

Bitcoin Price Prediction

```
In [2]: #yfinance is a popular Python library used for downloading historical market data from Yahoo Finance.  
#It simplifies the process of accessing financial data for various securities, including stocks, commodities, cryptocurrencies, and more
```

```
pip install yfinance
```

```
Requirement already satisfied: yfinance in c:\users\jan saida\anaconda3\lib\site-packages (0.2.52)  
Requirement already satisfied: pandas>=1.3.0 in c:\users\jan saida\anaconda3\lib\site-packages (from yfinance) (2.2.2)  
Requirement already satisfied: numpy>=1.16.5 in c:\users\jan saida\anaconda3\lib\site-packages (from yfinance) (1.26.4)  
Requirement already satisfied: requests>=2.31 in c:\users\jan saida\anaconda3\lib\site-packages (from yfinance) (2.32.2)  
Requirement already satisfied: multitasking>=0.0.7 in c:\users\jan saida\anaconda3\lib\site-packages (from yfinance) (0.0.11)  
Requirement already satisfied: lxml>=4.9.1 in c:\users\jan saida\anaconda3\lib\site-packages (from yfinance) (5.2.1)  
Requirement already satisfied: platformdirs>=2.0.0 in c:\users\jan saida\appdata\roaming\python\python312\site-packages (from yfinance) (4.3.6)  
Requirement already satisfied: pytz>=2022.5 in c:\users\jan saida\anaconda3\lib\site-packages (from yfinance) (2024.1)  
Requirement already satisfied: frozendict>=2.3.4 in c:\users\jan saida\anaconda3\lib\site-packages (from yfinance) (2.4.2)  
Requirement already satisfied: peewee>=3.16.2 in c:\users\jan saida\anaconda3\lib\site-packages (from yfinance) (3.17.8)  
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\users\jan saida\anaconda3\lib\site-packages (from yfinance) (4.12.3)  
Requirement already satisfied: html5lib>=1.1 in c:\users\jan saida\anaconda3\lib\site-packages (from yfinance) (1.1)  
Requirement already satisfied: soupsieve>1.2 in c:\users\jan saida\anaconda3\lib\site-packages (from beautifulsoup4>=4.11.1->yfinance) (2.5)  
Requirement already satisfied: six>=1.9 in c:\users\jan saida\appdata\roaming\python\python312\site-packages (from html5lib>=1.1->yfinance) (1.16.0)  
Requirement already satisfied: webencodings in c:\users\jan saida\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance) (0.5.1)  
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\jan saida\appdata\roaming\python\python312\site-packages (from pandas>=1.3.0->yfinance) (2.9.0.post0)  
Requirement already satisfied: tzdata>=2022.7 in c:\users\jan saida\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance) (2023.3)  
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\jan saida\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2.0.4)  
Requirement already satisfied: idna<4,>=2.5 in c:\users\jan saida\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (3.7)  
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\jan saida\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2.2.2)  
Requirement already satisfied: certifi>=2017.4.17 in c:\users\jan saida\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2024.7.4)  
Note: you may need to restart the kernel to use updated packages.
```

```
In [3]: import seaborn as sns  
import yfinance as yf  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.model_selection import train_test_split  
from sklearn.ensemble import RandomForestRegressor
```

```
In [4]: #The code fetches historical price data for Bitcoin, Ethereum, Tether, and Binance Coin for the past 5 years and keeps only the Close and Volume columns for each of these cryptocurrencies.  
#This cleaned data can then be used for further analysis or machine Learning tasks, such as predicting future prices.
```

```
btc=yf.Ticker('BTC-USD')  
prices1=btc.history(period='5y')  
prices1.drop(columns=['Open', 'High', 'Dividends', 'Stock Splits'], axis = 1, inplace = True)  
  
eth = yf.Ticker('ETH-USD')  
prices2 = eth.history(period='5y')  
prices2.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits'], axis = 1, inplace = True)  
  
usdt = yf.Ticker('USDT-USD')  
prices3 = usdt.history(period='5y')  
prices3.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits'], axis = 1, inplace = True)  
  
bnb = yf.Ticker('BNB-USD')
```

```
prices4 = bnb.history(period='5y')
prices4.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits'], axis = 1, inplace = True)
```

In [5]: *#The parameters lsuffix and rsuffix in the join method are used to add suffixes to overlapping column names when joining two DataFrames*
This is necessary to avoid column name conflicts when the two DataFrames have columns with the same name.

```
p1 = prices1.join(prices2, lsuffix = ' (BTC)', rsuffix = ' (ETH)')
p2 = prices3.join(prices4, lsuffix = ' (USDT)', rsuffix = ' (BNB)')
data = p1.join(p2, lsuffix = '_', rsuffix = '_')
```

In [6]: data.head()

Out[6]:

| | Low | Close (BTC) | Volume (BTC) | Close (ETH) | Volume (ETH) | Close (USDT) | Volume (USDT) | Close (BNB) | Volume (BNB) |
|---------------------------|-------------|-------------|--------------|-------------|--------------|--------------|---------------|-------------|--------------|
| Date | | | | | | | | | |
| 2020-02-04 00:00:00+00:00 | 9112.811523 | 9180.962891 | 29893183716 | 189.250595 | 11714191695 | 1.001745 | 36950713371 | 18.177366 | 213533737 |
| 2020-02-05 00:00:00+00:00 | 9163.704102 | 9613.423828 | 35222060874 | 204.230240 | 14865434435 | 1.002838 | 46504757742 | 19.215050 | 301486499 |
| 2020-02-06 00:00:00+00:00 | 9539.818359 | 9729.801758 | 37628823716 | 212.339081 | 16425589683 | 0.997216 | 46549174003 | 20.595263 | 394704716 |
| 2020-02-07 00:00:00+00:00 | 9726.002930 | 9795.943359 | 34522718159 | 222.726059 | 16673443564 | 0.997751 | 42966951015 | 22.062258 | 485529457 |
| 2020-02-08 00:00:00+00:00 | 9678.910156 | 9865.119141 | 35172043762 | 223.146515 | 16741203125 | 0.999534 | 44251729422 | 21.782419 | 376067881 |

In [7]: data.tail()

Out[7]:

| | Low | Close (BTC) | Volume (BTC) | Close (ETH) | Volume (ETH) | Close (USDT) | Volume (USDT) | Close (BNB) | Volume (BNB) |
|---------------------------|---------------|---------------|--------------|-------------|--------------|--------------|---------------|-------------|--------------|
| Date | | | | | | | | | |
| 2025-01-31 00:00:00+00:00 | 101543.882812 | 102405.023438 | 45732764360 | 3298.265137 | 30128115902 | 0.999771 | 96233410167 | 677.375366 | 1589542506 |
| 2025-02-01 00:00:00+00:00 | 100297.710938 | 100655.906250 | 27757944848 | 3118.328613 | 19917507079 | 0.999843 | 72792811129 | 653.432922 | 1544346912 |
| 2025-02-02 00:00:00+00:00 | 96216.078125 | 97688.976562 | 63091816853 | 2868.692871 | 42060930305 | 0.999760 | 147745588018 | 617.600647 | 2282982721 |
| 2025-02-03 00:00:00+00:00 | 91242.890625 | 101405.421875 | 115400897748 | 2884.566650 | 92453553253 | 1.001051 | 292831861639 | 617.120789 | 3960962742 |
| 2025-02-04 00:00:00+00:00 | 97963.304688 | 98712.531250 | 68290592768 | 2751.387939 | 46848655360 | 1.000552 | 150930571264 | 574.938049 | 2249767936 |

In [8]: data.shape

Out[8]: (1828, 9)

In [9]: data.info()

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1828 entries, 2020-02-04 00:00:00+00:00 to 2025-02-04 00:00:00+00:00
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  -
 0   Low                  1828 non-null   float64
 1   Close (BTC)          1828 non-null   float64
 2   Volume (BTC)         1828 non-null   int64  
 3   Close (ETH)          1828 non-null   float64
 4   Volume (ETH)         1828 non-null   int64  
 5   Close (USDT)         1828 non-null   float64
 6   Volume (USDT)        1828 non-null   int64  
 7   Close (BNB)          1828 non-null   float64
 8   Volume (BNB)         1828 non-null   int64  
dtypes: float64(5), int64(4)
memory usage: 142.8 KB
```

```
In [10]: data.isna().sum()
```

```
Out[10]: Low                0
Close (BTC)              0
Volume (BTC)             0
Close (ETH)              0
Volume (ETH)             0
Close (USDT)             0
Volume (USDT)            0
Close (BNB)              0
Volume (BNB)            0
dtype: int64
```

```
In [11]: data.describe()
```

```
Out[11]:
```

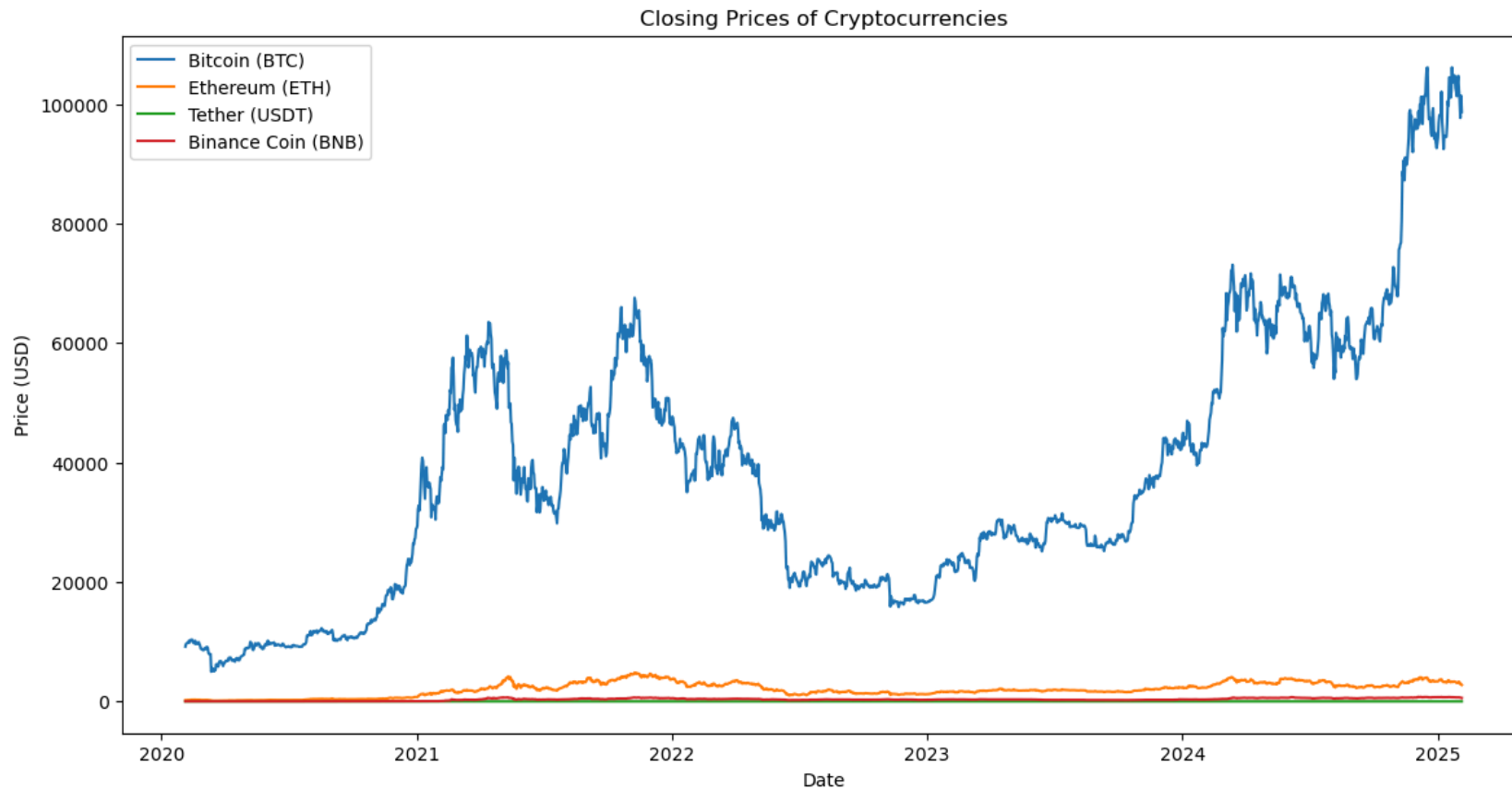
| | Low | Close (BTC) | Volume (BTC) | Close (ETH) | Volume (ETH) | Close (USDT) | Volume (USDT) | Close (BNB) | Volume (BNB) |
|-------|---------------|---------------|--------------|-------------|--------------|--------------|---------------|-------------|--------------|
| count | 1828.000000 | 1828.000000 | 1.828000e+03 | 1828.000000 | 1.828000e+03 | 1828.000000 | 1.828000e+03 | 1828.000000 | 1.828000e+03 |
| mean | 37141.454323 | 38053.590003 | 3.370915e+10 | 2041.098250 | 1.704349e+10 | 1.000366 | 6.083020e+10 | 319.393482 | 1.457044e+09 |
| std | 21963.516753 | 22498.661541 | 2.004917e+10 | 1132.951125 | 1.084360e+10 | 0.002095 | 3.966199e+10 | 198.762431 | 1.389998e+09 |
| min | 4106.980957 | 4970.788086 | 5.331173e+09 | 110.605873 | 2.081626e+09 | 0.974248 | 9.989859e+09 | 9.386050 | 1.365992e+08 |
| 25% | 19934.478027 | 20356.477051 | 2.064499e+10 | 1299.825958 | 9.650473e+09 | 0.999912 | 3.591407e+10 | 219.296188 | 4.799710e+08 |
| 50% | 33150.035156 | 34254.855469 | 3.021149e+10 | 1892.970093 | 1.488192e+10 | 1.000183 | 5.163867e+10 | 305.339066 | 1.144258e+09 |
| 75% | 52501.022461 | 53999.031250 | 4.097135e+10 | 2971.376770 | 2.107294e+10 | 1.000553 | 7.278883e+10 | 481.755943 | 1.917543e+09 |
| max | 105291.734375 | 106146.265625 | 3.509679e+11 | 4812.087402 | 9.245355e+10 | 1.053585 | 3.006686e+11 | 750.272644 | 1.798295e+10 |

Exploratory Data Analysis

```
In [13]: #Visualize the Closing Prices
# create a line plot to visualize the closing prices of all four cryptocurrencies over time:

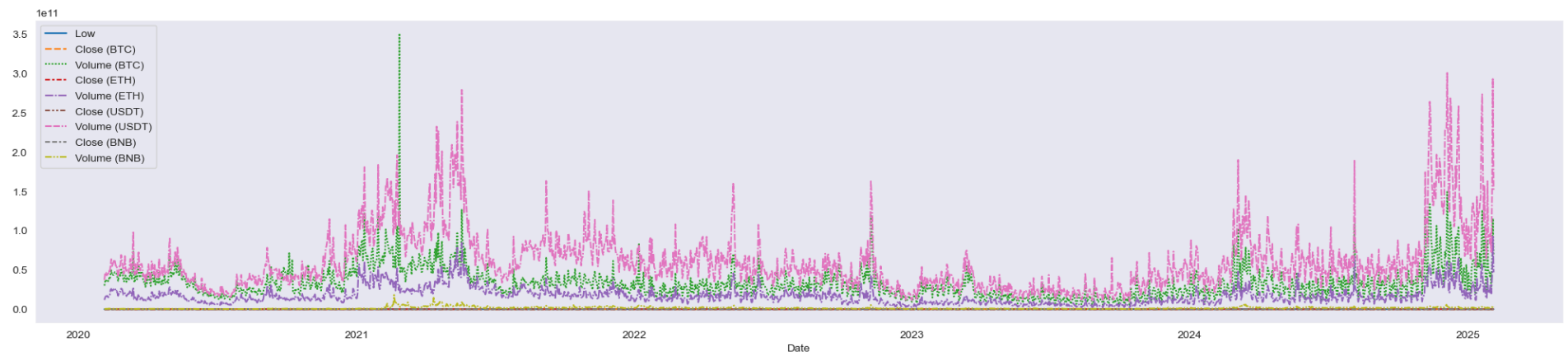
plt.figure(figsize=(14, 7))
plt.plot(data.index, data['Close (BTC)'], label='Bitcoin (BTC)')
```

```
plt.plot(data.index, data['Close (ETH)'], label='Ethereum (ETH)')
plt.plot(data.index, data['Close (USDT)'], label='Tether (USDT)')
plt.plot(data.index, data['Close (BNB)'], label='Binance Coin (BNB)')
plt.title('Closing Prices of Cryptocurrencies')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.show()
```



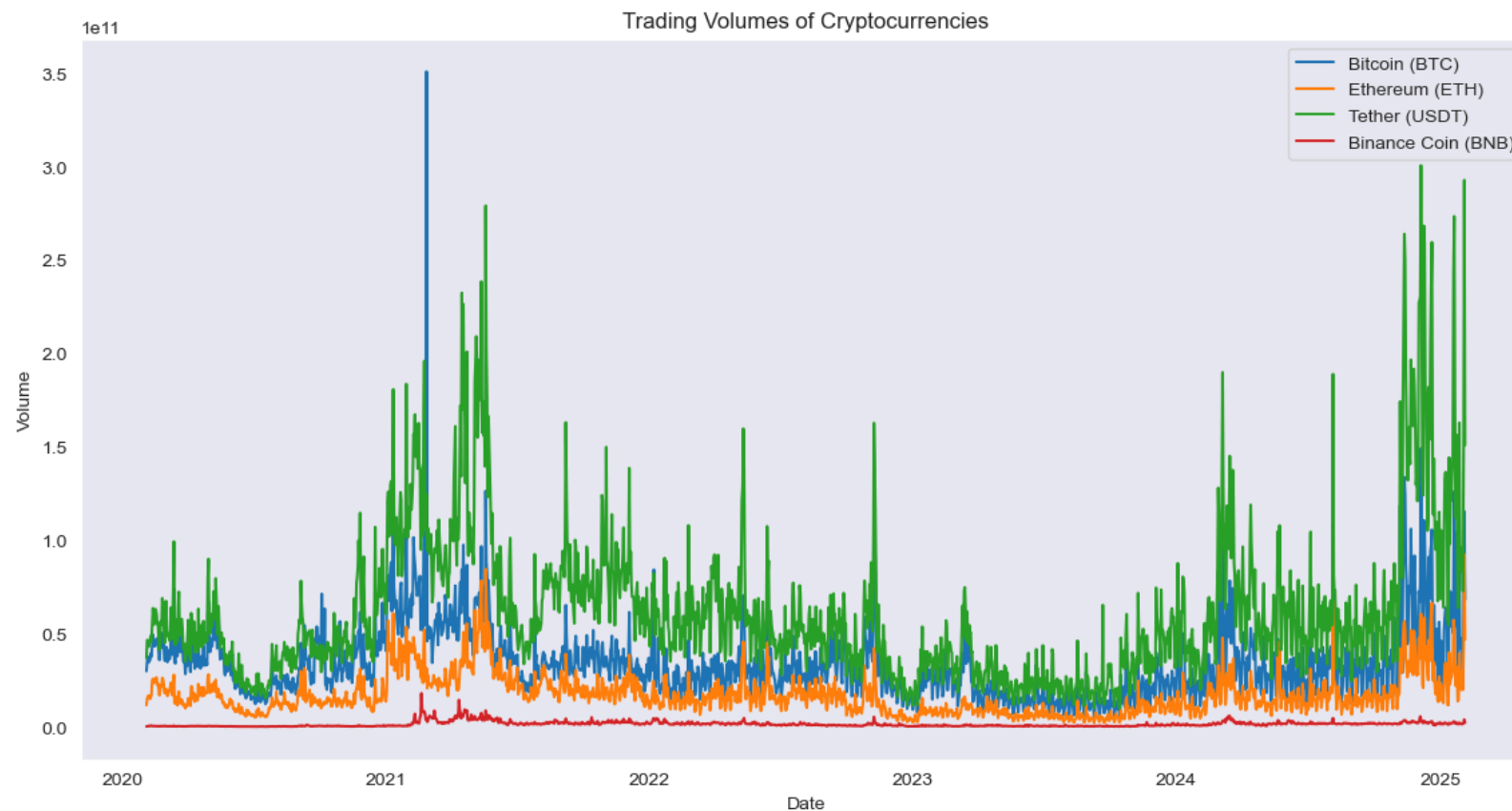
```
In [14]: plt.figure(figsize = (25, 5))
sns.set_style('dark')
sns.lineplot(data=data)
```

```
Out[14]: <Axes: xlabel='Date'>
```



```
In [15]: # Visualize the Trading Volumes
#Let's visualize the trading volumes of all four cryptocurrencies:

plt.figure(figsize=(14, 7))
plt.plot(data.index, data['Volume (BTC)'], label='Bitcoin (BTC)')
plt.plot(data.index, data['Volume (ETH)'], label='Ethereum (ETH)')
plt.plot(data.index, data['Volume (USDT)'], label='Tether (USDT)')
plt.plot(data.index, data['Volume (BNB)'], label='Binance Coin (BNB)')
plt.title('Trading Volumes of Cryptocurrencies')
plt.xlabel('Date')
plt.ylabel('Volume')
plt.legend()
plt.show()
```

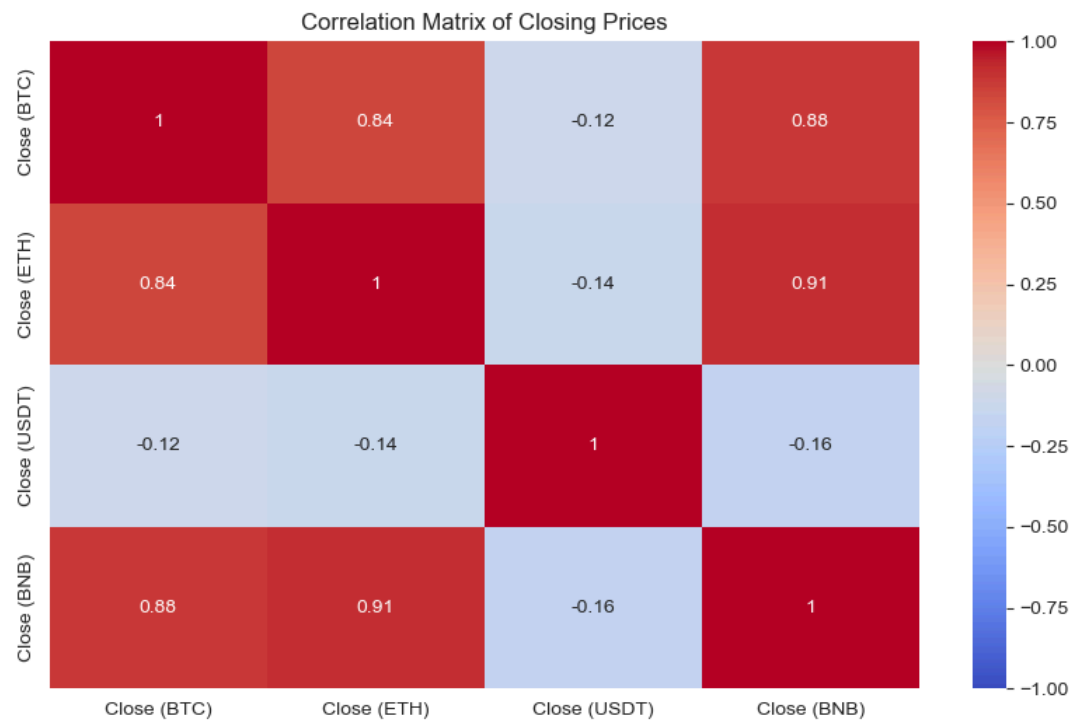


```
In [16]: #Correlation Analysis
#We'll analyze the correlation between the closing prices of the cryptocurrencies:
# Calculate the correlation matrix

corr_matrix = data[['Close (BTC)', 'Close (ETH)', 'Close (USDT)', 'Close (BNB)']].corr()

# Plot the heatmap

plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix of Closing Prices')
plt.show()
```



In [17]: *# Distribution of Closing Prices*
#Let's plot the distribution of closing prices for each cryptocurrency:

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

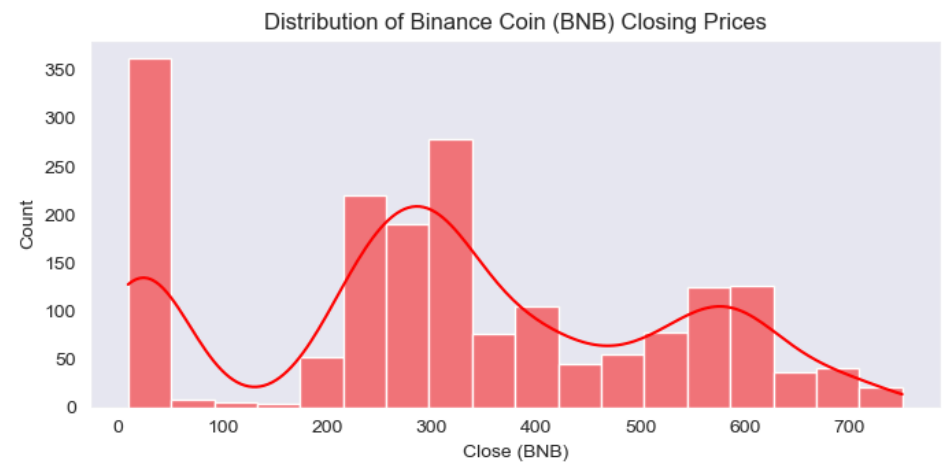
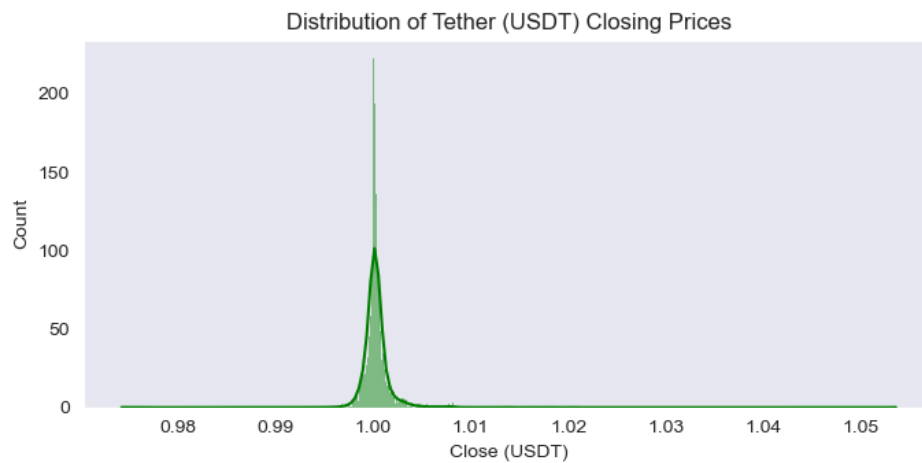
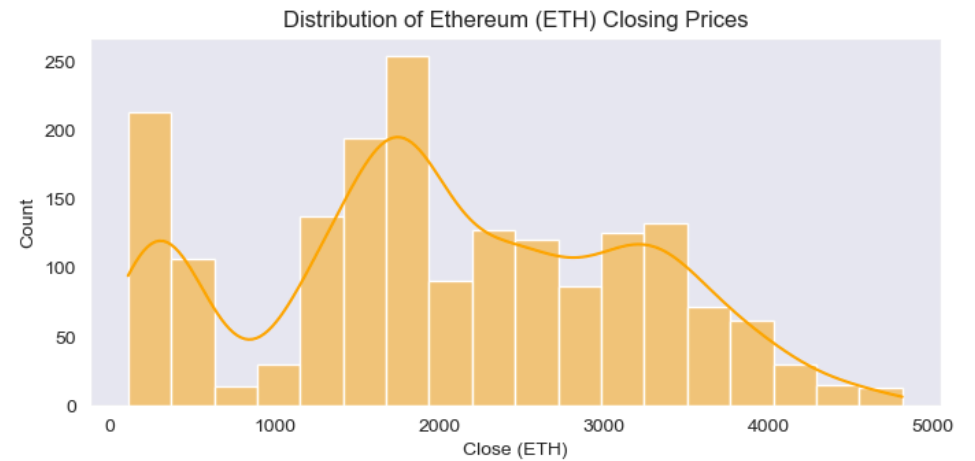
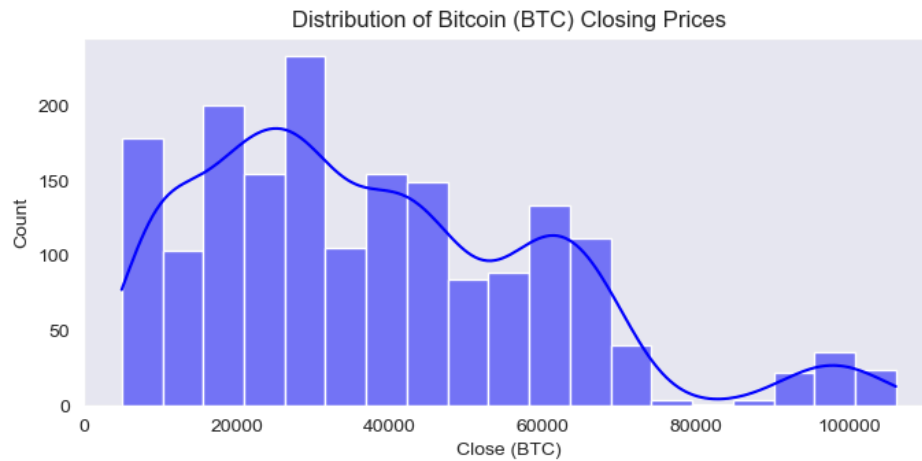
plt.subplot(2, 2, 1)
sns.histplot(data['Close (BTC)'], kde=True, color='blue')
plt.title('Distribution of Bitcoin (BTC) Closing Prices')

plt.subplot(2, 2, 2)
sns.histplot(data['Close (ETH)'], kde=True, color='orange')
plt.title('Distribution of Ethereum (ETH) Closing Prices')

plt.subplot(2, 2, 3)
sns.histplot(data['Close (USDT)'], kde=True, color='green')
plt.title('Distribution of Tether (USDT) Closing Prices')

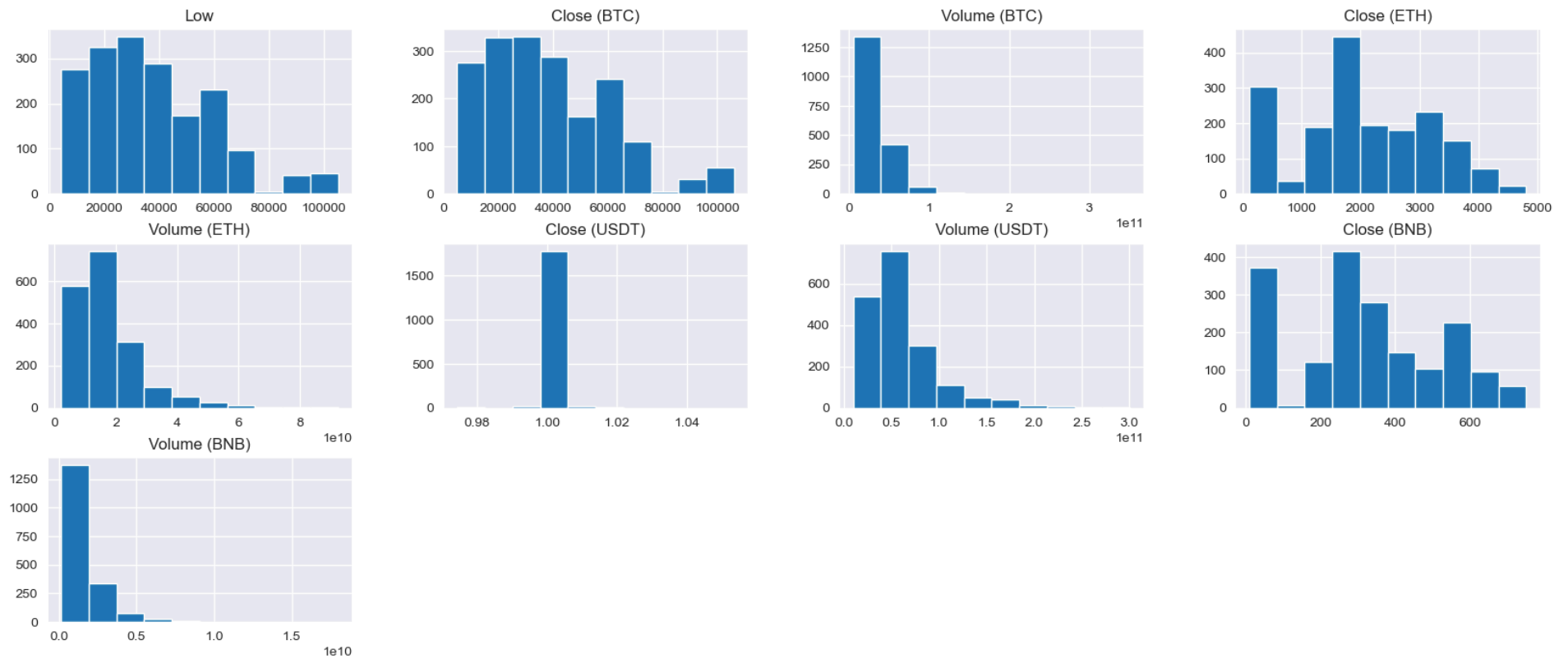
plt.subplot(2, 2, 4)
sns.histplot(data['Close (BNB)'], kde=True, color='red')
plt.title('Distribution of Binance Coin (BNB) Closing Prices')

plt.tight_layout()
plt.show()
```



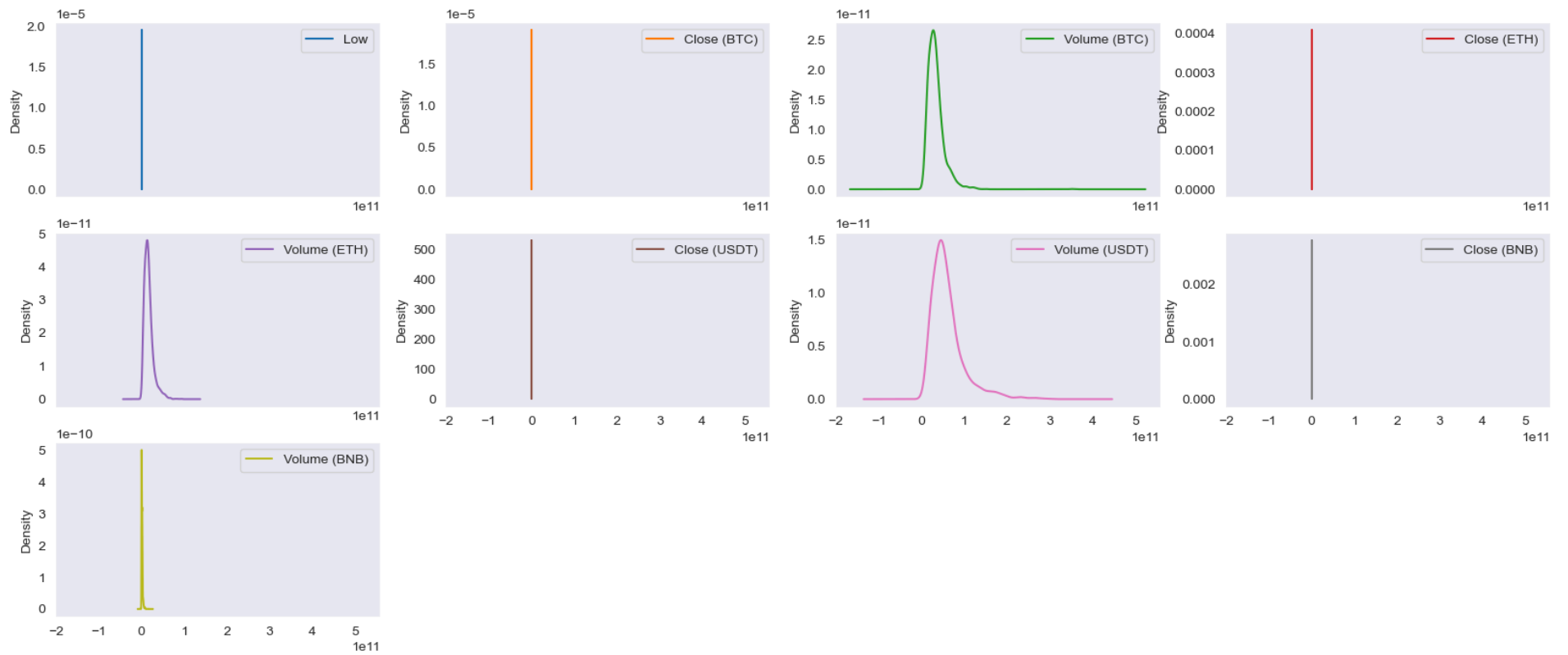
```
In [18]: data.hist(figsize=(20, 8), layout=(3, 4))
```

```
Out[18]: array([[<Axes: title={'center': 'Low'},>,
  <Axes: title={'center': 'Close (BTC)'>,
  <Axes: title={'center': 'Volume (BTC)'>,
  <Axes: title={'center': 'Close (ETH)'>]],
[<Axes: title={'center': 'Volume (ETH)'>,
  <Axes: title={'center': 'Close (USDT)'>,
  <Axes: title={'center': 'Volume (USDT)'>,
  <Axes: title={'center': 'Close (BNB)'>]],
[<Axes: title={'center': 'Volume (BNB)'>, <Axes: >, <Axes: >,
  <Axes: >]], dtype=object)
```

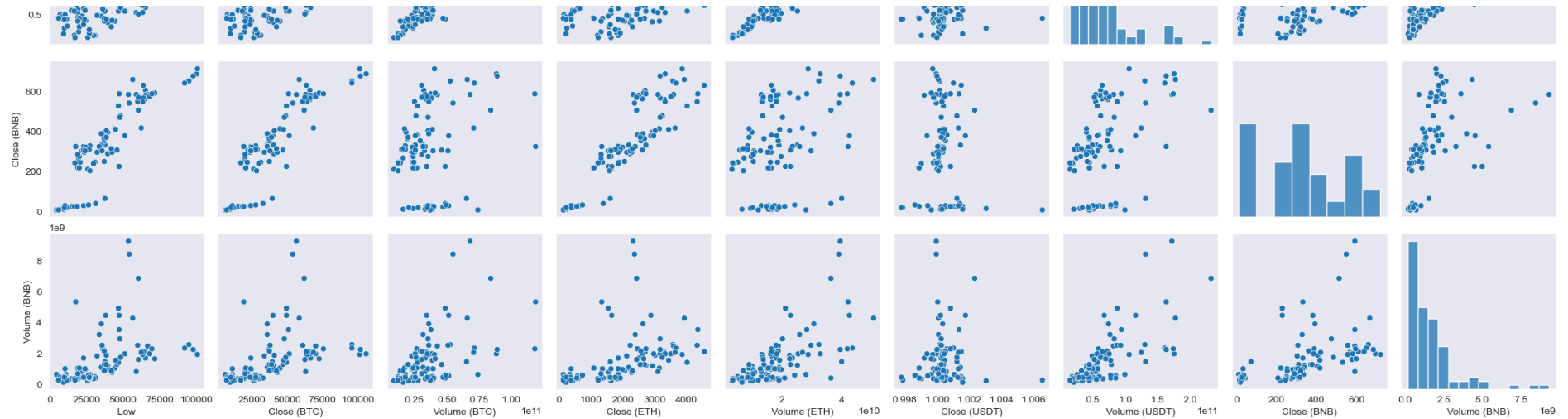
```
In [19]: data.plot(kind = "kde", subplots = True, layout = (3, 4), figsize = (20, 8))
```

```
Out[19]: array([[<Axes: ylabel='Density'>, <Axes: ylabel='Density'>,
  <Axes: ylabel='Density'>, <Axes: ylabel='Density'>],
  [<Axes: ylabel='Density'>, <Axes: ylabel='Density'>,
  <Axes: ylabel='Density'>, <Axes: ylabel='Density'>],
  [<Axes: ylabel='Density'>, <Axes: ylabel='Density'>,
  <Axes: ylabel='Density'>, <Axes: ylabel='Density'>]], dtype=object)
```



```
In [20]: sns.pairplot(data.sample(n=100));
```





Data Pre-processing

```
In [22]: X = data.drop(columns = ['Close (BTC)'], axis = 1)
Y = data.loc[:, 'Close (BTC)']
```

```
In [23]: X.head()
```

```
Out[23]:
```

| | Low | Volume (BTC) | Close (ETH) | Volume (ETH) | Close (USDT) | Volume (USDT) | Close (BNB) | Volume (BNB) |
|----------------------------------|-------------|--------------|-------------|--------------|--------------|---------------|-------------|--------------|
| 2020-02-04 00:00:00+00:00 | 9112.811523 | 29893183716 | 189.250595 | 11714191695 | 1.001745 | 36950713371 | 18.177366 | 213533737 |
| 2020-02-05 00:00:00+00:00 | 9163.704102 | 35222060874 | 204.230240 | 14865434435 | 1.002838 | 46504757742 | 19.215050 | 301486499 |
| 2020-02-06 00:00:00+00:00 | 9539.818359 | 37628823716 | 212.339081 | 16425589683 | 0.997216 | 46549174003 | 20.595263 | 394704716 |
| 2020-02-07 00:00:00+00:00 | 9726.002930 | 34522718159 | 222.726059 | 16673443564 | 0.997751 | 42966951015 | 22.062258 | 485529457 |
| 2020-02-08 00:00:00+00:00 | 9678.910156 | 35172043762 | 223.146515 | 16741203125 | 0.999534 | 44251729422 | 21.782419 | 376067881 |

```
In [24]: X.tail()
```

Out[24]:

| | Low | Volume (BTC) | Close (ETH) | Volume (ETH) | Close (USDT) | Volume (USDT) | Close (BNB) | Volume (BNB) |
|---------------------------|---------------|--------------|-------------|--------------|--------------|---------------|-------------|--------------|
| Date | | | | | | | | |
| 2025-01-31 00:00:00+00:00 | 101543.882812 | 45732764360 | 3298.265137 | 30128115902 | 0.999771 | 96233410167 | 677.375366 | 1589542506 |
| 2025-02-01 00:00:00+00:00 | 100297.710938 | 27757944848 | 3118.328613 | 19917507079 | 0.999843 | 72792811129 | 653.432922 | 1544346912 |
| 2025-02-02 00:00:00+00:00 | 96216.078125 | 63091816853 | 2868.692871 | 42060930305 | 0.999760 | 147745588018 | 617.600647 | 2282982721 |
| 2025-02-03 00:00:00+00:00 | 91242.890625 | 115400897748 | 2884.566650 | 92453553253 | 1.001051 | 292831861639 | 617.120789 | 3960962742 |
| 2025-02-04 00:00:00+00:00 | 97963.304688 | 68290592768 | 2751.387939 | 46848655360 | 1.000552 | 150930571264 | 574.938049 | 2249767936 |

In [25]:

```
Y.head()
```

Out[25]:

```
Date
2020-02-04 00:00:00+00:00    9180.962891
2020-02-05 00:00:00+00:00    9613.423828
2020-02-06 00:00:00+00:00    9729.801758
2020-02-07 00:00:00+00:00    9795.943359
2020-02-08 00:00:00+00:00    9865.119141
Name: Close (BTC), dtype: float64
```

In [26]:

```
Y.tail()
```

Out[26]:

```
Date
2025-01-31 00:00:00+00:00    102405.023438
2025-02-01 00:00:00+00:00    100655.906250
2025-02-02 00:00:00+00:00     97688.976562
2025-02-03 00:00:00+00:00    101405.421875
2025-02-04 00:00:00+00:00     98712.531250
Name: Close (BTC), dtype: float64
```

In [27]:

```
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

In [28]:

```
# Print the shapes of the resulting datasets

print(f'X_train shape: {X_train.shape}')
print(f'X_test shape: {X_test.shape}')
print(f'y_train shape: {Y_train.shape}')
print(f'y_test shape: {Y_test.shape}')
```

```
X_train shape: (1462, 8)
X_test shape: (366, 8)
y_train shape: (1462,)
y_test shape: (366,)
```

In [29]:

```
#SelectKBest
#SelectKBest is a feature selection method provided by scikit-Learn (sklearn) that selects the top k features based on a specified scoring function.
#This function evaluates each feature independently and selects those that have the strongest relationship with the target variable.

#Parameters
#k: Specifies the number of top features to select. In your case, k=4 indicates that you want to select the top 4 features

from sklearn.feature_selection import SelectKBest
```

```
fs = SelectKBest(k=4)
X_train = fs.fit_transform(X_train, Y_train)
X_test = fs.transform(X_test)
```

```
C:\Users\Jan Saida\anaconda3\Lib\site-packages\sklearn\feature_selection\_univariate_selection.py:109: RuntimeWarning: invalid value encountered in divide
msw = sswn / float(dfwn)
```

```
In [30]: mask = fs.get_support()
selected_features = X.columns[mask]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['Close (USDt)', 'Volume (USDt)', 'Close (BNB)', 'Volume (BNB)'], dtype='object')
```

```
In [31]: X_train
```

```
Out[31]: array([[1.00020003e+00, 4.25144241e+10, 2.84312153e+01, 3.60596998e+08],
 [9.99502003e-01, 4.72488257e+10, 2.73994808e+01, 4.82149967e+08],
 [1.00069594e+00, 7.57414313e+10, 4.13456207e+02, 1.47312139e+09],
 ...,
 [1.00029004e+00, 5.21447245e+10, 5.24015320e+02, 1.60440064e+09],
 [1.00043797e+00, 8.44380136e+10, 4.17470856e+02, 2.62057147e+09],
 [9.99988019e-01, 4.97521808e+10, 5.29972046e+02, 1.19958266e+09]])
```

```
In [32]: #MinMaxScaler is a preprocessing method in scikit-Learn that transforms features by scaling them to a specified range.
# It's often used when your data needs to be normalized within a specific range to ensure all features contribute equally to the analysis.
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
In [33]: # implementation of 10 different regression algorithms using scikit-Learn. Each algorithm is trained and evaluated on a sample dataset:
```

```
#Import Libraries and Generate Sample Data

from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

```
In [34]: #Define Models and Perform Training and Evaluation
```

```
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(alpha=1.0),
    'Lasso Regression': Lasso(alpha=1.0),
    'ElasticNet Regression': ElasticNet(alpha=1.0, l1_ratio=0.5),
    'Support Vector Regression (SVR)': SVR(kernel='rbf'),
    'Decision Tree Regression': DecisionTreeRegressor(),
    'Random Forest Regression': RandomForestRegressor(n_estimators=100),
    'Gradient Boosting Regression': GradientBoostingRegressor(n_estimators=100, learning_rate=0.1),
    'K-Nearest Neighbors Regression': KNeighborsRegressor(n_neighbors=5),
    'Neural Network Regression (MLP)': MLPRegressor(hidden_layer_sizes=(100, 50), activation='relu', solver='adam')
}
```

```

# Train and evaluate each model

results = {'Model': [], 'MSE': [], 'R-squared': []}

for name, model in models.items():

    # Train the model

    model.fit(X_train, Y_train)

    # Predict on test set

    Y_pred = model.predict(X_test)

    # Evaluate model

    mse = mean_squared_error(Y_test, Y_pred)
    r2 = r2_score(Y_test, Y_pred)

    # Store results

    results['Model'].append(name)
    results['MSE'].append(mse)
    results['R-squared'].append(r2)

    # Print results

    print(f"----- {name} -----")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"R-squared: {r2}")
    print()

# Convert results to DataFrame for visualization

results_df = pd.DataFrame(results)
print(results_df)

# Plotting the results

plt.figure(figsize=(12, 6))
plt.barh(results_df['Model'], results_df['R-squared'], color='skyblue')
plt.xlabel('R-squared')
plt.title('R-squared of Different Regression Models')
plt.xlim(-1, 1)
plt.gca().invert_yaxis()
plt.show()

```

----- Linear Regression -----
Mean Squared Error (MSE): 91822989.40739869
R-squared: 0.8225492739048273

----- Ridge Regression -----
Mean Squared Error (MSE): 92917900.31610283
R-squared: 0.8204333251972871

----- Lasso Regression -----
Mean Squared Error (MSE): 91802706.85481039
R-squared: 0.8225884705559829

----- ElasticNet Regression -----
Mean Squared Error (MSE): 413553613.0595567
R-squared: 0.20079503629417517

----- Support Vector Regression (SVR) -----
Mean Squared Error (MSE): 514735030.3557013
R-squared: 0.005258863028476557

----- Decision Tree Regression -----
Mean Squared Error (MSE): 83536513.99177702
R-squared: 0.8385631402444195

----- Random Forest Regression -----
Mean Squared Error (MSE): 38421657.53465013
R-squared: 0.9257489755963623

----- Gradient Boosting Regression -----
Mean Squared Error (MSE): 44683115.05804003
R-squared: 0.913648518062684

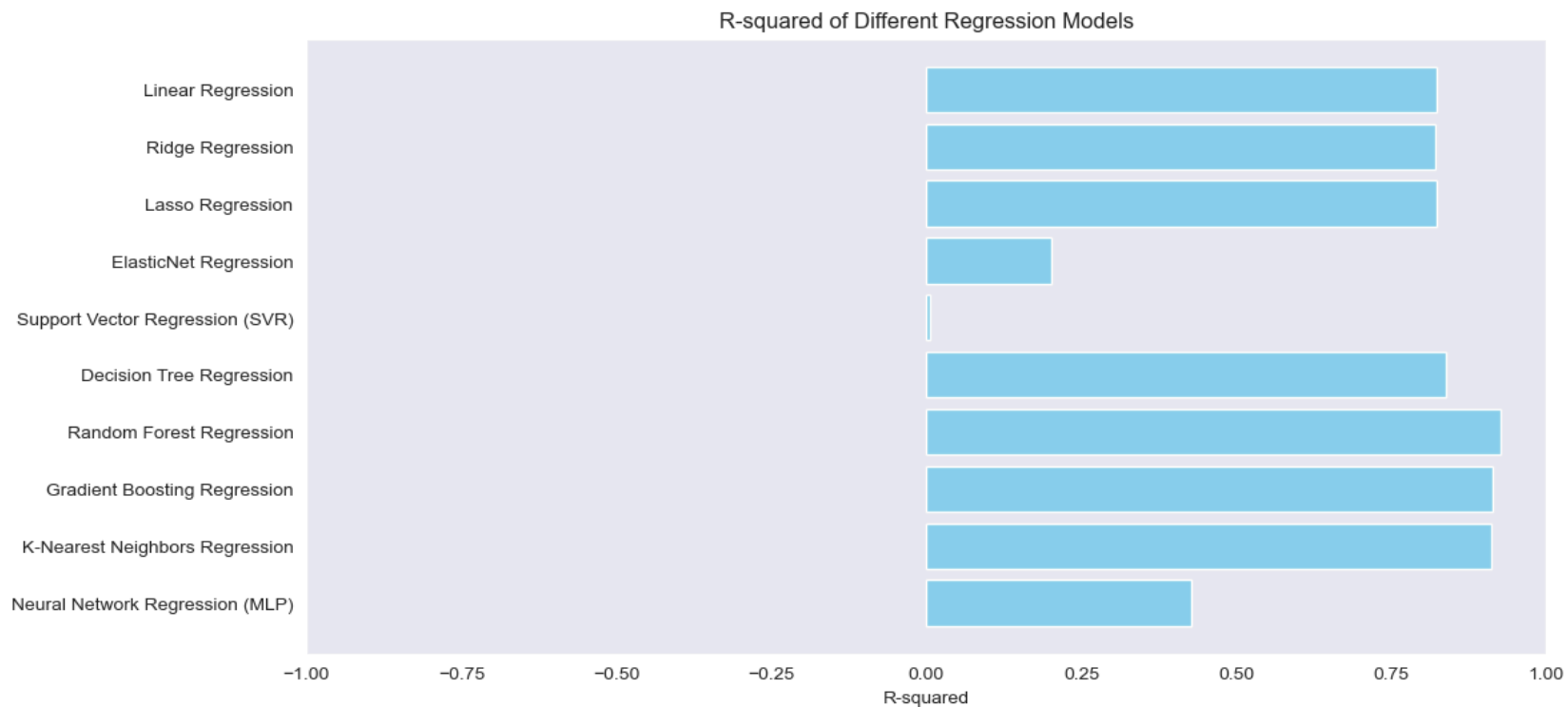
----- K-Nearest Neighbors Regression -----
Mean Squared Error (MSE): 45737035.18400449
R-squared: 0.9116117852923213

C:\Users\Jan Saida\anaconda3\Lib\site-packages\sklearn\normalization\multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.

warnings.warn(

----- Neural Network Regression (MLP) -----
Mean Squared Error (MSE): 296572607.20707595
R-squared: 0.4268643960681855

| | Model | MSE | R-squared |
|---|---------------------------------|--------------|-----------|
| 0 | Linear Regression | 9.182299e+07 | 0.822549 |
| 1 | Ridge Regression | 9.291790e+07 | 0.820433 |
| 2 | Lasso Regression | 9.180271e+07 | 0.822588 |
| 3 | ElasticNet Regression | 4.135536e+08 | 0.200795 |
| 4 | Support Vector Regression (SVR) | 5.147350e+08 | 0.005259 |
| 5 | Decision Tree Regression | 8.353651e+07 | 0.838563 |
| 6 | Random Forest Regression | 3.842166e+07 | 0.925749 |
| 7 | Gradient Boosting Regression | 4.468312e+07 | 0.913649 |
| 8 | K-Nearest Neighbors Regression | 4.573704e+07 | 0.911612 |
| 9 | Neural Network Regression (MLP) | 2.965726e+08 | 0.426864 |



Random Forest Regression is a powerful and versatile algorithm suitable for various regression tasks, offering robust performance and the ability to handle complex data relationships

Saving the model

```
In [37]: import pickle
import numpy as np
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score

# Generate sample data

X, Y = make_regression(n_samples=1000, n_features=10, noise=0.1, random_state=0)

# Scale the features (optional but recommended for some algorithms)
```

```

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize Random Forest Regressor

model_rf = RandomForestRegressor(n_estimators=100, random_state=0)

# Train the model

model_rf.fit(X_train, Y_train)

# Save the model to a file

filename = 'random_forest_model.pkl'
pickle.dump(model_rf, open(filename, 'wb'))

# Save scaler to a file

with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)

# Load the model from the file

loaded_model = pickle.load(open(filename, 'rb'))

# Predict using the loaded model

Y_pred = loaded_model.predict(X_test)

# Evaluate the loaded model

mse = mean_squared_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

print(f"Loaded Random Forest Regression - Mean Squared Error (MSE): {mse}")
print(f"Loaded Random Forest Regression - R-squared: {r2}")

```

Loaded Random Forest Regression - Mean Squared Error (MSE): 38086378.67246037
 Loaded Random Forest Regression - R-squared: 0.9263969122179415

In [71]: `import os`
`os.getcwd()`

Out[71]: 'C:\\Users\\Jan Saida'

In []:

Streamlit\BitCoin-Price-Pred_app.py

```
1 import streamlit as st
2 import pandas as pd
3 import pickle
4 from sklearn.ensemble import RandomForestRegressor
5
6 # Load the trained model
7 model_path = 'C:\\Users\\Jan Saida\\random_forest_model.pkl'
8 with open(model_path, 'rb') as file:
9     model_rf = pickle.load(file)
10
11 # Function to predict based on user inputs
12 def predict_btc_price(input_data):
13     # Make prediction using the model
14     prediction = model_rf.predict(input_data)
15     return prediction[0] # Assuming model returns a single prediction
16
17 def main():
18     # Title of your web app
19     st.title('Predict BTC Close Price')
20
21     # Sidebar for user inputs
22     st.sidebar.title('Input Features')
23
24     # Inputs for USDT, BNB closing prices and volumes
25     usdt_close = st.sidebar.number_input('USDT Close Price', min_value=0.0, format="%.2f")
26     usdt_volume = st.sidebar.number_input('USDT Volume', min_value=0.0, format="%.2f")
27     bnb_close = st.sidebar.number_input('BNB Close Price', min_value=0.0, format="%.2f")
28     bnb_volume = st.sidebar.number_input('BNB Volume', min_value=0.0, format="%.2f")
29
30     # Create input dataframe
31     input_data = pd.DataFrame({
32         'USDT_Close': [usdt_close],
33         'USDT_Volume': [usdt_volume],
34         'BNB_Close': [bnb_close],
35         'BNB_Volume': [bnb_volume]
36     })
37
38     # Button to trigger prediction
39     if st.button('Predict BTC Close Price'):
40         predicted_price = predict_btc_price(input_data)
41         st.write('Predicted BTC Close Price:', predicted_price)
42
43 if __name__ == '__main__':
44     main()
45
```

Input Features

USDT Close Price

0.10 - +

USDT Volume

0.10 - +

BNB Close Price

0.10 - +

BNB Volume

0.10 - +

Predict BTC Close Price

Predict BTC Close Price

Predicted BTC Close Price: 21623.362021484376

Input Features

USDT Close Price

0.13

-

+

USDT Volume

0.15

-

+

BNB Close Price

0.18

-

+

BNB Volume

0.21

-

+

Predict BTC Close Price

Predict BTC Close Price

Predicted BTC Close Price: 23999.289697265624