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

```
Requirement already satisfied: yfinance in c:\users\hanshu\anaconda3\lib\site-pac
kages (0.2.65)
Requirement already satisfied: pandas>=1.3.0 in c:\users\hanshu\anaconda3\lib\sit
e-packages (from yfinance) (2.2.2)
Requirement already satisfied: numpy>=1.16.5 in c:\users\hanshu\anaconda3\lib\sit
e-packages (from yfinance) (1.26.4)
Requirement already satisfied: requests>=2.31 in c:\users\hanshu\anaconda3\lib\si
te-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\hanshu\anaconda3\l
ib\site-packages (from yfinance) (0.0.12)
Requirement already satisfied: platformdirs>=2.0.0 in c:\users\hanshu\anaconda3\l
ib\site-packages (from yfinance) (3.10.0)
Requirement already satisfied: pytz>=2022.5 in c:\users\hanshu\anaconda3\lib\site
-packages (from yfinance) (2024.1)
Requirement already satisfied: frozendict>=2.3.4 in c:\users\hanshu\anaconda3\lib
\site-packages (from yfinance) (2.4.2)
Requirement already satisfied: peewee>=3.16.2 in c:\users\hanshu\anaconda3\lib\si
te-packages (from yfinance) (3.18.2)
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\users\hanshu\anaconda
3\lib\site-packages (from yfinance) (4.12.3)
Requirement already satisfied: curl_cffi>=0.7 in c:\users\hanshu\anaconda3\lib\si
te-packages (from yfinance) (0.13.0)
Requirement already satisfied: protobuf>=3.19.0 in c:\users\hanshu\anaconda3\lib
\site-packages (from yfinance) (6.32.0)
Requirement already satisfied: websockets>=13.0 in c:\users\hanshu\anaconda3\lib
\site-packages (from yfinance) (15.0.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\hanshu\anaconda3\lib\sit
e-packages (from beautifulsoup4>=4.11.1->yfinance) (2.5)
Requirement already satisfied: cffi>=1.12.0 in c:\users\hanshu\anaconda3\lib\site
-packages (from curl_cffi>=0.7->yfinance) (1.17.1)
Requirement already satisfied: certifi>=2024.2.2 in c:\users\hanshu\anaconda3\lib
\site-packages (from curl_cffi>=0.7->yfinance) (2025.4.26)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\hanshu\anaconda
3\lib\site-packages (from pandas>=1.3.0->yfinance) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in c:\users\hanshu\anaconda3\lib\si
te-packages (from pandas>=1.3.0->yfinance) (2023.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\hanshu\anacon
da3\lib\site-packages (from requests>=2.31->yfinance) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\hanshu\anaconda3\lib\site
-packages (from requests>=2.31->yfinance) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\hanshu\anaconda3\li
b\site-packages (from requests>=2.31->yfinance) (2.2.3)
Requirement already satisfied: pycparser in c:\users\hanshu\anaconda3\lib\site-pa
ckages (from cffi>=1.12.0->curl_cffi>=0.7->yfinance) (2.21)
Requirement already satisfied: six>=1.5 in c:\users\hanshu\anaconda3\lib\site-pac
kages (from python-dateutil>=2.8.2->pandas>=1.3.0->yfinance) (1.16.0)
```

In [2]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import seaborn as sns
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, confusion_ma
```

In [3]:

```
#The code fetches historical price data for Bitcoin, Ethereum, Tether, and Binan
#This cleaned data can then be used for further analysis or machine learning tas

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

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

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

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

In [4]:

```
#The parameters lsuffix and rsuffix in the join method are used to add suffixes
# This is necessary to avoid column name conflicts when the two DataFrames have

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]:

	Close(BTC)	Volume(BTC)	Close(ETH)	Volume(ETH)	Close(USDT)	Vo
Date						
2020-09-14 00:00:00+00:00	10680.837891	35453581940	377.268860	17536695361	1.001289	
2020-09-15 00:00:00+00:00	10796.951172	32509451925	364.839203	16140584321	1.002487	
2020-09-16 00:00:00+00:00	10974.905273	30769986455	365.812286	16107612177	1.003444	
2020-09-17 00:00:00+00:00	10948.990234	38151810523	389.019226	19899531080	1.001878	
2020-09-18 00:00:00+00:00	10944.585938	26341903912	384.364532	14108357740	0.999502	

In [6]:

```
data.tail()
```

Out[6]:

Date	Close(BTC)	Volume(BTC)	Close(ETH)	Volume(ETH)	Close(USDT)
2025-09-10 00:00:00+00:00	113955.359375	56377473784	4349.145996	39521365146	1.000138
2025-09-11 00:00:00+00:00	115507.539062	45685065332	4461.233398	35959212991	1.000266
2025-09-12 00:00:00+00:00	116101.578125	54785725894	4715.246094	43839753626	1.000618
2025-09-13 00:00:00+00:00	115950.507812	34549454947	4668.179688	34843845977	1.000319
2025-09-14 00:00:00+00:00	115852.859375	31612506112	4642.932617	28678072320	1.000401

In [7]:

```
data.shape
```

Out[7]:

```
(1827, 8)
```

In [8]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1827 entries, 2020-09-14 00:00:00+00:00 to 2025-09-14 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 [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()
```

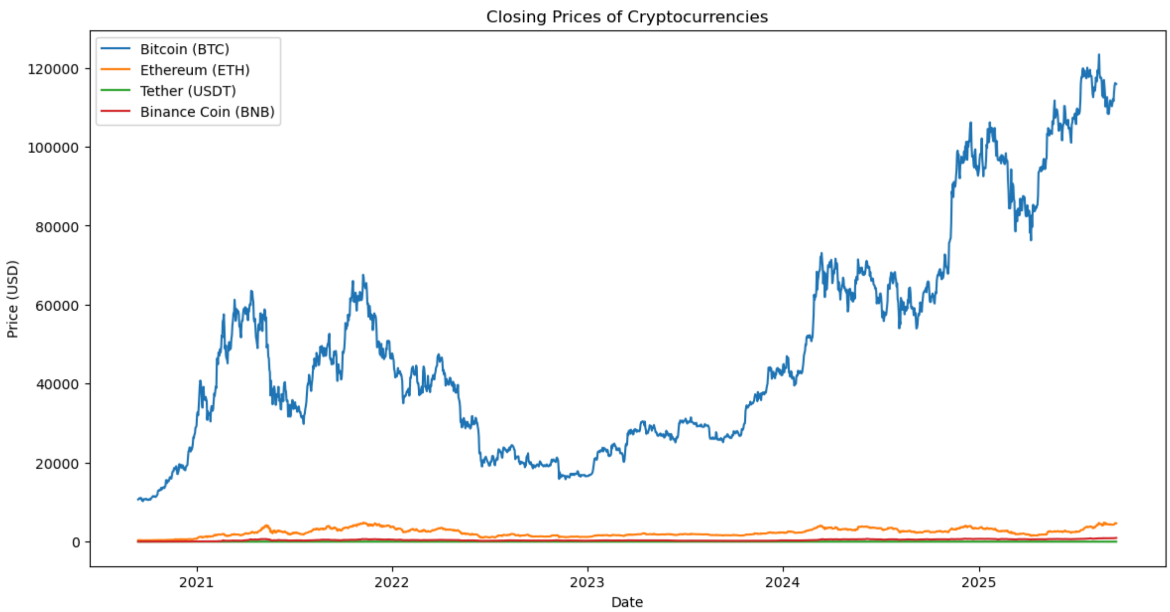
Out[10]:

	Close(BTC)	Volume(BTC)	Close(ETH)	Volume(ETH)	Close(USDT)	Volume(U
count	1827.000000	1.827000e+03	1827.000000	1.827000e+03	1827.000000	1.82700C
mean	49323.487513	3.571168e+10	2350.643495	1.842279e+10	1.000188	6.659633
std	28103.797749	2.156630e+10	983.483850	1.191476e+10	0.000737	4.142044
min	10246.186523	5.331173e+09	321.116302	2.081626e+09	0.995872	9.989859
25%	27055.889648	2.136936e+10	1649.178711	1.024022e+10	0.999922	3.915764
50%	42412.433594	3.115874e+10	2260.648682	1.581725e+10	1.000157	5.707433
75%	63842.345703	4.430770e+10	3107.366699	2.290538e+10	1.000430	8.167255
max	123344.062500	3.509679e+11	4831.348633	9.245355e+10	1.011530	3.006686

Exploratory Data Analysis

In [16]:

```python
#Visualize the Closing Prices
# create a line plot to visualize the closing prices of all four cryptocurrencie

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()
```



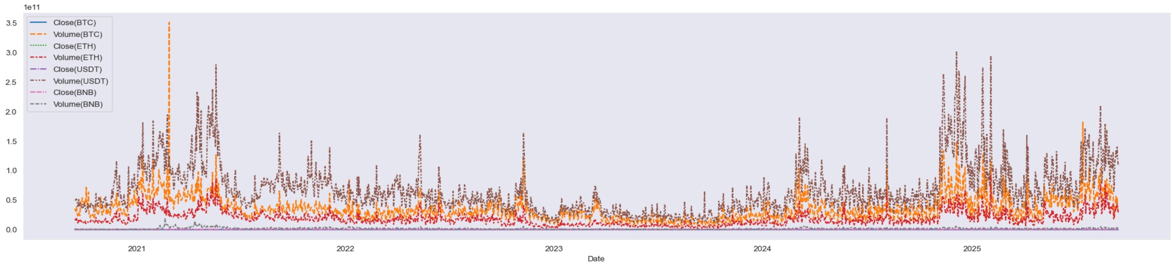
In [15]:

```python
print(data.columns)
```

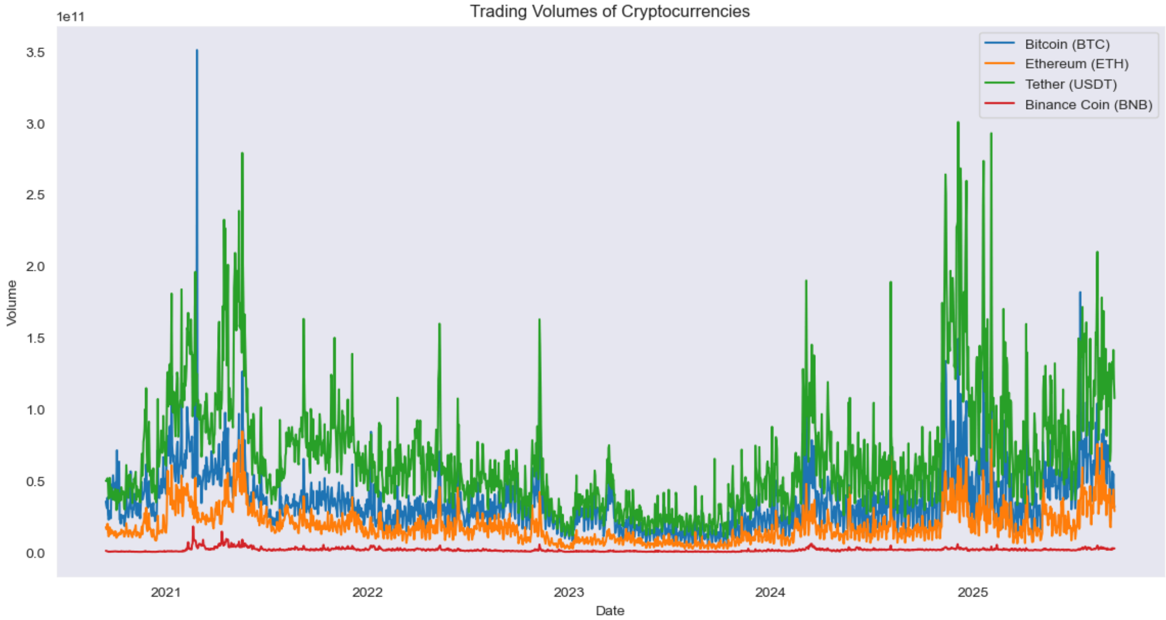
```
Index(['Close(BTC)', 'Volume(BTC)', 'Close(ETH)', 'Volume(ETH)', 'Close(USDT)',
       'Volume(USDT)', 'Close(BNB)', 'Volume(BNB)'],
      dtype='object')
```

In [18]:
```
plt.figure(figsize = (25,5))
sns.set_style('dark')
sns.lineplot(data=data)
```

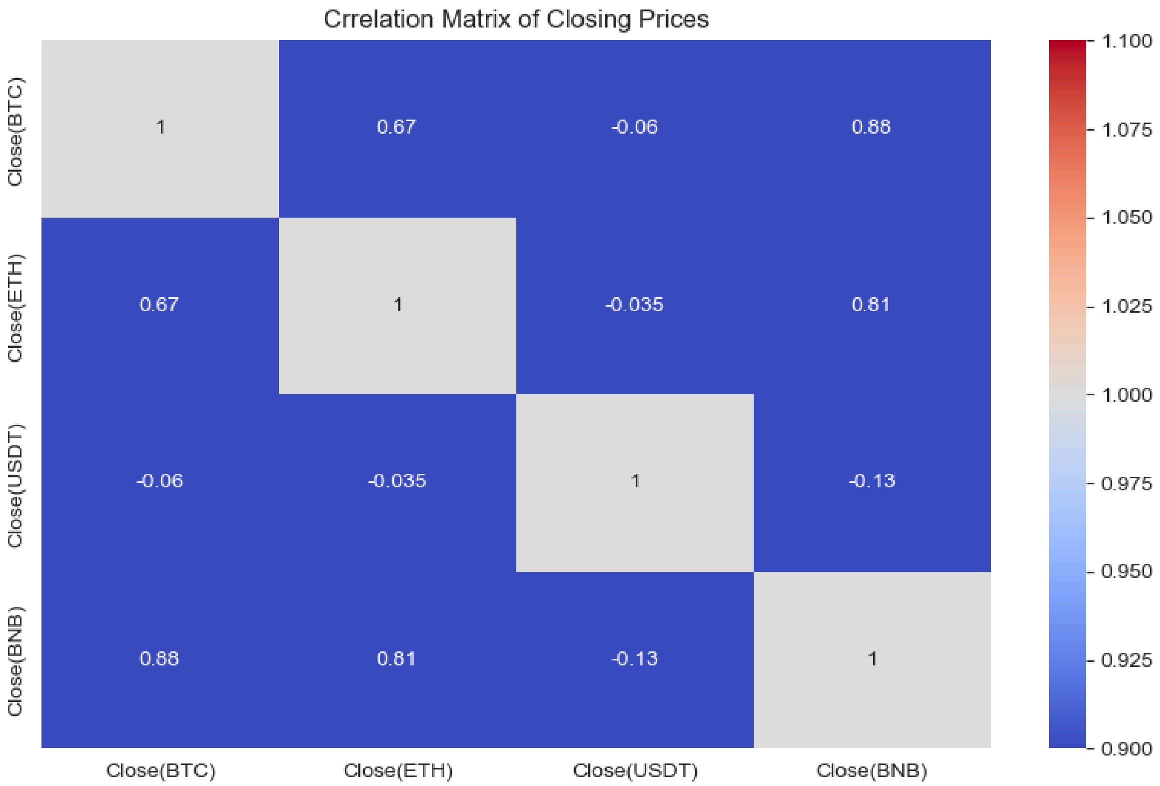
Out[18]: <Axes: xlabel='Date'>



In [21]:
```
# 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 [22]:
```
 #Correlation Analysis
#We'll analyze the correlation between the closing prices of the cryptocurrencie
# Calculate the correlation matrix

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

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

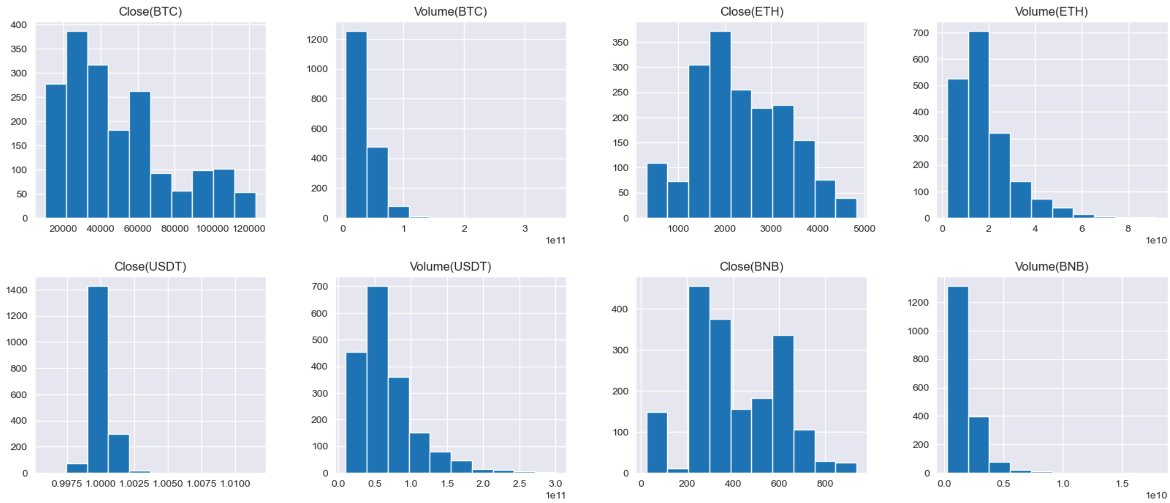


In [24]:

```
# 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()
```

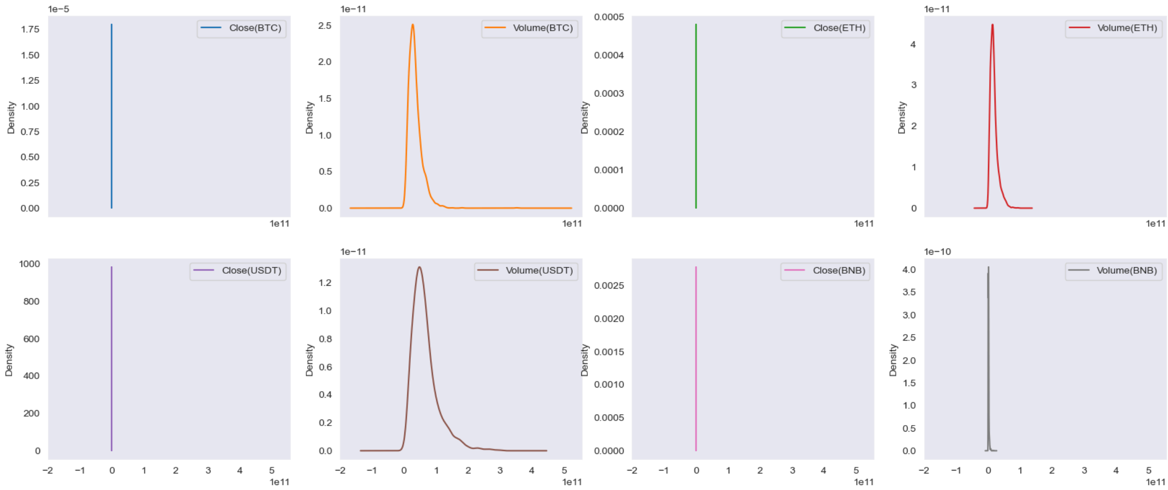
In [25]:
```
data.hist(figsize=(20,8), layout=(2,4))
```

Out[25]:
```
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 [26]:
```
data.plot(kind = "kde", subplots = True, layout = (2, 4), figsize = (20, 8))
```

Out[26]:
```
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 [28]:
```python
sns.pairplot(data.sample(n=100));
```

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

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

Out[30]:

Date	Volume(BTC)	Close(ETH)	Volume(ETH)	Close(USDT)	Volume(USDT)	C
2020-09-14 00:00:00+00:00	35453581940	377.268860	17536695361	1.001289	49936255991	
2020-09-15 00:00:00+00:00	32509451925	364.839203	16140584321	1.002487	49718173930	
2020-09-16 00:00:00+00:00	30769986455	365.812286	16107612177	1.003444	50682289026	
2020-09-17 00:00:00+00:00	38151810523	389.019226	19899531080	1.001878	51695424541	
2020-09-18 00:00:00+00:00	26341903912	384.364532	14108357740	0.999502	47248825663	

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

Out[31]:

Date	Volume(BTC)	Close(ETH)	Volume(ETH)	Close(USDT)	Volume(USDT)
2025-09-10 00:00:00+00:00	56377473784	4349.145996	39521365146	1.000138	133101421364
2025-09-11 00:00:00+00:00	45685065332	4461.233398	35959212991	1.000266	121507255807
2025-09-12 00:00:00+00:00	54785725894	4715.246094	43839753626	1.000618	141338448172
2025-09-13 00:00:00+00:00	34549454947	4668.179688	34843845977	1.000319	119042646333
2025-09-14 00:00:00+00:00	31612506112	4642.932617	28678072320	1.000401	107577327616

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

Out[32]:
```
Date
2020-09-14 00:00:00+00:00    10680.837891
2020-09-15 00:00:00+00:00    10796.951172
2020-09-16 00:00:00+00:00    10974.905273
2020-09-17 00:00:00+00:00    10948.990234
2020-09-18 00:00:00+00:00    10944.585938
Name: Close(BTC), dtype: float64
```

In [33]:
```
# 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_
```

In [34]:
```
# 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 [35]:
```
#SelectKBest
#SelectKBest is a feature selection method provided by scikit-learn (sklearn) th
#This function evaluates each feature independently and selects those that have

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

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\Hanshu\anaconda3\Lib\site-packages\sklearn\feature_selection\_univariate
_selection.py:109: RuntimeWarning: invalid value encountered in divide
  msw = sswn / 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(BN
B)'], dtype='object')
```

In [38]:
```
x_train
```

Out[38]:
```
array([[1.00051105e+00, 6.22330509e+10, 3.45933685e+02, 2.26992368e+09],
       [9.99966025e-01, 9.37161581e+10, 5.99706543e+02, 6.58591900e+09],
       [9.99065995e-01, 4.09658641e+10, 3.19609009e+02, 1.60429741e+09],
       ...,
       [1.00018799e+00, 5.84567573e+10, 6.29942871e+02, 2.23811852e+09],
       [1.00037599e+00, 5.91036805e+10, 4.30503265e+02, 1.49950509e+09],
       [1.00028896e+00, 5.74898269e+10, 2.87536133e+02, 1.57890533e+09]])
```

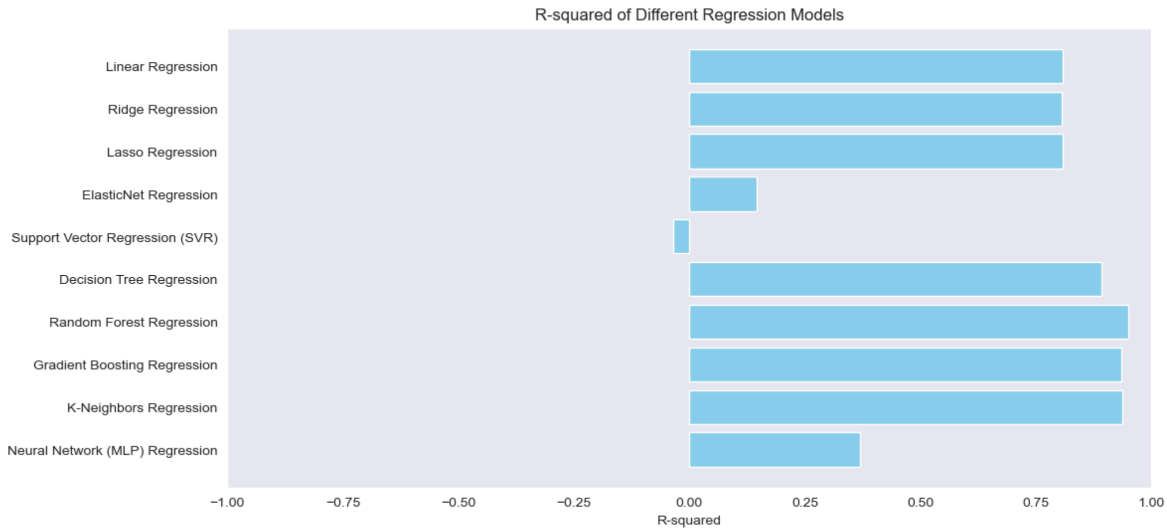
In [39]:
```
#MinMaxScaler is a preprocessing method in scikit-learn that transforms features
# It's often used when your data needs to be normalized within a specific range

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [42]:
```
# implementation of 10 different regression algorithms using scikit-learn. Each

#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 [49]:
```
#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,
    'K-Neighbors Regression': KNeighborsRegressor(n_neighbors=5),
    'Neural Network (MLP) Regression': MLPRegressor(hidden_layer_sizes=(100, 50)
    }

# 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 resultsre
    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): 150332109.8236788
R-squared: 0.8086771935356407

----- Ridge Regression -----
Mean Squared Error (MSE): 151893302.8553303
R-squared: 0.8066903137359858

----- Lasso Regression -----
Mean Squared Error (MSE): 150357484.84905022
R-squared: 0.8086448995628237

----- ElasticNet Regression -----
Mean Squared Error (MSE): 671371106.5323241
R-squared: 0.14556774044147236

----- Support Vector Regression (SVR) -----
Mean Squared Error (MSE): 813376297.152881
R-squared: -0.035157665686978756

----- Decision Tree Regression -----
Mean Squared Error (MSE): 85266293.85579413
R-squared: 0.8914843498409029

----- Random Forest Regression -----
Mean Squared Error (MSE): 39904889.378470555
R-squared: 0.9492143399271561

----- Gradient Boosting Regression -----
Mean Squared Error (MSE): 50841728.860370405
R-squared: 0.9352953785955983

----- K-Neighbors Regression -----
Mean Squared Error (MSE): 49073045.421996176
R-squared: 0.9375463247146522

c:\Users\Hanshu\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_pe
rceptron.py:690: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20
0) reached and the optimization hasn't converged yet.
  warnings.warn(
----- Neural Network (MLP) Regression -----
Mean Squared Error (MSE): 495536206.3573425
R-squared: 0.36934712207401244

                             Model           MSE  R-squared
0                Linear Regression  1.503321e+08   0.808677
1                 Ridge Regression  1.518933e+08   0.806690
2                 Lasso Regression  1.503575e+08   0.808645
3            ElasticNet Regression  6.713711e+08   0.145568
4  Support Vector Regression (SVR)  8.133763e+08  -0.035158
5         Decision Tree Regression  8.526629e+07   0.891484
6         Random Forest Regression  3.990489e+07   0.949214
7     Gradient Boosting Regression  5.084173e+07   0.935295
8           K-Neighbors Regression  4.907305e+07   0.937546
9  Neural Network (MLP) Regression  4.955362e+08   0.369347
```



In [ ]:

In [52]:

```
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)
```

```python
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): 38784117.98190298
Loaded Random Forest Regression - R-squared: 0.9506407093784183
```

In [ ]:

```python

```