

```
In [4]: pip install yfinance
```

```
Requirement already satisfied: yfinance in c:\users\santo\anaconda3\lib\site-packages (1.0)
Requirement already satisfied: pandas>=1.3.0 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (2.2.3)
Requirement already satisfied: numpy>=1.16.5 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (2.1.3)
Requirement already satisfied: requests>=2.31 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (0.0.12)
Requirement already satisfied: platformdirs>=2.0.0 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (4.3.7)
Requirement already satisfied: pytz>=2022.5 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (2024.1)
Requirement already satisfied: frozendict>=2.3.4 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (2.4.2)
Requirement already satisfied: peewee>=3.16.2 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (3.18.3)
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (4.12.3)
Requirement already satisfied: curl_cffi<0.14,>=0.7 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (0.13.0)
Requirement already satisfied: protobuf>=3.19.0 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (5.29.3)
Requirement already satisfied: websockets>=13.0 in c:\users\santo\anaconda3\lib\site-packages (from yfinance) (15.0.1)
Requirement already satisfied: cffi>=1.12.0 in c:\users\santo\anaconda3\lib\site-packages (from curl_cffi<0.14,>=0.7->yfinance) (1.17.1)
Requirement already satisfied: certifi>=2024.2.2 in c:\users\santo\anaconda3\lib\site-packages (from curl_cffi<0.14,>=0.7->yfinance) (2025.11.12)
Requirement already satisfied: soupsieve>1.2 in c:\users\santo\anaconda3\lib\site-packages (from beautifulsoup4>=4.11.1->yfinance) (2.5)
Requirement already satisfied: pycparser in c:\users\santo\anaconda3\lib\site-packages (from cffi>=1.12.0->curl_cffi<0.14,>=0.7->yfinance) (2.21)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\santo\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in c:\users\santo\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\santo\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas>=1.3.0->yfinance) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\santo\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\santo\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\santo\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2.3.0)
Note: you may need to restart the kernel to use updated packages.
```

```
In [7]: 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 [8]: btc = yf.Ticker('BTC-USD')
prices1 = btc.history(period='5y')
prices1.drop(columns=['Open', 'High', 'Low', '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 [9]: p1 = prices1.join(prices2, lsuffix = ' (BTC)', rsuffix = ' (ETH)')
p2 = prices3.join(prices4, lsuffix = ' (USDT)', rsuffix = ' (BNB)')
data = p1.join(p2, lsuffix = '_', rsuffix = '_')
```

```
In [10]: data.head()
```

Out[10]:

	Close (BTC)	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Volume (USDT)	Close (BNB)	Volume (BNB)
Date								
2021-01-03 00:00:00+00:00	32782.023438	78665235202	975.507690	45200463368	1.000514	120425679796	41.148979	758008613
2021-01-04 00:00:00+00:00	31971.914062	81163475344	1040.233032	56945985763	1.000128	125906387011	40.926353	807877171
2021-01-05 00:00:00+00:00	33992.429688	67547324782	1100.006104	41535932781	1.002202	101918715244	41.734600	644270927
2021-01-06 00:00:00+00:00	36824.363281	75289433811	1207.112183	44699914188	1.001528	116105139289	42.165955	641021601
2021-01-07 00:00:00+00:00	39371.042969	84762141031	1225.678101	40468027280	1.000400	129467601516	43.449490	829964770

In [11]: data.tail()

Out[11]:

	Close (BTC)	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Volume (USDT)	Close (BNB)	Volume (BNB)
Date								
2025-12-30 00:00:00+00:00	88430.132812	35586356225	2971.416748	18816704381	0.998867	74119035312	860.553711	2042633098
2025-12-31 00:00:00+00:00	87508.828125	33830210616	2967.037598	16451891101	0.998449	70259461189	863.257385	2539874072
2026-01-01 00:00:00+00:00	88731.984375	18849043990	3000.394287	10268796662	0.998745	50548666268	863.054626	1623168589
2026-01-02 00:00:00+00:00	89944.695312	46398906171	3124.422607	25242778003	0.999672	96128566387	880.844177	2274583336
2026-01-03 00:00:00+00:00	89751.359375	42751135744	3099.806396	23308302336	0.999661	92815785984	873.071350	2363838976

In [12]: data.shape

Out[12]: (1827, 8)

In [13]: data.info()

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1827 entries, 2021-01-03 00:00:00+00:00 to 2026-01-03 00:00:00+00:00
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  -
 0   Close (BTC)     1827 non-null   float64
 1   Volume (BTC)    1827 non-null   int64
 2   Close (ETH)     1827 non-null   float64
 3   Volume (ETH)    1827 non-null   int64
 4   Close (USDT)    1827 non-null   float64
 5   Volume (USDT)   1827 non-null   int64
 6   Close (BNB)     1827 non-null   float64
 7   Volume (BNB)    1827 non-null   int64
dtypes: float64(4), int64(4)
memory usage: 128.5 KB
```

```
In [14]: data.isna().sum()
```

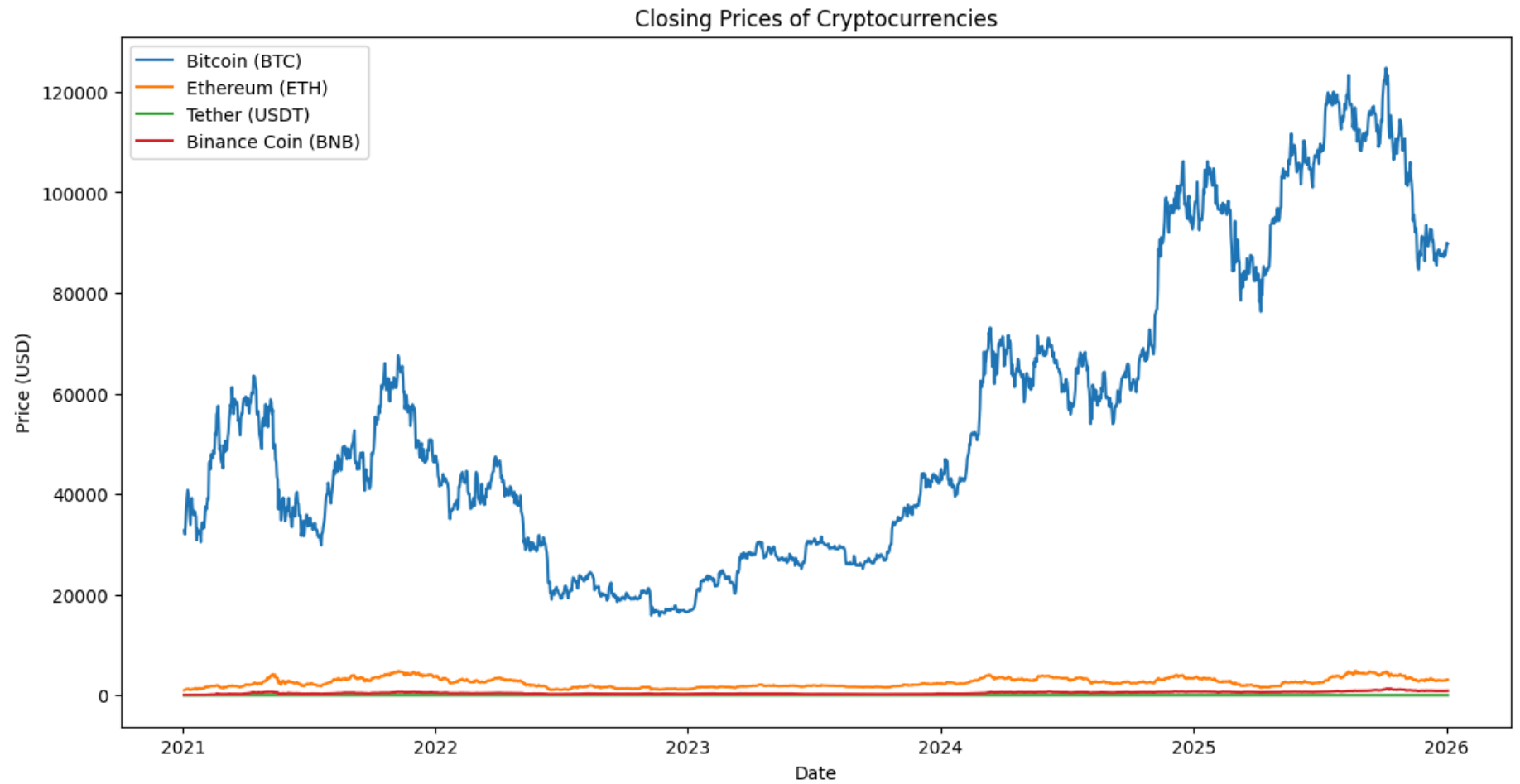
```
Out[14]: Close (BTC)      0
Volume (BTC)      0
Close (ETH)       0
Volume (ETH)      0
Close (USDT)      0
Volume (USDT)     0
Close (BNB)       0
Volume (BNB)      0
dtype: int64
```

```
In [15]: data.describe()
```

Out[15]:

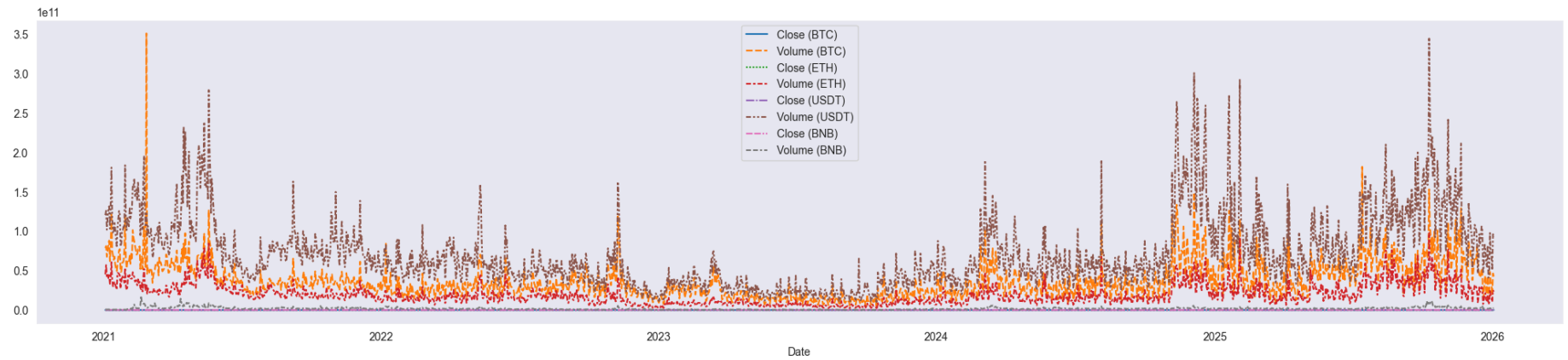
	Close (BTC)	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Volume (USDT)	Close (BNB)	Volume (BNB)
count	1827.000000	1.827000e+03	1827.000000	1.827000e+03	1827.000000	1.827000e+03	1827.000000	1.827000e+03
mean	54509.826626	3.715492e+10	2537.630664	1.954185e+10	1.000146	7.094099e+10	458.107801	1.847096e+09
std	29479.424357	2.303848e+10	909.649787	1.294683e+10	0.000710	4.503880e+10	220.918288	1.476398e+09
min	15787.284180	5.331173e+09	975.507690	2.081626e+09	0.995872	9.989859e+09	38.111610	2.038465e+08
25%	29412.204102	2.132182e+10	1792.933411	1.025234e+10	0.999894	4.011224e+10	287.430939	8.753381e+08
50%	46481.105469	3.180847e+10	2455.935059	1.645097e+10	1.000142	6.045086e+10	401.643890	1.600204e+09
75%	69297.523438	4.714782e+10	3234.704468	2.505416e+10	1.000385	8.832358e+10	599.994690	2.204831e+09
max	124752.531250	3.509679e+11	4831.348633	9.773662e+10	1.011530	3.443980e+11	1310.214355	1.798295e+10

```
In [16]: 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()
```

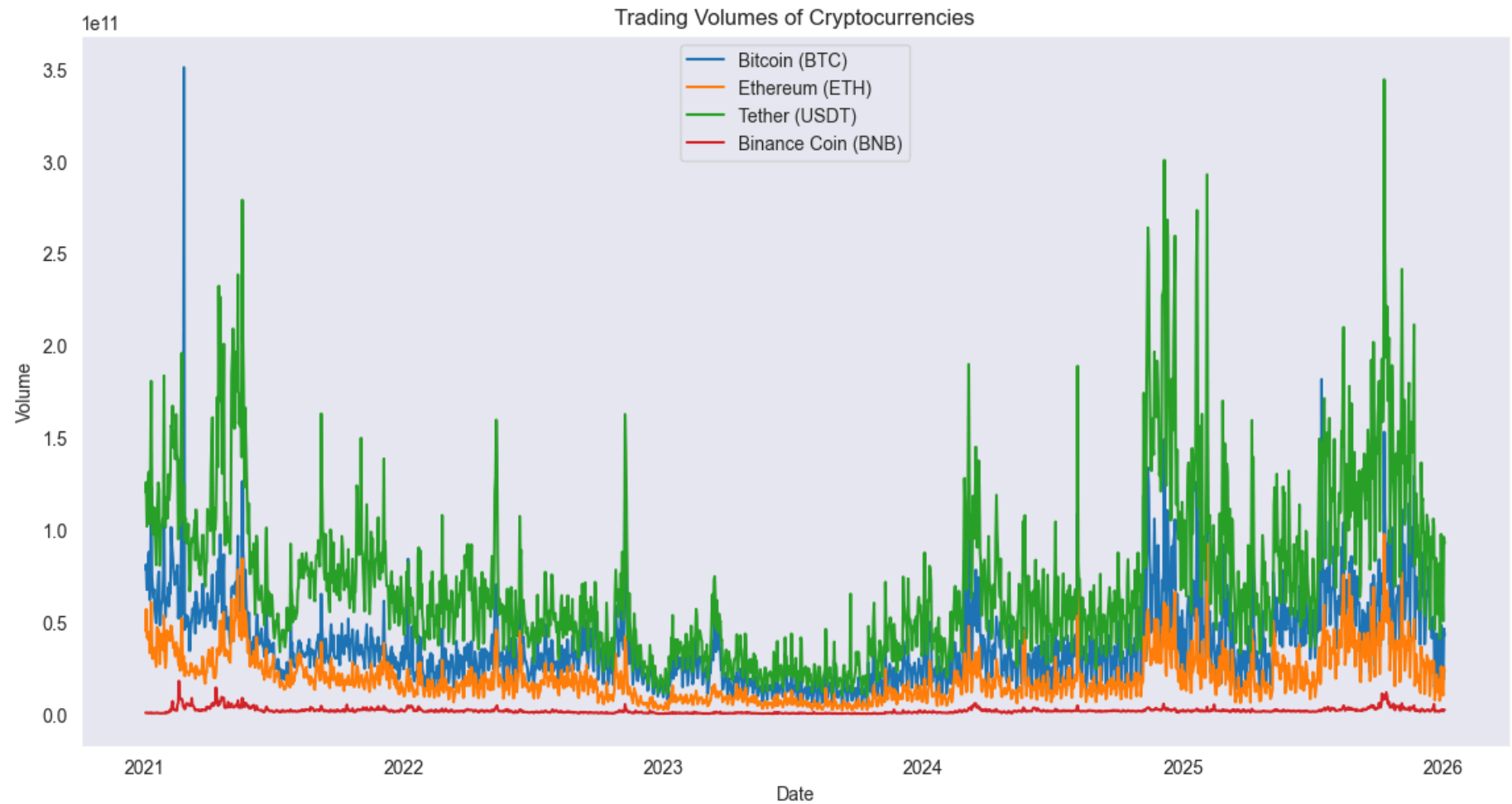


```
In [17]: plt.figure(figsize = (25, 5))  
sns.set_style('dark')  
sns.lineplot(data=data)
```

```
Out[17]: <Axes: xlabel='Date'>
```



```
In [18]: 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 [19]: 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 [20]: 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')

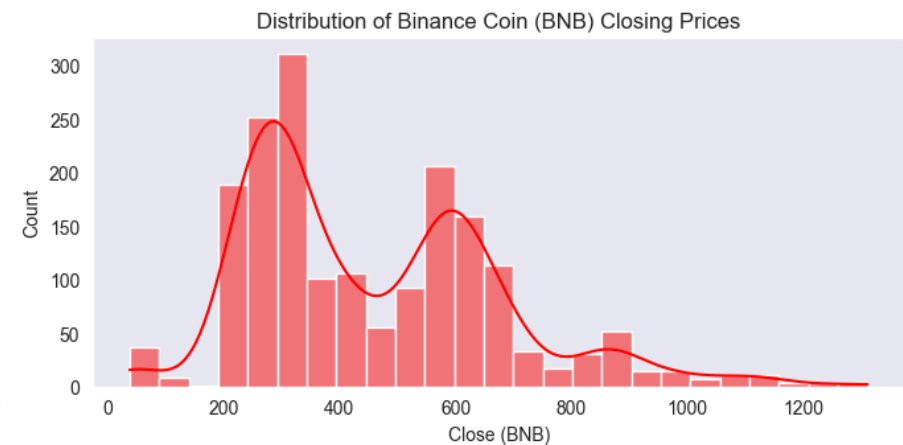
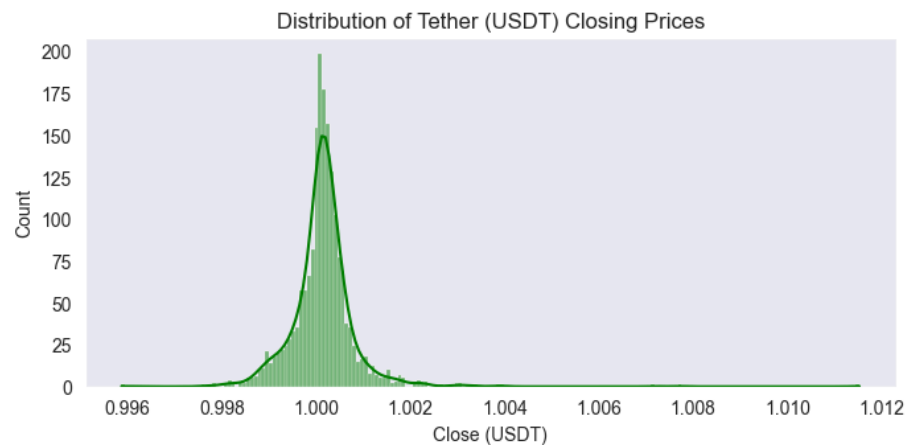
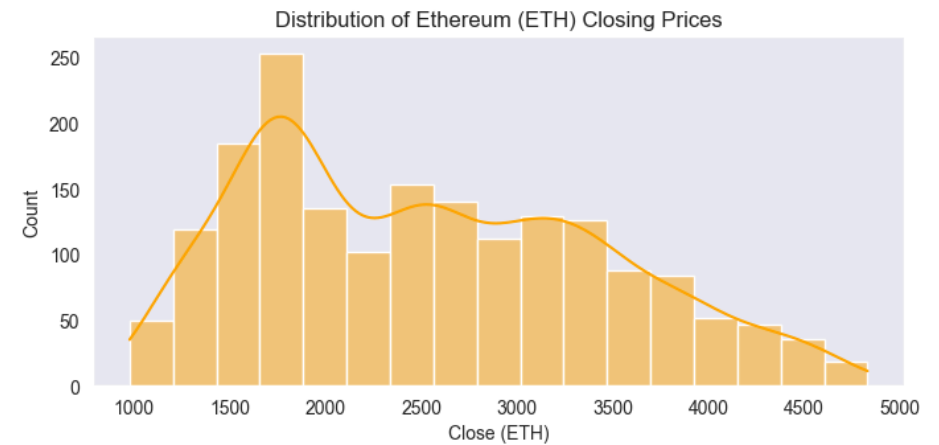
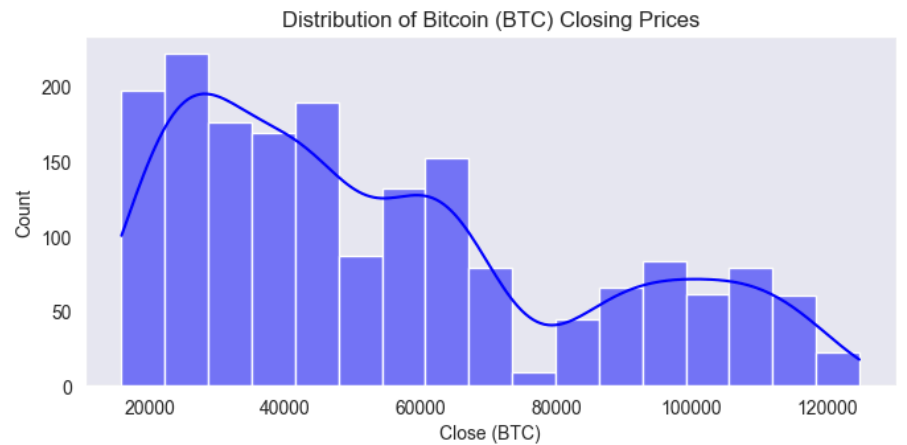
plt.subplot(2, 2, 2)
sns.histplot(data['Close (ETH)'], kde=True, color='orange')
```

```
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')

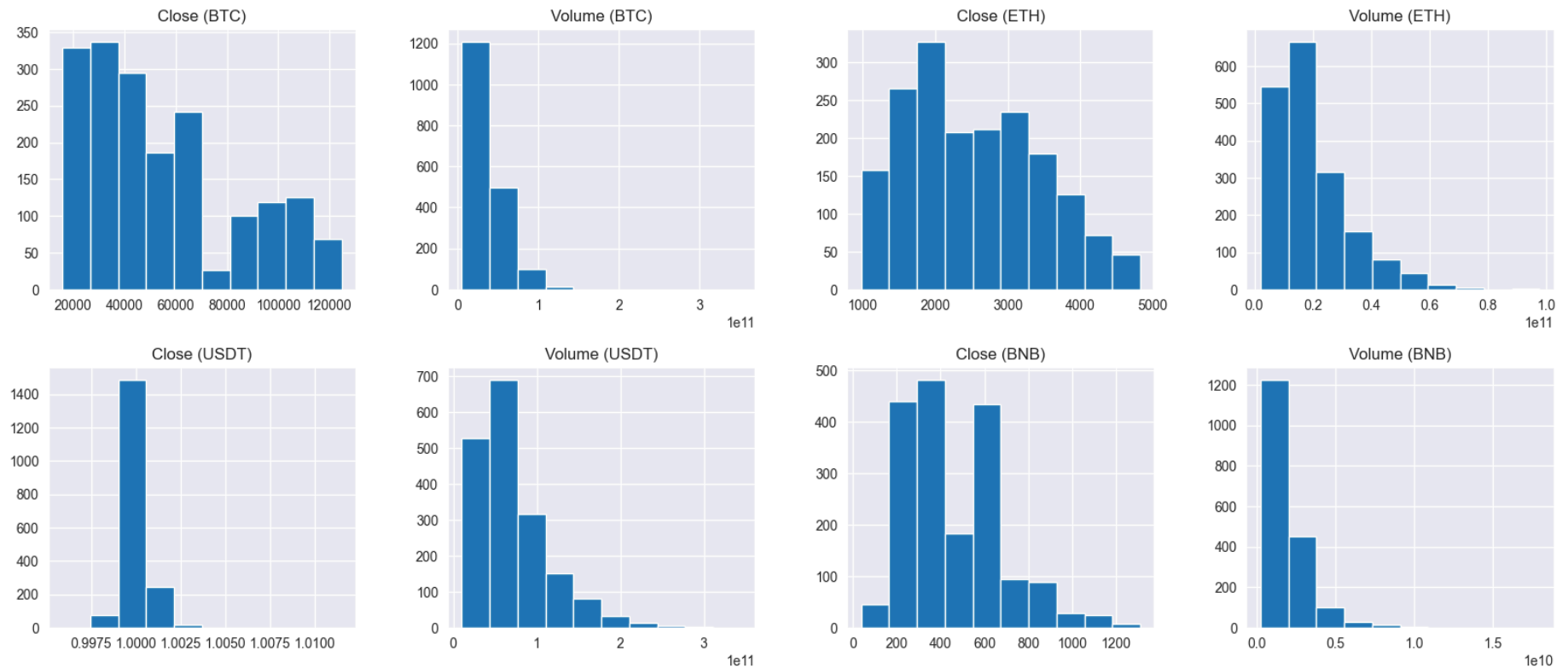
plt.subplot(2, 2, 4)
sns.histplot(data['Close (BNB)'], kde=True, color='red')
plt.title('Distribution of Binance Coin (BNB) Closing Prices')

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



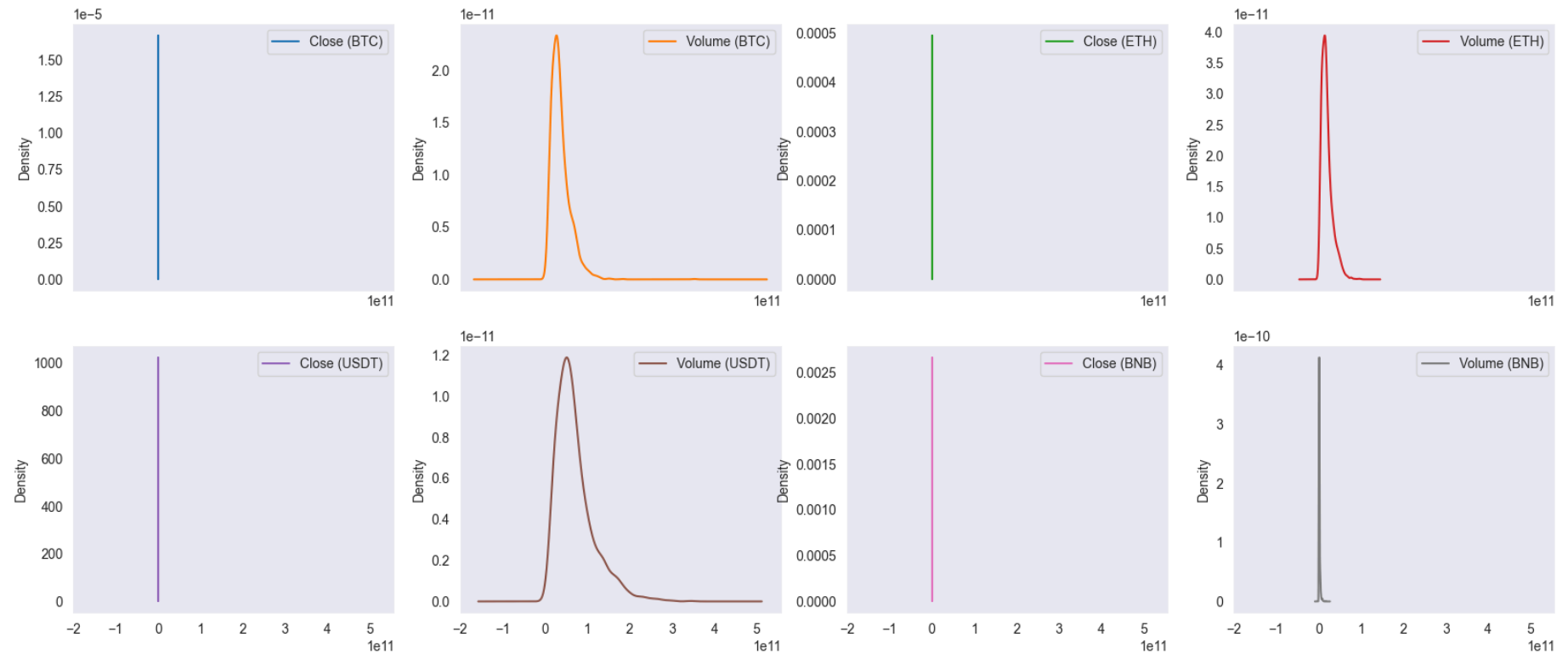
```
In [21]: data.hist(figsize=(20, 8), layout=(2, 4))
```

```
Out[21]: array([[<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)'>]], dtype=object)
```

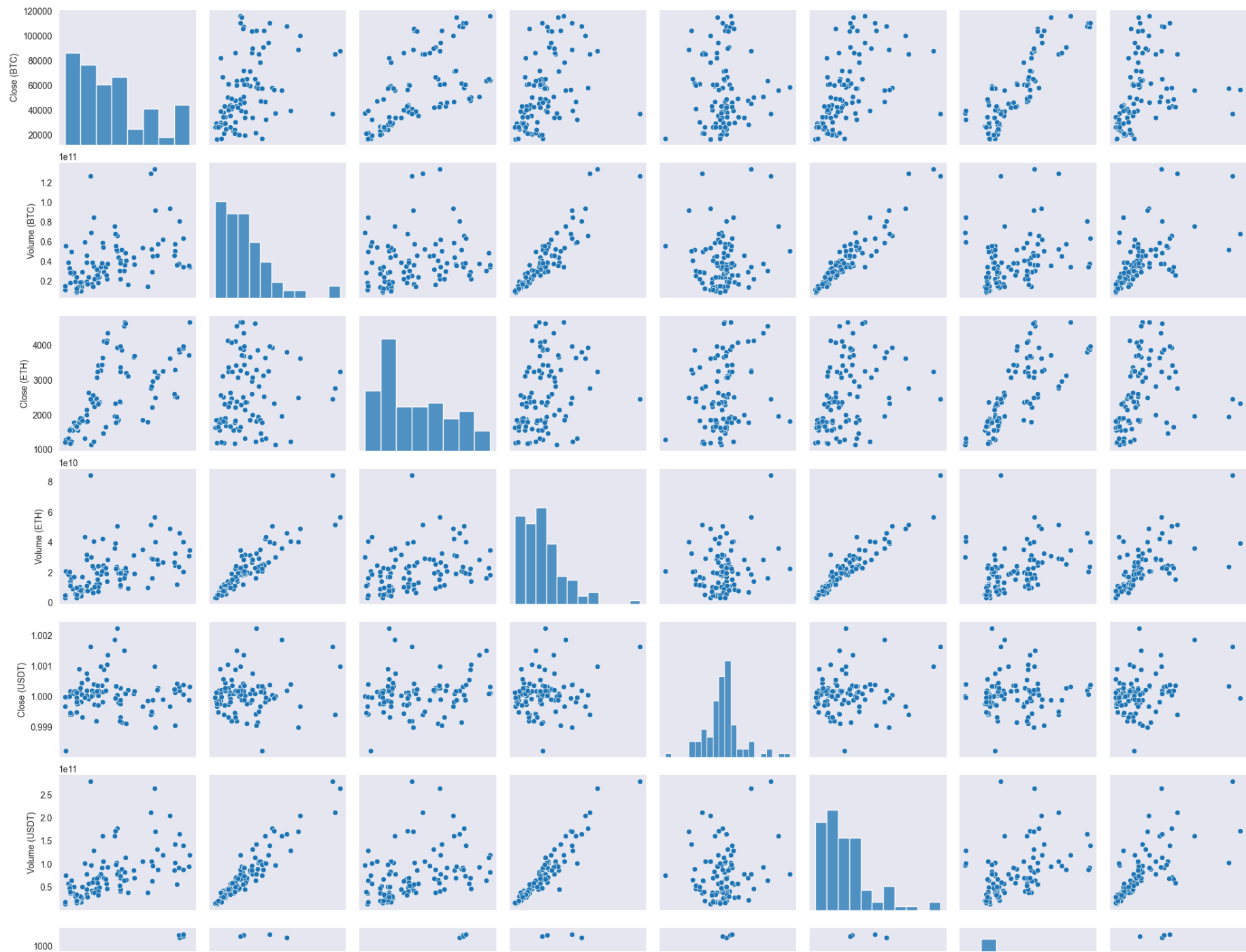


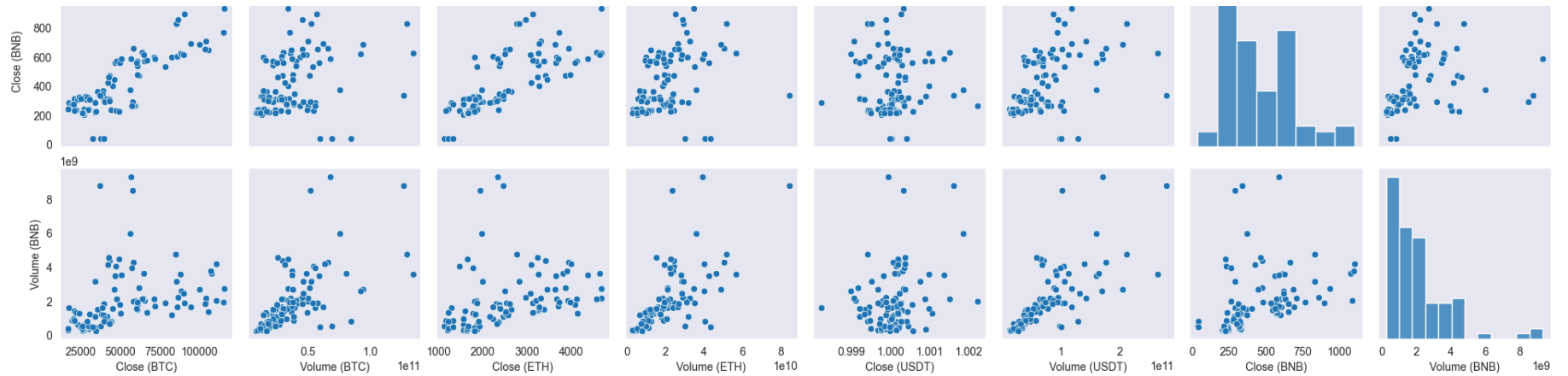
```
In [22]: data.plot(kind = "kde", subplots = True, layout = (2,4), figsize=(20, 8 ))
```

```
Out[22]: 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'>]], dtype=object)
```



```
In [23]: sns.pairplot(data.sample(n=100));
```





```
In [29]: X= data.drop(columns = ['Close (BTC)'], axis = 1)
Y= data.loc[:, 'Close (BTC)']
```

```
In [30]: X.head()
```

```
Out[30]:
```

	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Volume (USDT)	Close (BNB)	Volume (BNB)
Date							
2021-01-03 00:00:00+00:00	78665235202	975.507690	45200463368	1.000514	120425679796	41.148979	758008613
2021-01-04 00:00:00+00:00	81163475344	1040.233032	56945985763	1.000128	125906387011	40.926353	807877171
2021-01-05 00:00:00+00:00	67547324782	1100.006104	41535932781	1.002202	101918715244	41.734600	644270927
2021-01-06 00:00:00+00:00	75289433811	1207.112183	44699914188	1.001528	116105139289	42.165955	641021601
2021-01-07 00:00:00+00:00	84762141031	1225.678101	40468027280	1.000400	129467601516	43.449490	829964770

```
In [31]: X.tail()
```

Out[31]:

	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Volume (USDT)	Close (BNB)	Volume (BNB)
Date							
2025-12-30 00:00:00+00:00	35586356225	2971.416748	18816704381	0.998867	74119035312	860.553711	2042633098
2025-12-31 00:00:00+00:00	33830210616	2967.037598	16451891101	0.998449	70259461189	863.257385	2539874072
2026-01-01 00:00:00+00:00	18849043990	3000.394287	10268796662	0.998745	50548666268	863.054626	1623168589
2026-01-02 00:00:00+00:00	46398906171	3124.422607	25242778003	0.999672	96128566387	880.844177	2274583336
2026-01-03 00:00:00+00:00	42751135744	3099.806396	23308302336	0.999661	92815785984	873.071350	2363838976

In [32]: `Y.head()`

Out[32]:

```
Date
2021-01-03 00:00:00+00:00    32782.023438
2021-01-04 00:00:00+00:00    31971.914062
2021-01-05 00:00:00+00:00    33992.429688
2021-01-06 00:00:00+00:00    36824.363281
2021-01-07 00:00:00+00:00    39371.042969
Name: Close (BTC), dtype: float64
```

In [33]: `X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)`

```
In [34]: 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: (1461, 7)
X_test shape: (366, 7)
y_train shape: (1461,)
y_test shape: (366,)
```

```
In [35]: 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\santo\anaconda3\Lib\site-packages\sklearn\feature_selection_univariate_selection.py:108: RuntimeWarning: invalid value encountered in divide
msw = ssw / float(dfwn)

```
In [37]: 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 [38]: X_train
```

```
Out[38]: array([[1.00015497e+00, 6.59355365e+10, 3.87057343e+02, 1.79623565e+09],
 [1.00047398e+00, 7.95623781e+10, 3.98251465e+02, 2.28811705e+09],
 [1.00028098e+00, 5.35244431e+10, 2.96969055e+02, 8.51195253e+08],
 ...,
 [1.00002396e+00, 1.39888129e+11, 6.89416748e+02, 2.36227653e+09],
 [9.99773979e-01, 4.41267496e+10, 2.49593872e+02, 1.03098597e+09],
 [9.99162972e-01, 3.48790989e+10, 2.73919983e+02, 6.85307608e+08]])
```

```
In [39]: from sklearn.preprocessing import MinMaxScaler
Scaler = MinMaxScaler()
X_train = Scaler.fit_transform(X_train)
X_test = Scaler.transform(X_test)
```

```
In [40]: 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 [41]: #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): 152155696.33265093  
R-squared: 0.8230896542060595
```

```
----- Ridge Regression -----  
Mean Squared Error (MSE): 148532045.32976982  
R-squared: 0.8273028474509232
```

```
----- Lasso Regression -----  
Mean Squared Error (MSE): 152042186.10272157  
R-squared: 0.8232216317429639
```

```
----- ElasticNet Regression -----  
Mean Squared Error (MSE): 759035228.3909888  
R-squared: 0.11747513920964425
```

```
----- Support Vector Regression (SVR) -----  
Mean Squared Error (MSE): 900335147.83984  
R-squared: -0.046813272022108166
```

```
----- Decision Tree Regression -----  
Mean Squared Error (MSE): 100513027.55191538  
R-squared: 0.8831342178466345
```

```
----- Random Forest Regression -----  
Mean Squared Error (MSE): 54494906.25846648  
R-squared: 0.9366391601329428
```

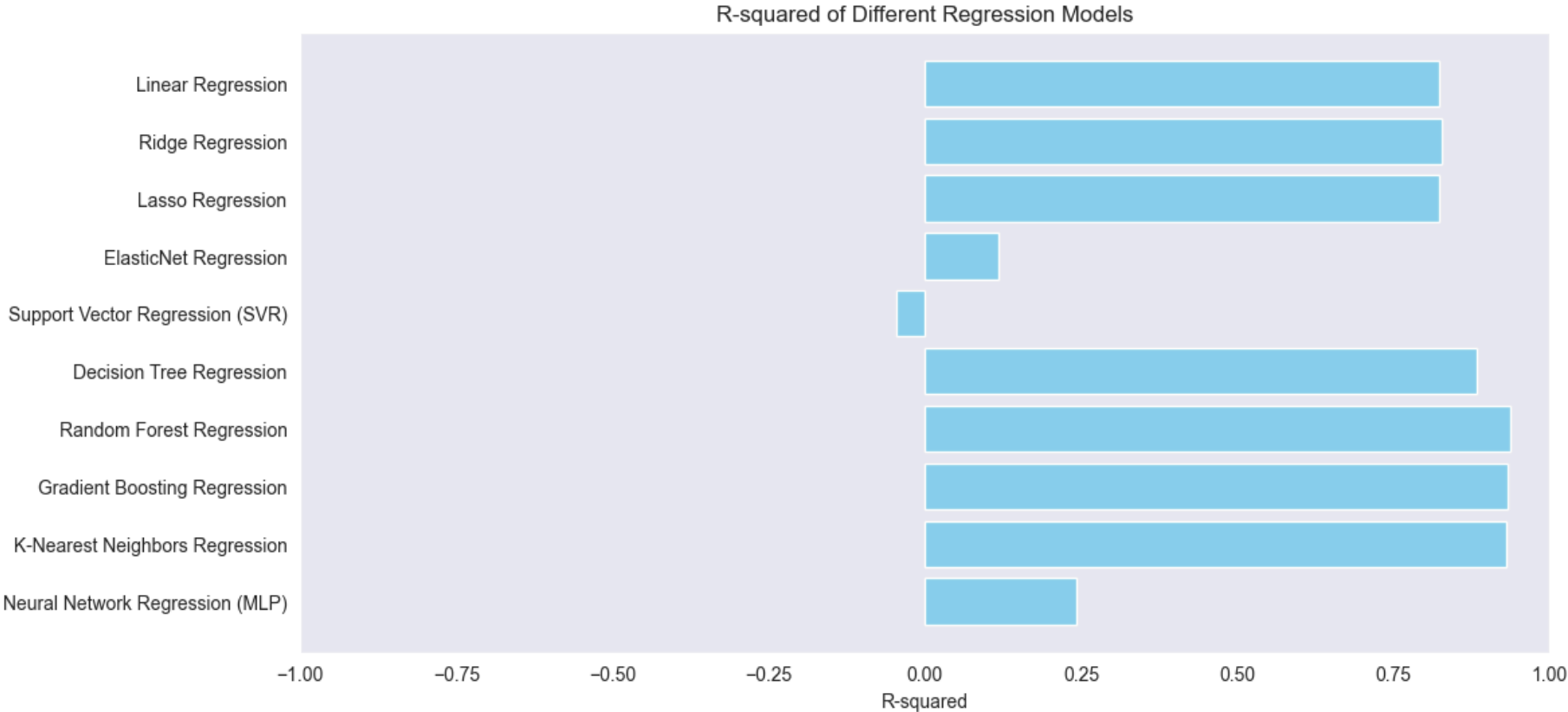
```
----- Gradient Boosting Regression -----  
Mean Squared Error (MSE): 57214062.40102676  
R-squared: 0.9334776166282135
```

```
----- K-Nearest Neighbors Regression -----  
Mean Squared Error (MSE): 59994509.78404195  
R-squared: 0.9302448102341915
```

```
C:\Users\santo\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): 651409997.6085432
R-squared: 0.24261022946777566

	Model	MSE	R-squared
0	Linear Regression	1.521557e+08	0.823090
1	Ridge Regression	1.485320e+08	0.827303
2	Lasso Regression	1.520422e+08	0.823222
3	ElasticNet Regression	7.590352e+08	0.117475
4	Support Vector Regression (SVR)	9.003351e+08	-0.046813
5	Decision Tree Regression	1.005130e+08	0.883134
6	Random Forest Regression	5.449491e+07	0.936639
7	Gradient Boosting Regression	5.721406e+07	0.933478
8	K-Nearest Neighbors Regression	5.999451e+07	0.930245
9	Neural Network Regression (MLP)	6.514100e+08	0.242610



```
In [42]: 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): 53297716.70196183

Loaded Random Forest Regression - R-squared: 0.9380311239142993

In []: