In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
```

In [2]:
```python
df=pd.read_csv('AAPL.csv')
#Basic Data information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10468 entries, 0 to 10467
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       10468 non-null  object 
 1   Open       10468 non-null  float64
 2   High       10468 non-null  float64
 3   Low        10468 non-null  float64
 4   Close      10468 non-null  float64
 5   Adj Close  10468 non-null  float64
 6   Volume     10468 non-null  int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 572.6+ KB
```

In [3]:
```python
df
```

Out[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	469033600
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	175884800
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	105728000
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	86441600
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	73449600
...	...	...	...	...	...	...	...
10463	2022-06-13	132.869995	135.199997	131.440002	131.880005	131.880005	122207100
10464	2022-06-14	133.130005	133.889999	131.479996	132.759995	132.759995	84784300
10465	2022-06-15	134.289993	137.339996	132.160004	135.429993	135.429993	91533000
10466	2022-06-16	132.080002	132.389999	129.039993	130.059998	130.059998	108123900
10467	2022-06-17	130.070007	133.080002	129.809998	131.559998	131.559998	134118500

10468 rows × 7 columns

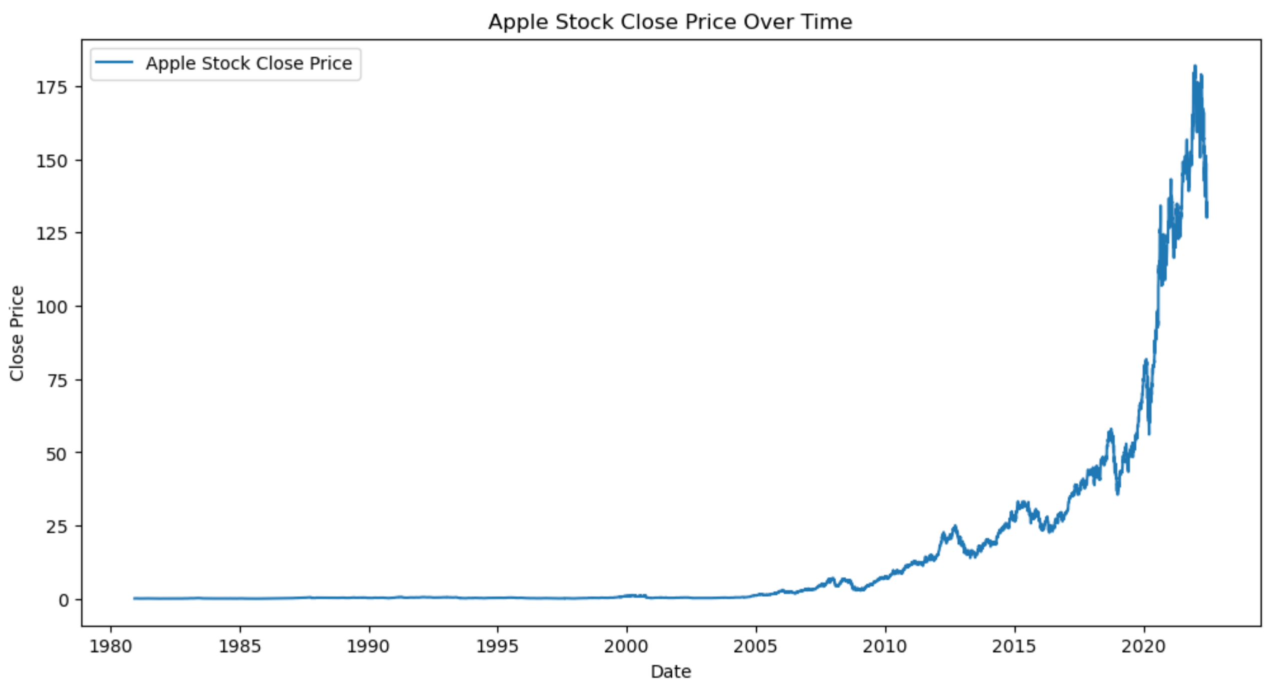
In [4]:
```python
# Display summary statistics
print(df.describe())
```

```
               Open          High           Low         Close     Adj Close  \
count  10468.000000  10468.000000  10468.000000  10468.000000  10468.000000   
mean      14.757987     14.921491     14.594484     14.763533     14.130431   
std       31.914174     32.289158     31.543959     31.929489     31.637275   
min        0.049665      0.049665      0.049107      0.049107      0.038329   
25%        0.283482      0.289286      0.276786      0.283482      0.235462   
50%        0.474107      0.482768      0.465960      0.475446      0.392373   
75%       14.953303     15.057143     14.692589     14.901964     12.835269   
max      182.630005    182.940002    179.119995    182.009995    181.511703   

             Volume  
count  1.046800e+04  
mean   3.308489e+08  
std    3.388418e+08  
min    0.000000e+00  
25%    1.237768e+08  
50%    2.181592e+08  
75%    4.105794e+08  
max    7.421641e+09  
```

In [5]:
```
# Check for missing values
print(df.isnull().sum())
```

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

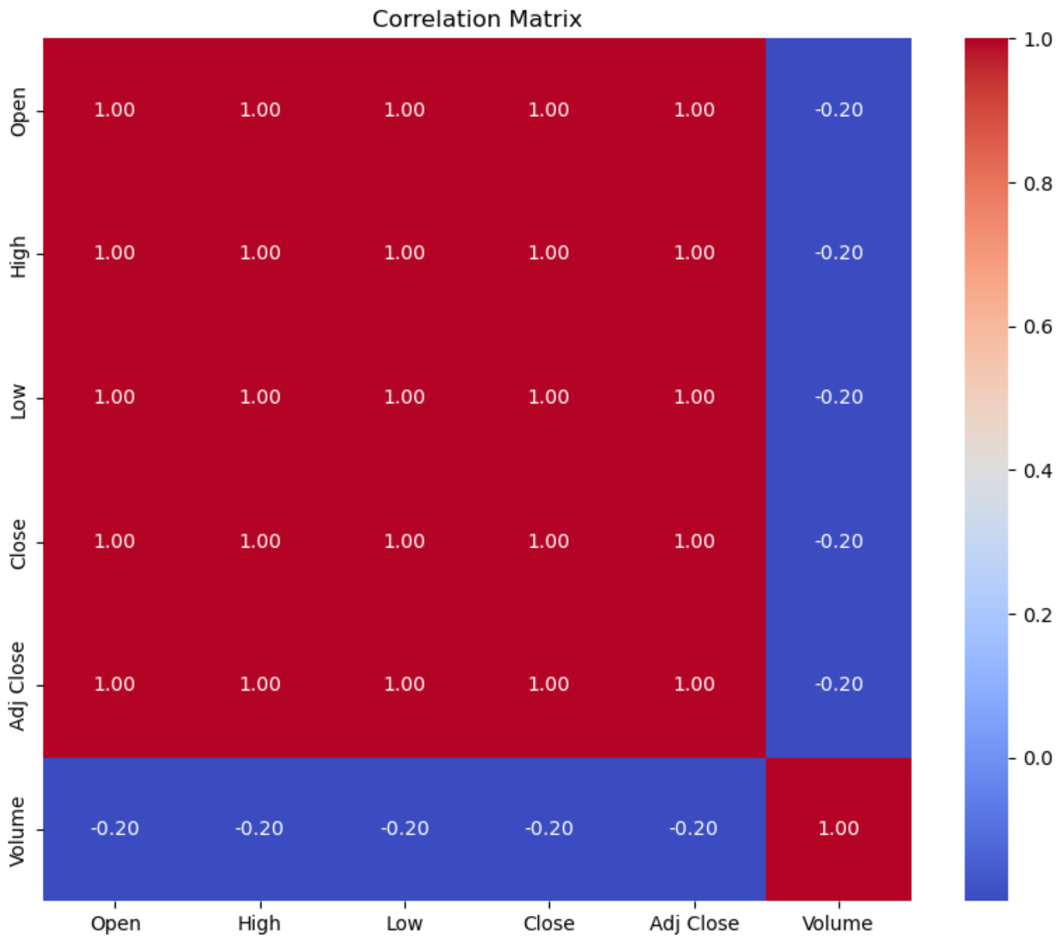
In [6]:
```
# Time Series analysis of the data and the target variable
# Convert 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Set 'Date' as the index
df.set_index('Date', inplace=True)

# Plot time series data
plt.figure(figsize=(12, 6))
plt.plot(df['Close'], label='Apple Stock Close Price')
plt.title('Apple Stock Close Price Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



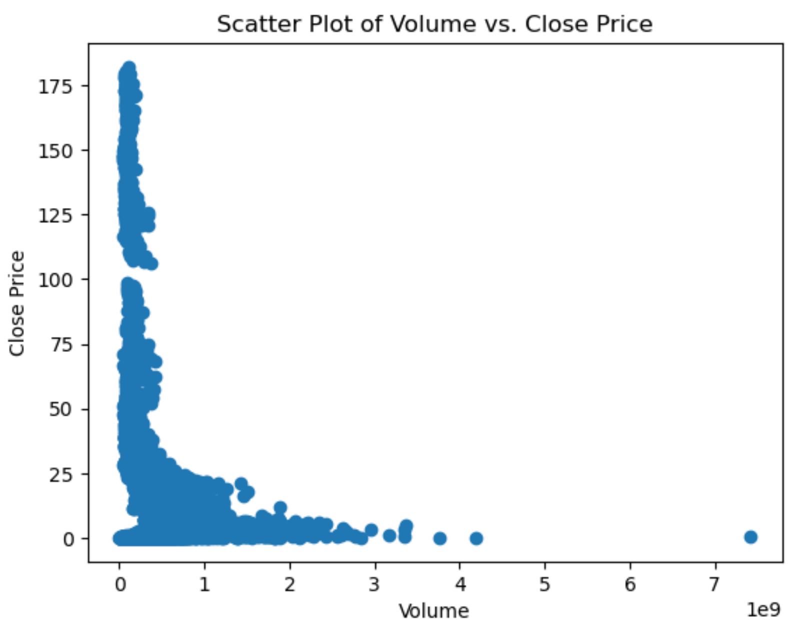
In [7]:
```
df.corr()['Close']
```

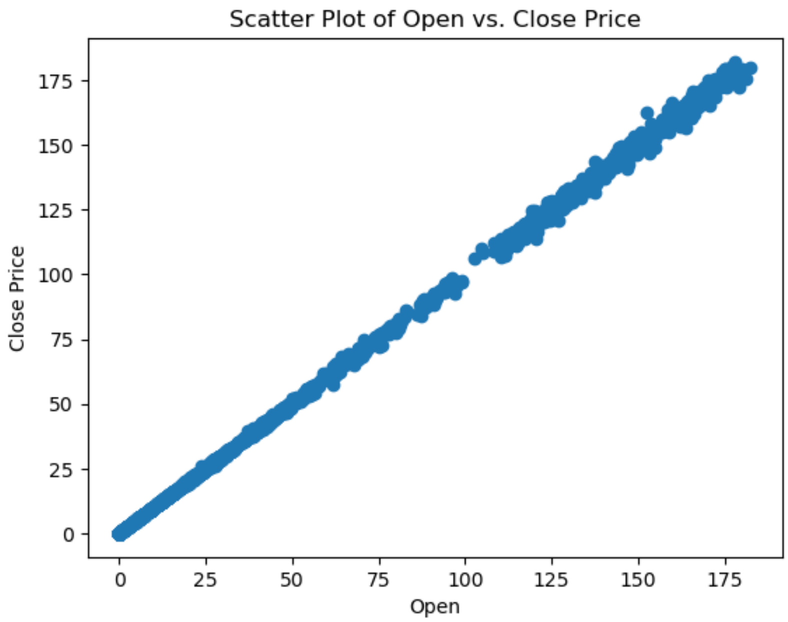
Out[7]:
```
Open         0.999850
High         0.999924
Low          0.999928
Close        1.000000
Adj Close    0.999671
Volume      -0.196411
Name: Close, dtype: float64
```

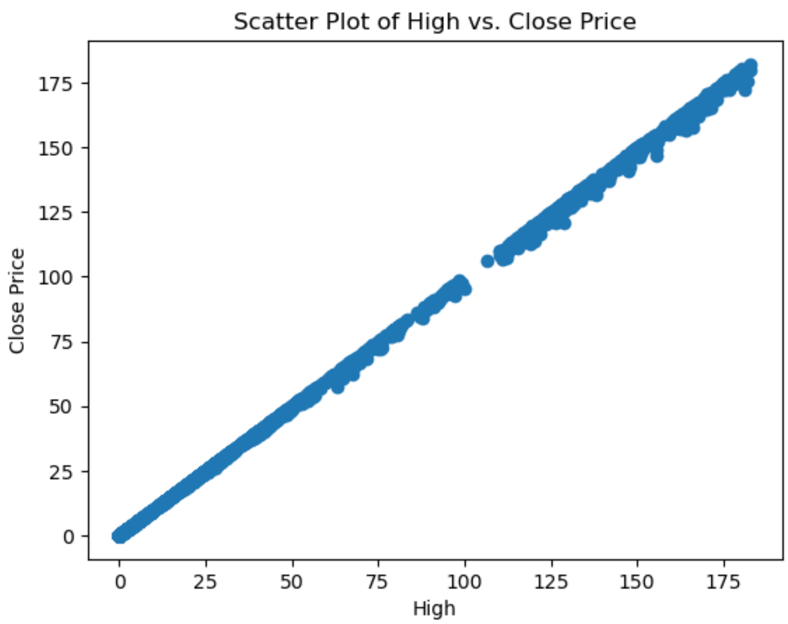
In [8]:
```
# Calculate correlation matrix

correlation_matrix = df.corr()

# Plot a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

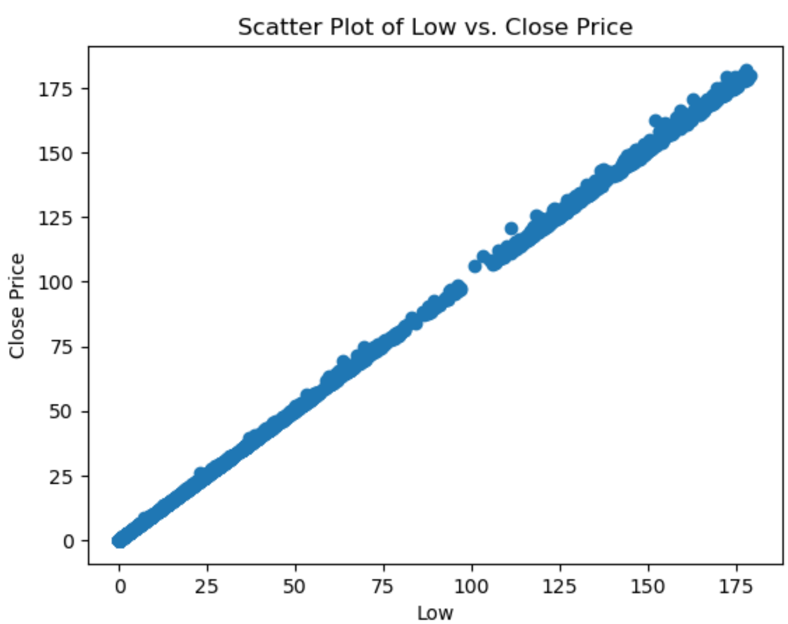


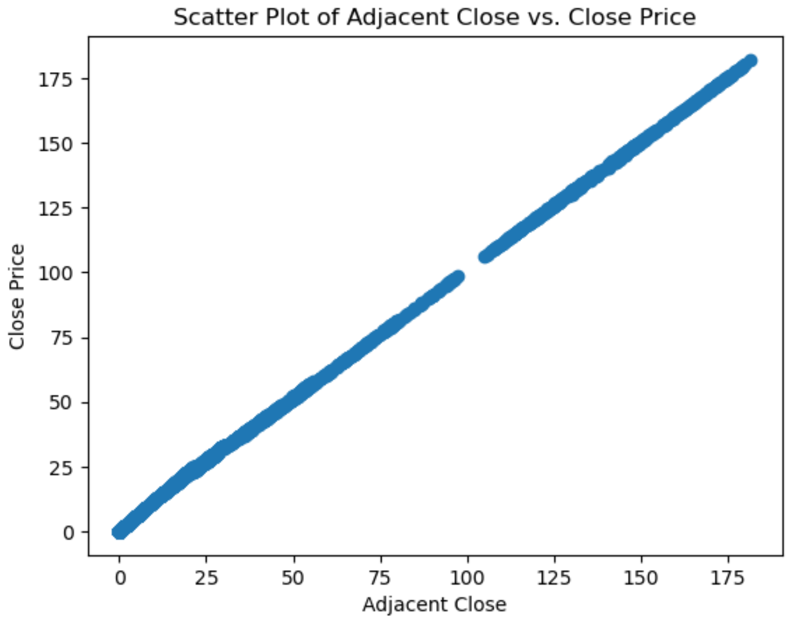
In [9]:
```
# Scatter plots wit all h variables

plt.scatter(df['Volume'], df['Close'])
plt.title('Scatter Plot of Volume vs. Close Price')
plt.xlabel('Volume')
plt.ylabel('Close Price')
plt.show()

plt.scatter(df['Open'], df['Close'])
plt.title('Scatter Plot of Open vs. Close Price')
plt.xlabel('Open')
plt.ylabel('Close Price')
plt.show()

plt.scatter(df['High'], df['Close'])
plt.title('Scatter Plot of High vs. Close Price')
plt.xlabel('High')
plt.ylabel('Close Price')
plt.show()

plt.scatter(df['Low'], df['Close'])
plt.title('Scatter Plot of Low vs. Close Price')
plt.xlabel('Low')
plt.ylabel('Close Price')
plt.show()

plt.scatter(df['Adj Close'], df['Close'])
plt.title('Scatter Plot of Adjacent Close vs. Close Price')
plt.xlabel('Adjacent Close')
plt.ylabel('Close Price')
plt.show()
```




Scatter Plot of Open vs. Close Price
Close Price
Open
0
25
50
75
100
125
150
175


Scatter Plot of High vs. Close Price
Close Price
High
0
25
50
75
100
125
150
175


Scatter Plot of Low vs. Close Price
Close Price
Low
0
25
50
75
100
125
150
175


Scatter Plot of Adjacent Close vs. Close Price
Close Price
Adjacent Close
0
25
50
75
100
125
150
175

In [10]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset (assuming 'AAPL.csv' contains the necessary columns)
df = pd.read_csv('AAPL.csv')

# Assume 'Close' is the column representing stock prices
data = df[['Open', 'High', 'Low', 'Adj Close']]  # Exclude 'Close' from predictors
target = df['Close'].values.reshape(-1, 1)

# Normalize the data
scaler_data = MinMaxScaler(feature_range=(0, 1))
scaler_target = MinMaxScaler(feature_range=(0, 1))

data_normalized = scaler_data.fit_transform(data)
target_normalized = scaler_target.fit_transform(target)

# Function to prepare the data for linear regression
def create_dataset(data, target, look_back=1):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:(i + look_back), :])
        y.append(target[i + look_back, 0])
    return np.array(X), np.array(y)

# Set the look-back period (number of time steps to look back)
look_back = 20

# Create the dataset
X, y = create_dataset(data_normalized, target_normalized, look_back)

# Set the number of folds for cross-validation
k_folds = 5

# Create a k-fold cross-validation object
kf = KFold(n_splits=k_folds, shuffle=True, random_state=42)

# Initialize the linear regression model
model = LinearRegression()

# Initialize lists to store MSE and R-squared scores
mse_scores = []
r2_scores = []

# Perform cross-validation and obtain scores
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model on the training set
    model.fit(X_train.reshape(-1, look_back * data.shape[1]), y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test.reshape(-1, look_back * data.shape[1]))

    # Denormalize the predictions and actual values
    y_pred_denormalized = scaler_target.inverse_transform(y_pred.reshape(-1, 1))
    y_test_denormalized = scaler_target.inverse_transform(y_test.reshape(-1, 1))

    # Calculate Mean Squared Error (MSE) on the test set
    mse_test = mean_squared_error(y_test_denormalized, y_pred_denormalized)

    # Calculate R-squared on the test set
    r2_test = r2_score(y_test_denormalized, y_pred_denormalized)

    mse_scores.append(mse_test)
    r2_scores.append(r2_test)

# Print the mean and standard deviation of the MSE and R-squared scores
print(f'Mean MSE: {np.mean(mse_scores)}')
print(f'Standard Deviation MSE: {np.std(mse_scores)}')
print(f'Mean R-squared: {np.mean(r2_scores)}')
print(f'Standard Deviation R-squared: {np.std(r2_scores)}')

# No need to train the model again on the entire dataset after cross-validation

# 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, shuffle=False)

# Make predictions on the test set
y_pred = model.predict(X_test.reshape(-1, look_back * data.shape[1]))

# Denormalize the predictions and actual values
y_pred_denormalized = scaler_target.inverse_transform(y_pred.reshape(-1, 1))
y_test_denormalized = scaler_target.inverse_transform(y_test.reshape(-1, 1))

# Calculate Mean Squared Error (MSE) on the test set
mse_test = mean_squared_error(y_test_denormalized, y_pred_denormalized)

# Calculate R-squared on the test set
r2_test = r2_score(y_test_denormalized, y_pred_denormalized)

print(f'Mean Squared Error (MSE) on Test Set: {mse_test}')
print(f'R-squared on Test Set: {r2_test}')

# Plot actual vs. predicted values on the test set
# Plot actual vs. predicted values on the test set

df['Date'] = pd.to_datetime(df['Date'])

# Plot for actual values and the predictions.
plt.figure(figsize=(12, 6))
plt.plot(df['Date'][-len(y_test_denormalized):], y_test_denormalized, label='Actual Closing Price', color='blue')
plt.plot(df['Date'][-len(y_test_denormalized):], y_pred_denormalized, label='Predicted Closing Price', color='orange')
plt.title('Apple Stock Close Price Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

```
Mean MSE: 0.5365784710577322
Standard Deviation MSE: 0.07546522768126811
Mean R-squared: 0.999475381709283
Standard Deviation R-squared: 6.340824985322405e-05
Mean Squared Error (MSE) on Test Set: 2.1335148126186203
R-squared on Test Set: 0.9989716080512189
```