q-learning-bitcoins

December 21, 2023

[65]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

[66]:
```
btc_data = pd.read_csv('/kaggle/input/binance-top-cryptocurrencies/BTC.csv')
btc_data.head()
```

[66]:

	unix	date	symbol	open	high	low	close	Volume BTC	Volume USDT	tradecount
0	1.610496e+12	2021-01-13 00:00:00	BTC/USDT	34049.15	34049.15	33589.57	33606.99	655.275245	2.215246e+07	11838.0
1	1.610410e+12	2021-01-12 00:00:00	BTC/USDT	35410.37	36628.00	32531.00	34051.24	133948.151996	4.651302e+09	2674145.0
2	1.610323e+12	2021-01-11 00:00:00	BTC/USDT	38150.02	38264.74	30420.00	35404.47	249131.539943	8.426880e+09	4431451.0
3	1.610237e+12	2021-01-10 00:00:00	BTC/USDT	40088.22	41350.00	35111.11	38150.02	118209.544503	4.604035e+09	2628050.0
4	1.610150e+12	2021-01-09 00:00:00	BTC/USDT	40586.96	41380.00	38720.00	40088.22	75785.979675	3.054779e+09	1998156.0

[67]:
```
btc_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1247 entries, 0 to 1246
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   unix         1247 non-null   float64
 1   date         1247 non-null   object
 2   symbol       1247 non-null   object
 3   open         1247 non-null   float64
 4   high         1247 non-null   float64
 5   low          1247 non-null   float64
 6   close        1247 non-null   float64
 7   Volume BTC   1247 non-null   float64
```

```
 8   Volume USDT  1247 non-null   float64
 9   tradecount   1123 non-null   float64
dtypes: float64(8), object(2)
memory usage: 97.5+ KB
```

[68]:
```
btc_data.describe()
```

[68]:
```
                unix          open          high           low         close  \
count   1.247000e+03   1247.000000   1247.000000   1247.000000   1247.000000
mean    1.406850e+12   8819.920313   9085.641099   8518.824587   8843.271716
std     4.679284e+11   4586.887774   4794.111621   4340.497940   4638.470396
min     1.502928e+09   3189.020000   3276.500000   2817.000000   3189.020000
25%     1.529755e+12   6387.980000   6532.575000   6253.105000   6389.485000
50%     1.556669e+12   8175.630000   8348.620000   7917.000000   8175.640000
75%     1.583582e+12  10195.095000  10438.500000   9869.230000  10208.395000
max     1.610496e+12  40586.960000  41950.000000  38720.000000  40582.810000

           Volume BTC   Volume USDT    tradecount
count     1247.000000  1.247000e+03  1.123000e+03
mean     45316.815018  4.528942e+08  5.037790e+05
std      34754.098993  6.010777e+08  4.463849e+05
min        225.760000  9.677533e+05  1.183800e+04
25%      25262.445595  1.697613e+08  2.156135e+05
50%      38815.409893  3.048624e+08  3.561340e+05
75%      58523.946985  5.148775e+08  6.378765e+05
max     402201.673764  8.426880e+09  4.431451e+06
```

[69]:
```
btc_data.isnull().sum()
```

[69]:
```
unix             0
date             0
symbol           0
open             0
high             0
low              0
close            0
Volume BTC       0
Volume USDT      0
tradecount     124
dtype: int64
```

[70]:
```
# Handle missing values in 'tradecount' column
btc_data['tradecount'].fillna(btc_data['tradecount'].mean(), inplace=True)
btc_data.isnull().sum()
```

[70]:
```
unix           0
date           0
```

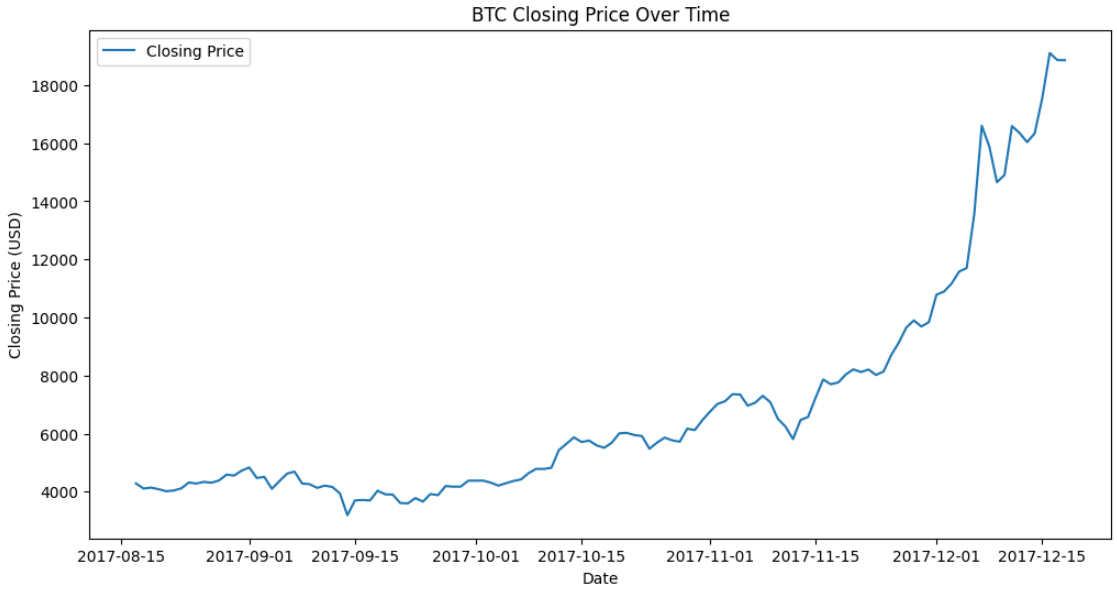
```
symbol         0
open           0
high           0
low            0
close          0
Volume BTC     0
Volume USDT    0
tradecount     0
dtype: int64
```

[71]:
```
btc_data.dtypes
```

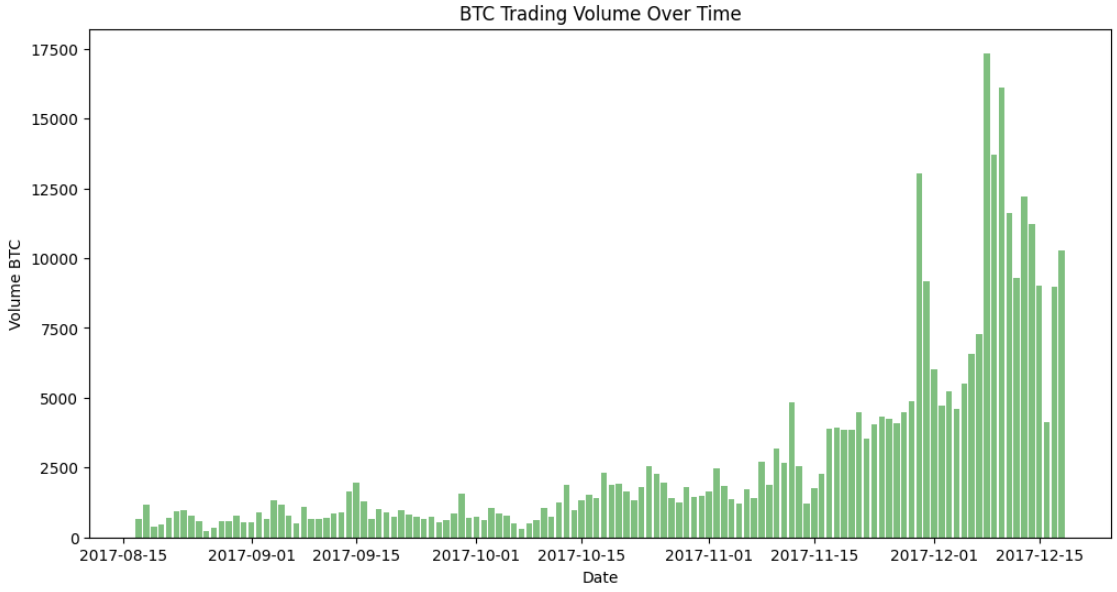
[71]:
```
unix           float64
date            object
symbol          object
open           float64
high           float64
low            float64
close          float64
Volume BTC     float64
Volume USDT    float64
tradecount     float64
dtype: object
```

[72]:
```
# convert date format
btc_data['date'] = pd.to_datetime(btc_data['date'], format="%Y-%m-%d",␣
 ↪errors='coerce')
```

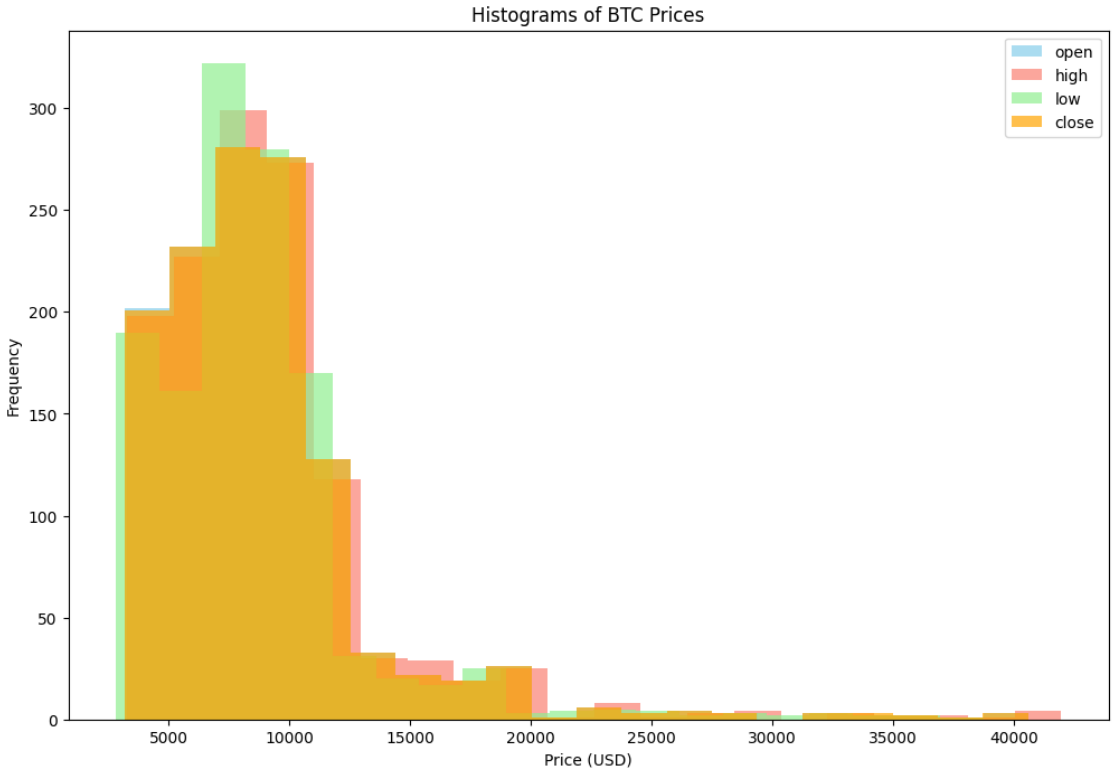
[73]:
```
# Visualize data
# Time series plot
plt.figure(figsize=(12, 6))
plt.plot(btc_data['date'], btc_data['close'], label='Closing Price')
plt.title('BTC Closing Price Over Time')
plt.xlabel('Date')
plt.ylabel('Closing Price (USD)')
plt.legend()
plt.show()
```



[74]:
```
#Plot the trading volume over time.

plt.figure(figsize=(12, 6))
plt.bar(btc_data['date'], btc_data['Volume BTC'], color='green', alpha=0.5)
plt.title('BTC Trading Volume Over Time')
plt.xlabel('Date')
plt.ylabel('Volume BTC')
plt.show()
```



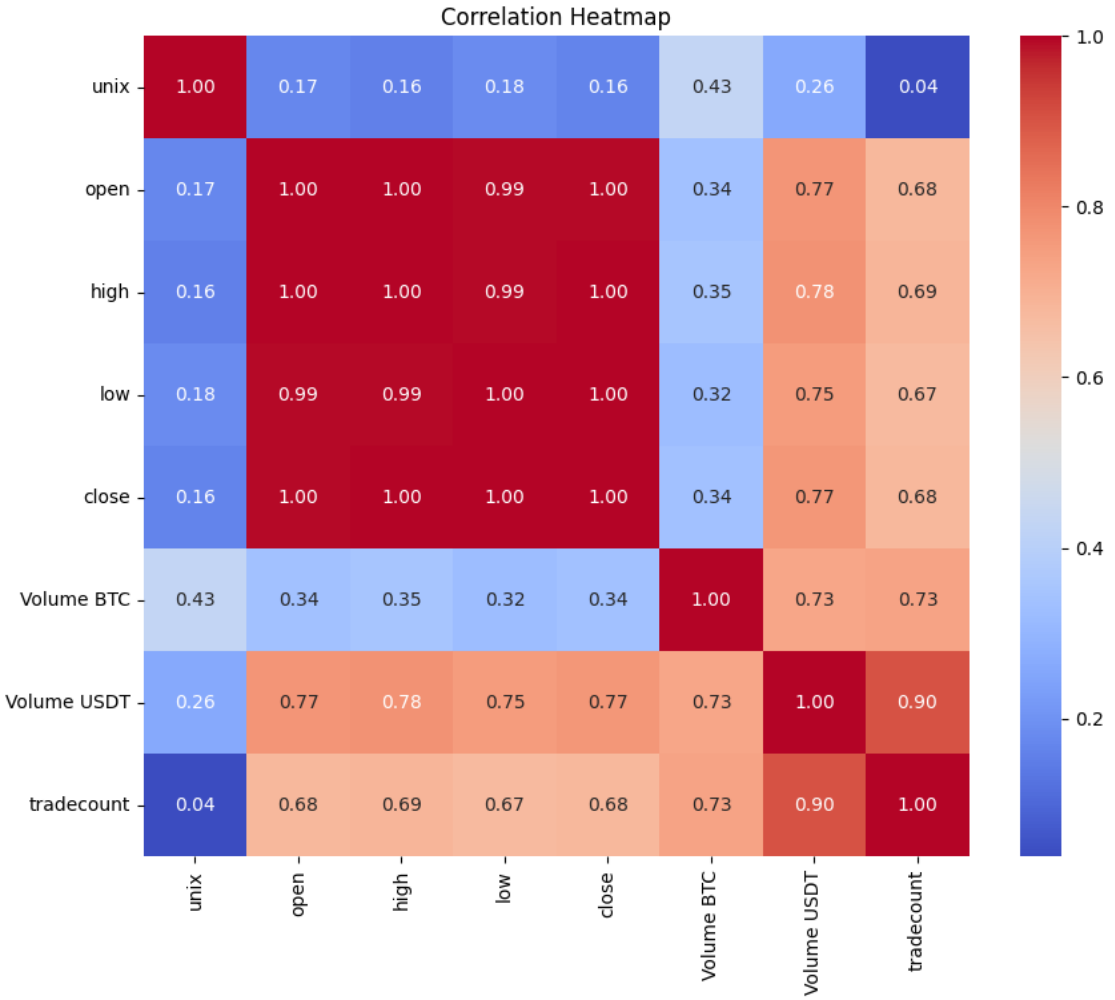
[75]:
```
#Histograms
colors = ['skyblue', 'salmon', 'lightgreen', 'orange']
plt.figure(figsize=(12, 8))

for i, column in enumerate(['open', 'high', 'low', 'close']):
    plt.hist(btc_data[column], bins=20, alpha=0.7, color=colors[i],␣
 ↪label=column)

plt.title('Histograms of BTC Prices')
plt.xlabel('Price (USD)')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



[76]:
```
# Select only numeric columns for correlation matrix
num_col = btc_data.select_dtypes(include=['float64']).columns
corr = btc_data[num_col].corr()
```

[77]:
```
# Plot the correlation heatmap
import seaborn as sns
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



[78]:
```
# Feature scaling using Min-Max scaling
scaler = MinMaxScaler()
fs = ['open', 'high', 'low', 'close', 'Volume BTC', 'Volume USDT', 'tradecount']

# Apply Min-Max scaling to selected features
btc_data[fs] = scaler.fit_transform(btc_data[fs])
```

[79]:
```
# Feature engineering
window_size = 10
btc_data['SMA'] = btc_data['close'].rolling(window=window_size).mean()
```

[80]:
```
# Display the preprocessed data
print("Preprocessed Data:")
print(btc_data.head())
```

```
Preprocessed Data:
           unix date    symbol      open      high       low     close  \
0  1.610496e+12  NaT  BTC/USDT  0.825183  0.795704  0.857103  0.813450
1  1.610410e+12  NaT  BTC/USDT  0.861581  0.862386  0.827619  0.825330
2  1.610323e+12  NaT  BTC/USDT  0.934838  0.904708  0.768822  0.861519
3  1.610237e+12  NaT  BTC/USDT  0.986664  0.984486  0.899482  0.934941
4  1.610150e+12  NaT  BTC/USDT  1.000000  0.985261  1.000000  0.986773

   Volume BTC  Volume USDT  tradecount  SMA
0    0.001069     0.002514    0.000000  NaN
1    0.332663     0.551909    0.602385  NaN
2    0.619206     1.000000    1.000000  NaN
3    0.293510     0.546299    0.591955  NaN
4    0.187972     0.362431    0.449433  NaN
```

1 Define Q-learning parameters

[81]:
```
gamma = 0.9  # Discount factor
alpha = 0.1  # Learning rate
epsilon = 0.1  # Exploration-exploitation trade-off
```

2 Define Q-learning Algorithm

[82]:
```
def q_learning(Q, state, action, reward, next_state):
    current_value = Q[state, action]
    max_future_value = np.max(Q[next_state, :])
    new_value = (1 - alpha) * current_value + alpha * (reward + gamma *␣
 ↪max_future_value)
    Q[state, action] = new_value
    return Q
```

[83]:
```
train_data, test_data = train_test_split(btc_data, test_size=0.2,␣
 ↪random_state=42)
```

[84]:
```
# Initialize Q-table
num_states = 100
num_actions = 2

Q = np.zeros((num_states, num_actions))
```

3 Training the Q-learning model

[85]:
```
for index, row in train_data.iterrows():
    state = int(index % num_states)
    action = 1 if row['close'] < row['SMA'] else 0  # Buy if the price is below␣
 ↪the moving average, else sell
    reward = row['close'] - row['SMA']  # Reward is the difference from the␣
 ↪moving average
    next_state = int((index + 1) % num_states)  # Simple next state␣
 ↪representation, adjust accordingly

    Q = q_learning(Q, state, action, reward, next_state)
```

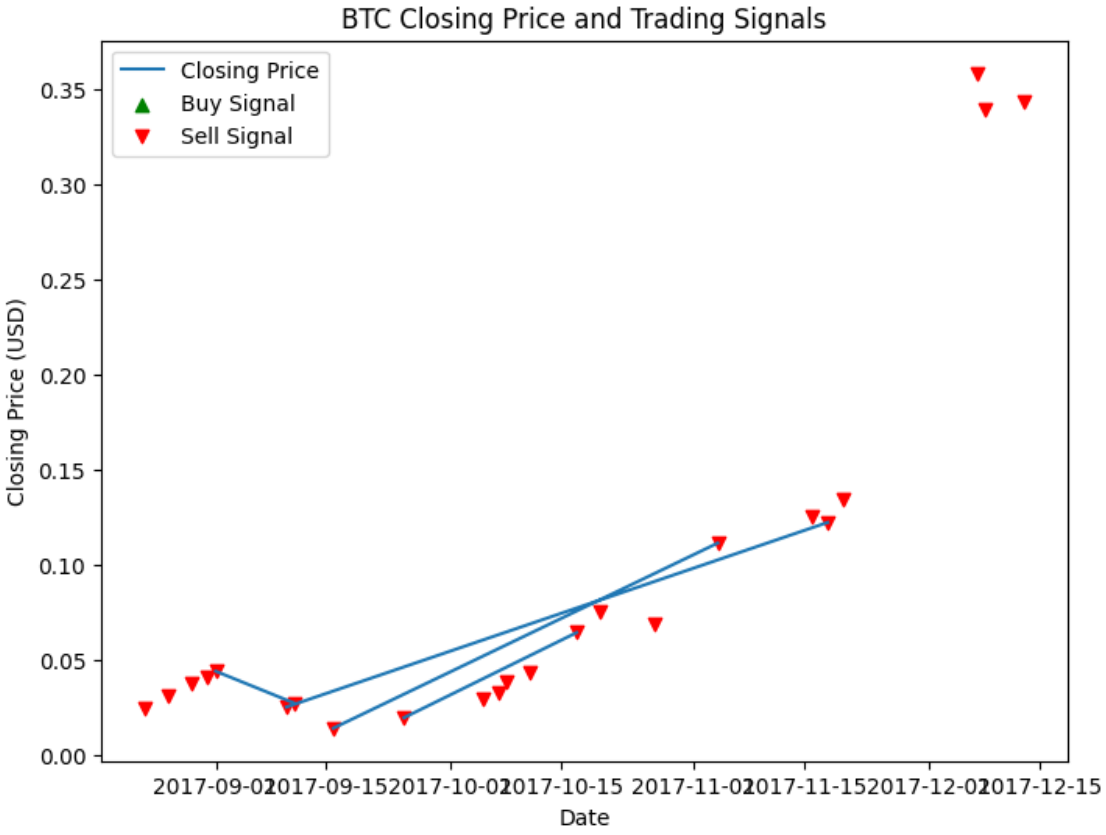
4 Testing the Q-learning model

[86]:
```
# Testing the Q-learning model
total_reward = 0
actions_taken = []

for index, row in test_data.iterrows():
    state = int(index % num_states)
    action = np.argmax(Q[state, :])
    actions_taken.append(action)
    reward = row['close'] - row['SMA']
    total_reward += reward
```

5 Visualize actions taken during testing

[87]:
```
plt.figure(figsize=(8, 6))
plt.plot(test_data['date'], test_data['close'], label='Closing Price')

# Filter buy signals
buy_signals = test_data['date'][np.array(actions_taken) == 1]
plt.scatter(buy_signals, test_data.loc[test_data['date'].
 ↪isin(buy_signals)]['close'],
            color='green', label='Buy Signal', marker='^')

# Filter sell signals
sell_signals = test_data['date'][np.array(actions_taken) == 0]
plt.scatter(sell_signals, test_data.loc[test_data['date'].
 ↪isin(sell_signals)]['close'],
            color='red', label='Sell Signal', marker='v')

plt.title('BTC Closing Price and Trading Signals')
plt.xlabel('Date')
```

```
plt.ylabel('Closing Price (USD)')
plt.legend()
plt.show()
```



[88]:
```
# Visualize actions taken during testing with annotations
plt.figure(figsize=(8, 6))
plt.plot(test_data['date'], test_data['close'], label='Closing Price')

# Filter buy signals
buy_signals = test_data['date'][np.array(actions_taken) == 1]
plt.scatter(buy_signals, test_data['close'].loc[test_data['date'].
 ↪isin(buy_signals)],
            color='green', label='Buy Signal', marker='^')

# Filter sell signals
sell_signals = test_data['date'][np.array(actions_taken) == 0]
plt.scatter(sell_signals, test_data['close'].loc[test_data['date'].
 ↪isin(sell_signals)],
            color='red', label='Sell Signal', marker='v')
```

```
# Annotate Buy signals
for date, price in zip(buy_signals, test_data['close'].loc[test_data['date'].
 ↪isin(buy_signals)]):
    plt.annotate('Buy', (date, price), textcoords="offset points", xytext=(0,␣
 ↪10), ha='center', fontsize=8, color='green')

# Annotate Sell signals
for date, price in zip(sell_signals, test_data['close'].loc[test_data['date'].
 ↪isin(sell_signals)]):
    plt.annotate('Sell', (date, price), textcoords="offset points", xytext=(0,␣
 ↪-10), ha='center', fontsize=8, color='red')

plt.title('BTC Closing Price and Trading Signals with Annotations')
plt.xlabel('Date')
plt.ylabel('Closing Price (USD)')
plt.legend()
plt.show()
```