*# This Python 3 environment comes with many helpful analytics libraries installed*

*# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python*

*# For example, here's several helpful packages to load*

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

*# Input data files are available in the read-only "../input/" directory*

*# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory*

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

*# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"*

*# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session*

/kaggle/input/bitcoin-historical-data/bitstampUSD\_1-min\_data\_2012-01-01\_to\_2021-03-31.csv



**Bitcoin: A Brief Overview**

**1. Introduction**

* Bitcoin is a digital currency created in 2009 by an unknown person or group using the pseudonym Satoshi Nakamoto.

**2. Decentralized Nature**

* Bitcoin operates on a decentralized network called the blockchain, facilitating peer-to-peer transactions without the need for intermediaries like banks.

**3. Limited Supply**

* Bitcoin has a capped supply of 21 million coins, making it deflationary and designed to mimic the scarcity of precious metals like gold.

**4. Volatility and Speculation**

* Bitcoin's price can be highly volatile, attracting investors seeking high returns. It is often considered both a store of value and a speculative asset.

**5. Pseudonymity and Transparency**

* While transactions are pseudonymous, recorded on the blockchain, the technology is transparent, allowing anyone to view transaction history.

**6. Implications**

* Bitcoin represents a decentralized alternative to traditional currencies, challenging the existing financial system and offering potential applications beyond finance.

**This Python script imports necessary libraries for data analysis and visualization, including pandas for data manipulation, matplotlib.pyplot for basic plotting, seaborn for advanced visualization, numpy for numerical operations, and train\_test\_split from sklearn.model\_selection for splitting data into training and testing sets.**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from sklearn.model\_selection import train\_test\_split

**READING THE CSV FILE**

show\_data = pd.read\_csv('../input/bitcoin-historical-data/bitstampUSD\_1-min\_data\_2012-01-01\_to\_2021-03-31.csv')

**show\_data.info() provides a concise overview of the dataset's structure, including the number of rows and columns, data types, memory usage, and presence of missing values. It's a handy tool for quick data inspection and identifying potential issues.**

show\_data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 4857377 entries, 0 to 4857376

Data columns (total 8 columns):

# Column Dtype

--- ------ -----

0 Timestamp int64

1 Open float64

2 High float64

3 Low float64

4 Close float64

5 Volume\_(BTC) float64

6 Volume\_(Currency) float64

7 Weighted\_Price float64

dtypes: float64(7), int64(1)

memory usage: 296.5 MB

**VIEWING THE DATASET**

show\_data

|  | Timestamp | Open | High | Low | Close | Volume\_(BTC) | Volume\_(Currency) | Weighted\_Price |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1325317920 | 4.39 | 4.39 | 4.39 | 4.39 | 0.455581 | 2.000000 | 4.390000 |
| 1 | 1325317980 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | 1325318040 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | 1325318100 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | 1325318160 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 4857372 | 1617148560 | 58714.31 | 58714.31 | 58686.00 | 58686.00 | 1.384487 | 81259.372187 | 58692.753339 |
| 4857373 | 1617148620 | 58683.97 | 58693.43 | 58683.97 | 58685.81 | 7.294848 | 428158.146640 | 58693.226508 |
| 4857374 | 1617148680 | 58693.43 | 58723.84 | 58693.43 | 58723.84 | 1.705682 | 100117.070370 | 58696.198496 |
| 4857375 | 1617148740 | 58742.18 | 58770.38 | 58742.18 | 58760.59 | 0.720415 | 42332.958633 | 58761.866202 |
| 4857376 | 1617148800 | 58767.75 | 58778.18 | 58755.97 | 58778.18 | 2.712831 | 159417.751000 | 58764.349363 |

4857377 rows × 8 columns

**This code generates a line plot of Bitcoin's weighted price over time. It also fills the area between the low and high prices to represent the price range. The plot includes a title, axis labels, a legend, and a grid for reference.**

import pandas as pd

import matplotlib.pyplot as plt

plt.figure(figsize=(14, 7))

plt.plot(show\_data['Timestamp'], show\_data['Weighted\_Price'], color='blue', label='Weighted Price')

plt.fill\_between(show\_data['Timestamp'], show\_data['Low'], show\_data['High'], color='skyblue', alpha=0.3, label='Price Range')

plt.title('Bitcoin Weighted Price Over Time')

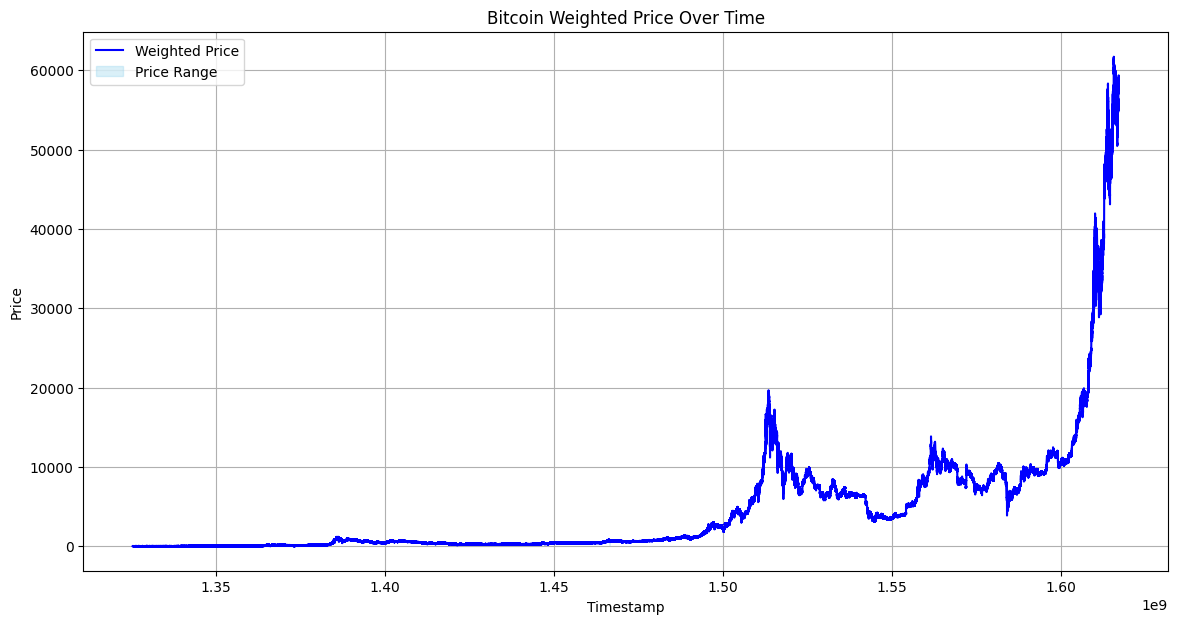
plt.xlabel('Timestamp')

plt.ylabel('Price')

plt.legend()

plt.grid(True)

plt.show()



**This code generates a scatter plot comparing Bitcoin's high and low prices, providing a visual representation of their relationship.**

plt.figure(figsize=(10, 6))

plt.scatter(show\_data['High'], show\_data['Low'], color='orange', alpha=0.1)

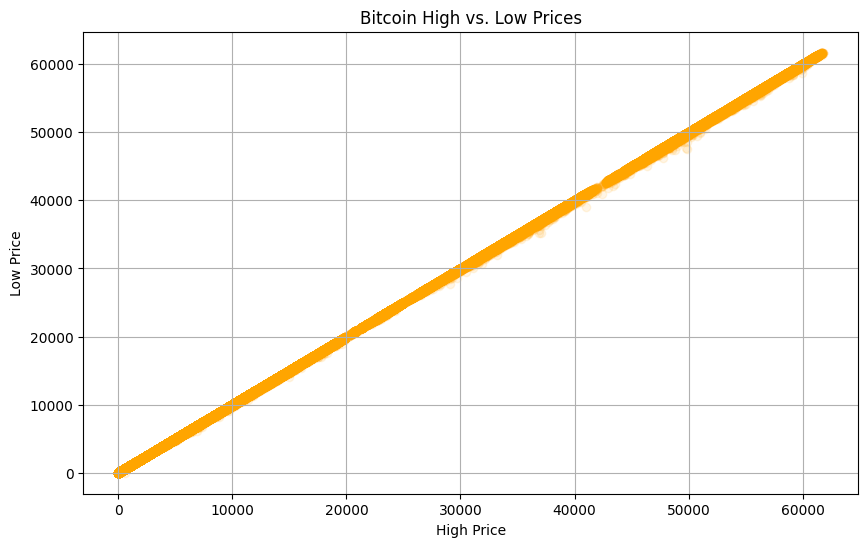
plt.title('Bitcoin High vs. Low Prices')

plt.xlabel('High Price')

plt.ylabel('Low Price')

plt.grid(True)

plt.show()



**Display basic information about the dataset**

print(show\_data.describe())

Timestamp Open High Low Close \

count 4.857377e+06 3.613769e+06 3.613769e+06 3.613769e+06 3.613769e+06

mean 1.471301e+09 6.009024e+03 6.013357e+03 6.004488e+03 6.009014e+03

std 8.428019e+07 8.996247e+03 9.003521e+03 8.988778e+03 8.996360e+03

min 1.325318e+09 3.800000e+00 3.800000e+00 1.500000e+00 1.500000e+00

25% 1.398179e+09 4.438600e+02 4.440000e+02 4.435200e+02 4.438600e+02

50% 1.471428e+09 3.596970e+03 3.598190e+03 3.595620e+03 3.597000e+03

75% 1.544288e+09 8.627270e+03 8.632980e+03 8.621090e+03 8.627160e+03

max 1.617149e+09 6.176356e+04 6.178183e+04 6.167355e+04 6.178180e+04

Volume\_(BTC) Volume\_(Currency) Weighted\_Price

count 3.613769e+06 3.613769e+06 3.613769e+06

mean 9.323249e+00 4.176284e+04 6.008935e+03

std 3.054989e+01 1.518248e+05 8.995992e+03

min 0.000000e+00 0.000000e+00 3.800000e+00

25% 4.097759e-01 4.521422e+02 4.438306e+02

50% 1.979811e+00 3.810124e+03 3.596804e+03

75% 7.278216e+00 2.569821e+04 8.627637e+03

max 5.853852e+03 1.390067e+07 6.171621e+04

**Calculate the number of missing values in each column**

missing\_values = show\_data.isnull().sum()

print("Missing Values:")

print(missing\_values)

Missing Values:

Timestamp 0

Open 1243608

High 1243608

Low 1243608

Close 1243608

Volume\_(BTC) 1243608

Volume\_(Currency) 1243608

Weighted\_Price 1243608

dtype: int64

**Calculate the number of unique values in each column**

unique\_values = show\_data.nunique()

print("Unique Values:")

print(unique\_values)

Unique Values:

Timestamp 4857377

Open 1059732

High 1030826

Low 1046536

Close 1058685

Volume\_(BTC) 3181865

Volume\_(Currency) 3562603

Weighted\_Price 3046022

dtype: int64

**Calculate the correlation matrix**

correlation\_matrix = show\_data.corr()

print("Correlation Matrix:")

print(correlation\_matrix)

Correlation Matrix:

Timestamp Open High Low Close \

Timestamp 1.000000 0.650719 0.650644 0.650803 0.650714

Open 0.650719 1.000000 0.999999 0.999999 0.999999

High 0.650644 0.999999 1.000000 0.999998 0.999999

Low 0.650803 0.999999 0.999998 1.000000 0.999999

Close 0.650714 0.999999 0.999999 0.999999 1.000000

Volume\_(BTC) -0.103926 -0.051860 -0.051717 -0.052040 -0.051877

Volume\_(Currency) 0.228190 0.344074 0.344498 0.343541 0.344036

Weighted\_Price 0.650737 0.999999 0.999999 1.000000 1.000000

Volume\_(BTC) Volume\_(Currency) Weighted\_Price

Timestamp -0.103926 0.228190 0.650737

Open -0.051860 0.344074 0.999999

High -0.051717 0.344498 0.999999

Low -0.052040 0.343541 1.000000

Close -0.051877 0.344036 1.000000

Volume\_(BTC) 1.000000 0.352038 -0.051887

Volume\_(Currency) 0.352038 1.000000 0.344010

Weighted\_Price -0.051887 0.344010 1.000000

**# Fill missing values with the mean of each column**

show\_data\_filled = show\_data.fillna(show\_data.mean())

**Display the first few rows of the filled DataFrame**

print("DataFrame after filling missing values:")

print(show\_data\_filled.head())

DataFrame after filling missing values:

Timestamp Open High Low Close \

0 1325317920 4.39000 4.390000 4.390000 4.390000

1 1325317980 6009.02368 6013.357082 6004.488004 6009.013545

2 1325318040 6009.02368 6013.357082 6004.488004 6009.013545

3 1325318100 6009.02368 6013.357082 6004.488004 6009.013545

4 1325318160 6009.02368 6013.357082 6004.488004 6009.013545

Volume\_(BTC) Volume\_(Currency) Weighted\_Price

0 0.455581 2.000000 4.3900

1 9.323249 41762.842397 6008.9348

2 9.323249 41762.842397 6008.9348

3 9.323249 41762.842397 6008.9348

4 9.323249 41762.842397 6008.9348

**This code will generate a heatmap displaying the correlation coefficients between different numerical columns in the show\_data DataFrame. Adjust the figsize, colormap (cmap), and other parameters as needed to customize the appearance of the heatmap.**

linkcode

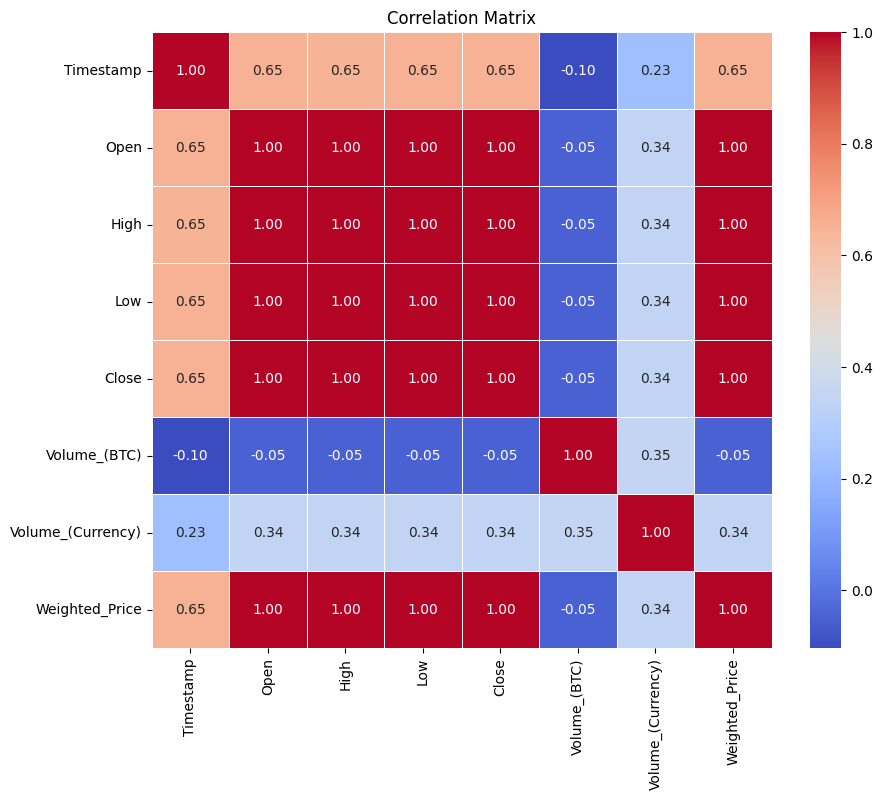
correlation\_matrix = show\_data.corr()

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

plt.title('Correlation Matrix')

plt.show()



**BITCOIN LOGARITHMIC GROWTH RATE**

linkcode

