MSE (0.28519784 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.
* 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.
* Normalized the data using ​MinMaxScaler ​ helper function from Scikit-Learn.
* Stored the normalized data in ​ gold\_spot\_preprocessed\_normalized.csv ​ file for future reusability.
* Split the dataset into the training (68.53%) and test (31.47%) 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 (82.95%) and test (17.05%) datasets for LSTM model. The Split was of following shape:

x\_train (7809, 50, 2)

y\_train (7809,)

x\_test (446, 50, 2)

y\_test (446,)

## Implementation

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

I have thoroughly specified all the steps to build, train and test model and its predictions in the notebook itself.

Some code implementation insight:

### Benchmark model:

Step 1 : ​Split into train and test model :

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

## Refinement

Fine tuning parameters of LSTM to get better predictions. I did the 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 ​0.01153170 MSE ​ to 0.00093063 MSE.

The predicted plot difference can be seen as follows:

# Result

## Model Evaluation and Validation

## Justification

# Conclusion

## Visualization

## Reflection

## Improvement