Python Libraries

In [1]:
```
import datetime
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import requests
import statsmodels.api as sm
```

In [2]:
```
def get_ethereum_data():
    # Define the base URL for the CoinGecko API
    base_url = "https://api.coingecko.com/api/v3"

    # Get current timestamp for the 'to' parameter
    end_date = int(datetime.datetime.now().timestamp())

    # Get timestamp for 30 days ago for the 'from' parameter
    start_date = int((datetime.datetime.now() - datetime.timedelta(days=30)).t
imestamp())

    # Endpoint for getting Ethereum's market chart data (prices and volumes)
    url = f"{base_url}/coins/ethereum/market_chart/range?vs_currency=usd&from=
{start_date}&to={end_date}"

    # Make the request to the CoinGecko API
    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()

        # Prepare lists to store the data
        timestamps = []
        prices = []
        volumes = []

        # Process and store the data
        for i in range(len(data['prices'])):
            timestamp = data['prices'][i][0] / 1000  # Convert ms timestamp to
seconds
            date = datetime.datetime.utcfromtimestamp(timestamp).strftime('%Y-
%m-%d')
            price = data['prices'][i][1]
            volume = data['total_volumes'][i][1]

            timestamps.append(date)
            prices.append(price)
            volumes.append(volume)

        # Create a DataFrame
        df = pd.DataFrame({
            'Date': timestamps,
            'Price': prices,
            'Volume': volumes
        })

        return df
    else:
        print(f"Error fetching data from CoinGecko API: {response.status_cod
e}")
```

In [3]:
```
# Example: Fetch the first 10 Ethereum transactional data i.e., Price and Volu
me
transaction_data = get_ethereum_data()
transaction_data
```

Out[3]:

	Date	Price	Volume
0	2024-10-15	2661.786216	1.965807e+10
1	2024-10-15	2562.429365	1.978650e+10
2	2024-10-15	2586.534484	1.865791e+10
3	2024-10-15	2592.031112	2.114254e+10
4	2024-10-15	2590.483536	2.111182e+10
...	...	...	...
715	2024-11-14	3204.751549	5.929675e+10
716	2024-11-14	3209.832086	5.974423e+10
717	2024-11-14	3193.446176	5.988718e+10
718	2024-11-14	3185.999223	5.708326e+10
719	2024-11-14	3190.733281	5.945789e+10

720 rows × 3 columns

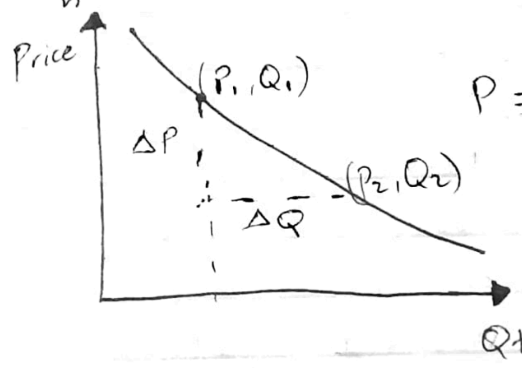
In [4]:
```
df = pd.DataFrame(transaction_data)
# Display the transactions dataframe
df.head()
```

Out[4]:

	Date	Price	Volume
0	2024-10-15	2661.786216	1.965807e+10
1	2024-10-15	2562.429365	1.978650e+10
2	2024-10-15	2586.534484	1.865791e+10
3	2024-10-15	2592.031112	2.114254e+10
4	2024-10-15	2590.483536	2.111182e+10

<u>Demand Curve</u>

Sh



$P = \frac{k}{Q}$

$Q_2 > Q_1$

$P_1 > P_2$

Price ↑ Qty ↓ ⇒ Inverse relationship.

$\frac{\Delta P}{\Delta Q} = \frac{P_2 - P_1}{Q_2 - Q_1} = -K$ since $P_1 > P_2$, $Q_2 > Q_1$

<u>In Practical scenario</u>

P = Token Price

Q = Qty = Volume

Obtaining K through Regressional Analysis.

In [5]:

```
"OLS Regression"
def perform_regression(df):
    # Log transform the Price and Volume
    df['Log_Price'] = df['Price']
    #df['Log_Price'] = np.log(df['Price'])
    df['Log_Volume'] = df['Volume']
    #df['Log_Volume'] = np.log(df['Volume'])

    # Define the independent variable (Log_Volume) and add a constant
    X = sm.add_constant(df['Log_Volume'])

    # Define the dependent variable (Log_Price)
    y = df['Log_Price']

    # Perform the regression
    model = sm.OLS(y, X).fit()

    # Print the regression results
    print(model.summary())

    return model,X,y
# Fetch Ethereum data

if df is not None:
    # Perform regression analysis
    model,X,y = perform_regression(df)
```

```
                            OLS Regression Results
==============================================================================
=
                                                                    lapse
Dep. Variable:              Log_Price   R-squared:                       0.74
9
Model:                            OLS   Adj. R-squared:                  0.74
9
Method:                 Least Squares   F-statistic:                     214
5.
Date:                Thu, 14 Nov 2024   Prob (F-statistic):          7.32e-21
8
Time:                        16:19:53   Log-Likelihood:                -4529.
1
No. Observations:                 720   AIC:                             906
2.
Df Residuals:                     718   BIC:                             907
1.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
-
const       2337.2280      9.277    251.949      0.000    2319.016    2355.44
1
Log_Volume   1.47e-08   3.17e-10     46.318      0.000    1.41e-08    1.53e-0
8
==============================================================================
=
Omnibus:                       16.560   Durbin-Watson:                   0.04
3
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               17.12
3
Skew:                           0.355   Prob(JB):                     0.00019
1
Kurtosis:                       3.260   Cond. No.                     5.57e+1
0
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 5.57e+10. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

Demand Curve Plot

In [7]:
```
#plt.scatter(X['Log_Volume'],y)
y_model= (2336+248.740)-1.469e-08*X['Log_Volume']

#y_model= 4.8210 + 0.1293*X['Log_Volume']
fig=plt.plot(X['Log_Volume'],y_model,lw=4,c='blue',label="OLS Regression")
plt.title("Demand Curve Plot of Ethereum Token")
plt.xlabel("Volume - Qty Demanded")
plt.ylabel("Price")
plt.show()
```