# AWS Machine Learning Engineer Nanodegree

## Capstone Project - Global X Copper Miners ETF Price Prediction

**Roger Jaccaud**

January 14, 2022

## 1. Project Definition

### Project Overview

Since 2021, inflation in the United States of America has increased dramatically. Indeed, from 1.2% in 2020, the US inflation rate rose to 6.8% at the end of November 2021, its highest since 1982.


In these uncertain times, for those of us who have a stake in the stock market, how do we protect our investments from the negative impacts of inflation? We do by investing in assets that performs well during inflationary periods.

Historically, copper is one of the best performing assets during inflationary periods. As a result, predicting changes in the value of copper has obvious financial benefit including knowing when to buy or sell.

Using historical stock prices for the Global X Copper Miners Exchange Traded Fund (ETF), that we obtained from Yahoo! Finance, we created three machine learning models tasked to predict the next day's closing price for copper.

The three models that I used were a Linear Regression model, a Random Forest model and a Long Short-Term Memory model. By comparing the capabilities of our models to predict the ETF next day's closing price, we were able to determine that our first model (the Linear Regression) would be the model that we would deployed.

### Problem Statement

The problem, that we attempted to solve with this project, is to predict the next day's closing price of copper based on historical data. In order to achieve our goal, we took the following steps:

- downloaded the historical prices of the Global X Copper Miners ETF (ticker: COPX) and stored them into a data set;
- added some technical indicators (used by traders for technical analysis of a stock) to the data set;
- implemented and trained the following machine learning models:
    1. a Linear Regression model;
    2. a Random Forest model;
    3. a Long Short-Term Memory model;
- tested the aptitude of our models at predicting the next day's closing price of my chosen ETF;
- compared the predictions' results to actual closing prices;
- based on our model performance comparison, chose and deployed the best performing model.

## Evaluation Metrics

In order to evaluate the accuracy of our models, we are using to two evaluation metrics. The first metric used is the **Root Mean Square Error** (RMSE). We chose this metrics because it is one of the most commonly used measures for evaluating the quality of predictions. It shows how far predictions fall from measured true values using Euclidean distance.

The **RMSE** equation is:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}}$$

The second metric used is the **Coefficient of Determination** ($R^2$). The Coefficient of Determination tells us how well the data fits the model. In the context of regression it is a statistical measure of how well the regression line approximates the actual data. It can take any values from 0 to 1, the closer to 1 the better the model fit.

The **$R^2$** equation is:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

# 2. Analysis

## Data Exploration

For this project, we have used the historical prices of the Global X Copper Miners ETF starting from the fund inception date (04/19/2010) to the end of 2021. Utilizing the `get_data_yahoo()` function of the **pandas_datareader** Python library, the data was retrieved from **Yahoo! Finance**.

The downloaded COPX data set is comprised of the following six data fields:

- Date - the entry date in a 'yyyy-mm-dd' format;
- High - the daily high price in USD;

- Low - the daily low price in USD;
- Open - the opening price in USD;
- Close - the closing price in USD;
- Volume - the volume of trades executed during the day;
- Adj Close - the Close price adjusted for splits, dividend and/or capital gain distributions.

It should be noted that our 'Date' field will not be used as a column within our date set but instead it will be its index.

To these seven original data field, we added a couple technical indicators (technical indicators are mathematical tools used to forecast a stock future price movements). They are:

- EMA3 - the 3-day Exponential Moving Average;
- EMA9 - the 9-day Exponential Moving Average.

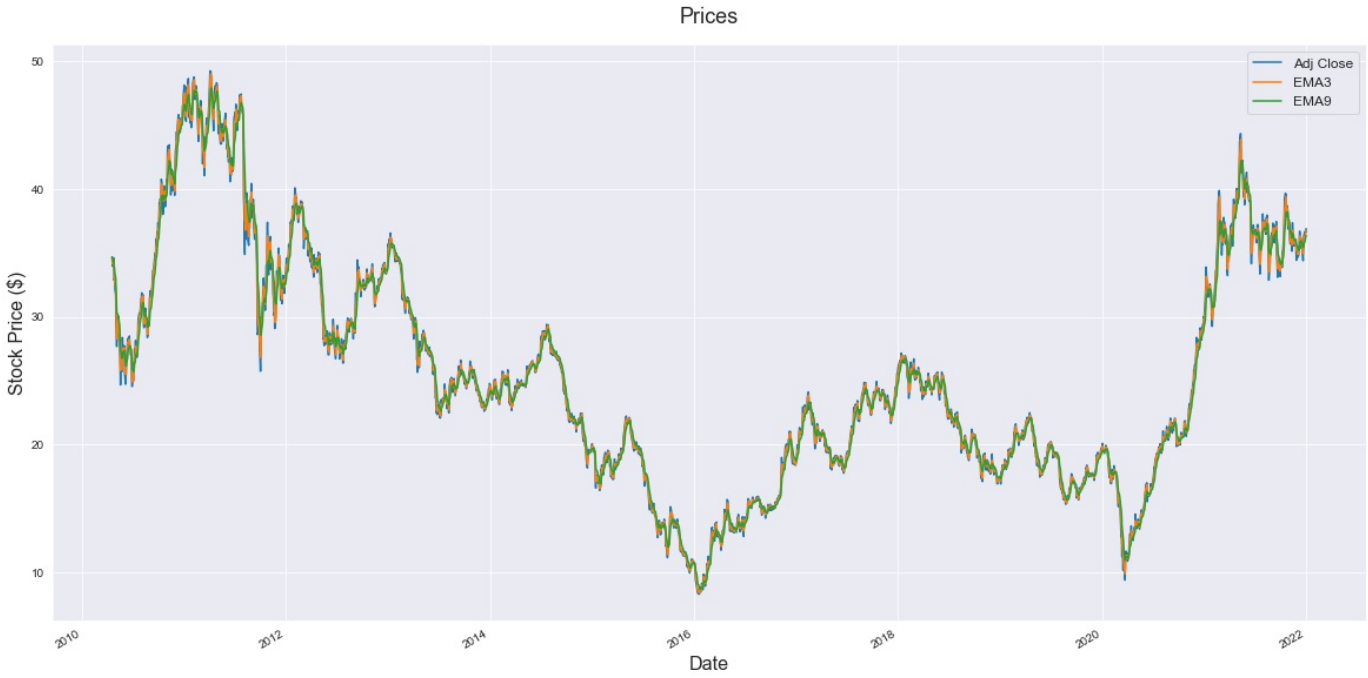
On a price chart, a moving average (MA) creates a single, flat line that effectively eliminates any variations due to random price fluctuations. An exponential moving average (EMA) is a type of moving average that places a greater weight and significance on the most recent data points.

Since the purpose of this project is to attempt to predict the next day price of a stock, we will drop most of the fields from our data set and keep only three fields:

- Adj Close;
- EMA3;
- EMA9.

## Data Visualization

In terms of data visualization, we created a chart plotting the closing prices as well as the 3-day Exponential Moving Average and the 9-day Exponential Moving Average from April 2010 to the end of 2021.




To better represent the smoothing out effect of any variations due to random price fluctuations obtained by the

use of Exponential Moving Averages, we created an additional chart plotting the closing prices as well as the 3-day Exponential Moving Average and the 9-day Exponential Moving Average for 2021 only.


## Benchmark

The benchmark that we are use for this project is based on the naive method of forecasting. This method dictates that one use a previous period to forecast for the next period. In our case, we will be using the **previous adjusted closing price** of the stock to predict its **next day closing price**.

# 3. Methodology

## Data Preprocessing

Because the source of our data is Yahoo! Finance (a media property that provides financial news, data and commentary including stock quotes, press releases, financial reports, and original content), our data is clean and requires very little preprocessing.

```
# Display a concise summary of the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2948 entries, 2010-04-20 to 2021-12-31
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   High       2948 non-null   float64
 1   Low        2948 non-null   float64
 2   Open       2948 non-null   float64
 3   Close      2948 non-null   float64
 4   Volume     2948 non-null   float64
 5   Adj Close  2948 non-null   float64
dtypes: float64(6)
memory usage: 161.2 KB
```

Having clean data, we do not have to deal with missing values. Actually, the preprocessing that we will be

performing will be the addition of two news fields:

- EMA3 - the 3-day Exponential Moving Average;
- EMA9 - the 9-day Exponential Moving Average.

The Exponential Moving Average (EMA) is a specific type of moving average that points towards the importance of the most recent data and information from the market.

$$EMA_{Today} = (Value_{Today} \times (\frac{Smoothing}{1 + Days})) + EMA_{Yesterday} \times (1 - (\frac{Smoothing}{1 + Days}))$$

EMA is a type of technical indicator that is used to get buy and sell indicators based on historical averages. Traders usually use 50 days and 200-day moving averages. However, since we are attempting to predict the next day closing price, we are using averages on a smaller number of days.

Finally, we added a new column to our data set. This new column (that we named 'Next Day Price') will be used as our label during our supervised training sessions. The value of this new column is tomorrow's closing price assigned to today's record.

Our desire to predict the next day closing price led us to shed most of our data set columns and only keep the ones that would help us achieve our goal. Upon completion of this restructuring step, we are left with four columns of data:

- Adj Close;
- EMA3;
- EMA9;
- Next Day Price.

```
            Adj Close       EMA3       EMA9  Next Day Price
Date
2010-04-20  34.699127  34.699127  34.699127       34.155464
2010-04-21  34.155464  34.427296  34.590395       33.966381
2010-04-22  33.966381  34.196838  34.465592       34.084560
2010-04-23  34.084560  34.140699  34.389386       34.604576
2010-04-26  34.604576  34.372638  34.432424       32.879074
...               ...        ...        ...             ...
2021-12-23  36.398113  36.010707  35.724753       36.685104
2021-12-27  36.685104  36.347906  35.916823       36.536659
2021-12-28  36.536659  36.442282  36.040791       36.695004
2021-12-29  36.695004  36.568643  36.171633       36.650002
2021-12-30  36.650002  36.609322  36.267307       36.910000

[2947 rows x 4 columns]
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2947 entries, 2010-04-20 to 2021-12-30
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Adj Close       2947 non-null   float64
 1   EMA3            2947 non-null   float64
 2   EMA9            2947 non-null   float64
 3   Next Day Price  2947 non-null   float64
dtypes: float64(4)
memory usage: 115.1 KB
None
```

At this point, we separated our **independent variables** (also referred to as **features**) from the **dependent variable** into two distinct DataFrames. Our independent variables ('**EMA3**', '**EMA9**') will be the input to our machine learning models. Whereas our dependent variable ('**Next Day Price**') will be the values to which we will compare the output of our models.

Finally, we split our two DataFrames into three distinct sets of two DataFrames: a set of two for training purposes, a set of two for testing purposes and a set of two for validation purposes. Our training data set is made up of the first 76% of the historical trading records. The testing data set holds 19% of the records. Meanwhile, the validation data set contains the the remaining 5% of records. Because we are dealing historical data, we did not shuffle the data while splitting it. Indeed, shuffling the data while splitting it could have potentially introduced information about the future in the training data relative to the testing data, giving it an unfair advantage.


## Implementation

Upon completion of the data preprocessing, we proceeded by creating three distinct machine learning models. To create our models, we used ***Scikit-learn*** (a free machine learning library for Python).

We created:

- a **Linear Regression** model (a linear approach for modeling the relationship between a dependent variable and one or more independent variables);
- a **Random Forest** model (an ensemble learning method for regression that operates by constructing a multitude of decision trees at training time);
- a **Long Short-Term Memory** model (a type of recurrent neural network capable of learning order dependence in sequence prediction problems).

Each model was trained on the same set of data (the 'training' data set) and their performance evaluated on the same set of data (the 'testing' data set). Two metrics were used to judge their accuracy at predicting the next day closing price of our EFT:

- the **Root Mean Square Error** (RMSE) which shows how far predictions fall from measured true values using Euclidean distance;

- the **Coefficient of Determination** ($R^2$) which tells us how well the data fits the model.

Having compared the accuracy of our three models, it was determined that our **Linear Regression** model, even though not perfect, was our best performing model and the one that we should deploy.

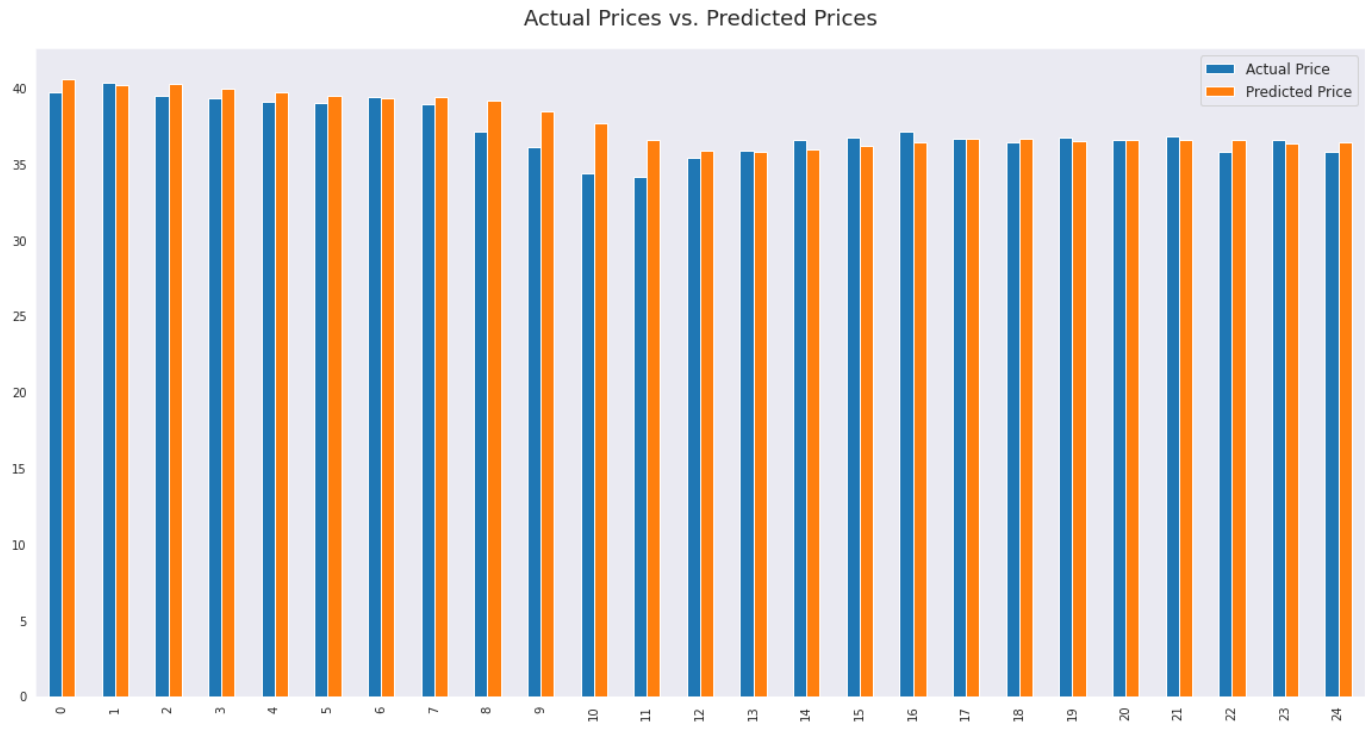
We, then, proceeded to created a deployable version of our best performing model using AWS SageMaker. Using a **SageMaker Estimator** (a training script), we are able to create our Linear Regression model. To the script, we provide hyperparameters and input data as its command line arguments.

```
# Set up the hyperparameters
linear_model.set_hyperparameters(feature_dim=X_train.shape[1],
                                 epochs=20,
                                 num_models=32,
                                 loss='absolute_loss',
                                 predictor_type='regressor',
                                 mini_batch_size=32,
                                 normalize_data=True,
                                 normalize_label=False)
```

## Hyperparameters explained

- the ***feature_dim***, the number of columns (2 in our case) of the feature DataFrame (X_train);
- the ***predictor_type***, the type of target variable (**regression** in our case)
- the ***loss***, the loss function which depend on the value of predictor_type (**absolute_loss** in our case)
- the ***normalize_data***, if true, it shifts the data for each feature to have a mean of zero and scales it to have unit standard deviation.

The script is then executed via its **fit()** method. Upon completion of its training session, SageMaker will automatically select and save the best model for us. We may then proceed to deploy it and test it on our third set data: our 'validation' data set.




As we may see on the graph below, our model is fairly accurate in its stock price predictions. Its results could be better but they are sufficient for our purpose.

## Refinement

Prior to creating and deploying our best performing model using AWS SageMaker, we attempted to fine tune both our **Random Forest** and **Long Short-Term Memory** models. Unfortunately, our fine tuning was not successful at improving our models accuracy. As a result, we abandoned this approach and deployed our Linear Regression model.

# 4. Results

## Model Evaluation and Validation

The table below shows how each of our original three models performed from the point of view of our two principal metrics: the **Root Mean Square Error** and the **Coefficient of Determination**.

| | Model | Mean Absolute Percentage Error | Root Mean Square Error | Coefficient of Determination |
|---|---|---|---|---|
| 0 | Linear Regression - Test | 1.884101 | 0.598425 | 0.994158 |
| 1 | Linear Regression - Val | 1.926614 | 0.874210 | 0.674198 |
| 2 | Random Forest - Test | 2.335735 | 0.693306 | 0.992158 |
| 3 | Random Forest - Val | 2.515219 | 1.095089 | 0.488764 |
| 4 | LSTM - Test | 2.463661 | 0.730451 | 0.991295 |
| 5 | LSTM - Val | 2.416549 | 1.046492 | 0.533132 |

As we know, in the context of regression, the **Coefficient of Determination** is a statistical measure of how well the regression line approximates the actual data. It can take any values from 0 to 1, the closer to 1 the better the model fit. In our case, our Linear Regression model had the best Coefficient of Determination for both its testing and validation sessions.

One way to assess how well a regression model fits a dataset is to calculate the **Root Mean Square Error**, which is a metric that tells us the average distance between the predicted values from the model and the actual values in the dataset. The lower the RMSE, the better a given model is able to 'fit' a dataset. Once again, our Linear Regression model had the best Root Mean Square Error for both its testing and validation sessions.

## Justification

Both metrics show that our model could benefit form further refinements. Nevertheless, their accuracy, especially the one from our Linear Regression model, is sufficient for our purpose.

# 5. Conclusion

## Reflection

Attempting to predict the next day closing price of our ETF of choice was both challenging and rewarding.

The challenges encountered were diverse and present all along this project. The first challenge was to choose a project. Fortunately, the current level of inflation in the country was very helpful in our decision. The second challenge was to acquire, clean and preprocess the historical stock prices for our EFT of choice. Luckily, Yahoo! Finance allows us to access without to much hassle the desired data. The third challenge was to create, train and evaluate three different machine learning models: a Linear Regression model; a Random Forest model; a Long

Short-Term Memory. Finally, the fourth challenge was to implement and deploy our winning model (the Linear Regression one) using AWS SageMaker.

Despite these challenges, driving the project to its conclusion was enriching and a great training experience in the domain of machine learning.

## Improvement

Obviously no project is perfect at first. The same goes for this project. Having more time and resources, we might have been able to achieve better accuracy. For example, if we keep in mind that the stock market is driven by more than historical stock prices, we might have been able to add a sentiment analysis based on external elements such as news about the economy, the perceived mood of the population to our project.

In addition, it might have been judicious to spend more time trying to fine tune our models.