Multiple Linear Regression with Ethereum Stock Price Report

Introduction and Project Outline

The general purpose of this project is to build and evaluate a Multiple Linear Regression model to predict the price of the Ethereum cryptocurrency (synonymous with Ether or ETH). The regression model should show us what factors influence the price of Ethereum. This project will focus on the price of the Ethereum cryptocurrency on the stock exchange although the Ethereum platform with in which it is used will be briefly described below to get a context of the real-world applications of the cryptocurrency.

Ethereum can also be viewed as a blockchain platform where the cryptocurrency is used to host its own applications with a programming language called Solidity. As a blockchain network, Ethereum is a decentralised public leger for verifying and recording transactions using cryptography and a proof of work consensus protocol.

Ethereum was created to enable developers to build and plublish smart contracts, distributed applications (dApps) and NFTs (Non-fungible tokens) that can be used without the risk of downtime or interference from a third party.

The building of the Linear Regression model aspect of the project will be completed with Python and its various libraries, mainly Scikit-Learn for the functions required for implementing linear regression.

Software used

Python 3.8.8

Python is a high level, object-oriented programming language which is used for a variety of purposes. It has an elegant syntax that is easy to code in. Anaconda is an open-source distribution for the Python language which simplifies package management and deployment. It has a large number of open-source libraries built into the distribution making it easy to expand programs by importing new modules. Python also has many volunteers constantly improving the language and open-source libraries.

In this project, we will be using Python for implementing the Data Science and Machine Learning aspects of the project. There is a large amount of online support for Python in this specific application, and many of the standard Python libraries have built-in functions which will be used.

Scikit-Learn 1.0.1

Scikit-learn is a simple and effective tool for predictive data analysis built on top of the library SciPy. The library is cross-platform and was released under the open-source BSD License so it's free to use for both academic and commercial use. It was initially developed by a French data science David Courtnapeau, as a Google Sumnmer of Code Project.

Most of the Multiple Linear Regression algorithm used in this project will be implemented through the Scikit-Learn library.

Background Theory

Regression analysis is a way to statistically estimate the relationship between variables. They can have one or more independent (predictor) variables, and one dependent (outcome) variable. Regression models comprise of two types of regression: Simple Linear Regression and Multiple Linear Regression.

Simple linear regression is not commonly seen often as it is rare for an outcome to only be influenced by just one predictor variable. It is more commonly seen in real-world applications that outcomes are influenced by multiple different independent (predictor) variables which is then known as Multiple Linear Regression. With multiple linear regression, we have the ability to narrow down which variables affect the outcome, and to what extent with coefficient variables.

Multiple linear regression can hence be defined as a statistical technique that is used to predict the outcome of a variable based on the value of two or more variables. It is an extension of Simple linear Regression.

The formula for the Multiple Linear Regression is:

$$y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n + \varepsilon$$

where:

- y – is the predicted value of the dependent variable
- β_0 – is the y intercept (ie. The value of y when all parameters are set to 0)
- $\beta_1 X_1$ – is the regression coefficient (β_1) of the first independent variable (X_1)
- … – continue for all the independent variables are being tested
- $\beta_n X_n$ – is the regression coefficient of the last independent variable
- ε – is the model error (how much variation there is in the estimate of y)

Each independent variable has its own β coefficient which reflects the different influences it has on the outcome of the model.

Finding the values of these β constants is done by minimising the error function and fitting a hyperplane. This is done by minimising the Residual Sum of Squares (RSS) to train the model and fit the best hyperplane to the data. The Residual Sum of Squares (RSS) of the training data is calculated by squaring the difference between actual and predicted outcomes in the pursuit of determining the dispersion of the data points to check how well the data series can be fitted by a hyperplane. The equation for RSS can be denoted as the following:

$$RSS = \sum_{i}^{n} \left(Actual - Predicted \right)^2 = \sum_{i}^{n} \left(y_i - \hat{y}_i \right)^2$$

where:

- y_i is the obsersed values from the dataset
- $\hat{y}_i$ is the predicted values by our regression model
- n is the number of values in the dataset

Due to fact this method finds the least sum of squares, it is also known as the Ordinary Least Squares (OLS) method.

Evaluating the model

A Multiple Linear Regression model is assessed by metrics to provide a measure of how well the predicted outputs are being generated by the model.

Evaluating the model is essential in understanding its performance to both the developer and user. Its aim is to measure how well outputs predicted by the model compare to the actual values of the observed outputs from the dataset. In essence, the best model for the dataset is the model with the least error.

The main evaluation metrics useful for Linear Regression model are:

- Coefficient of Determination (R^2)/ Adjusted R^2
- Mean Squared Error (MSE)/ Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE)/ Mean Absolute Percentage Error (MAPE)

Coefficient of Determination (R^2)

The coefficient of determination (R^2) is a measure of how close the data is fitted to the regression line or hyperplane. It is typically a value between 0 and 1, but can be negative.

One of the things to keep note about the R^2 metric is that each time a independent variable is added to the model, the metrics value increases and moves closer to 1. This can cause the metric to become inaccuate. In order to counter this, the R^2 metric will have to be adjusted which would be described below.

The overall equation is:

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum \left(y_i - \hat{y}_i\right)^2}{\sum \left(y_i - \bar{y}_i\right)^2}$$

where:

- y_i is the actual output values
- $\hat{y}_i$ is the predicted values from our model
- $\bar{y}_i$ is the mean values in the dataset

Residual Sum of Squares (RSS) is defined as the sum of squares of the residual for each data point in the plot/data. It is the measure of the difference between the expected and the actual observed output.

Total Sum of Squares (TSS) is defined as the sum of errors of the data points from the mean of the response variable.

The closer the R2 score is to 1, the better the regressions predictions fit to the data.

Adjusted R^2

Adjusted R^2 is a modified version of R^2 that adds precision and reliability to the metric by accounting for the impact additional independent variable that can cause R^2 to become inaccurate. It can be denoted as following:

$$R^2_{adj} = 1 - \left[\frac{(1 - R^2)(n - 1)}{n - k - 1} \right]$$

where:

- R^2 is the coefficient of determination of the model
- n is the number of samples in the dataset
- k is the number of independent variables (predictors)

The Adjusted R^2 is always lower than the R squared of the model. The closer the metric is to 0 can also indicate good performance. Though it is important to be cautious of the contents in which each of these metrics are used as a high score might not necessarily mean that the model is accurate such as is seen in some Non-stationary Time Series datasets.

Mean Squared Error (MSE)* and *Root Mean Squared Error (RMSE)

Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) is the mean of the square of residuals and the squared root of the mean squared errors (MSE) respectively. The square root helps to reduce the impact of scale of the independent variables on the metric allows us to compare the results with the Mean Absolute Error (MAE) metric.

Root Mean Squared Error can also be described as the standard deviation of the residuals, which is the difference between the predicted values and the regression line (or hyperplane). It gives an estimate of the spread of observed datapoints across the predicted regression line.

The Root Mean Squared Error (RMSE) can be denoted as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Actual - Predicted)^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where:

- y_i is the actual output values
- $\hat{y}_i$ is the predicted values from our model
- n is the number of values in the dataset

The lower the RMSE the better the regression model indicates higher accuracy of the model.

Mean Absolute Error (MAE)* and *Mean Absolute Percentage Error (MAPE)

Mean Absolute Error (MAE) is the average of all the absolute errors which leads it to be reboust against outliers in our data. The absolute error is the difference between the true value and the predicted value. It can be denoted as following:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |Actual - Predicted| = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

where:

- y_i is the actual output values
- $\hat{y}_i$ is the predicted values from our model
- n is the number of values in the dataset

The metric will output the average difference between the predicted values from the regression model between the actual outputs from the dataset. The lower the MAE metric is to 0, the less the overall error for the model, hence better performance.

This metric is often converted to a percentage value ascribed as *Mean Absolute Percentage Error* (MAPE). Its useful when evaluating a model if the datasets value vary by a wide range.

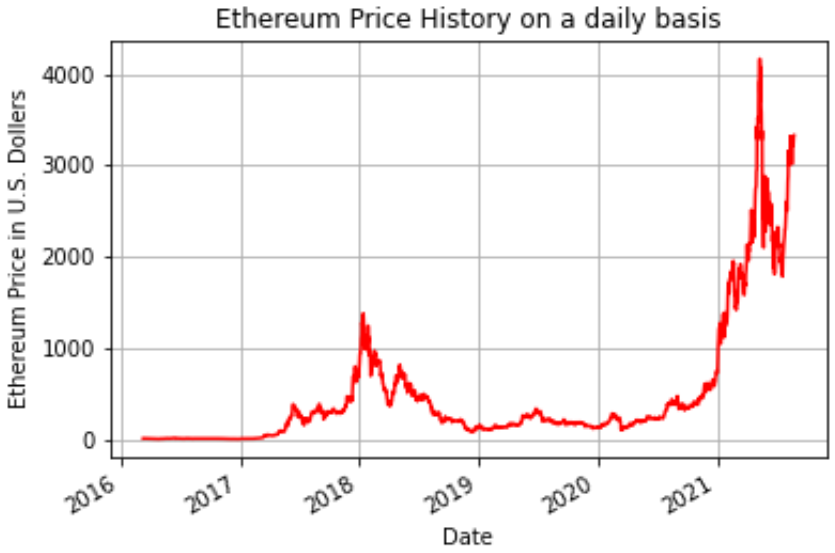
The formula is updated below:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n} \left| \frac{Actual - Predicted}{Actual} \right| \times 100 = \frac{1}{n}\sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100$$

Dataset

The dataset used in this project represents the historical data from Ethereum, a cryptocurrency that is used in the Ethereum blockchain platform.

Our dataset spans almost the entirety of Ethereums existence, starting from 10/03/2016 and ending in 24/08/2021 containing 1994 data points in total.



The dataset was obtained from ‘Kaggle’, which is a public data platform with a huge inventory of datasets. The datasets from Kaggle are generally clean which is useful for us as it allows us to focus on the problem at hand without having to edit to dataset, that is ‘Implementing a Multiple Linear Regression model to a suitable datasets’. This data from the dataset was collected from '*investing.com*', which is a solid stock investing website that provides real-time information on Ethereum.

SNo	Date	Price	Open	High	Low	Vol.	Change %
1	2016-03-10	11.75	11.20	11.85	11.07	0.0	4.91
2	2016-03-11	11.95	11.75	11.95	11.75	180.0	1.70
3	2016-03-12	12.92	11.95	13.45	11.95	830.0	8.12
4	2016-03-13	15.07	12.92	15.07	12.92	1300.0	16.64
5	2016-03-14	12.50	15.07	15.07	11.40	92180.0	-17.05
6	2016-03-15	13.06	12.63	13.42	11.98	39730.0	4.48
7	2016-03-16	12.88	13.06	13.89	12.62	19240.0	-1.38
8	2016-03-17	10.74	12.58	12.61	10.44	89450.0	-16.61
9	2016-03-18	10.74	11.09	11.20	8.34	157370.0	0.00
10	2016-03-19	10.31	10.97	11.10	9.78	67550.0	-4.00

Figure 1: First 10 rows from our dataset

The dataset contains the following variables:

- **'Date':** Date of the cypto prices in NY EST Standard Time
- **'Price':** Price of Ethereum (in U.S. Dollers)
- **'Open':** Opening Price of Ethereum on the respective date
- **'High':** Highest Price of Ethereum on the respective date
- **'Low':** Lowest Price of Ethereum on the respective date
- **'Vol.':** Volume of Ethereum transacted on the respective date
- **'Change':** Percentage of Change in Ethereum Prices on the respective date

The ‘Date’, ‘Vol.’ and ‘Change’ variables will be removed from the dataset as they are not useful when implementing the Multiple Linear Regression.

The aim of the regression model in this context will be used to predict the price of the Ethereum cryptocurrency by using the ‘Open’, ‘High’ and ‘Low’ variables. It should also reveal which variables are influential and how so when predicting the price of Ethereum.

A validation set will be used at the end of this project to test our model on new unseen data from the same source as the original dataset. It will contain information about the price of the Ethereum cryptocurrency 16 days prior to when the project was finished.

Testing the model with a validation set in this manor will give us more insight into how well the model performs at predicing the price of Ethereum using only the 'Open', 'High' and 'Low' variables. As well as if the model would be useful when implemented on the real world price of the cryptocurrency for a shorter time period.

	Date	Price	Open	High	Low	Vol.	Change %
0	2021-11-13	4688.46	4665.76	4705.02	4585.37	377110	0.45%
1	2021-11-12	4667.31	4720.50	4807.16	4511.96	634590	-1.13%
2	2021-11-11	4720.87	4633.98	4778.17	4578.66	418610	1.87%
3	2021-11-10	4634.03	4731.83	4864.06	4498.78	761900	-2.07%
4	2021-11-09	4731.83	4808.34	4836.69	4715.43	437410	-1.59%
5	2021-11-08	4808.38	4612.05	4822.97	4612.05	673210	4.26%
6	2021-11-07	4612.06	4517.27	4634.39	4502.70	337810	2.10%
7	2021-11-06	4517.36	4475.00	4526.75	4330.29	442600	0.95%
8	2021-11-05	4475.00	4534.96	4569.31	4439.67	409750	-1.33%
9	2021-11-04	4535.11	4601.93	4604.83	4421.87	495860	-1.46%
10	2021-11-03	4602.21	4586.96	4661.88	4458.87	634680	0.35%
11	2021-11-02	4586.06	4320.85	4598.38	4287.36	630260	6.14%
12	2021-11-01	4320.85	4287.41	4376.94	4157.51	574260	0.78%
13	2021-10-31	4287.56	4322.00	4393.61	4169.48	555320	-0.77%
14	2021-10-30	4320.65	4413.70	4428.41	4250.24	494870	-2.11%
15	2021-10-29	4413.70	4284.90	4458.30	4268.21	781570	3.01%

Figure 2: The Validation Set

Hypothesis testing

One of the main questions that needs to answered when applying Multiple Linear Regression to the dataset is, does at least one of the independent variables help us to predict the output?

In this case, that question translates to ‘does any of the 'Open', 'High' and 'Low' variables from our dataset help to predict the ‘Price’ variable?’ which is the closing price of Ethereum on that day’.

The Multiple Linear Regression model can only give us coefficients (or β values) to establish a close enough linear relationship between the independent variables. Although this in of itself cannot definitively prove the existence of a relationship, it is a strong indication. Whenever the coefficient is 0, there is no relationship between the other independent variables used to make the predictions.

The null and the alternative hypothesis for the Multiple Linear Regression can be denoted as the following:

H_0**:** The 'Price' of the crypotocurrency Ethereum at any given day cannot be predicted by the 'Open', 'High' and 'Low' variables from our dataset for that day. The variables refer to the Opening Price, Highest Price and Lowest Price of Ethereum on that day.

$$H_0: \beta_1 = \beta_2 = \beta_3 = 0$$

H_A**:** The ‘Price’ of the crypotocurrency Ethereum at any given day can be predicted by the ‘Open’, ‘High’ and ‘Low’ variables from our dataset for that day.

$$H_A: At\ least\ one\ \beta_i\ is\ non\ -\ zero$$

In order to test the null hypothesis, we need to first determine what the β coefficients are for each independent variable which will show the likelihood of their influence on the model and then calculate their p-value. The ‘p-value’ of a β coefficient is a representation of the popbability that the β coefficient is actually zero.

If the p-value < 0.05 of at least one of the β coefficient is less than 0.05 we can reject the null hypothesis since at least one of the independent variables can be use to predict the ‘Price’ of Ethereum on that day.

Implementing the Multiple Linear Regression

Python and Scikit Learn

Steps required for implementing the Multiple Linear Regression to our dataset as follows:

1. Loading the dataset
2. Visualising the dataset
3. Preparing the dataset for the Model
4. Hypothesis testing
5. Splitting the dataset into training and testing sets
6. Training the model with our Training set
7. Evaluating the model
8. Testing our model with a validation set

Loading the dataset

First, we will load the '*Ethereum_Price_History_USD.csv*' file as a DataFrame, using the '*SNo*' as the index column.

Then we will print the first 10 Rows in the 'Eth_Price_Data' DataFrame to ensure that everything is as expected.

```
import pandas as pd

# Loads the dataset
Eth_Price_Data = pd.read_csv('Ethereum_Price_History_USD.csv',
index_col='SNo')

# Prints the first 10 Rows from the dataset
Eth_Price_Data.head(10)
```

SNo	Date	Price	Open	High	Low	Vol.	Change %
1	2016-03-10	11.75	11.20	11.85	11.07	0.0	4.91
2	2016-03-11	11.95	11.75	11.95	11.75	180.0	1.70
3	2016-03-12	12.92	11.95	13.45	11.95	830.0	8.12
4	2016-03-13	15.07	12.92	15.07	12.92	1300.0	16.64
5	2016-03-14	12.50	15.07	15.07	11.40	92180.0	-17.05
6	2016-03-15	13.06	12.63	13.42	11.98	39730.0	4.48
7	2016-03-16	12.88	13.06	13.89	12.62	19240.0	-1.38
8	2016-03-17	10.74	12.58	12.61	10.44	89450.0	-16.61
9	2016-03-18	10.74	11.09	11.20	8.34	157370.0	0.00
10	2016-03-19	10.31	10.97	11.10	9.78	67550.0	-4.00

```
print('Number of the days of Ethereum prices contained in the dataset: \n',
len(Eth_Price_Data['Date']), 'days' )
```

```
Number of the days of Ethereum prices contained in the dataset:
 1994 days
```

Visualising the dataset

We will plot the dataset to analyse the price of Ethereum for a period of over 5 years.

In order to plot the Ethereum dataset as we have loaded it, we will need to:

- Create a new DataFrame contining only the 'Date' and 'Price' Variables
- Creates a seperate DataFrame containing only the 'Date' variable
- Convert the 'Date' DataFrame to datetime
- Plot the final updated variables using matplotlib

Creating new DataFrame for plotting the Ethereum price history purposes only containing the 'Date' and 'Price' Variables as it won't affect the original dataset.

We will then seperate the 'Date' variables with a unique DataFrame to convert the variable to datetime which is needed for the 'autofmt_xdate()' function used to make the plot more readable.

Finally we can plot the complete dataset with the updated variables to observe the daily price for Ethereum since it entered the Stock exchange in 2016.

```
from matplotlib import pyplot as plt

## Styling the plots for the project
plt.style.use(style="seaborn")
%matplotlib inline

# Creates a new DataFrame containing only the 'Date'
# and 'Price' Variables from the dataset
Plot_Eth_Price = pd.DataFrame(Eth_Price_Data, columns = ['Date', 'Price'])

# Creates a new DataFrame containing only the 'Date' variable
# from the above 'Plot_Eth_Price' DataFrame
Plot_Eth_Date = Plot_Eth_Price['Date']

# Converts the content in the 'Plot_Eth_Date' variable to
# datetime. This is needed for the 'autofmt_xdate()' function
# used to make the plot of the dataset more readable
Plot_Eth_Date = pd.to_datetime(Plot_Eth_Date)

plt.title('Ethereum Price History on a daily basis')
plt.xlabel('Date')
plt.ylabel('Ethereum Price in U.S. Dollers')
plt.grid(True)
plt.plot(Plot_Eth_Date, Plot_Eth_Price['Price'], color = 'red')
plt.gcf().autofmt_xdate()
plt.savefig('Ethereum_Price_History.png')

plt.show()
```

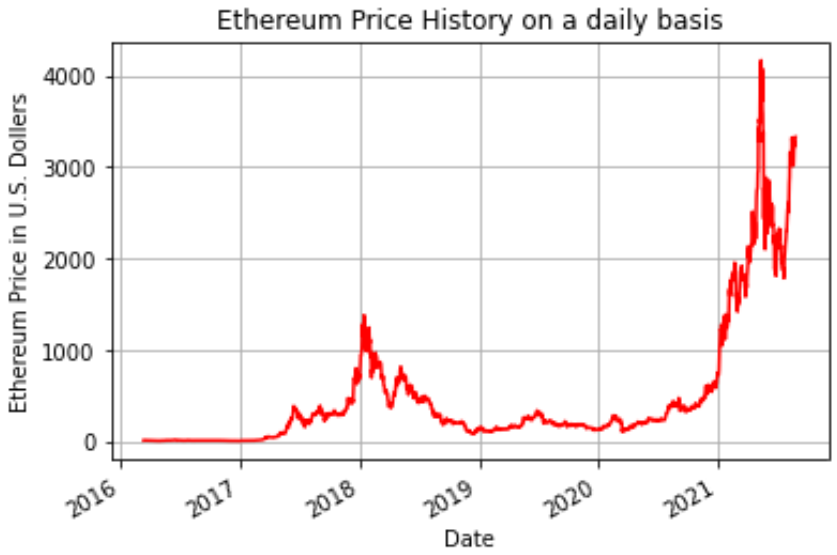


Figure 3: Ethereum Price History on a daily basis

Preparing the dataset for the Model

In order to prepare the data for the Regression model we will need remove the unwanted variables. In this case, we will remove 'Vol.' and 'Change'

We will also split the 'Eth_Price_Data' DataFrame to eventually create training and testing set:

- 'data_X' will contain all the columns except the 'Price'
- 'data_Y' will contain only the 'Price' column

```
# Creates a new DataFrame from the dataset that only contains the 'Price',
# 'High', 'Low' and 'Open' Columns
Eth_Price_Data = Eth_Price_Data[['Price', 'High', 'Low', 'Open']].dropna()

# Split the 'Eth_Price_Data' DataFrame to eventually create
# training and testing set
# 'data_X' will contain all the columns except the 'Price'
# 'data_Y' will contain only the 'Price' column
data_X = Eth_Price_Data.loc[:,Eth_Price_Data.columns !=  'Price' ]
data_Y = Eth_Price_Data['Price']
```

Hypothesis testing

One of the main question that needs to be answered before applying Multiple Linear Rgression to a dataset is, does at least one of the indepedent varibales help us in predicting the output.

Just a brief over

H_0: The 'Price' of the crypotocurrency Ethereum at any given day cannot be predicted by the 'Open', 'High' and 'Low' variables from our dataset for that day. The variables refer to the Opening Price, Highest Price and Lowest Price of Ethereum on that day.

$$H_0 = \beta_1 = \beta_2 = \beta_3 = 0$$

H_A: The 'Price' of the crypotocurrency Ethereum at any given day cannot be predicted by the 'Open', 'High' and 'Low' variables from our dataset for that day.

$$H_0: \text{At least one } \beta_i \text{ is non-zero}$$

To perform a hypothesis test on the dataset, we will use the *'f_regression'* function:

```
# Imports the 'f_regression' function which computes the F statistics
from sklearn.feature_selection import f_regression
```

```
(F_statistic, p_values) = f_regression(data_X, data_Y)
```

Note:

- β_1 refers to the 'High' Variable influence on the model
- β_2 refers to the 'Low' Variable influence on the model
- β_3 refers to the 'Open' Variable influence on the model

```
print('P-value for beta_1:        ', p_values[0])
print('P-value for beta_2:        ', p_values[1])
print('P-value for beta_3:        ', p_values[2])
```

```
P-value for beta_1:         0.0
P-value for beta_2:         0.0
P-value for beta_3:         0.0
```

The p-values are used for the hypothesis test. The 'High', 'Low' and 'Open' variables have a statistically significant p-value.

The p-value for all the coefficients are 0 so we can reject the null hypothesis. This basically proves that there is a relationship between the 'High', 'Low' and 'Open' variables can be used to predict the 'Price' of Ethereum on that day.

Splitting the dataset into training and testing set

Scikit Learn has a function '*train_test_split*' to split the dataset into two categories: training and testing set. It will create four seperarte arrays beased on these splits: *train_X*, *test_X*, *train_Y*, *test_Y*.

In this case, we are randomly putting 75% of the dataset into the training set and the remaining 25% into our testing set. The random nature in this process helps to vary the type of data in each set.

```
from sklearn.model_selection import train_test_split

# Splitting the dataset into training and testing set
train_X, test_X, train_Y,test_Y = train_test_split(data_X, data_Y,
test_size=0.25, random_state=0)
```

Training the model with our Training set

Implementing linear regression with Scikit learn to the training set from our dataset.

The 'LinearRegression' function fits the linear model with coefficients (β_1, ..., β_p) to minimise the residual sum of squares between the outputs and the independent variables in the dataset.

Here we create a Linear Regressor model object and subsequently trains our regression model with our training set with the fit() method.

```
from sklearn.linear_model import LinearRegression

# Trains the model on our training dataset
regressor = LinearRegression()
regressor.fit(train_X,train_Y)
```

Our final linear regression model will be:

$$Price = \beta_0 + \beta_1 . High + \beta_2 . Low + \beta_3 . Open$$

where:

- β_0 is the intercept that shows the output of model when all the independent variables are 0.
- β_1 is a coeffieient that shows the influence the '*High*' variable has on the predictions.
- β_2 is a coeffieient that shows the influence the '*Low*' variable has on the predictions.
- β_3 is a coeffieient that shows the influence the '*Open*' variable has on the predictions.

Testing the model with our Testing set

We will now test our model to predict the values for our testing set. This is done with the '*.predict*' method from Scikit-Learn.

Once our model has finished predicting the price values, we can measure the performance the accuracy of the model using Scikit-Learn's '*r2_score*' function.

```
import numpy as np

# Use the model to predict the values for our testing set
```

```
predict_Y = regressor.predict(test_X)

print('Accuracy score with our test set: ' , regressor.score(test_X,test_Y))

## The intercept is the expected mean value of Y when all X=0
print('Intercept: ', regressor.intercept_)

# Coefficient are the weights
print('Coefficients: \n')
print(regressor.coef_)

print('')

coefficients = pd.DataFrame(regressor.coef_, train_X.columns,
columns=['Coefficient'])
coefficients
```

```
Accuracy score with our test set:  0.9992289474063466
Intercept:  -0.1380558838586694
Coefficients:

[ 0.92015022  0.65675643 -0.58033529]
```

	Coefficient
High	0.920150
Low	0.656756
Open	-0.580335

From our results, the '*High*' variables has the greatest influence on the price of Ethereum on a given day as it has a β coefficient of over 0.92.

The model appears to perform really well with a R^2 score of over 99.9% which indicates that the model is relatively good at predicting Ethereums price give the highest price, lowest price and opening price on a given day.

β_1: The '*High*' Variable influence on the model	0.920150
β_2: The '*Low*' Variable influence on the model	0.656756
β_3: The '*Open*' Variable influence on the model	-0.580335

- β_1 refers to the 'High' Variable influence on the model
- β_2 refers to the 'Low' Variable influence on the model
- β_3 refers to the 'Open' Variable influence on the model

Evaluating the model

A Multiple Linear Regression model is assessed by metrics to provide a measure of how well the predicted outputs are being generated by the model.

Evaluating the model is essential in understanding its performance to both the developer and user. Its aim is to measure how well outputs predicted by the model compare to the actual values of the observed outputs from the dataset. In essence, the best model for the dataset is the model with the least error.

The evalulation metrics used in the project:

- Coefficient of Deterimintion on testing set

- Mean Squared Error (MSE)/ Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE)/ Mean Absolute Percentage Error (MAPE)
- Adjusted R^2

Below, we will use Scikit-Learns '*.metrics*' method to easily canculate the models metrics.

```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error,
mean_absolute_percentage_error

# Calculates the mean absolute error of our model
meanAbError = mean_absolute_error(test_Y, predict_Y)

# Calculates the mean absolute percentage error of our model
meanPerAbError = mean_absolute_percentage_error(test_Y, predict_Y)

# Calculates the mean squared error of our model
meanSqError = mean_squared_error(test_Y, predict_Y)

# Calculates the R2 score of our model
r2 = r2_score(test_Y, predict_Y) # R2 Score
```

```
# Calculates the adjusted R2 score of our  model
n = len(test_X)        # Number of data points in our testing set
p = 3                  # Number of independent variables used in model
adjusted_r2 = 1 - ((1-r2) * (n-1) / (n-p-1))

print('Coefficient of Determination on training: ', r2)
print('Mean Squared Error:                       ', meanSqError)
print('Root Mean Squared Error:                  ', np.sqrt(meanSqError))
print('Mean Absolute Error:                      ', meanAbError)
print('Adjusted R2 Squared:                      ', adjusted_r2)

print('Mean Absolute Percentage Error:           ', meanPerAbError)
```

```
Coefficient of Determination on training:  0.9992289474063466
Mean Squared Error:                        391.1387125906281
Root Mean Squared Error:                   19.77722712087385
Mean Absolute Error:                       8.096399055124106
Adjusted R2 Squared:                       0.9992242743603245
Mean Absolute Percentage Error:            0.01635415475872566
```

Some takeaways from our evaluation:

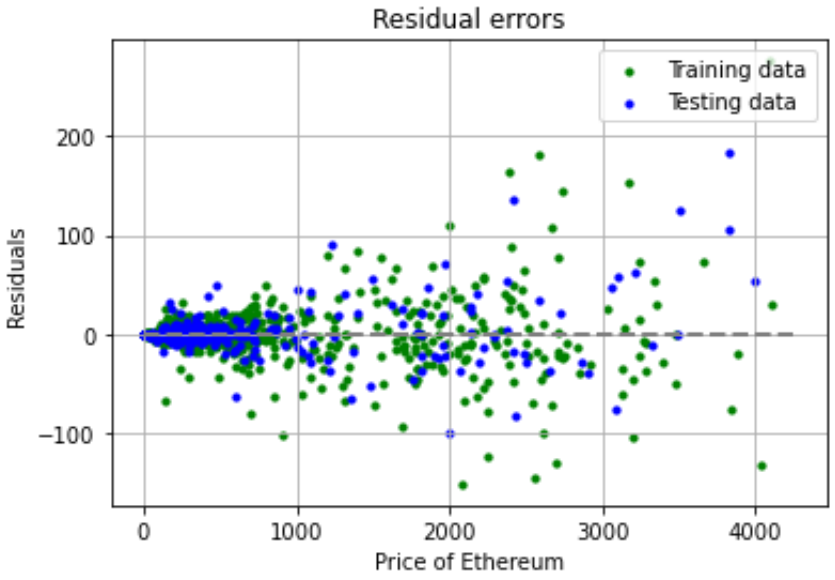
- The R^2 metric for the model on the training set is 0.99922 which indicates that it is really likely that the predicted value will be close to the actual value. This means the hyperplane is fitted close to the price data. There was no major change in the adjusted R^2 value implying that the amount of independent variables used in the Regression model didn't negatively impact the models accuracy.
- The Mean Squared Error (MSE) and Root Mean Square Error (RMSE) metrics for the model are 391.1387 and 19.7772 respectively which shows that there are many large outliers in the models predictions.
- The Mean Absolute Error (MAE) is 8.096399 meaning that the average amount our predicted value differed from the actual amount attributed to a $8.10 price difference for the cryptocurrency. This isn't too bad when considering that fact the price of Ethereum is in the 1000's of Dollars for a lot of the Dataset.

- This error can be better observed as a percentage which is accomplished with the Mean Absolute Percentage Error (MAPE) metric. For this model, the metric showed us that the error was roughly 1.63% across the testing set in relation to the price of Ethereum on that day.

Residual Plots

Residual plots are useful for visualising the errors in your data. If the model performed well the data in the plot should be randomly scattered around line zero.

```
# Plotting the residual errors from our training data
plt.scatter(regressor.predict(train_X), regressor.predict(train_X) - train_Y,
color = 'green', s = 10, label = 'Training data')

# Plotting the residual errors from our testing data
plt.scatter(regressor.predict(test_X), regressor.predict(test_X) - test_Y,
color = 'blue', s = 10, label = 'Testing data')

# Plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 4250, linewidth = 2, linestyles = '--',
color = 'gray')

plt.legend(loc = 'upper right')
plt.title('Residual errors')
plt.ylabel('Residuals')

plt.show()
```



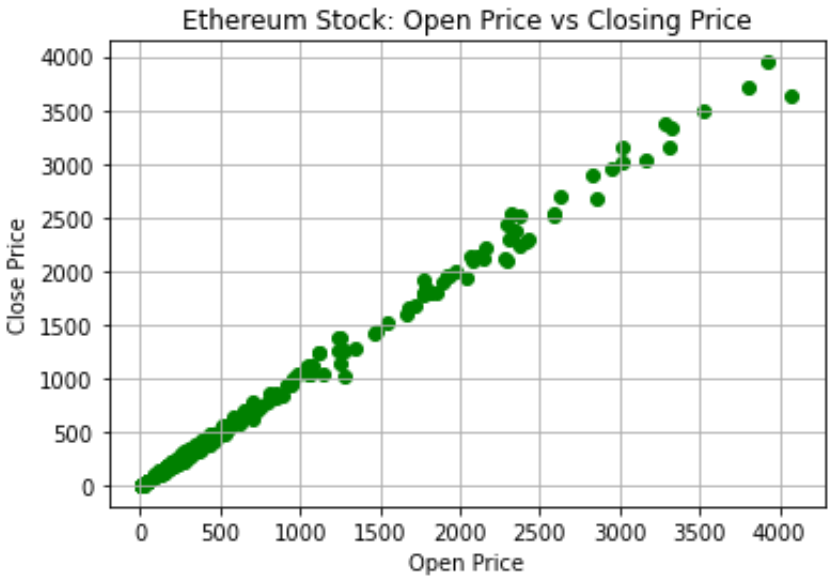
The residual errors are larger as the price of Ethereums grows in value. This is underdstandable as higher values of the price have a larger error even with the same performance from the model.

Toward the end of the plot, it shows that the model makes less accurate predictions as the price of ethereum increases. This is most likely due to the fact that the price of Ethereum drastically increased on a short period of time. For example, the price of Ethereum grow by over 25 times in a period of 14 months which is unprecedented starting from the March 2020. It would be unrealistic for our model to be able to adjust for such a change appropriately since most of the data for the independent variables in the dataset were taken when the cryptocurrency was worth under $500. This results in a model that's really good at predict the price of Ethereum when it is valued under $500 but it underperforms when the value of Ethereum is higher than $500.

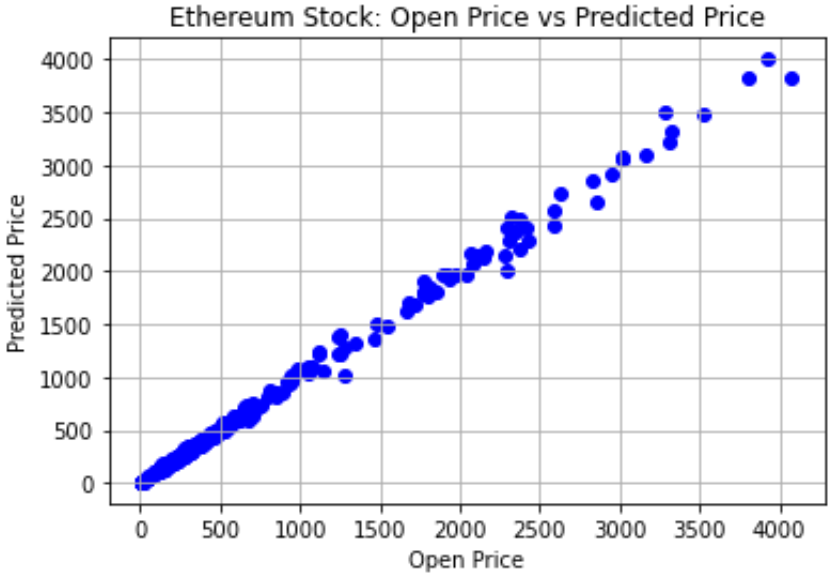
We will plot both the 'Open' price and Predicted price vs the closing price.

```
# Plot the opening price values and the closing price values

plt.title('Ethereum Stock: Open Price vs Closing Price')
plt.xlabel('Open Price')
plt.ylabel('Close Price')
plt.grid(True)
plt.scatter(test_X['Open'], test_Y, marker='o', c= 'green');
```

```
plt.show()
```




```
# Plot the predicted and the expected values

plt.title('Ethereum Stock: Open Price vs Predicted Price')
plt.xlabel('Open Price')
plt.ylabel('Predicted Price')
plt.grid(True)
plt.scatter(test_X['Open'], predict_Y, marker='o', c='blue');
plt.show()
```



Both plots are done seperately because the data overlaps too much. It just goes to show how similar they are.

Finally, we will create a new DataFrame containing all the actual prices and the predicted prices in our testing set.

```
model_result = pd.DataFrame({'Actual Price' :test_Y, 'Predicted Price'
:predict_Y, 'Difference' :predict_Y-test_Y })

model_result
```

SNo	Actual Price	Predicted Price	Difference
361	19.35	19.481926	0.131926
1170	244.68	238.155149	-6.524851
258	9.84	10.549752	0.709752
15	11.20	11.734867	0.534867
1561	233.88	231.225114	-2.654886
...	...	...	...
642	515.25	519.566089	4.316089
490	223.92	214.116071	-9.803929
1785	1247.62	1265.856414	18.236414
1080	147.97	147.162067	-0.807933
1199	294.72	292.286383	-2.433617

499 rows × 3 columns

```
print('Number of days of data in our test set:          ', len(test_Y))
print('Sum of the Differences between our Predicted')
print('Prices and Acutal Prices:                        ',
sum(model_result['Difference']))

# Calculates the average difference between the predicted
# price and the actual price
avg_diff = sum(model_result['Difference']) / len(model_result['Difference'])

print('Average Difference between the predicted')
print('Price and the Actual Price:                       ', avg_diff)
```

```
Number of days of data in our test set:          499
Sum of the Differences between our Predicted
Prices and Acutal Prices:                        119.27323462832778
Average Difference between the predicted
Price and the Actual Price:                       0.23902451829324206
```

The model suggests that the ‘High’, ‘Low’ and ‘Open’variables are indicative of the ‘Price’ variable which translates to the closing price of Ethereum the that same day. Based on the found coefficients, we conclude that the high-price is most influential on the closing price.

Testing of our model with a validation set

Further testing of our model with a validation set is important because it will elaborate the usefullness of the model.

```
# Loading the validation set 'Ethereum_Validation_Set.csv'
validation_set = pd.read_csv('Ethereum_Validation_Set.csv')

# Issolating the actual price of the data in the
# validation set
validation_actual_output = validation_set['Price']

# Prints the entire validation set
validation_set.head(20)
```

	Date	Price	Open	High	Low	Vol.	Change %
0	2021-11-13	4688.46	4665.76	4705.02	4585.37	377110	0.45%
1	2021-11-12	4667.31	4720.50	4807.16	4511.96	634590	-1.13%
2	2021-11-11	4720.87	4633.98	4778.17	4578.66	418610	1.87%
3	2021-11-10	4634.03	4731.83	4864.06	4498.78	761900	-2.07%
4	2021-11-09	4731.83	4808.34	4836.69	4715.43	437410	-1.59%
5	2021-11-08	4808.38	4612.05	4822.97	4612.05	673210	4.26%
6	2021-11-07	4612.06	4517.27	4634.39	4502.70	337810	2.10%
7	2021-11-06	4517.36	4475.00	4526.75	4330.29	442600	0.95%
8	2021-11-05	4475.00	4534.96	4569.31	4439.67	409750	-1.33%
9	2021-11-04	4535.11	4601.93	4604.83	4421.87	495860	-1.46%
10	2021-11-03	4602.21	4586.96	4661.88	4458.87	634680	0.35%
11	2021-11-02	4586.06	4320.85	4598.38	4287.36	630260	6.14%
12	2021-11-01	4320.85	4287.41	4376.94	4157.51	574260	0.78%
13	2021-10-31	4287.56	4322.00	4393.61	4169.48	555320	-0.77%
14	2021-10-30	4320.65	4413.70	4428.41	4250.24	494870	-2.11%
15	2021-10-29	4413.70	4284.90	4458.30	4268.21	781570	3.01%

```
# Removing all the unnecessary columns from the validation set
validating_set = pd.DataFrame(validation_set[['High', 'Low', 'Open']])

# Use the original model to predict the 'Price' of
# of Ethereum on new dates
validation_Y = regressor.predict(validating_set)

# Creates a new DataFrame containing all the results from our model
validation_result = pd.DataFrame({'Date' :validation_set['Date'], 'Actual
Price' :validation_actual_output, 'Predicted Price' :validation_Y,
'Difference' :validation_Y-validation_actual_output })

validation_result
```

	Date	Actual Price	Predicted Price	Difference
0	2021-11-13	4688.46	4632.953195	-55.506805
1	2021-11-12	4667.31	4646.957295	-20.352705
2	2021-11-11	4720.87	4714.298404	-6.571596
3	2021-11-10	4634.03	4684.082594	50.052594
4	2021-11-09	4731.83	4756.782911	24.952911
5	2021-11-08	4808.38	4790.176984	-18.203016
6	2021-11-07	4612.06	4599.842918	-12.217082
7	2021-11-06	4517.36	4412.097344	-105.262656
8	2021-11-05	4475.00	4488.298053	13.298053
9	2021-11-04	4535.11	4470.426469	-64.683531
10	2021-11-03	4602.21	4555.908647	-46.301353
11	2021-11-02	4586.06	4539.271836	-46.788164
12	2021-11-01	4320.85	4269.640360	-51.209640
13	2021-10-31	4287.56	4272.766841	-14.793159
14	2021-10-30	4320.65	4304.610973	-16.039027
15	2021-10-29	4413.70	4418.663361	4.963361

The model doesn't perform as well on the validation set as it did on the testing set from the original dataset. This is most likely due to the variation in data in the training set used to train the model, it is very unlike the data in the validation set.

```
val_r2 = r2_score(validation_actual_output, validation_Y) # R2 Score
print('R2 Score of model on the validation set: ' , val_r2)
```

```
R2 Score of model on the validation set:  0.9199773264059619
```

The R^2 score on the validation set is 0.919977 which is reasonably accurate, though not as accurate on the Testing set. The hyperplane fits close to the data but the predictions are often unacceptably large, such as on '06-11-21' when the model underestimated the Ethereum price by over $100.

Getting some more information about our results:

```
print('Number of days of data in our validation set:     ',
len(validating_model['High']))
print('Sum of the Differences between the Predicted')
print('Price and Acutal Price:                           ',
sum(validation_result['Difference']))

# Calculates the average difference between the predicted
# price and the actual price
avg_diff = sum(validation_result['Difference']) /
len(validation_result['Difference'])

print('Average Difference between the predicted')
print('price and the acutal Price:                       ', avg_diff)
```

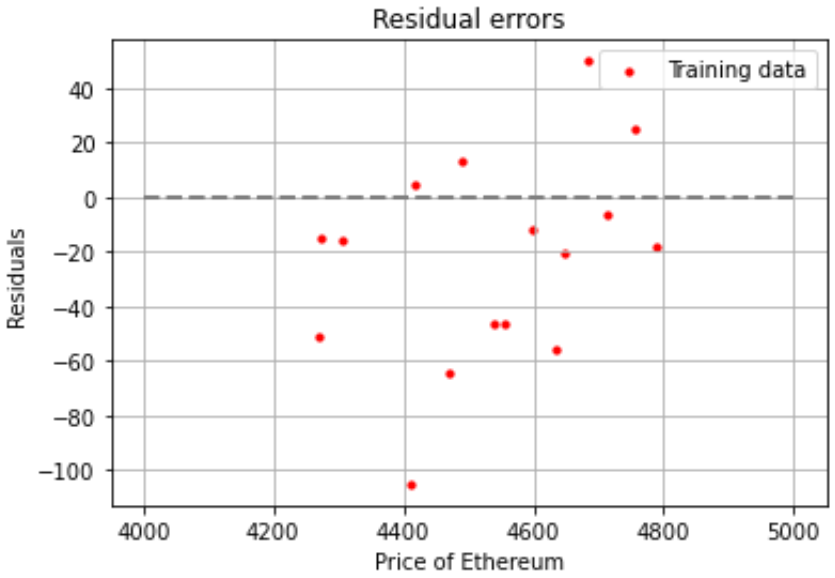
```
Number of days of data in our validation set:      16
Sum of the Differences between the Predicted
Price and Acutal Price:                            -364.66181517900804
Average Difference between the predicted
price and the acutal Price:                        -22.791363448688003
```

```
# Calculates the mean absolute error of our model
val_meanAbError = mean_absolute_error(validation_actual_output, validation_Y)

# Calculates the mean absolute percentage error of our model
val_meanPerAbError = mean_absolute_percentage_error(validation_actual_output,
validation_Y)

# Calculates the mean squared error of our model
val_meanSqError = mean_squared_error(validation_actual_output, validation_Y)

# Calculates the R2 score of our model
val_r2 = r2_score(validation_actual_output, validation_Y) # R2 Score

# Calculates the adjusted R2 score of our  model
n = len(validation_actual_output)        # Number of data points in our
testing set
p = 3                                    # Number of independent variables
used in model
val_adjusted_r2 = 1 - ((1-val_r2) * (n-1) / (n-p-1))

print('Coefficient of Determination on training: ', val_r2)
print('Mean Squared Error:                       ', val_meanSqError)
print('Root Mean Squared Error:                  ', np.sqrt(val_meanSqError))
print('Mean Absolute Error:                      ', val_meanAbError)
```

```
print('Adjusted R2 Squared:                          ', val_adjusted_r2)

print('Mean Absolute Percentage Error:              ', val_meanPerAbError)
```

```
Coefficient of Determination on training:  0.9199773264059619
Mean Squared Error:                        1877.7538960779648
Root Mean Squared Error:                   43.33305777438242
Mean Absolute Error:                       34.44972823008095
Adjusted R2 Squared:                       0.8999716580074524
Mean Absolute Percentage Error:            0.0075649261190462985
```

```
# Plotting the residual error
# Plotting the rediual errors in the validation data
plt.scatter(regressor.predict(validating_set),
regressor.predict(validating_set) - validation_actual_output, color = 'red',
s = 10, label = 'Validation data')

# Plotting line for zero residual error
plt.hlines(y = 0, xmin = 4000, xmax = 5000, linewidth = 2, linestyles = '--',
color = 'gray')

plt.legend(loc = 'upper right')
plt.title('Residual errors')
plt.xlabel('Price of Ethereum')
plt.ylabel('Residuals')
plt.grid(True)

plt.show()
```



When test the model with a validation set, things to note:

- The R^2 metric for the model on the validation set is 0.919977 which indicates that the model performs worse than on the testing set. Therefore, the model predicts prices that are further from the actual price of Ethereum on that date.
- The model contains a lot of outliers for the validation set based on the Root Mean Squared Error (RMSE) and Residual plot, which is not ideal for a prediction model.

Overview of the results

Our final model

Our final linear regression model will be:

$$Price = \beta_0 + \beta_1 . High + \beta_2 . Low + \beta_3 . Open$$

Our training data

On the validation set

Conclusion

References