Predictive Modelling and Oil Prices with Machine Learning And Python

By observing the markets you learn everything about people, and most importantly this knowledge is provided in such a way where it is perfect for data scientists to put their hands on it. The main problem that tends to appear with markets, is that they are often unpredictable.
The mathematical models might all show that the value of a certain commodity will go up and then something unpredictable happens (e.x. COVID-19) and everything changes.
Thus, it becomes obvious that the markets are extremely prone to external influence and factors. Nevertheless, I am going to attempt, by using **linear *regression*,** to predict the price of Brent oil

Key Terms

Regression
In simple terms, *regression* is a Machine Learning algorithm that can be trained to predict certain real-numbered results. The prediction is based on the interaction of the different variables found in the dataset.

Features
A *feature* is a measurable property of an object we are analyzing. In a dataset, *features* appear as columns and are the different characteristics of an object (e.x. price, location, id)

Explanatory Variables
The *explanatory variables* are the different *features* we are going to be using to predict the price of Oil at any given time.

Dependent Variable
The dependent variable depends on the values of the *explanatory variables*. In simple words, the *dependant variable* in our case scenario is the price of oil we are trying to predict.

NaN values
In computing, *NaN* is an unknown value

Preparing our dataset and work environment

Dataset

Having the right dataset is undoubtedly one of the most important aspects of any data science project. In this case scenario, the dataset we need is an archive of oil prices for the last years. The way you acquire said database depends entirely on your preference. **Yahoo Finance and iexfinance** are both great ways to retrieve such financial information (you can manually download the data as well).

In this instance, I will be using **Quandl** as I find it both easy to use and with many interesting functionalities.

*In order to use Quandl, an **API key is required**. Do not worry though! The process of obtaining an API key takes **less than two minutes** and **no form of verification is required.**

Instructions on how to obtain a Quandl API key can be found **here**.

Coding

Now that we have both our libraries and dataset set-up, it is time to launch our text editors (I will be using a jupyter notebook/Google Colab).

We will begin by importing all necessary libraries:

```
# A machine learning library used for linear regression
from sklearn.linear_model import LinearRegression
# numpy and pandas will be used for data manipulation
import numpy as np
import pandas as pd
# matplotlib will be used for visually representing our data
import matplotlib.pyplot as plt
# Quandl will be used for importing historical oil prices
!pip install Quandl
import quandl
```

Now that we have all libraries ready, we can begin by importing our historical data using Quandl. We are going to use **twenty years** of **Crude Oil Brent** prices, starting from January 1st, 2000 up to Maech 1st, 2022.

Quandl Code

FRED/DCOILBRENTEU

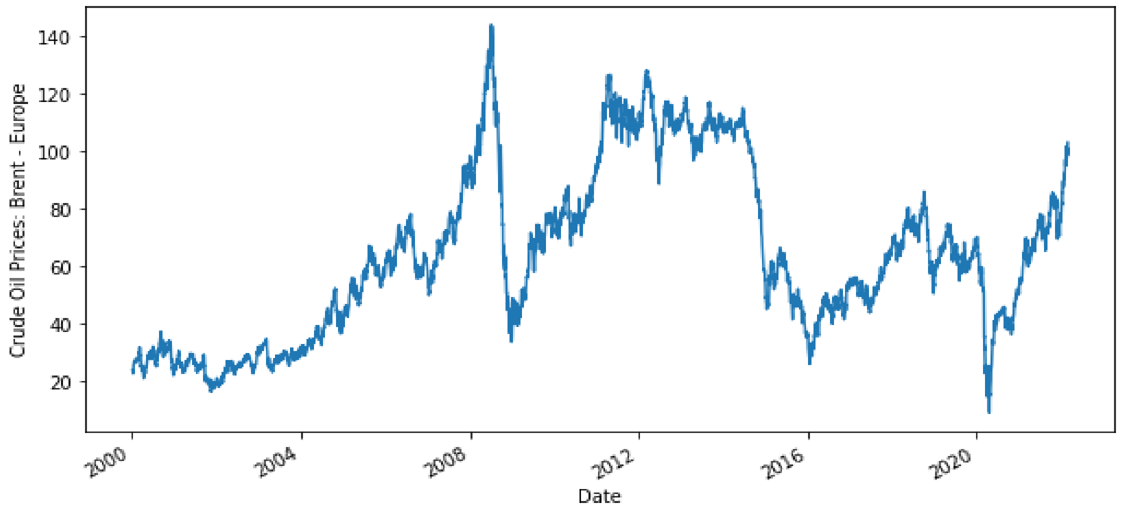
In our situation, we need something more than a ticker. Quandl classifies its data into different datasets. From the image above, we can see that the Quandl Code for “Crude Oil Prices: Brent-Europe” is **“FRED/DCOILBRENTEU”.**

```
# Setting our API key
quandl.ApiConfig.api_key = 'Your_Key'

# Importing our data
data = quandl.get("FRED/DCOILBRENTEU", start_date="2000-01-01", end_dat
e="2022-03-01")
```

We now have all Brent oil price information for the last 20 years under "data". It is now time to see how our data looks in order to determine what *features* we want to keep and which ones we want to discard.

It appears that we are quite lucky! The dataset only contains the date and the price, thus there is no sort of tampering needed on our part.
(If you are using another dataset, you must model the data and keep only what is necessary!)
We are now going to proceed by visually plotting our data in the form of a graph for better analysis.

```
# Setting the text on the Y-axis
plt.ylabel("Crude Oil Prices: Brent - Europe")

# Setting the size of our graph
data.Value.plot(figsize=(10,5))
```



Now that we have properly visualized our data, we are going to define our *explanatory variables* — the *features* we are going to use to predict the price of oil. The variables we will be using at this stage, are the **moving averages** for the past **three** and nine **days**.

```
data['MA3'] = data['Value'].shift(1).rolling(window=3).mean()
data['MA9']= data['Value'].shift(1).rolling(window=9).mean()
```

We must now drop all of the *NaN values* and store the two variables we have created into X and view its contents.

```
# Dropping the NaN values
data = data.dropna()

# Initialising X and assigning the two feature variables
X = data[['MA3','MA9']]

# Getting the head of the data
X.head()
```

By having set-up the *explanatory variables*, it is time to initialize our *dependant variable*, y.

```
Setting-up the dependent variable
```

```
y = data['Value']
# Getting the head of the data
y.head()
```

Training-Testing

Now that everything is set-up, we are going to split our dataset into two distinct parts. We are going to assign 80% of our data into a training set, responsible for training the model. The rest 20% will be assigned to a testing set, used to estimate the accuracy of the model. This way, by connecting the input from the training set with the expected result from the testing set, we create a *linear regression model*.

```
# Setting the training set to 80% of the data
training = 0.8
t = int(training*len(data))

# Training dataset
X_train = X[:t]
y_train = y[:t]

# Testing dataset
X_test = X[t:]
y_test = y[t:]
```

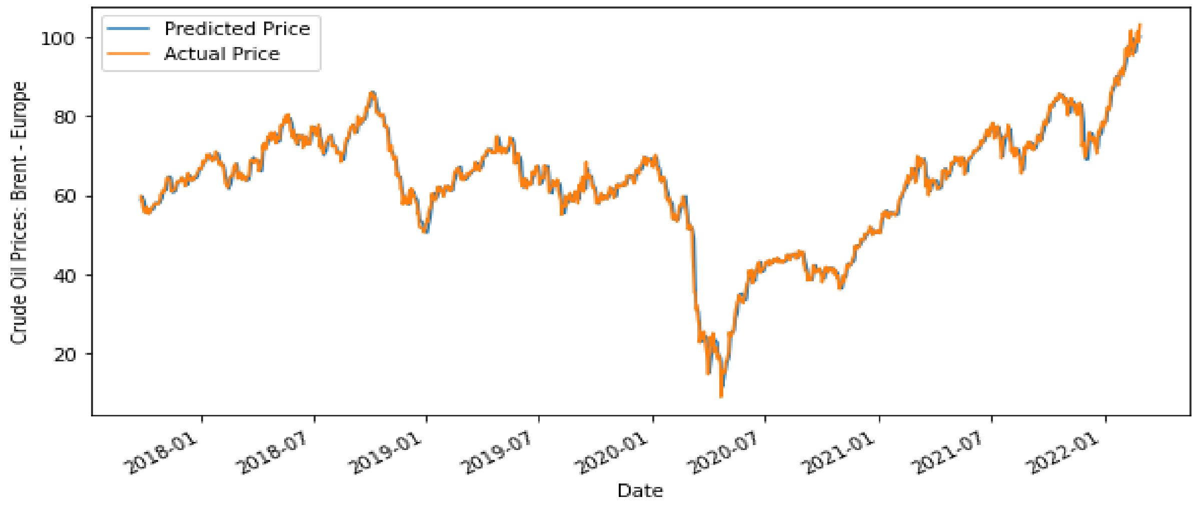
In order to create the *linear regression model*, we are going to fit the *dependant* and independent variables and create a constant and a coefficient for the *regression*.

```
model = LinearRegression().fit(X_train,y_train)
```

We are almost done! It is now time to check if our model works and plot the price our model predicted, as well as the actual price on the same graph for comparison.

```
predicted_price =
model.predict(X_test)
                    predicted_price =
                    pd.DataFrame(predicted_price,index=y_test.index,columns = ['price'])
                    predicted_price.plot(figsize=(10,5))
                    y_test.plot()
                    plt.legend(['Predicted Price','Actual Price'])
                    plt.ylabel("Crude Oil Prices: Brent - Europe")
                    plt.show()
```

Voilà! We have plotted a graph which shows the predicted and actual price of oil each time. Although the graph gives us a lot of information on whether we have succeeded in carrying out our task or not, we will compute the accuracy of our model



```
# Computing the accuracy of our model
R_squared_score =model.score(X[t:],y[t:])*100
accuracy = ("{0:.2f}".format(R_squared_score))
print ("The model has a " + accuracy + "% accuracy.")
```

The model has a 98.67% accuracy.

```
print(f'alpha = {model.intercept_}')
print(f'betas = {model.coef_}')
```

```
alpha = 0.12151703363885247
betas = [ 1.22804036 -0.22978307]
```

This is simply a linear regression model with more than one predictor, and is modelled by:
$\boldsymbol{Y_e = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta p X p}$, where p is the number of predictors.

$$Therefore\ Y = 0.122 + 1.23*MA3 - 0.23*MA9.$$

What is the meaning of a Moving Average (MA3, MA9)?

A moving average is a technical indicator that market analysts and investors may use to determine the direction of a trend. It sums up the data points of a financial security over a specific time period and divides the total by the number of data points to arrive at an average. It is called a “moving” average because it is continually recalculated based on the latest price data.

Analysts use the moving average to examine support and resistance by evaluating the movements of an asset’s price. A moving average reflects the previous price action/movement of a security. Analysts or investors then use the information to determine the potential direction of the asset price. It is known as a lagging indicator because it trails the price action of the underlying asset to produce a signal or show the direction of a given trend.