# Bitcoin Price Prediction

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 [ ]:
```
data = pd.read_csv(r"C:\Users\JANHAVI\Desktop\crypto_data_updated_13_november.csv")
```

In [7]:
```
btc = yf.Ticker('BTC-USD')
prices1 = btc.history(period='5y')
prices1.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits'], axis = 1

eth = yf.Ticker('ETH-USD')
prices2 = eth.history(period='5y')
prices2.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits'], axis = 1

usdt = yf.Ticker('USDT-USD')
prices3 = usdt.history(period='5y')
prices3.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits'], axis = 1

bnb = yf.Ticker('BNB-USD')
prices4 = bnb.history(period='5y')
prices4.drop(columns=['Open', 'High', 'Low', 'Dividends', 'Stock Splits'], axis = 1
```

In [8]:
```
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) | Volume (USDT) | ( |
|---|---|---|---|---|---|---|---|
| 2020-09-14 00:00:00+00:00 | 10680.837891 | 35453581940 | 377.268860 | 17536695361 | 1.001289 | 49936255991 | 31.1 |
| 2020-09-15 00:00:00+00:00 | 10796.951172 | 32509451925 | 364.839203 | 16140584321 | 1.002487 | 49718173930 | 27.2 |
| 2020-09-16 00:00:00+00:00 | 10974.905273 | 30769986455 | 365.812286 | 16107612177 | 1.003444 | 50682289026 | 27.9 |
| 2020-09-17 00:00:00+00:00 | 10948.990234 | 38151810523 | 389.019226 | 19899531080 | 1.001878 | 51695424541 | 26.9 |
| 2020-09-18 00:00:00+00:00 | 10944.585938 | 26341903912 | 384.364532 | 14108357740 | 0.999502 | 47248825663 | 27.3 |

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

Out[10]:

| Date | Close (BTC) | Volume (BTC) | Close (ETH) | Volume (ETH) | Close (USDT) | Volume (USDT) | |
|---|---|---|---|---|---|---|---|
| 2025-09-10 00:00:00+00:00 | 113955.359375 | 56377473784 | 4349.145996 | 39521365146 | 1.000138 | 133101421364 | ε |
| 2025-09-11 00:00:00+00:00 | 115507.539062 | 45685065332 | 4461.233398 | 35959212991 | 1.000266 | 121507255807 | ς |
| 2025-09-12 00:00:00+00:00 | 116101.578125 | 54785725894 | 4715.246094 | 43839753626 | 1.000618 | 141338448172 | ς |
| 2025-09-13 00:00:00+00:00 | 115950.507812 | 34549454947 | 4668.179688 | 34843845977 | 1.000319 | 119042646333 | ς |
| 2025-09-14 00:00:00+00:00 | 115639.640625 | 30940768256 | 4626.679688 | 27470587904 | 1.000392 | 104634056704 | ς |

In [11]:
```
data.shape
```
Out[11]:
```
(1827, 8)
```

In [12]:
```
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+00:0
0
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: 193.0 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()
```

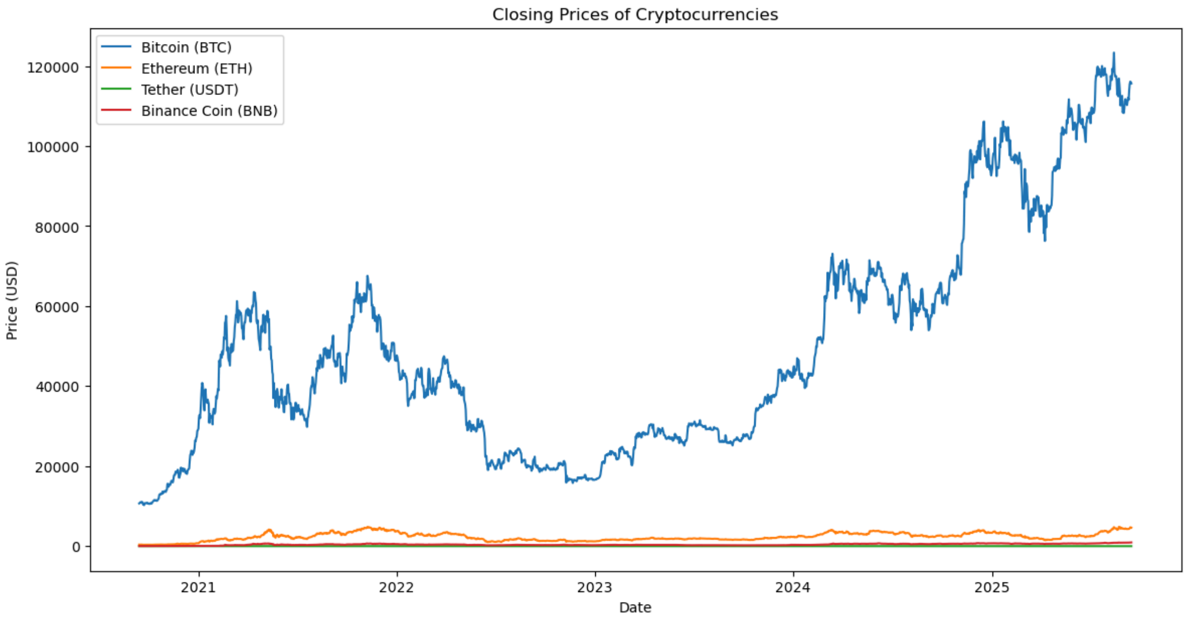
Out[14]:

| | Close (BTC) | Volume (BTC) | Close (ETH) | Volume (ETH) | Close (USDT) | Volume (USDT) | Close |
|---|---|---|---|---|---|---|---|
| **count** | 1827.000000 | 1.827000e+03 | 1827.000000 | 1.827000e+03 | 1827.000000 | 1.827000e+03 | 1827.0 |
| **mean** | 49323.370808 | 3.571132e+10 | 2350.634599 | 1.842213e+10 | 1.000188 | 6.659472e+10 | 400.3 |
| **std** | 28103.521768 | 2.156637e+10 | 983.463178 | 1.191423e+10 | 0.000737 | 4.141890e+10 | 197.7 |
| **min** | 10246.186523 | 5.331173e+09 | 321.116302 | 2.081626e+09 | 0.995872 | 9.989859e+09 | 22.8 |
| **25%** | 27055.889648 | 2.136936e+10 | 1649.178711 | 1.024022e+10 | 0.999922 | 3.915764e+10 | 264.8 |
| **50%** | 42412.433594 | 3.114168e+10 | 2260.648682 | 1.581725e+10 | 1.000157 | 5.707433e+10 | 348.2 |
| **75%** | 63842.345703 | 4.430770e+10 | 3107.366699 | 2.290538e+10 | 1.000430 | 8.167255e+10 | 579.6 |
| **max** | 123344.062500 | 3.509679e+11 | 4831.348633 | 9.245355e+10 | 1.011530 | 3.006686e+11 | 933.8 |

# Exploratory Data Analysis

In [15]:

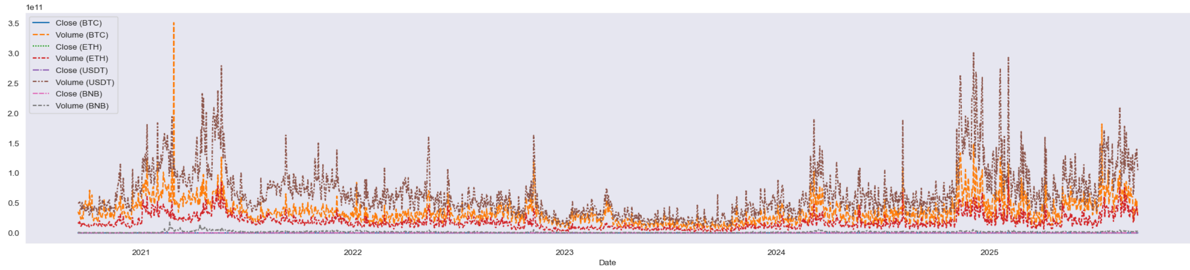
```
plt.figure(figsize=(14, 7))
plt.plot(data.index, data['Close (BTC)'], label='Bitcoin (BTC)')
plt.plot(data.index, data['Close (ETH)'], label='Ethereum (ETH)')
plt.plot(data.index, data['Close (USDT)'], label='Tether (USDT)')
plt.plot(data.index, data['Close (BNB)'], label='Binance Coin (BNB)')
plt.title('Closing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



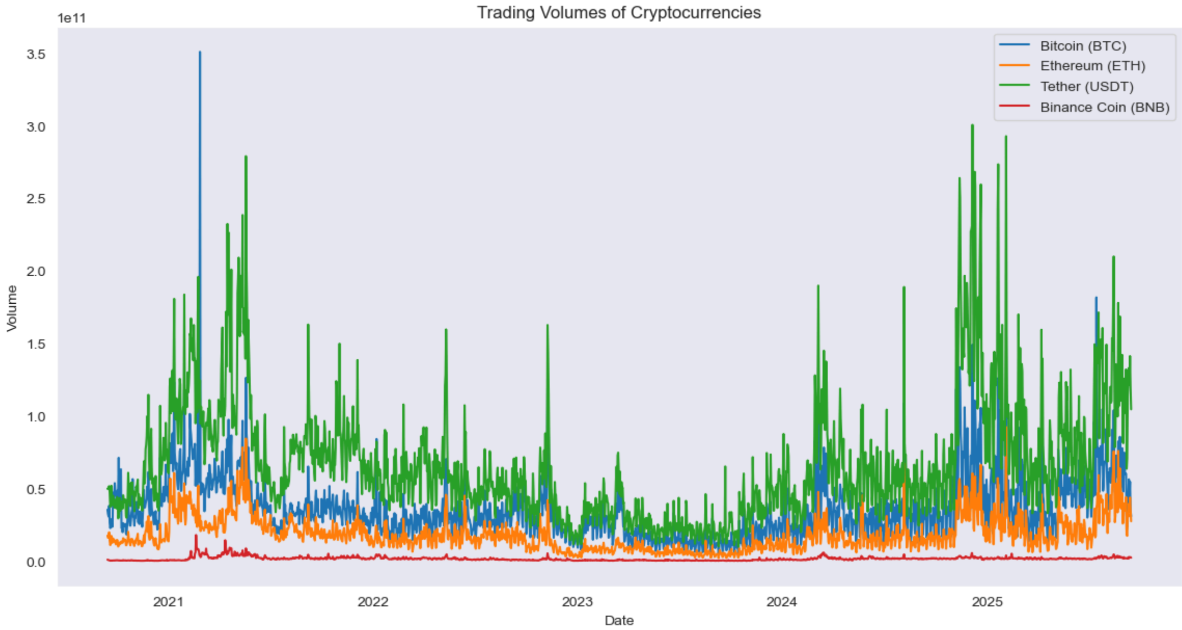

In [16]:

```
plt.figure(figsize = (25, 5))
sns.set_style('dark')
sns.lineplot(data=data)
```

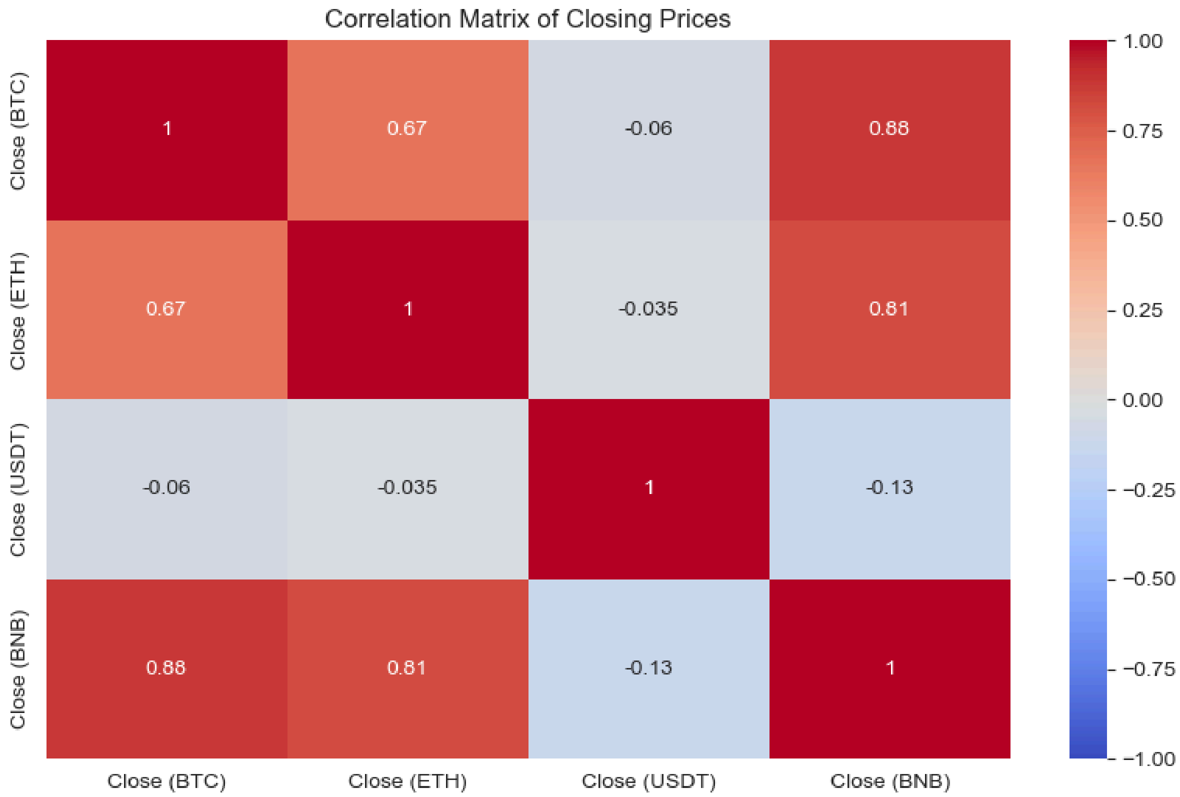
Out[16]: <Axes: xlabel='Date'>




In [17]:
```python
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()
```

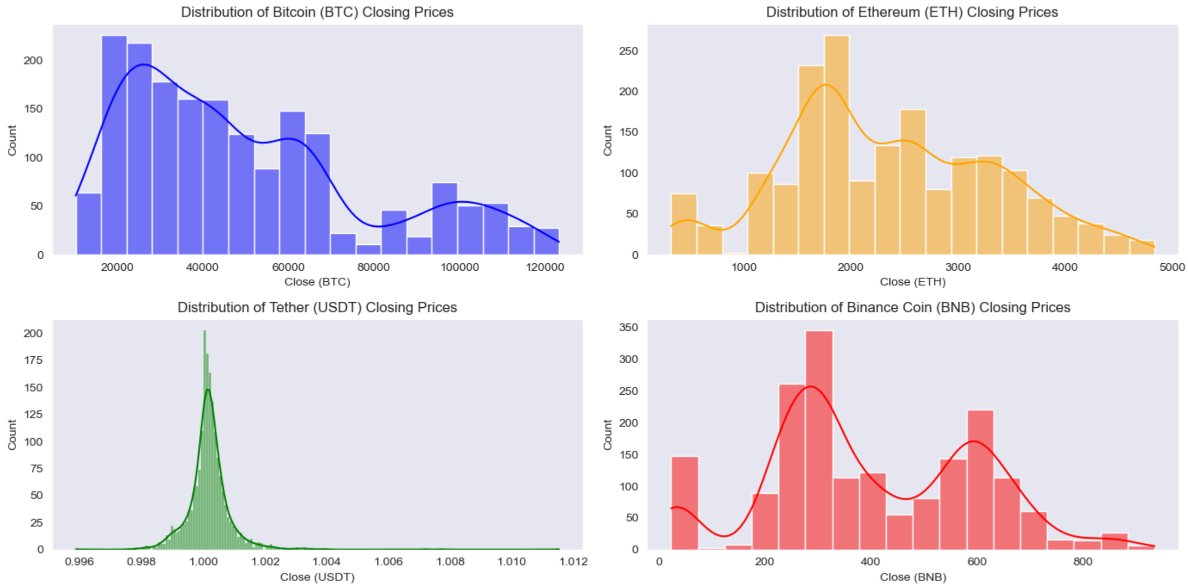



In [18]:
```python
corr_matrix = data[['Close (BTC)', 'Close (ETH)', 'Close (USDT)', 'Close (BNB)']].c

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



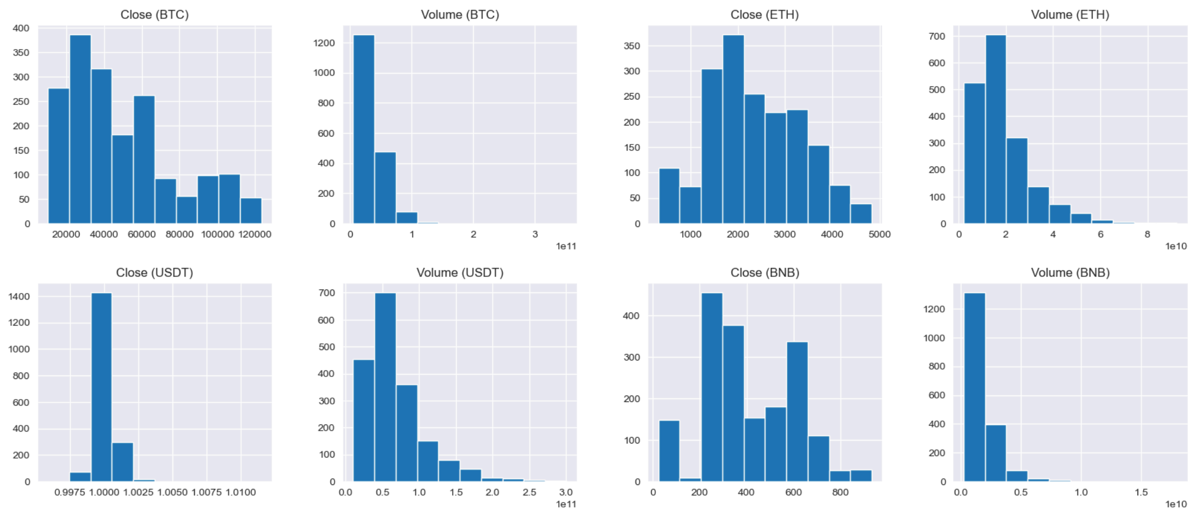

In [20]:

```
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()
```



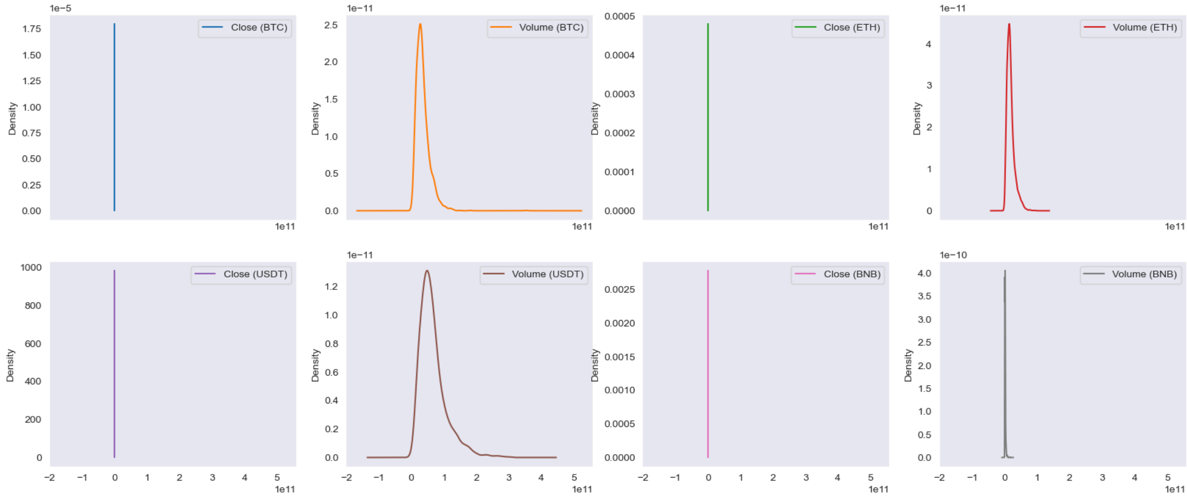

In [21]:
```
data.hist(figsize=(20, 8), layout=(2, 4))
```

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)
```

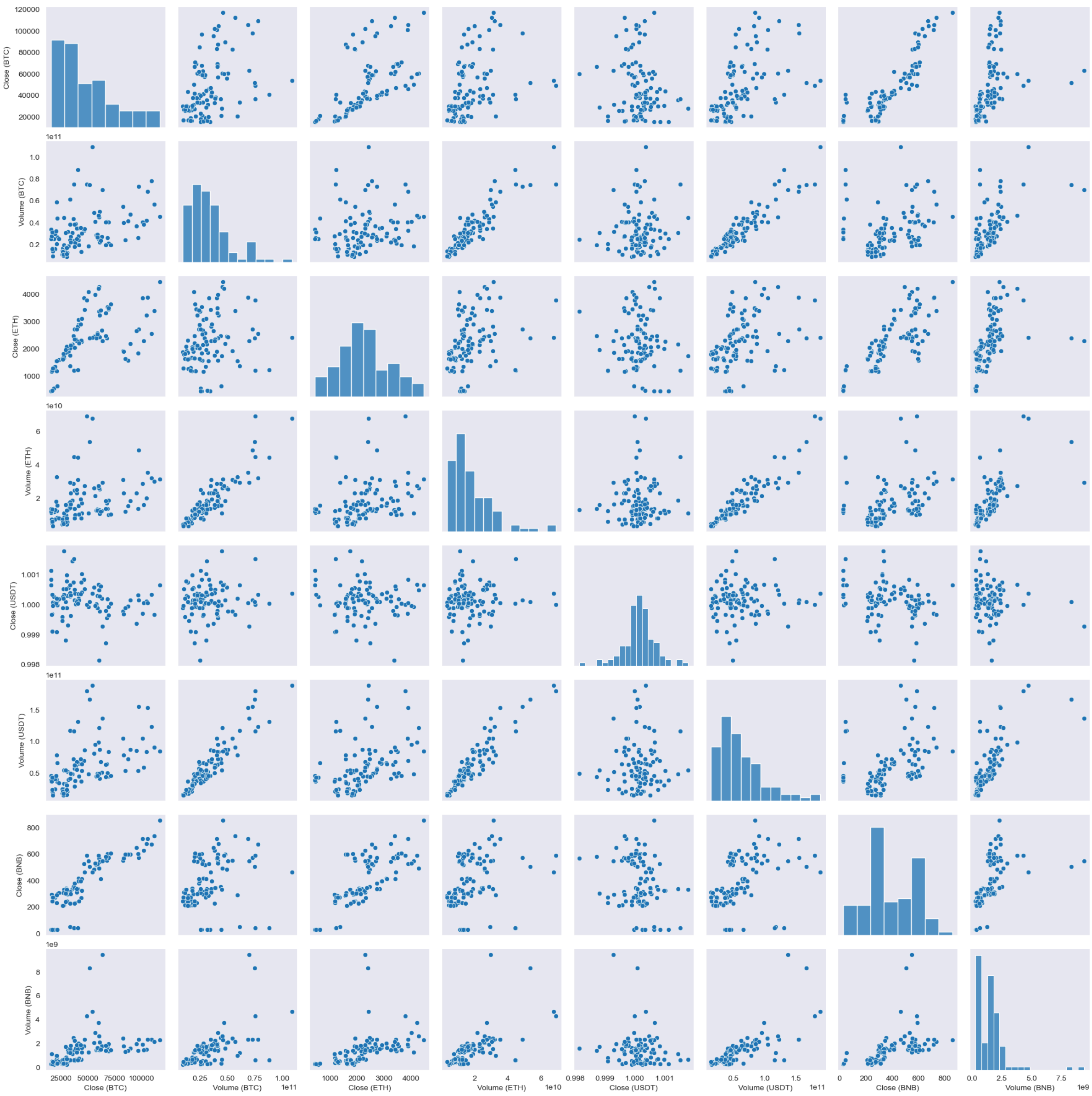



In [22]:
```
data.plot(kind = "kde", subplots = True, layout = (2, 4), figsize = (20, 8))
```

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));
```




# Data Preprocessing

In [24]:
```
X = data.drop(columns = ['Close (BTC)'], axis = 1)
Y = data.loc[:, 'Close (BTC)']
```

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

Out[25]:

| Date | Volume (BTC) | Close (ETH) | Volume (ETH) | Close (USDT) | Volume (USDT) | Close (BNB) | Volu (B |
|---|---|---|---|---|---|---|---|
| 2020-09-14 00:00:00+00:00 | 35453581940 | 377.268860 | 17536695361 | 1.001289 | 49936255991 | 31.178642 | 1009392 |
| 2020-09-15 00:00:00+00:00 | 32509451925 | 364.839203 | 16140584321 | 1.002487 | 49718173930 | 27.202391 | 861821 |
| 2020-09-16 00:00:00+00:00 | 30769986455 | 365.812286 | 16107612177 | 1.003444 | 50682289026 | 27.964594 | 664539 |
| 2020-09-17 00:00:00+00:00 | 38151810523 | 389.019226 | 19899531080 | 1.001878 | 51695424541 | 26.993130 | 512578 |
| 2020-09-18 00:00:00+00:00 | 26341903912 | 384.364532 | 14108357740 | 0.999502 | 47248825663 | 27.399481 | 482149 |

In [26]:
```
X.tail()
```

Out[26]:

| Date | Volume (BTC) | Close (ETH) | Volume (ETH) | Close (USDT) | Volume (USDT) | Close (BNB) | V |
|---|---|---|---|---|---|---|---|
| 2025-09-10 00:00:00+00:00 | 56377473784 | 4349.145996 | 39521365146 | 1.000138 | 133101421364 | 893.566589 | 2868 |
| 2025-09-11 00:00:00+00:00 | 45685065332 | 4461.233398 | 35959212991 | 1.000266 | 121507255807 | 902.983337 | 2250 |
| 2025-09-12 00:00:00+00:00 | 54785725894 | 4715.246094 | 43839753626 | 1.000618 | 141338448172 | 925.030701 | 2648 |
| 2025-09-13 00:00:00+00:00 | 34549454947 | 4668.179688 | 34843845977 | 1.000319 | 119042646333 | 933.899658 | 2744 |
| 2025-09-14 00:00:00+00:00 | 30940768256 | 4626.679688 | 27470587904 | 1.000392 | 104634056704 | 928.769043 | 2386 |

In [27]:
```
Y.head()
```

Out[27]:
```
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 [28]:
```
# 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_sta
```

In [29]:
```
# 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 [33]:
```
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)
```

In [36]:
```
import pandas as pd
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import train_test_split

# Sample dataset for demonstration (replace this with your actual data)
data = {
    'feature1': [1, 2, 3, 4, 5],
    'feature2': [5, 4, 3, 2, 1],
    'feature3': [2, 3, 4, 5, 6],
    'feature4': [7, 8, 9, 10, 11],
    'feature5': [1, 3, 5, 7, 9],
    'feature6': [9, 7, 5, 3, 1],
    'feature7': [4, 4, 4, 4, 4]
}
df = pd.DataFrame(data)

# Target variable
y = pd.Series([0, 1, 0, 1, 0])

# Features
X = df

# Split the data (optional but recommended)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Feature selection: select best 4 features based on ANOVA F-value
fs = SelectKBest(score_func=f_classif, k=4)
fs.fit(X_train, y_train)

# Get the mask and ensure it matches the columns
mask = fs.get_support()

print("X_train shape:", X_train.shape)
print("Mask shape:", mask.shape)

# Select features from the training set using the mask
selected_features = X_train.columns[mask]

print("Selected Features:", selected_features)
```

```
X_train shape: (4, 7)
Mask shape: (7,)
Selected Features: Index(['feature3', 'feature4', 'feature5', 'feature6'], dtype
='object')
```

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

Out[37]:

| | feature1 | feature2 | feature3 | feature4 | feature5 | feature6 | feature7 |
|---|---|---|---|---|---|---|---|
| **4** | 5 | 1 | 6 | 11 | 9 | 1 | 4 |
| **2** | 3 | 3 | 4 | 9 | 5 | 5 | 4 |
| **0** | 1 | 5 | 2 | 7 | 1 | 9 | 4 |
| **3** | 4 | 2 | 5 | 10 | 7 | 3 | 4 |

In [38]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [39]:
```
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 [42]:
```
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
from sklearn.model_selection import train_test_split
import pandas as pd
import matplotlib.pyplot as plt

# Example: Load your dataset
# X, Y = your data here

# Split the dataset properly
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_sta

print("X_train shape:", X_train.shape)
print("Y_train shape:", Y_train.shape)

# Define models
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, lea
    'K-Nearest Neighbors Regression': KNeighborsRegressor(n_neighbors=5),
    'Neural Network Regression (MLP)': MLPRegressor(hidden_layer_sizes=(100, 50), a
}

# Train and evaluate
results = {'Model': [], 'MSE': [], 'R-squared': []}

for name, model in models.items():
    model.fit(X_train, Y_train)
```

```
    Y_pred = model.predict(X_test)
    mse = mean_squared_error(Y_test, Y_pred)
    r2 = r2_score(Y_test, Y_pred)
    results['Model'].append(name)
    results['MSE'].append(mse)
    results['R-squared'].append(r2)
    print(f"----- {name} -----")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"R-squared: {r2}")
    print()

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

# Plot 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()
```

```
X_train shape: (800, 10)
Y_train shape: (800,)
----- Linear Regression -----
Mean Squared Error (MSE): 0.011095733155955564
R-squared: 0.9999993922864031

----- Ridge Regression -----
Mean Squared Error (MSE): 0.05345544205131381
R-squared: 0.9999970722440326

----- Lasso Regression -----
Mean Squared Error (MSE): 10.094820186755193
R-squared: 0.999447106432796

----- ElasticNet Regression -----
Mean Squared Error (MSE): 2615.3001058323466
R-squared: 0.8567599444000146

----- Support Vector Regression (SVR) -----
Mean Squared Error (MSE): 14560.025852262783
R-squared: 0.20254700102511236

----- Decision Tree Regression -----
Mean Squared Error (MSE): 12183.987057665921
R-squared: 0.3326827083142224

----- Random Forest Regression -----
Mean Squared Error (MSE): 3706.7534086372325
R-squared: 0.7969810182913404

----- Gradient Boosting Regression -----
Mean Squared Error (MSE): 2080.9395293400867
R-squared: 0.8860268872325012

----- K-Nearest Neighbors Regression -----
Mean Squared Error (MSE): 4109.233337384071
R-squared: 0.7749371820053983

----- Neural Network Regression (MLP) -----
Mean Squared Error (MSE): 40.12253897592816
R-squared: 0.9978024874847411

                            Model           MSE  R-squared
0               Linear Regression      0.011096   0.999999
1                Ridge Regression      0.053455   0.999997
2                Lasso Regression     10.094820   0.999447
3           ElasticNet Regression   2615.300106   0.856760
4  Support Vector Regression (SVR)  14560.025852   0.202547
5        Decision Tree Regression  12183.987058   0.332683
6        Random Forest Regression   3706.753409   0.796981
7    Gradient Boosting Regression   2080.939529   0.886027
8   K-Nearest Neighbors Regression   4109.233337   0.774937
9  Neural Network Regression (MLP)     40.122539   0.997802
```


In [44]:
```
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): 3435.5407260447582
Loaded Random Forest Regression - R-squared: 0.8118353440520176
```