In [1]:
```
pip install yfinance
```

```
Requirement already satisfied: yfinance in /opt/anaconda3/lib/python3.13/s
ite-packages (1.0)
Requirement already satisfied: pandas>=1.3.0 in /opt/anaconda3/lib/python
3.13/site-packages (from yfinance) (2.2.3)
Requirement already satisfied: numpy>=1.16.5 in /opt/anaconda3/lib/python
3.13/site-packages (from yfinance) (2.1.3)
Requirement already satisfied: requests>=2.31 in /opt/anaconda3/lib/python
3.13/site-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in /opt/anaconda3/lib/p
ython3.13/site-packages (from yfinance) (0.0.12)
Requirement already satisfied: platformdirs>=2.0.0 in /opt/anaconda3/lib/p
ython3.13/site-packages (from yfinance) (4.3.7)
Requirement already satisfied: pytz>=2022.5 in /opt/anaconda3/lib/python3.
13/site-packages (from yfinance) (2024.1)
Requirement already satisfied: frozendict>=2.3.4 in /opt/anaconda3/lib/pyt
hon3.13/site-packages (from yfinance) (2.4.2)
Requirement already satisfied: peewee>=3.16.2 in /opt/anaconda3/lib/python
3.13/site-packages (from yfinance) (3.18.3)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /opt/anaconda3/li
b/python3.13/site-packages (from yfinance) (4.12.3)
Requirement already satisfied: curl_cffi<0.14,>=0.7 in /opt/anaconda3/lib/
python3.13/site-packages (from yfinance) (0.13.0)
Requirement already satisfied: protobuf>=3.19.0 in /opt/anaconda3/lib/pyth
on3.13/site-packages (from yfinance) (5.29.3)
Requirement already satisfied: websockets>=13.0 in /opt/anaconda3/lib/pyth
on3.13/site-packages (from yfinance) (15.0.1)
Requirement already satisfied: cffi>=1.12.0 in /opt/anaconda3/lib/python3.
13/site-packages (from curl_cffi<0.14,>=0.7->yfinance) (1.17.1)
Requirement already satisfied: certifi>=2024.2.2 in /opt/anaconda3/lib/pyt
hon3.13/site-packages (from curl_cffi<0.14,>=0.7->yfinance) (2025.8.3)
Requirement already satisfied: soupsieve>1.2 in /opt/anaconda3/lib/python
3.13/site-packages (from beautifulsoup4>=4.11.1->yfinance) (2.5)
Requirement already satisfied: pycparser in /opt/anaconda3/lib/python3.13/
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 /opt/anaconda3/li
b/python3.13/site-packages (from pandas>=1.3.0->yfinance) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in /opt/anaconda3/lib/python
3.13/site-packages (from pandas>=1.3.0->yfinance) (2025.2)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.13/s
ite-packages (from python-dateutil>=2.8.2->pandas>=1.3.0->yfinance) (1.17.
0)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/
lib/python3.13/site-packages (from requests>=2.31->yfinance) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.
13/site-packages (from requests>=2.31->yfinance) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/lib/py
thon3.13/site-packages (from requests>=2.31->yfinance) (2.3.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
#pip install yfinance
```

```
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
#from sklearn import metrics
#from sklearn.metrics import accuracy_score, classification_report, confu
```

In [3]:
```
btc = yf.Ticker('BTC-USD')
prices1 = btc.history(period='5y')
prices1.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits']

eth = yf.Ticker('ETH-USD')
prices2 = eth.history(period='5y')
prices2.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits']

usdt = yf.Ticker('USDT-USD')
prices3 = usdt.history(period='5y')
prices3.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits']

bnb = yf.Ticker('BNB-USD')
prices4 = bnb.history(period='5y')
prices4.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits']
```

In [4]:
```
p1 = prices1.join(prices2, lsuffix = ' (BTC)', rsuffix = ' (ETH)')
p2 = prices3.join(prices4, lsuffix = ' (USDT)', rsuffix = ' (BNB)')
data = p1.join(p2, lsuffix = '_', rsuffix = '_')
```

In [5]:
```
data.head()
```

Out[5]:

Date	Close (BTC)	Volume (BTC)	Close (ETH)	Volume (ETH)	Clos (USD
2021-01-04 00:00:00+00:00	31971.914062	81163475344	1040.233032	56945985763	1.00012
2021-01-05 00:00:00+00:00	33992.429688	67547324782	1100.006104	41535932781	1.00220
2021-01-06 00:00:00+00:00	36824.363281	75289433811	1207.112183	44699914188	1.00152
2021-01-07 00:00:00+00:00	39371.042969	84762141031	1225.678101	40468027280	1.00040
2021-01-08 00:00:00+00:00	40797.609375	88107519480	1224.197144	44334826666	1.00004

In [6]:
```
data.tail()
```

Out[6]:

Date	Close (BTC)	Volume (BTC)	Close (ETH)	Volume (ETH)	Clo (USD
2025-12-31 00:00:00+00:00	87508.828125	33830210616	2967.037598	16451891101	0.9984
2026-01-01 00:00:00+00:00	88731.984375	18849043990	3000.394287	10268796662	0.9987
2026-01-02 00:00:00+00:00	89944.695312	46398906171	3124.422607	25242778003	0.9996
2026-01-03 00:00:00+00:00	90603.187500	20774828592	3125.917480	11460707919	0.9996
2026-01-04 00:00:00+00:00	91364.398438	24750749696	3140.739990	12534880256	0.9994

In [7]:
```
data.shape
```

Out[7]: (1827, 8)

In [8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1827 entries, 2021-01-04 00:00:00+00:00 to 2026-01-04 00:0
0:00+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 [9]:
```
data.isna().sum()
```

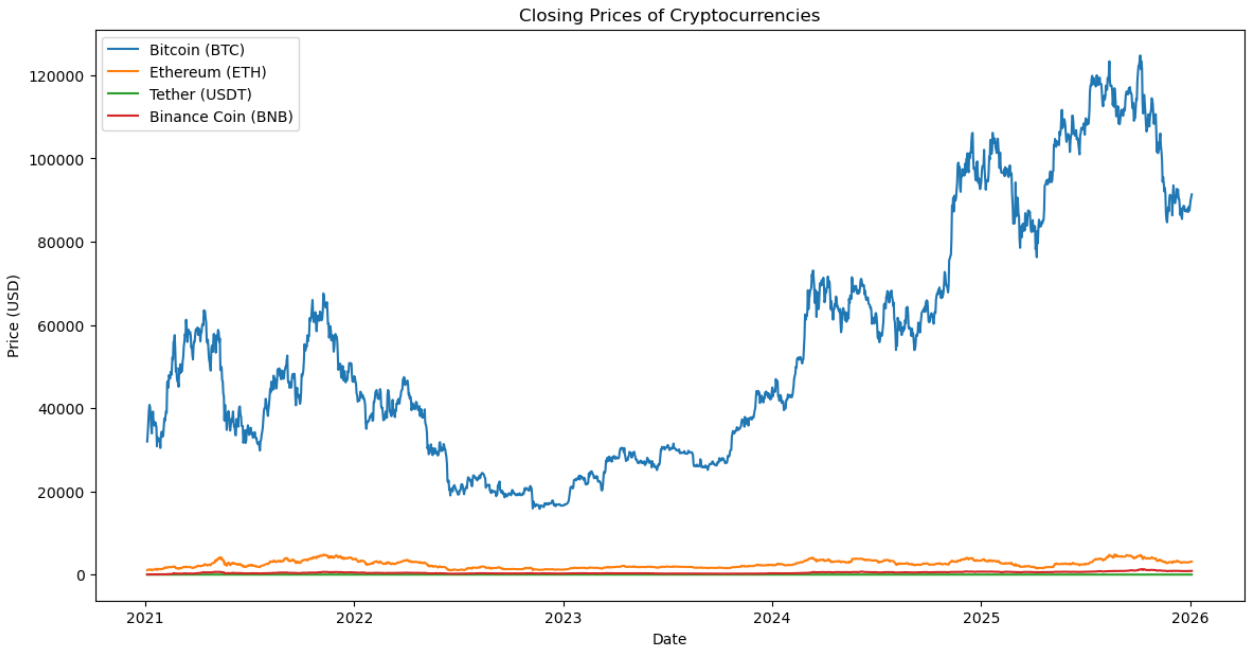
Out[9]:
```
Close (BTC)      0
Volume (BTC)     0
Close (ETH)      0
Volume (ETH)     0
Close (USDT)     0
Volume (USDT)    0
Close (BNB)      0
Volume (BNB)     0
dtype: int64
```

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

Out[10]:

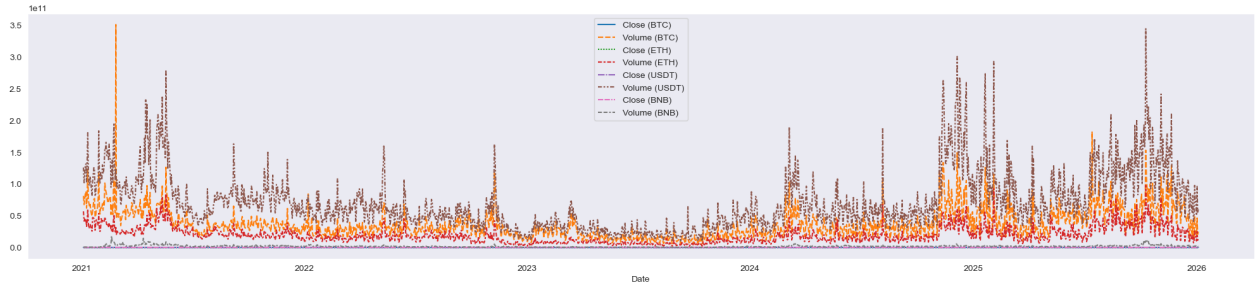
	Close (BTC)	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	
count	1827.000000	1.827000e+03	1827.000000	1.827000e+03	1827.000000	1.8
mean	54542.357662	3.711338e+10	2538.830085	1.951748e+10	1.000145	7.0
std	29488.200718	2.302261e+10	909.032765	1.293499e+10	0.000710	4.5
min	15787.284180	5.331173e+09	993.636780	2.081626e+09	0.995872	9.9
25%	29412.204102	2.130695e+10	1793.287048	1.025234e+10	0.999893	4.(
50%	46612.632812	3.175896e+10	2456.425293	1.643428e+10	1.000142	6.0
75%	69324.179688	4.708497e+10	3234.704468	2.502669e+10	1.000383	8.8
max	124752.531250	3.509679e+11	4831.348633	9.773662e+10	1.011530	3.4

In [11]:
```
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()
```

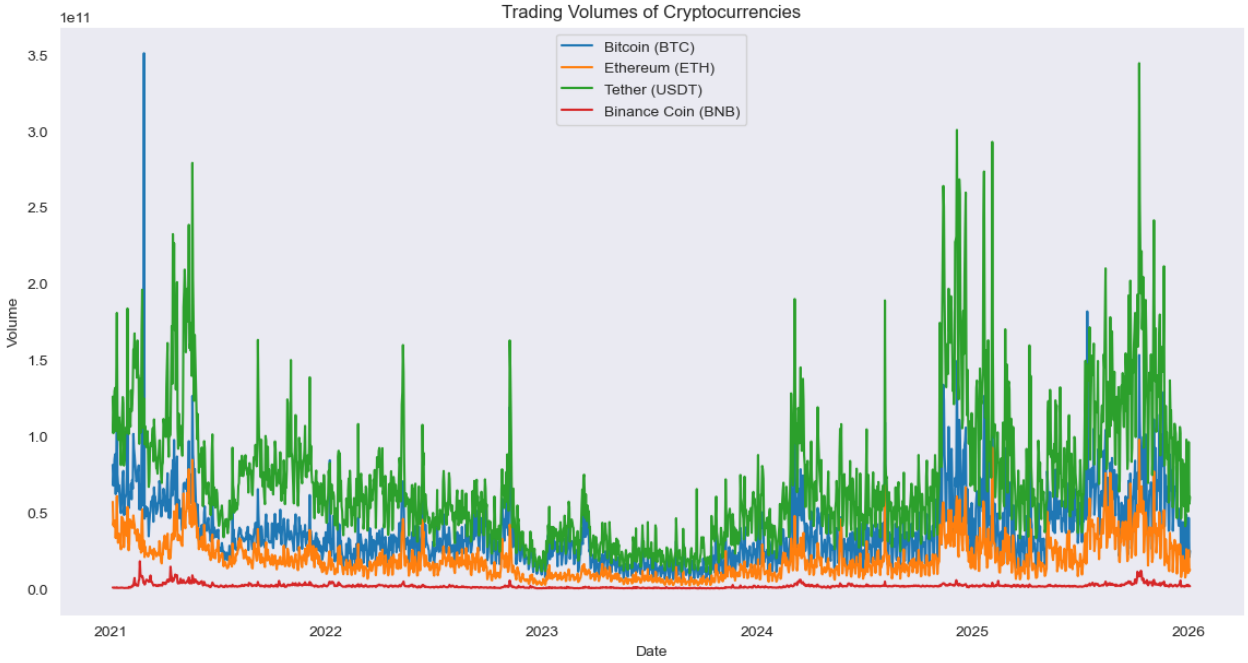


In [12]:
```
plt.figure(figsize = (25, 5))
sns.set_style('dark')
sns.lineplot(data=data)
```

Out[12]: <Axes: xlabel='Date'>

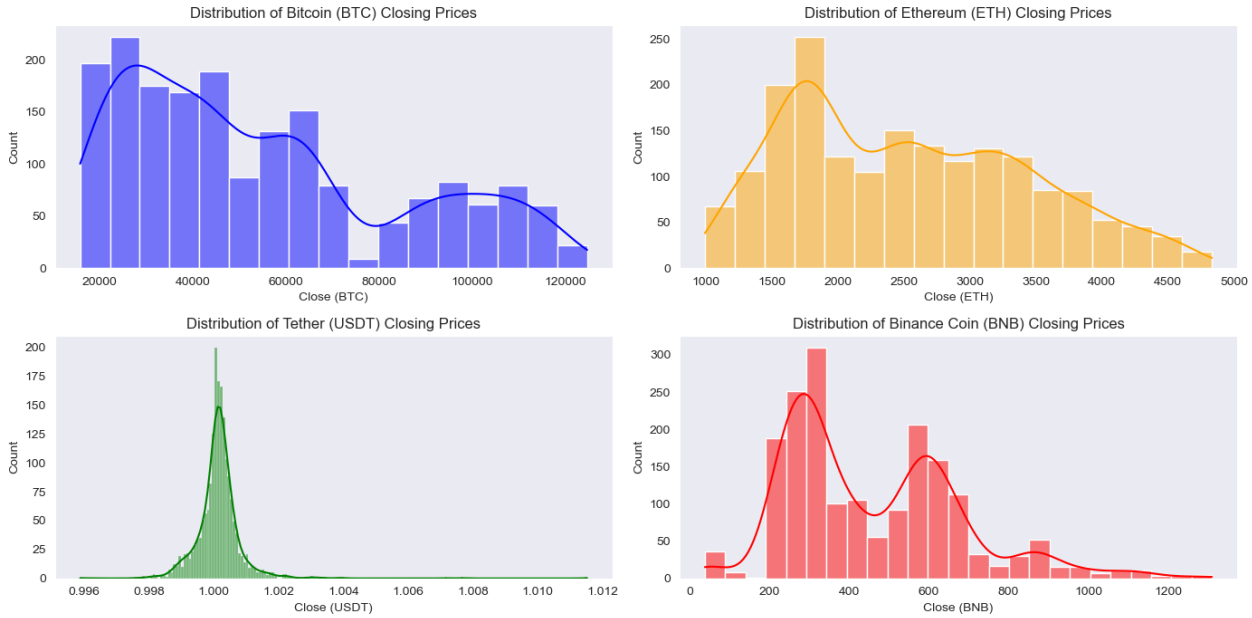


In [13]:
```
# Visualize the Trading Volumes
#Let's visualize the trading volumes of all four cryptocurrencies:
plt.figure(figsize=(14, 7))
plt.plot(data.index, data['Volume (BTC)'], label='Bitcoin (BTC)')
plt.plot(data.index, data['Volume (ETH)'], label='Ethereum (ETH)')
plt.plot(data.index, data['Volume (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()
```



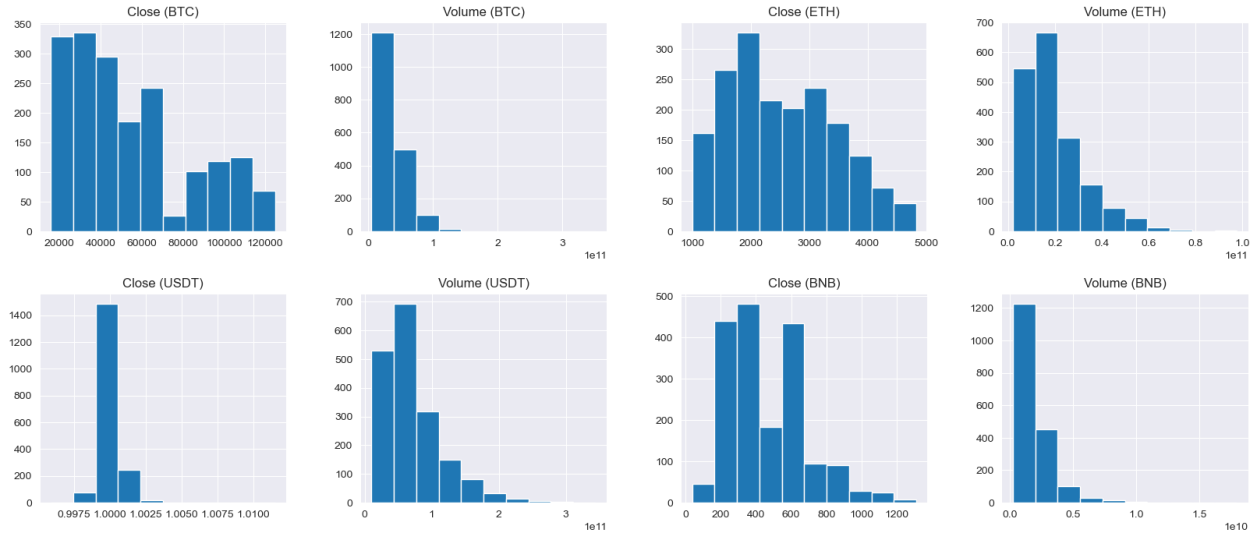
In [14]:
```
#Correlation Analysis
#We'll analyze the correlation between the closing prices of the cryptocu
# Calculate the correlation matrix
corr_matrix = data[['Close (BTC)', 'Close (ETH)', 'Close (USDT)', 'Close

# 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 [15]:
```
# 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()
```



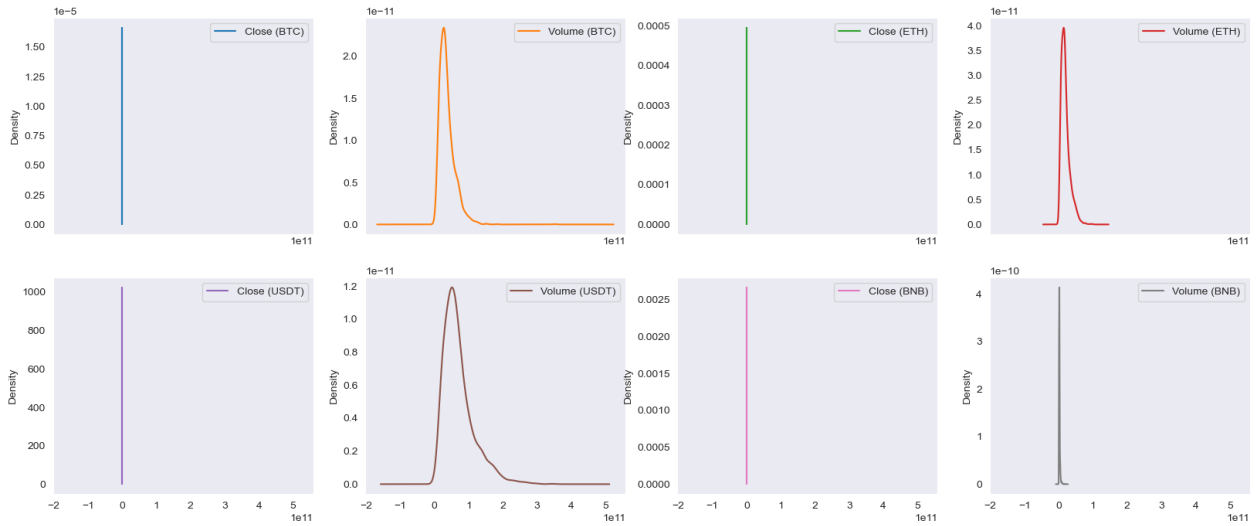
In [16]:
```
data.hist(figsize=(20, 8), layout=(2, 4))
```

Out[16]:
```
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)
```

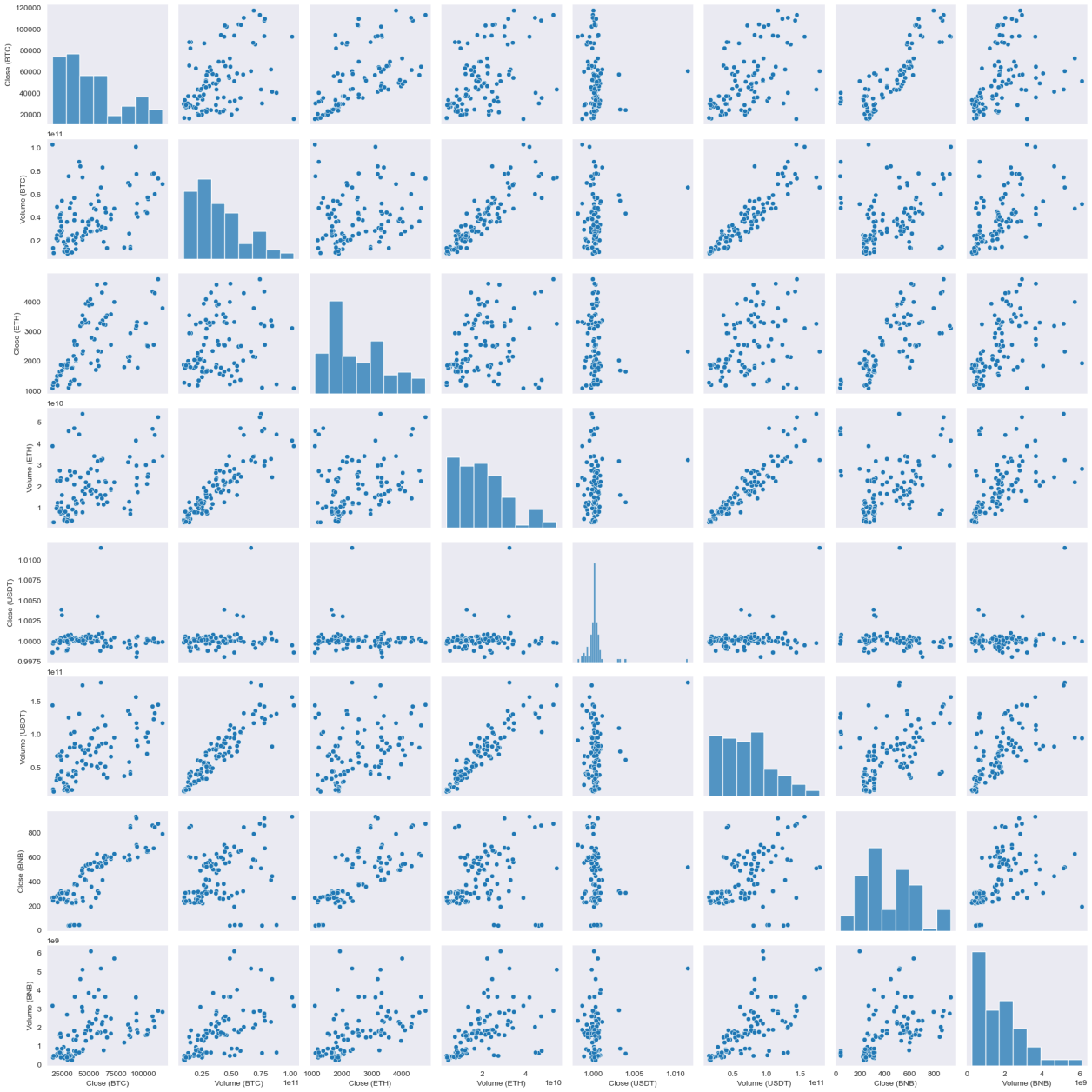


In [17]:
```
data.plot(kind = "kde", subplots = True, layout = (2, 4), figsize = (20,
```

Out[17]:
```
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=obje
ct)
```



In [18]:
```
sns.pairplot(data.sample(n=100));
```



In [19]:
```
#data preprocessing
X = data.drop(columns = ['Close (BTC)'], axis = 1)
Y = data.loc[:, 'Close (BTC)']
```

In [20]:
```
X.head()
```

Out[20]:

Date	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Volum (USDT
2021-01-04 00:00:00+00:00	81163475344	1040.233032	56945985763	1.000128	1259063870
2021-01-05 00:00:00+00:00	67547324782	1100.006104	41535932781	1.002202	10191871524
2021-01-06 00:00:00+00:00	75289433811	1207.112183	44699914188	1.001528	11610513928
2021-01-07 00:00:00+00:00	84762141031	1225.678101	40468027280	1.000400	12946760151
2021-01-08 00:00:00+00:00	88107519480	1224.197144	44334826666	1.000045	13155596174

In [21]:
```
X.tail()
```

Out[21]:

Date	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Volun (USD
2025-12-31 00:00:00+00:00	33830210616	2967.037598	16451891101	0.998449	702594611
2026-01-01 00:00:00+00:00	18849043990	3000.394287	10268796662	0.998745	505486662
2026-01-02 00:00:00+00:00	46398906171	3124.422607	25242778003	0.999672	961285663
2026-01-03 00:00:00+00:00	20774828592	3125.917480	11460707919	0.999635	556601048
2026-01-04 00:00:00+00:00	24750749696	3140.739990	12534880256	0.999491	601305128

In [22]:
```
Y.head()
```

Out[22]:
```
Date
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
2021-01-08 00:00:00+00:00    40797.609375
Name: Close (BTC), dtype: float64
```

In [23]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
```

In [24]:
```
# 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: (1461, 7)
X_test shape: (366, 7)
y_train shape: (1461,)
y_test shape: (366,)
```

In [25]:
```
#SelectKBest is a feature selection method provided by scikit-learn (skle
#This function evaluates each feature independently and selects those tha

#Parameters
#k: Specifies the number of top features to select. In your case, k=4 ind

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)
```

```
/opt/anaconda3/lib/python3.13/site-packages/sklearn/feature_selection/_uni
variate_selection.py:108: RuntimeWarning: invalid value encountered in div
ide
  msw = sswn / float(dfwn)
```

In [26]:
```
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 [28]:
```
X_train
```

Out[28]:
```
array([[1.00017798e+00, 8.14116305e+10, 4.21643188e+02, 2.16249338e+09],
       [9.99684989e-01, 7.95351909e+10, 4.30988861e+02, 2.54921533e+09],
       [1.00027597e+00, 4.71600623e+10, 2.95161865e+02, 7.00835025e+08],
       ...,
       [1.00007701e+00, 1.47110596e+11, 6.89921143e+02, 2.16720240e+09],
       [9.99733984e-01, 4.21892486e+10, 2.47848480e+02, 1.05086167e+09],
       [9.99101996e-01, 2.32884391e+10, 2.72397675e+02, 5.51451228e+0
8]])
```

In [29]:
```
# implementation of 10 different regression algorithms using scikit-learn

#Import Libraries and Generate Sample Data

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

In [30]:
```
#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_estimator
    'K-Nearest Neighbors Regression': KNeighborsRegressor(n_neighbors=5),
    'Neural Network Regression (MLP)': MLPRegressor(hidden_layer_sizes=(1
}
```

In [31]:
```
# 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)
```

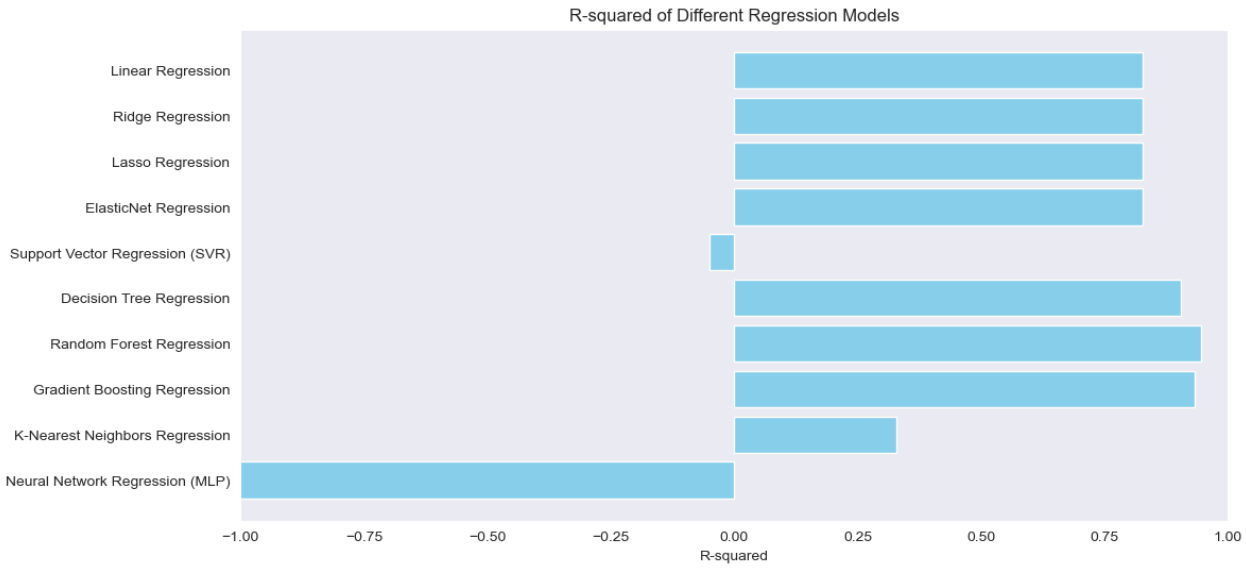
```
----- Linear Regression -----
Mean Squared Error (MSE): 146760904.2750877
R-squared: 0.8280020362883744

----- Ridge Regression -----
Mean Squared Error (MSE): 146760922.93241867
R-squared: 0.8280020144227218

----- Lasso Regression -----
Mean Squared Error (MSE): 146760891.12623718
R-squared: 0.8280020516983061

----- ElasticNet Regression -----
Mean Squared Error (MSE): 146760170.7525191
R-squared: 0.8280028959477554
```

```
/opt/anaconda3/lib/python3.13/site-packages/sklearn/linear_model/_ridge.p
y:215: LinAlgWarning: Ill-conditioned matrix (rcond=3.38058e-25): result m
ay not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
----- Support Vector Regression (SVR) -----
Mean Squared Error (MSE): 896787670.5722206
R-squared: -0.05099961043437684

----- Decision Tree Regression -----
Mean Squared Error (MSE): 82170575.52742752
R-squared: 0.9036993418816857

----- Random Forest Regression -----
Mean Squared Error (MSE): 45693775.198592745
R-squared: 0.9464487063003957

----- Gradient Boosting Regression -----
Mean Squared Error (MSE): 57532595.88657175
R-squared: 0.9325740776236555

----- K-Nearest Neighbors Regression -----
Mean Squared Error (MSE): 572425710.990455
R-squared: 0.329139751810248

----- Neural Network Regression (MLP) -----
Mean Squared Error (MSE): 4506816475109.622
R-squared: -5280.810304792754
```

	Model	MSE	R-squared
0	Linear Regression	1.467609e+08	0.828002
1	Ridge Regression	1.467609e+08	0.828002
2	Lasso Regression	1.467609e+08	0.828002
3	ElasticNet Regression	1.467602e+08	0.828003
4	Support Vector Regression (SVR)	8.967877e+08	-0.051000
5	Decision Tree Regression	8.217058e+07	0.903699
6	Random Forest Regression	4.569378e+07	0.946449
7	Gradient Boosting Regression	5.753260e+07	0.932574
8	K-Nearest Neighbors Regression	5.724257e+08	0.329140
9	Neural Network Regression (MLP)	4.506816e+12	-5280.810305

In [32]:
```
# 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()
```



In [33]:
```
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 da
```

In [34]:
```
X, Y = make_regression(n_samples=1000, n_features=10, noise=0.1, random_s


# 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'))
```

In [35]:
```
# 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): 45253629.01095
4395
Loaded Random Forest Regression - R-squared: 0.946964540189422
```

In [ ]: