# [Data science] Crypto Currency Predition

# 1 - Ethereum Historical Data Extraction

Here is an example of how you could retrieve historical data on Ethereum using the CryptoCompare API in Python using the basic plan:

In [8]:
```
import requests
import json

# Define the endpoint for the CryptoCompare API
url = 'https://min-api.cryptocompare.com/data/histoday'

# Define the parameters for the API request
parameters = {
    'fsym': 'ETH',
    'tsym': 'USD',
    'limit': '2000'
}

# Make the API request
response = requests.get(url, params=parameters)

# Parse the response
data = json.loads(response.text)

# Print the historical data
print(data['Data'][:10])   #Only Showing first 10 values just for confirmation

#The data variable contains 2000 records which have been extrected from the CryptoCompare API
#Note: This Free web-API doesn't allow us to extract more than 2000 records
```

```
[{'time': 1500940800, 'high': 225.98, 'low': 192.64, 'open': 225.48, 'volumefrom': 1382784.11, 'volumeto': 285374761.65, 'close': 203.59, 'conversionType': 'direct', 'conversionSymbol': ''}, {'time': 1501027200, 'high': 209.29, 'low': 193.1, 'open': 203.59, 'volumefrom': 809346.64, 'volumeto': 161659275.04, 'close': 202.88, 'conversionType': 'direct', 'conversionSymbol': ''}, {'time': 1501113600, 'high': 205.68, 'low': 198.93, 'open': 202.88, 'volumefrom': 504777.47, 'volumeto': 102009361.91, 'close': 202.93, 'conversionType': 'direct', 'conversionSymbol': ''}, {'time': 1501200000, 'high': 203.93, 'low': 189.77, 'open': 202.93, 'volumefrom': 631780.7, 'volumeto': 123425467.06, 'close': 191.21, 'conversionType': 'direct', 'conversionSymbol': ''}, {'time': 1501286400, 'high': 209.57, 'low': 177.69, 'open': 191.21, 'volumefrom': 1010020.82, 'volumeto': 192325102.02, 'close': 206.14, 'conversionType': 'direct', 'conversionSymbol': ''}, {'time': 1501372800, 'high': 209.88, 'low': 194.12, 'open': 206.14, 'volumefrom': 553749.07, 'volumeto': 110586575.75, 'close': 196.78, 'conversionType': 'direct', 'conversionSymbol': ''}, {'time': 1501459200, 'high': 201.78, 'low': 190.53, 'open': 196.78, 'volumefrom': 528822.67, 'volumeto': 103641349.39, 'close': 201.33, 'conversionType': 'direct', 'conversionSymbol': ''}, {'time': 1501545600, 'high': 232.55, 'low': 201.27, 'open': 201.33, 'volumefrom': 1444224.2, 'volumeto': 314260050.78, 'close': 225.9, 'conversionType': 'direct', 'conversionSymbol': ''}, {'time': 1501632000, 'high': 228.98, 'low': 215.71, 'open': 225.9, 'volumefrom': 557162.98, 'volumeto': 122967812.49, 'close': 218.12, 'conversionType': 'direct', 'conversionSymbol': ''}, {'time': 1501718400, 'high': 228.08, 'low': 217.84, 'open': 218.12, 'volumefrom': 417327.69, 'volumeto': 93201219.53, 'close': 224.39, 'conversionType': 'direct', 'conversionSymbol': ''}]
```

## Converting JSON to CSV and Exporting as .csv

You can use the pandas library to convert the JSON data returned from the API into a CSV file. Here's an example of how you can do this:

In [9]:
```
import pandas as pd

# Load the data into a pandas DataFrame
df = pd.DataFrame(data['Data'])

# Convert the DataFrame to a CSV file
df.to_csv('eth_data.csv', index=False)
```

This code creates a new CSV file called "eth_data.csv" in the same directory as your script, with the data from the API response. The index=False argument tells pandas to not include the DataFrame's index in the output CSV file.

Please note that the data structure of the API response may be different, you may need to adjust the code accordingly.

# 2 - Machine Learning

For machine learning model you can use any algorithm such as Linear Regression, Random Forest, LSTM etc.

In [10]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pandas as pd

# Load the data into a pandas DataFrame
df = pd.read_csv('eth_data.csv')

# Define the features and target
X = df[['high', 'low', 'open','volumefrom','volumeto', 'close']]
y = df['close']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Use the model to make predictions on the test data
y_pred = model.predict(X_test)

print(y_pred[:30])   #Only Showing first 30 predictions just for confirmation
```

```
[2442.89        432.97       1805.49        169.98000001 1597.44
 1367.65        258.79       2963.17        302.77        242.08
  283.          206.78       2531.75        135.56        417.76
  661.45        206.          434.85       2114.51       3569.26
  254.66       1848.69       1210.59        239.54       1833.26
  289.42        346.94        111.34000001  170.38000001  317.4       ]
```

In [11]:
```
# Evaluate the model's performance
score = model.score(X_test, y_test)
print(f'R^2 score: {score}')
```

```
R^2 score: 1.0
```

Please be aware that using a free/basic plan API will limit the amount of data you can retrieve and the accuracy of predictions will be low. Also, for machine learning model, you need to pre-process the data and use feature engineering techniques to improve the accuracy of predictions.

# 3 - Predict Future Prices and Store in a DB

Yes, you can use the trained model to make predictions on new, unseen data and store those predictions in a database.

Here is an example of how you could use a Jupyter notebook to make predictions using a trained model and store those predictions in a SQLite database:

In [12]:
```
#Make predictions using the trained model
y_pred = model.predict(X_test)

# Convert predictions to a DataFrame
predictions_df = pd.DataFrame(y_pred, columns=['predicted_price'])

# Import the SQLite3 library
import sqlite3

# Connect to a SQLite database
conn = sqlite3.connect('predictions.db')

# Store the predictions in a SQLite table
predictions_df.to_sql('predictions', conn, if_exists='replace')
```

This code makes predictions on the test data using the trained model, and converts the predictions to a DataFrame. Then, it imports the SQLite3 library, connects to a SQLite database called 'predictions.db', and stores the predictions in a table called 'predictions' in the database.

It's worth noting that you can also use other databases such as MySQL, PostgreSQL, MongoDB

### Getting records from the Database

In [13]:
```
# Read the 'predictions' table into a DataFrame
predictions_df = pd.read_sql_query('SELECT * FROM predictions', conn)

# Print the first 5 rows of the DataFrame
print(predictions_df.head())
```

```
   index  predicted_price
0      0          2442.89
1      1           432.97
2      2          1805.49
3      3           169.98
4      4          1597.44
```

# 4 - Evaluation

Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) are commonly used metrics for evaluating the accuracy of time series predictions.

MAE measures the average difference between the predicted and actual values. It gives an idea of how far off the predictions are from the actual values. MSE measures the average squared difference between the predicted and actual values. It gives more weight to larger errors. RMSE is the square root of MSE. It is used to interpret the model's performance in the same units as the original data. You can use the mean_absolute_error(), mean_squared_error(), and sqrt() functions from the sklearn.metrics module to calculate these metrics:

In [14]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
from math import sqrt

# Calculate MAE
mae = mean_absolute_error(y_test, predictions_df['predicted_price'])

# Calculate MSE
mse = mean_squared_error(y_test, predictions_df['predicted_price'])

# Calculate RMSE
rmse = sqrt(mse)

# Print the results
print(f'MAE: {mae}')
print(f'MSE: {mse}')
print(f'RMSE: {rmse}')
```

```
MAE: 1.9405040832718e-09
MSE: 7.260987360392698e-18
RMSE: 2.6946219327380043e-09
```

## Accuracy

For regression problems, instead of using accuracy as a performance metric, you can use metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). These metrics give you an idea of how close the predicted values are to the actual values.

Another way to check the accuracy of your prediction is to check the correlation between your predicted values and the actual values. You can use the r2_score() function from sklearn.metrics to check the correlation coefficient.

In [15]:
```
from sklearn.metrics import r2_score

# Calculate R-squared score
r2 = r2_score(y_test, predictions_df['predicted_price'])

# Print the R-squared score
print(f'R-squared score: {r2}')
```

```
R-squared score: 1.0
```

This will give you a value between -1 and 1, where 1 means a perfect positive correlation and -1 means a perfect negative correlation. The closer the value is to 1, the better the model is at predicting the target variable.

In [ ]:
```

```