Bitcoin Price Prediction

In [5]:
```
#yfinance is a popular Python library used for downloading historical market dat
#It simplifies the process of accessing financial data for various securities, i

# !pip install yfinance
```

In [6]:
```
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 [7]:
```
#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 [8]:
```
#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 [9]:
```
data.head()
```

Out[9]:

Date	Close (BTC)	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Vo (U
2020-09-11 00:00:00+00:00	10400.915039	45201121775	374.695587	27296269329	1.001517	460094:
2020-09-12 00:00:00+00:00	10442.170898	36750077324	387.183105	13295405814	1.001307	435063ł
2020-09-13 00:00:00+00:00	10323.755859	36506852789	365.570007	15005899191	0.999213	463306
2020-09-14 00:00:00+00:00	10680.837891	35453581940	377.268860	17536695361	1.001289	499362
2020-09-15 00:00:00+00:00	10796.951172	32509451925	364.839203	16140584321	1.002487	497181

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

Out[10]:

Date	Close (BTC)	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	
2025-09-07 00:00:00+00:00	111167.617188	24618007520	4305.347656	17426783536	1.000093	707
2025-09-08 00:00:00+00:00	112071.429688	40212813407	4308.072266	32277142378	0.999906	1147
2025-09-09 00:00:00+00:00	111530.546875	45984480722	4309.041504	30703320925	1.000065	1245
2025-09-10 00:00:00+00:00	113955.359375	56377473784	4349.145996	39521365146	1.000138	1331
2025-09-11 00:00:00+00:00	114178.812500	51175231488	4445.397461	41069117440	0.999936	1309

In [11]:
```
data.shape
```

Out[11]: (1827, 8)

In [12]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1827 entries, 2020-09-11 00:00:00+00:00 to 2025-09-11 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 [13]:
```
data.isna().sum()
```

Out[13]:
```
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 [14]:
```
data.describe()
```

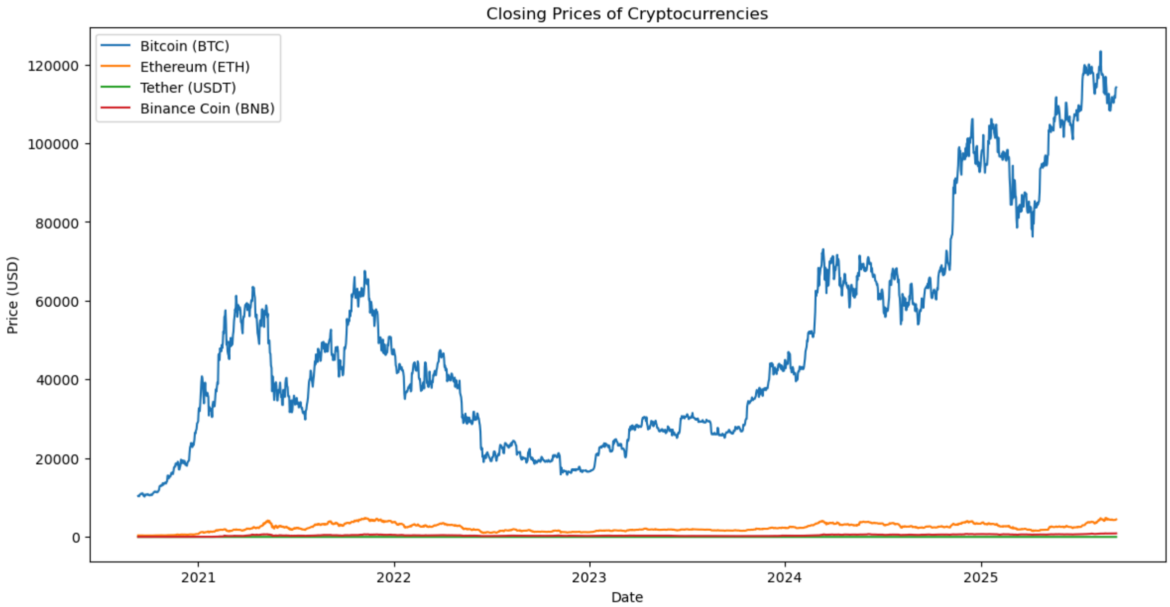
Out[14]:

	Close (BTC)	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Volu (US
count	1827.000000	1.827000e+03	1827.000000	1.827000e+03	1827.000000	1.827000e
mean	49149.395104	3.571333e+10	2343.574669	1.839726e+10	1.000188	6.647445e
std	28015.908477	2.156439e+10	982.181444	1.189867e+10	0.000738	4.137050e
min	10246.186523	5.331173e+09	321.116302	2.081626e+09	0.995872	9.989859e
25%	26984.352539	2.136936e+10	1646.813049	1.024022e+10	0.999920	3.915764e
50%	42358.808594	3.115874e+10	2245.430420	1.578392e+10	1.000157	5.703328e
75%	63806.531250	4.430770e+10	3103.291260	2.287331e+10	1.000430	8.139610e
max	123344.062500	3.509679e+11	4831.348633	9.245355e+10	1.011530	3.006686e

Exploratory Data Analysis

In [15]:
```
# Visualise the Closing Price
# Create a line plot to visualise 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)')
```

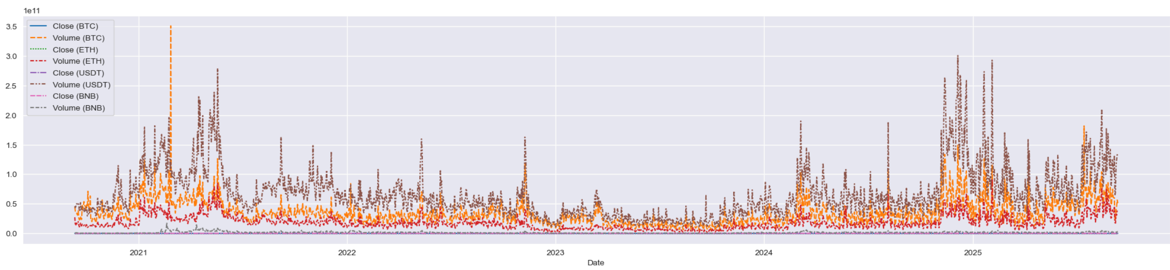
```python
plt.plot(data.index, data['Close (USDT)'], label='Tether (USDT)')
plt.plot(data.index, data['Close (BNB)'], label='Binance Coin (BNB)')
plt.title('Closing Prices of Cryptocurrencies')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.show()
```



In [16]:
```python
plt.figure(figsize=(25,5))
sns.set_style('darkgrid')
sns.lineplot(data=data)
```

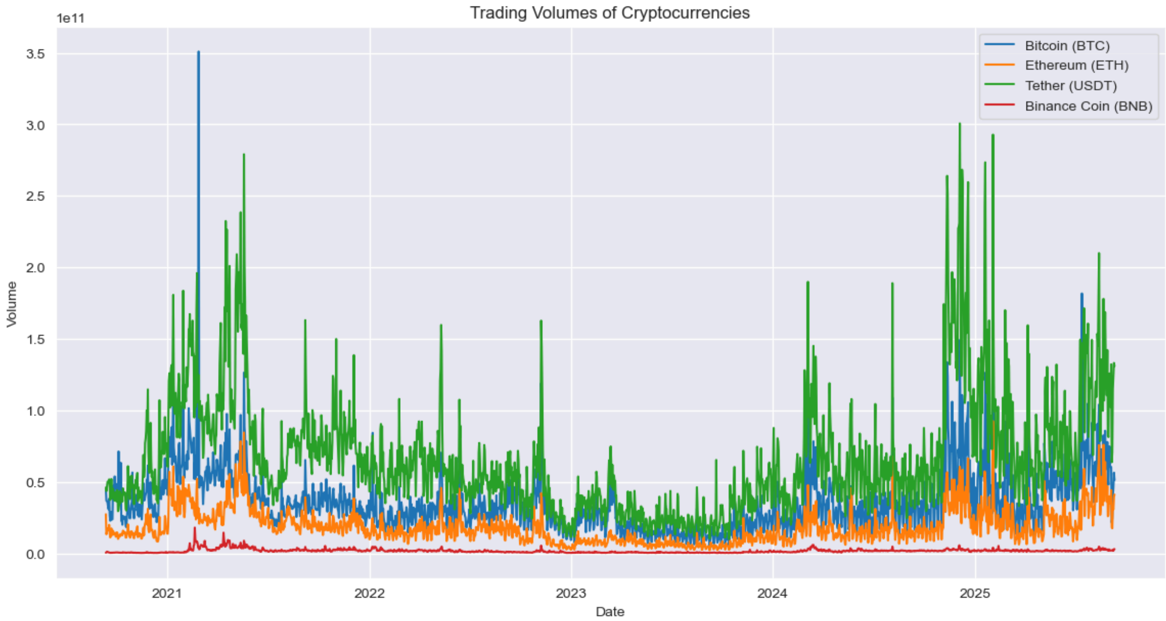
```
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[16]: <Axes: xlabel='Date'>



In [17]:
```python
# 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')
```

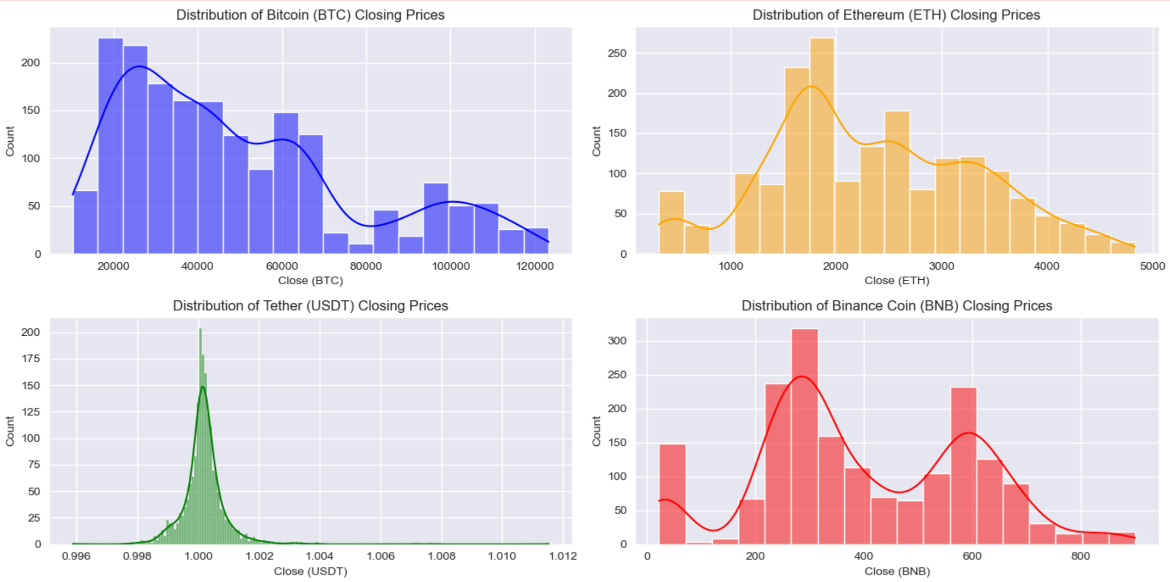
```
plt.legend()
plt.show()
```



In [18]:
```
# 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)']]

# 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()
```

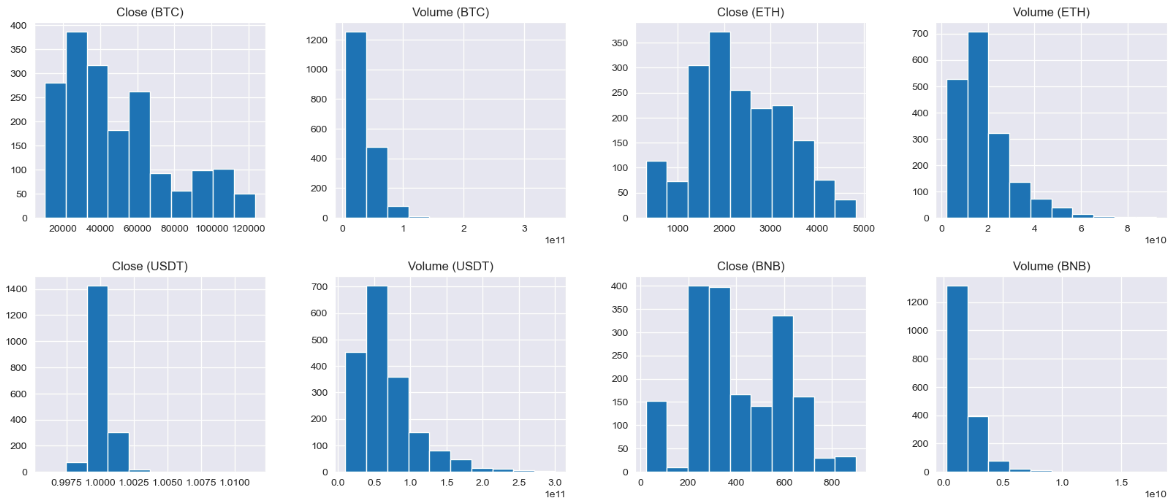
In [19]:
```python
# Distribution of Closing Prices
#Let's plot the distribution of closing prices for each cryptocurrency:
plt.figure(figsize=(14, 7))

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()
```

```
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



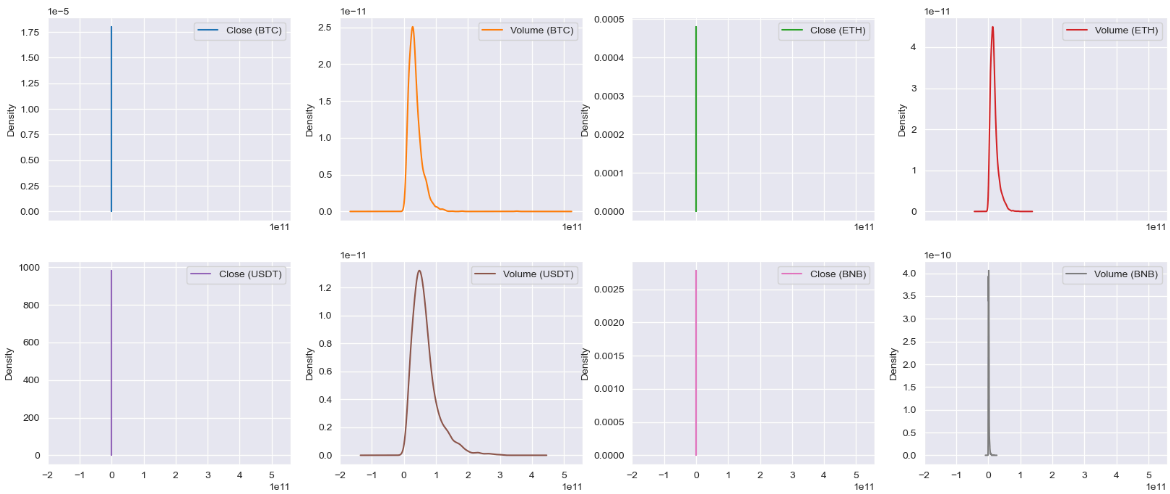
In [20]:
```python
data.hist(figsize=(20,8),layout=(2,4))
```

```
Out[20]: 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)
```



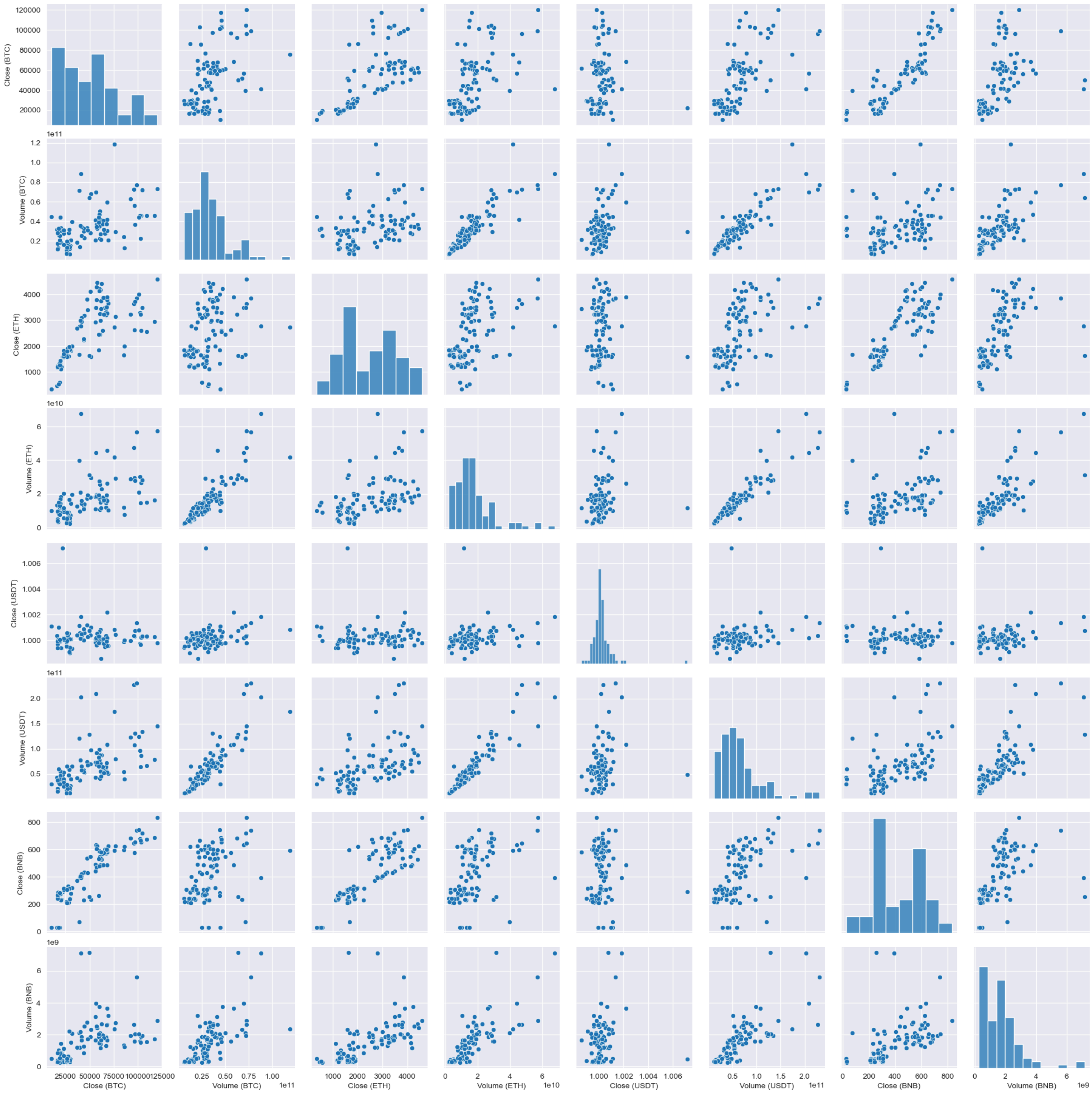

```
In [21]: data.plot(kind="kde", subplots = True, layout=(2,4), figsize=(20,8))
```

```
Out[21]: 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 [22]: sns.pairplot(data.sample(n=100));
```

```
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarni
ng: use_inf_as_na option is deprecated and will be removed in a future version. C
onvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Prachi\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Data Pre-processing

In [23]:
```
X = data.drop(columns = ['Close (BTC)'], axis=1)
Y =data.loc[:, 'Close (BTC)']
```

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

Out[24]:

Date	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Volume (USDT)	Clo (BNI
2020-09-11 00:00:00+00:00	45201121775	374.695587	27296269329	1.001517	46009434500	25.4514
2020-09-12 00:00:00+00:00	36750077324	387.183105	13295405814	1.001307	43506381696	28.5298
2020-09-13 00:00:00+00:00	36506852789	365.570007	15005899191	0.999213	46330693824	31.0586
2020-09-14 00:00:00+00:00	35453581940	377.268860	17536695361	1.001289	49936255991	31.1786
2020-09-15 00:00:00+00:00	32509451925	364.839203	16140584321	1.002487	49718173930	27.2023

In [25]:
```
X.tail()
```

Out[25]:

Date	Volume (BTC)	Close (ETH)	Volume (ETH)	Close (USDT)	Volume (USDT)	(
2025-09-07 00:00:00+00:00	24618007520	4305.347656	17426783536	1.000093	70719934554	880.6
2025-09-08 00:00:00+00:00	40212813407	4308.072266	32277142378	0.999906	114790407663	878.2
2025-09-09 00:00:00+00:00	45984480722	4309.041504	30703320925	1.000065	124541484302	880.0
2025-09-10 00:00:00+00:00	56377473784	4349.145996	39521365146	1.000138	133101421364	893.5
2025-09-11 00:00:00+00:00	51175231488	4445.397461	41069117440	0.999936	130951036928	899.4

In [26]:
```
Y.head()
```

Out[26]:
```
Date
2020-09-11 00:00:00+00:00    10400.915039
2020-09-12 00:00:00+00:00    10442.170898
2020-09-13 00:00:00+00:00    10323.755859
2020-09-14 00:00:00+00:00    10680.837891
2020-09-15 00:00:00+00:00    10796.951172
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_
```

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: (1461, 7)
X_test shape: (366, 7)
y_train shape: (1461,)
y_test shape: (366,)
```

In [29]:
```
#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\Prachi\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([[1.00118995e+00, 9.67272038e+10, 3.54001373e+02, 4.49112231e+09],
       [1.00004196e+00, 1.14746493e+11, 5.34577698e+02, 4.07286688e+09],
       [9.98970985e-01, 5.96518368e+10, 3.06791565e+02, 1.64688295e+09],
       ...,
       [9.99877989e-01, 3.30019976e+10, 6.26574402e+02, 1.18274185e+09],
       [1.00059998e+00, 7.78429044e+10, 4.14126984e+02, 1.83145122e+09],
       [1.00027394e+00, 6.01045912e+10, 2.71504364e+02, 1.40551024e+09]])
```

In [32]:
```
#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 [33]:
```
# 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
```

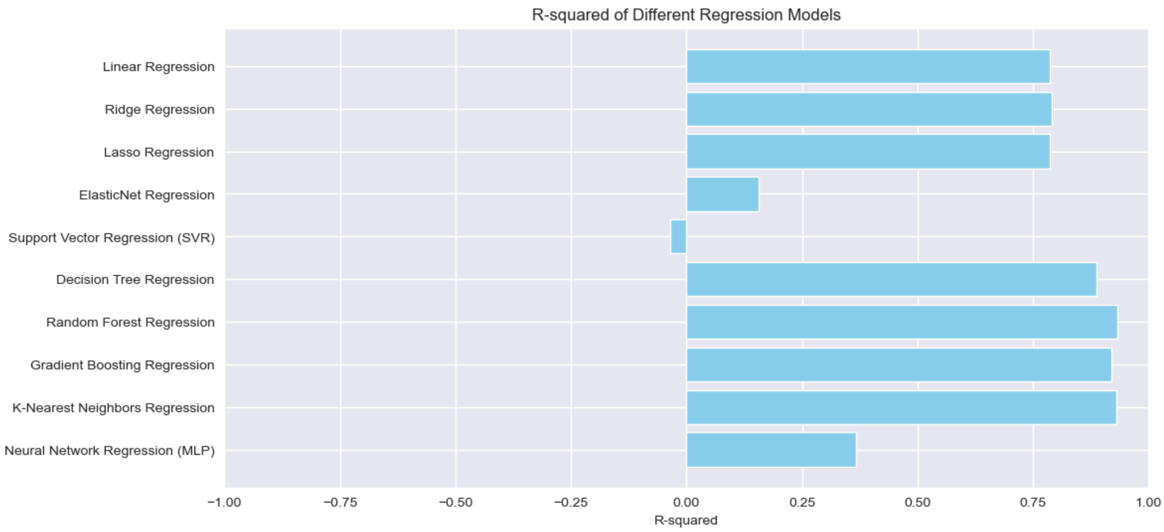
```python
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

In [34]:
```python
# 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-Nearest Neighbors Regression': KNeighborsRegressor(n_neighbors=5),
    'Neural Network Regression (MLP)': 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 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): 163492150.0312961
R-squared: 0.7866434530827466

----- Ridge Regression -----
Mean Squared Error (MSE): 161056560.76835373
R-squared: 0.7898218865105942

----- Lasso Regression -----
Mean Squared Error (MSE): 163425165.5279935
R-squared: 0.7867308675691234

----- ElasticNet Regression -----
Mean Squared Error (MSE): 645848338.6043692
R-squared: 0.15717071840760588

----- Support Vector Regression (SVR) -----
Mean Squared Error (MSE): 793375059.4484557
R-squared: -0.03535101264366913

----- Decision Tree Regression -----
Mean Squared Error (MSE): 86401190.56297068
R-squared: 0.8872468209358904

----- Random Forest Regression -----
Mean Squared Error (MSE): 50895765.2658949
R-squared: 0.9335812470032123

----- Gradient Boosting Regression -----
Mean Squared Error (MSE): 62031191.505255446
R-squared: 0.9190495640421223

----- K-Nearest Neighbors Regression -----
Mean Squared Error (MSE): 54098342.736261815
R-squared: 0.9294018972902035
```

```
c:\Users\Prachi\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 Regression (MLP) -----
Mean Squared Error (MSE): 485773821.82147366
R-squared: 0.366067268753961

                            Model           MSE  R-squared
0               Linear Regression  1.634922e+08   0.786643
1                Ridge Regression  1.610566e+08   0.789822
2                Lasso Regression  1.634252e+08   0.786731
3           ElasticNet Regression  6.458483e+08   0.157171
4  Support Vector Regression (SVR)  7.933751e+08  -0.035351
5         Decision Tree Regression  8.640119e+07   0.887247
6         Random Forest Regression  5.089577e+07   0.933581
7     Gradient Boosting Regression  6.203119e+07   0.919050
8   K-Nearest Neighbors Regression  5.409834e+07   0.929402
9   Neural Network Regression (MLP)  4.857738e+08   0.366067
```



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 [36]:
```
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)
```

```python
# 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): 51501201.45687739
Loaded Random Forest Regression - R-squared: 0.9327911553990463
```

In [ ]:

In [ ]: