Bitcoin Price Prediction Model

In [1]:
```
# Import necessary libraries for data manipulation
import numpy as np
import pandas as pd
from warnings import simplefilter
simplefilter('ignore')#ignores warnings
```

In [2]:
```
# Create a path
path = ('C:/Users/Sanayak/Desktop/archive (11)/BTC.csv')

# Load to a pandas datafarme
df_btc = pd.read_csv(path)

# inspect the the first 5 entries of the dataset
df_btc.head()
```

Out[2]:

	ticker	date	open	high	low	close
0	BTC	2010-07-17	0.04951	0.04951	0.04951	0.04951
1	BTC	2010-07-18	0.04951	0.08585	0.04951	0.08584
2	BTC	2010-07-19	0.08584	0.09307	0.07723	0.08080
3	BTC	2010-07-20	0.08080	0.08181	0.07426	0.07474
4	BTC	2010-07-21	0.07474	0.07921	0.06634	0.07921

In [3]:
```
# Convert date to datetime
df_btc['Date'] = pd.to_datetime(df_btc['date'])

# Create additional features
df_btc['Day'] = df_btc['Date'].dt.day
df_btc['Month'] = df_btc['Date'].dt.month
df_btc['Year'] = df_btc['Date'].dt.year
df_btc['Day_of_Week'] = df_btc['Date'].dt.dayofweek
df_btc.head()
```

Out[3]:

	ticker	date	open	high	low	close	Date	Day	Month	Year	Day_of_Week
0	BTC	2010-07-17	0.04951	0.04951	0.04951	0.04951	2010-07-17	17	7	2010	5
1	BTC	2010-07-18	0.04951	0.08585	0.04951	0.08584	2010-07-18	18	7	2010	6
2	BTC	2010-07-19	0.08584	0.09307	0.07723	0.08080	2010-07-19	19	7	2010	0
3	BTC	2010-07-20	0.08080	0.08181	0.07426	0.07474	2010-07-20	20	7	2010	1
4	BTC	2010-07-21	0.07474	0.07921	0.06634	0.07921	2010-07-21	21	7	2010	2

In [4]:
```
# Create a new dataframe with relevant feature and target variable
new_df = df_btc[['Day', 'Month', 'Year','Day_of_Week', 'close']]
new_df.head()
```

Out[4]:

	Day	Month	Year	Day_of_Week	close
0	17	7	2010	5	0.04951
1	18	7	2010	6	0.08584
2	19	7	2010	0	0.08080
3	20	7	2010	1	0.07474
4	21	7	2010	2	0.07921

In [5]:
```
# Inspect the dataset for any null entries, duplicates and shape
print('Shape of Dataset:')
print(new_df.shape)
print('Numbers of duplicate:')
print(new_df.duplicated().sum())
print(new_df.info())
print(new_df.isnull().sum())
```

```
Shape of Dataset:
(5260, 5)
Numbers of duplicate:
0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5260 entries, 0 to 5259
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Day          5260 non-null   int32
 1   Month        5260 non-null   int32
 2   Year         5260 non-null   int32
 3   Day_of_Week  5260 non-null   int32
 4   close        5260 non-null   float64
dtypes: float64(1), int32(4)
memory usage: 123.4 KB
None
Day            0
Month          0
Year           0
Day_of_Week    0
close          0
dtype: int64
```

In [6]:
```
# Obtain statistical information about the dataset
new_df.describe()
```

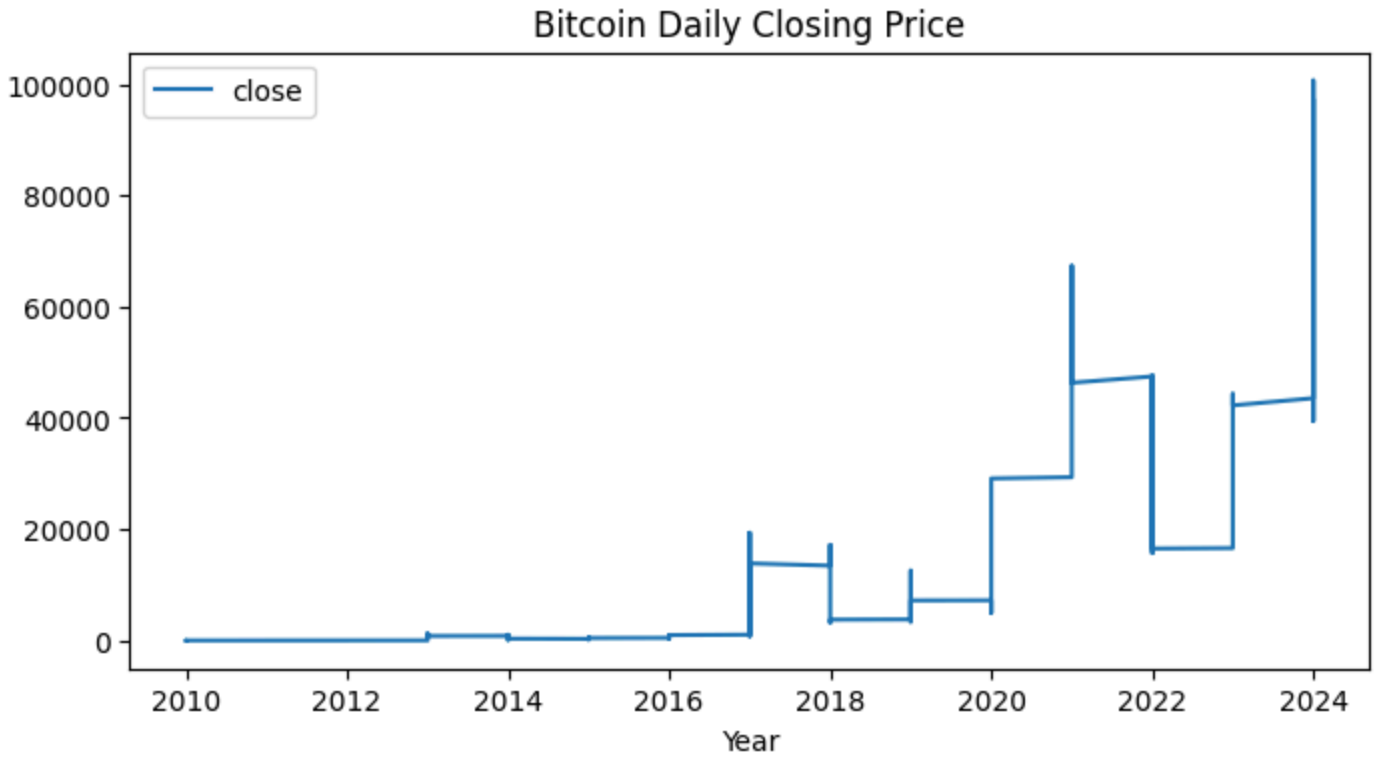
Out[6]:

	Day	Month	Year	Day_of_Week	close
count	5260.000000	5260.000000	5260.00000	5260.000000	5260.000000
mean	15.736692	6.602091	2017.23365	3.000380	13622.865358
std	8.807488	3.441748	4.16584	2.000665	19941.962706
min	1.000000	1.000000	2010.00000	0.000000	0.049510
25%	8.000000	4.000000	2014.00000	1.000000	235.827500
50%	16.000000	7.000000	2017.00000	3.000000	3649.575000
75%	23.000000	10.000000	2021.00000	5.000000	20355.075000
max	31.000000	12.000000	2024.00000	6.000000	100648.000000

In [7]:
```
# Import  relevant libraries for data Visualisation
import matplotlib.pyplot as plt
import seaborn as sns
```

In [8]:
```
# Visualise the closing price against date
fig, ax = plt.subplots(figsize=(8,4))
new_df.plot('Year', 'close', ax = ax)
ax.set(title='Bitcoin Daily Closing Price')
```

Out[8]:
```
[Text(0.5, 1.0, 'Bitcoin Daily Closing Price')]
```



In [9]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error as MSE, r2_score as R_Squared, mean_absolute_error as MAE
```

In [10]:
```
# Manually splitting the data by Date to avoid LOOK AHEAD BIAS
train_data = new_df[new_df['Year'] < 2024]
test_data = new_df[new_df['Year'] == 2024]

# Identify feture and target columns
feature_columns = ['Year', 'Day', 'Month' , 'Day_of_Week']
target_column = ['close']

X_train1 = train_data[feature_columns]
y_train1 = train_data[target_column]
X_test1 = test_data[feature_columns]
y_test1 = test_data[target_column]
```

```
# Instantiate the MinMaxScaler of scikit-learn
scaler = MinMaxScaler(feature_range=(0,1))

# fit transform to scale the train set
X_train_scale = scaler.fit_transform(X_train1)
X_test_scale = scaler.transform(X_test1)

# Inspect the shape of the scaled dataset
X_train_scale.shape, X_test_scale.shape, y_train1.shape, y_test1.shape
```

Out[10]: ((4916, 4), (344, 4), (4916, 1), (344, 1))

In [11]:
```
# Initialize Models
models = {
          'Random Forest': RandomForestRegressor(n_estimators = 100, random_state = 42),
          'XGBoost': XGBRegressor(n_estimators = 100, random_state = 42)}

#v Create a dictionary to store the results and predictions
results = {}
predictions = {}

# Use a for loop to loop through the models and make predictions
for name, model in models.items():
    model.fit(X_train_scale, y_train1)
    train_pred = model.predict(X_train_scale)
    test_pred = model.predict(X_test_scale)
    predictions[name] = test_pred

    # Calculate metrics
    train_mse = MSE(y_train1, train_pred)
    train_rmse = train_mse**0.5
    test_mse = MSE(y_test1, test_pred)
    test_rmse = test_mse**0.5
    test_r2 = R_Squared(y_test1, test_pred)
    test_mae = MAE(y_test1, test_pred)

    results[name] = {'Train Mean Squared Error': train_mse,
                     'Train Root Mean Squared Error': train_rmse,
                     'Test Mean Squared Error': test_mse,
                     'Test Root Mean Squared': test_rmse,
```

```
                    'Test R_Squared': test_r2,
                    'Test Mean Absolute Error': test_mae,}
```

In [12]:
```
# Print the predictions
print('\nModel Performance Metrics:')
for model_name, metrics in results.items():
    print(f'\n{model_name}:')
    for metric_name, value in metrics.items():
        print(f'{metric_name}: {value:.2f}')
```

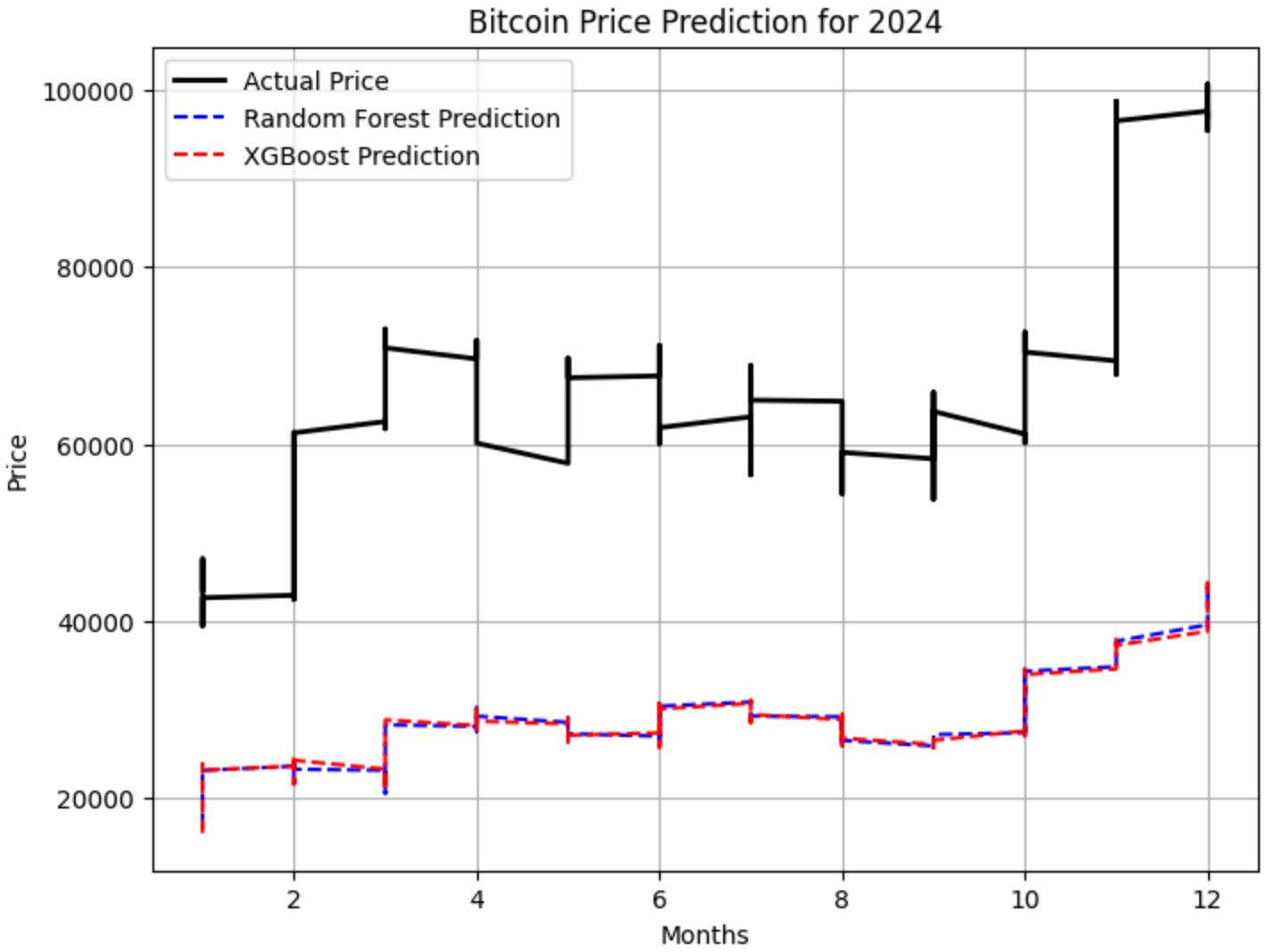
```
Model Performance Metrics:

Random Forest:
Train Mean Squared Error: 53880.23
Train Root Mean Squared Error: 232.12
Test Mean Squared Error: 1372820554.84
Test Root Mean Squared: 37051.59
Test R_Squared: -7.78
Test Mean Absolute Error: 35956.80

XGBoost:
Train Mean Squared Error: 83380.52
Train Root Mean Squared Error: 288.76
Test Mean Squared Error: 1371159810.27
Test Root Mean Squared: 37029.18
Test R_Squared: -7.77
Test Mean Absolute Error: 35939.32
```

In [13]:
```
# plot the Actual and Predicted Prices
plt.figure(figsize=(8,6))

#Plot actual test values
plt.plot(test_data['Month'], test_data['close'], label='Actual Price', color= 'black',linewidth=2)
colors = ['blue', 'red']
for (name, pred), color in zip(predictions.items(), colors):
    plt.plot(test_data['Month'], pred, label=f'{name} Prediction', color=color,linestyle='--')
    plt.title('Bitcoin Price Prediction for 2024')
    plt.xlabel('Months')
    plt.ylabel('Price')
    plt.grid(True)
    plt.legend()
plt.show()
```



In [14]:
```
# define aa functin that predicts future price
def predict_future_price1(date_str, model_name='Random Forest'):

    #Convert input data to features
    date = pd.to_datetime(date_str)
    features = pd.DataFrame({'Year':[date.year],
                             'Month':[date.month],
                             'Day': [date.day],
                             'Day_of_Week':[date.dayofweek]})
```

```
    # Make predictions
    model = models[model_name]
    prediction = model.predict(features)[0]
    return prediction
```

In [15]:
```
# Example prediction
future_date1 = '2025-01-05'
predicted_price1 = predict_future_price1(future_date1,'Random Forest')
print(predicted_price1)
```

```
42228.30199999995
```

I will explore the traditional train_test_split module of scikit-learn. In this case, the model will have information about the future and compare with the above result

In [16]:
```
from sklearn.model_selection import train_test_split
```

In [17]:
```
# Make a duplicate of the dataset
df_new = new_df.copy()
df_new.head()
```

Out[17]:

	Day	Month	Year	Day_of_Week	close
0	17	7	2010	5	0.04951
1	18	7	2010	6	0.08584
2	19	7	2010	0	0.08080
3	20	7	2010	1	0.07474
4	21	7	2010	2	0.07921

In [18]:
```
# Identify the feature and target variable
X = df_new.drop('close', axis=1)
y = df_new['close']

# #scale the X variable
X_scaled2 = scaler.fit_transform(X)
```

```
# Split the dataset
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_scaled2, y, test_size = 0.2, random_state = 42)

# Inspect the shapes
X_train2.shape, X_test2.shape, y_train2.shape, y_test2.shape
```

Out[18]: ((4208, 4), (1052, 4), (4208,), (1052,))

In [19]:
```
# Instantiate the Random Forest Model and fit to training data
rf = RandomForestRegressor(n_estimators = 100, random_state = 42)
rf.fit(X_train2, y_train2)
```

Out[19]:
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

In [20]:
```
# Make prediction on the test set
y_pred = rf.predict(X_test2)
```

In [21]:
```
# Calculate metrics
mse = MSE(y_test2, y_pred)
rmse = mse**0.5
mae = MAE(y_test2, y_pred)
r2 =  R_Squared(y_test2, y_pred)
```

In [22]:
```
# Print predictions
print(f'Mean Squared Error:{mse}')
print(f'Root Mean Squared Error:{rmse}')
print(f'Mean Absolute Error:{mae}')
print(f'R_Squared:{r2}')
```

```
Mean Squared Error:863458.409729081
Root Mean Squared Error:929.2246282407075
Mean Absolute Error:375.55550296673
R_Squared:0.9977905279093465
```

In [23]:
```
# Define a funtion that predicts future price of Bitcoin
def predict_future_price2(date_str, model):

    #Convert input data to features
```

```
    date = pd.to_datetime(date_str)
    features2 = pd.DataFrame({'Year':[date.year],
                              'Month':[date.month],
                              'Day': [date.day],
                              'Day_of_Week':[date.dayofweek]})

    # Make predictions
    predicted_price = model.predict(features2)[0]
    return predicted_price
```

In [24]:
```
# Example prediction
future_date2 = '2025-01-05'
predicted_price2 = predict_future_price2(future_date2, rf)
print(predicted_price2)
```

```
97578.646
```

The traditional train test split showed a better R_Squared 0f (0.998) as against the date based splitting(-7.78)

TRAINED AND PREPARED BY VICTOR ITINAH INIOBONG