ice-prediction-using-decision-tree

January 14, 2024

Stock Market Price Prediction using decision tree

Importing important libraries

[227]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score,␣
↪precision_score,f1_score, roc_auc_score
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

Tesla Dataset

In this project, I selected tesla datasets which play a crucial role as they form the foundation for training and evaluating machine learning models i.e decision tree for stock market price prediction. Dataset represents historical stock market data for different assets, and understanding their characteristics is vital for building effective predictive models.

Features of dataset * Date: Essential for organizing data chronologically and identifying trends over time. * Open: The opening price of Datasets on a given day. * High: The highest price of Datasets on a given day. * Low: The lowest price of Datasets on a given day. * Close: The closing price of Datasets on a given day. * Adj Close: The adjusted closing price of Datasets on a given day, considering dividends, stock splits, etc. * Volume: The volume of Datasets traded on a given day.

Load the dataset

[228]:
```
data = pd.read_csv('TESLA.csv')
print(data)
```

```
           Date        Open        High         Low       Close   Adj Close  \
0    2021-09-29  259.933319  264.500000  256.893341  260.436676  260.436676
1    2021-09-30  260.333344  263.043335  258.333344  258.493347  258.493347
2    2021-10-01  259.466675  260.260010  254.529999  258.406677  258.406677
```

```
3    2021-10-04  265.500000  268.989990  258.706665  260.510010  260.510010
4    2021-10-05  261.600006  265.769989  258.066681  260.196655  260.196655
..          …           …           …           …           …           …
248  2022-09-23  283.089996  284.500000  272.820007  275.329987  275.329987
249  2022-09-26  271.829987  284.089996  270.309998  276.010010  276.010010
250  2022-09-27  283.839996  288.670013  277.510010  282.940002  282.940002
251  2022-09-28  283.079987  289.000000  277.570007  287.809998  287.809998
252  2022-09-29  282.760010  283.649994  265.779999  268.209991  268.209991

       Volume
0    62828700
1    53868000
2    51094200
3    91449900
4    55297800
..        …
248  63615400
249  58076900
250  61925200
251  54664800
252  77393100

[253 rows x 7 columns]
```

Quick peek at functions:

[229]:
```python
data.shape
```

[229]:
```
(253, 7)
```

[230]:
```python
data.columns
```

[230]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

[231]:
```python
print(data.describe())
```

```
             Open        High         Low       Close   Adj Close  \
count  253.000000  253.000000  253.000000  253.000000  253.000000
mean   300.136008  307.486021  292.114058  299.709104  299.709104
std     46.139272   46.789896   44.685331   45.788283   45.788283
min    207.949997  217.973328  206.856674  209.386673  209.386673
25%    266.513336  273.166656  260.723328  266.923340  266.923340
50%    298.500000  303.709991  289.130005  296.666656  296.666656
75%    335.600006  344.950012  327.510010  336.336670  336.336670
max    411.470001  414.496674  405.666656  409.970001  409.970001

             Volume
```

```
count  2.530000e+02
mean   8.050938e+07
std    2.546595e+07
min    3.504270e+07
25%    6.255570e+07
50%    7.695630e+07
75%    9.347310e+07
max    1.885563e+08
```

[232]:
```python
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253 entries, 0 to 252
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       253 non-null    object
 1   Open       253 non-null    float64
 2   High       253 non-null    float64
 3   Low        253 non-null    float64
 4   Close      253 non-null    float64
 5   Adj Close  253 non-null    float64
 6   Volume     253 non-null    int64
dtypes: float64(5), int64(1), object(1)
memory usage: 14.0+ KB
None
```

Data Preprocessing

Handling Missing Values:

[233]:
```python
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)
data = data.dropna()
print("Missing Values After Handling:\n", data.isnull().sum())
```

```
Missing Values:
 Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
Missing Values After Handling:
 Date         0
Open         0
High         0
```

```
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

Feature Scaling

[234]:
```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data[['High']] = scaler.fit_transform(data[['High']])
data[['Low']] = scaler.fit_transform(data[['Low']])
data
```

[234]:
```
           Date        Open      High       Low       Close   Adj Close  \
0    2021-09-29  259.933319  0.236749  0.251681  260.436676  260.436676
1    2021-09-30  260.333344  0.229337  0.258924  258.493347  258.493347
2    2021-10-01  259.466675  0.215174  0.239793  258.406677  258.406677
3    2021-10-04  265.500000  0.259596  0.260802  260.510010  260.510010
4    2021-10-05  261.600006  0.243211  0.257583  260.196655  260.196655
..          …         …         …         …         …           …
248  2022-09-23  283.089996  0.338518  0.331791  275.329987  275.329987
249  2022-09-26  271.829987  0.336432  0.319166  276.010010  276.010010
250  2022-09-27  283.839996  0.359737  0.355381  282.940002  282.940002
251  2022-09-28  283.079987  0.361416  0.355683  287.809998  287.809998
252  2022-09-29  282.760010  0.334193  0.296380  268.209991  268.209991

       Volume
0    62828700
1    53868000
2    51094200
3    91449900
4    55297800
..        …
248  63615400
249  58076900
250  61925200
251  54664800
252  77393100

[253 rows x 7 columns]
```

Feature Engineering

[235]:
```python
data['DailyReturn'] = data['Adj Close'].pct_change() * 100
data['MovingAverage'] = data['Adj Close'].rolling(window=5).mean()
data['PriceToVolumeRatio'] = data['Adj Close']/data['Volume']
data=data.dropna()
```

```
data
```

[235]:

```
           Date        Open      High       Low       Close   Adj Close  \
4    2021-10-05  261.600006  0.243211  0.257583  260.196655  260.196655
5    2021-10-06  258.733337  0.225147  0.255939  260.916656  260.916656
6    2021-10-07  261.820007  0.256255  0.272974  264.536682  264.536682
7    2021-10-08  265.403320  0.241634  0.268833  261.829987  261.829987
8    2021-10-11  262.549988  0.249877  0.276529  263.980011  263.980011
..          …          …         …         …          …           …
248  2022-09-23  283.089996  0.338518  0.331791  275.329987  275.329987
249  2022-09-26  271.829987  0.336432  0.319166  276.010010  276.010010
250  2022-09-27  283.839996  0.359737  0.355381  282.940002  282.940002
251  2022-09-28  283.079987  0.361416  0.355683  287.809998  287.809998
252  2022-09-29  282.760010  0.334193  0.296380  268.209991  268.209991

       Volume  DailyReturn  MovingAverage  PriceToVolumeRatio
4    55297800    -0.120285     259.608673            0.000005
5    43898400     0.276714     259.704669            0.000006
6    57587400     1.387426     260.913336            0.000005
7    50215800    -1.023183     261.597998            0.000005
8    42600900     0.821153     262.291998            0.000006
..        …          …            …                  …
248  63615400    -4.594757     296.503998            0.000004
249  58076900     0.246985     289.891998            0.000005
250  61925200     2.510776     284.733997            0.000005
251  54664800     1.721212     282.135999            0.000005
252  77393100    -6.810051     278.059998            0.000003

[249 rows x 10 columns]
```

Data Splitting and Model Training

[236]:

```
features = ['Open', 'High', 'Low', 'Volume', 'DailyReturn', 'MovingAverage',␣
 ↪'PriceToVolumeRatio']
X = data[features]
Y = data['Close']
print(X.columns)
print(Y.name)
```

```
Index(['Open', 'High', 'Low', 'Volume', 'DailyReturn', 'MovingAverage',
       'PriceToVolumeRatio'],
      dtype='object')
Close
```

[237]:

```
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
```

[238]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2,␣
↪random_state=42)
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (199, 7)
X_test shape: (50, 7)
y_train shape: (199,)
y_test shape: (50,)
```

[239]:
```
model = DecisionTreeRegressor()
```

Model Evaluation

[240]:
```
model.fit(X_train, y_train)
predictions = model.predict(X_test)
predictions
```

[240]:
```
array([355.983337, 272.773346, 262.369995, 352.26001 , 274.820007,
       233.070007, 239.706665, 236.473328, 300.980011, 275.609985,
       266.679993, 300.980011, 268.573334, 303.996674, 349.869995,
       303.083344, 237.036667, 352.420013, 233.070007, 383.196655,
       287.809998, 399.926666, 312.23999 , 352.420013, 233.070007,
       352.26001 , 340.790009, 290.253326, 349.869995, 355.983337,
       276.01001 , 364.66333 , 310.      , 216.759995, 342.320007,
       264.536682, 366.523346, 239.706665, 224.473328, 234.516663,
       334.763336, 268.193329, 276.01001 , 227.263336, 244.919998,
       343.503326, 233.066666, 366.523346, 231.733337, 381.816681])
```

[241]:
```
y_test
```

[241]:
```
141    336.260010
10     270.359985
101    254.679993
64     356.779999
116    280.076660
184    235.070007
200    240.546661
187    232.663330
13     290.036682
108    279.429993
203    271.706665
219    300.029999
205    268.433319
243    303.350006
71     352.706665
```

```
227    297.096680
197    237.039993
19     339.476654
166    235.910004
28     387.646667
250    282.940002
23     402.863342
118    301.796661
72     354.799988
179    233.000000
49     356.320007
59     336.290009
14     288.089996
34     351.576660
124    363.946655
204    272.243347
37     379.019989
144    292.140015
163    224.966660
29     341.166656
115    267.296661
22     371.333344
199    240.066666
177    215.736664
162    221.300003
122    337.973328
100    273.843323
112    279.433319
188    228.490005
194    250.763336
20     345.953339
198    238.313339
73     368.739990
176    232.229996
42     378.996674
Name: Close, dtype: float64
```

[242]:
```
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 44.04801666119099
```

[243]:
```
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error: {mae:.2f}')
```

```
Mean Absolute Error: 4.91
```

[244]:
```python
# Calculate R-squared
r2 = r2_score(y_test, predictions)
print(f'R-squared: {r2:.2f}')
```

```
R-squared: 0.98
```

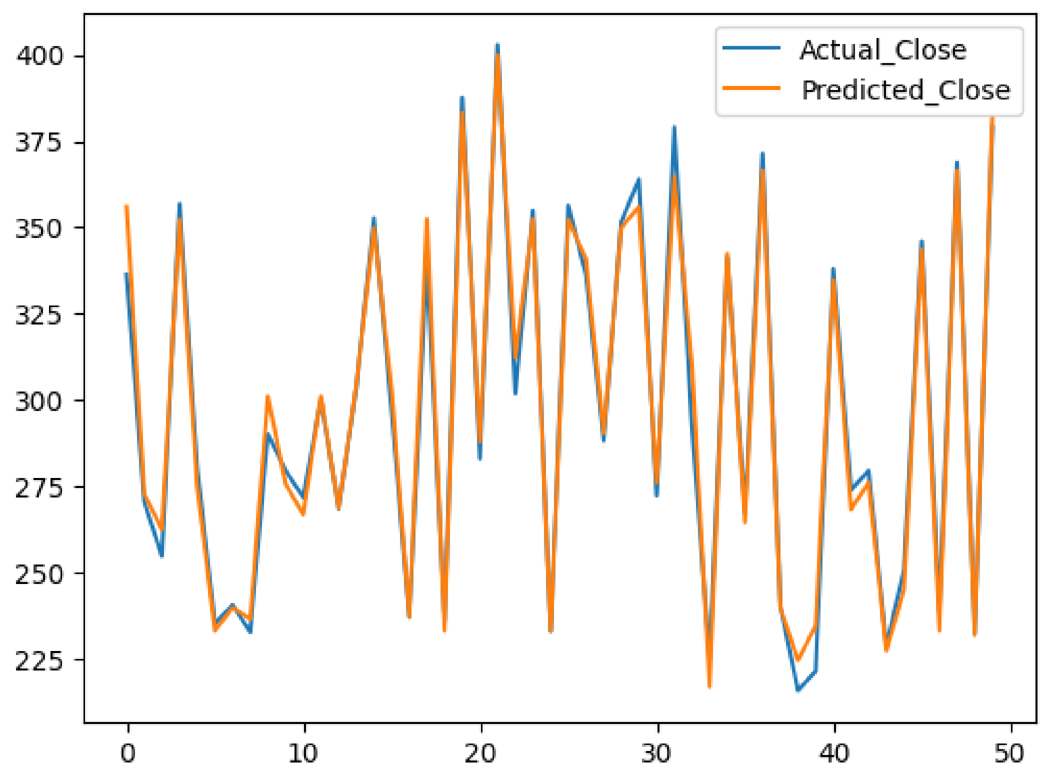
[245]:
```python
df_results = pd.DataFrame({'Date': pd.to_datetime(y_test.index,␣
↪format='%Y-%m-%d'),
                           'Actual_Close': y_test.values,
                           'Predicted_Close': predictions})

# Display the new DataFrame
print(df_results)
```

```
                            Date  Actual_Close  Predicted_Close
0  1970-01-01 00:00:00.000000141    336.260010       355.983337
1  1970-01-01 00:00:00.000000010    270.359985       272.773346
2  1970-01-01 00:00:00.000000101    254.679993       262.369995
3  1970-01-01 00:00:00.000000064    356.779999       352.260010
4  1970-01-01 00:00:00.000000116    280.076660       274.820007
5  1970-01-01 00:00:00.000000184    235.070007       233.070007
6  1970-01-01 00:00:00.000000200    240.546661       239.706665
7  1970-01-01 00:00:00.000000187    232.663330       236.473328
8  1970-01-01 00:00:00.000000013    290.036682       300.980011
9  1970-01-01 00:00:00.000000108    279.429993       275.609985
10 1970-01-01 00:00:00.000000203    271.706665       266.679993
11 1970-01-01 00:00:00.000000219    300.029999       300.980011
12 1970-01-01 00:00:00.000000205    268.433319       268.573334
13 1970-01-01 00:00:00.000000243    303.350006       303.996674
14 1970-01-01 00:00:00.000000071    352.706665       349.869995
15 1970-01-01 00:00:00.000000227    297.096680       303.083344
16 1970-01-01 00:00:00.000000197    237.039993       237.036667
17 1970-01-01 00:00:00.000000019    339.476654       352.420013
18 1970-01-01 00:00:00.000000166    235.910004       233.070007
19 1970-01-01 00:00:00.000000028    387.646667       383.196655
20 1970-01-01 00:00:00.000000250    282.940002       287.809998
21 1970-01-01 00:00:00.000000023    402.863342       399.926666
22 1970-01-01 00:00:00.000000118    301.796661       312.239990
23 1970-01-01 00:00:00.000000072    354.799988       352.420013
24 1970-01-01 00:00:00.000000179    233.000000       233.070007
25 1970-01-01 00:00:00.000000049    356.320007       352.260010
26 1970-01-01 00:00:00.000000059    336.290009       340.790009
27 1970-01-01 00:00:00.000000014    288.089996       290.253326
28 1970-01-01 00:00:00.000000034    351.576660       349.869995
29 1970-01-01 00:00:00.000000124    363.946655       355.983337
30 1970-01-01 00:00:00.000000204    272.243347       276.010010
31 1970-01-01 00:00:00.000000037    379.019989       364.663330
```

```
32 1970-01-01 00:00:00.000000144    292.140015       310.000000
33 1970-01-01 00:00:00.000000163    224.966660       216.759995
34 1970-01-01 00:00:00.000000029    341.166656       342.320007
35 1970-01-01 00:00:00.000000115    267.296661       264.536682
36 1970-01-01 00:00:00.000000022    371.333344       366.523346
37 1970-01-01 00:00:00.000000199    240.066666       239.706665
38 1970-01-01 00:00:00.000000177    215.736664       224.473328
39 1970-01-01 00:00:00.000000162    221.300003       234.516663
40 1970-01-01 00:00:00.000000122    337.973328       334.763336
41 1970-01-01 00:00:00.000000100    273.843323       268.193329
42 1970-01-01 00:00:00.000000112    279.433319       276.010010
43 1970-01-01 00:00:00.000000188    228.490005       227.263336
44 1970-01-01 00:00:00.000000194    250.763336       244.919998
45 1970-01-01 00:00:00.000000020    345.953339       343.503326
46 1970-01-01 00:00:00.000000198    238.313339       233.066666
47 1970-01-01 00:00:00.000000073    368.739990       366.523346
48 1970-01-01 00:00:00.000000176    232.229996       231.733337
49 1970-01-01 00:00:00.000000042    378.996674       381.816681
```

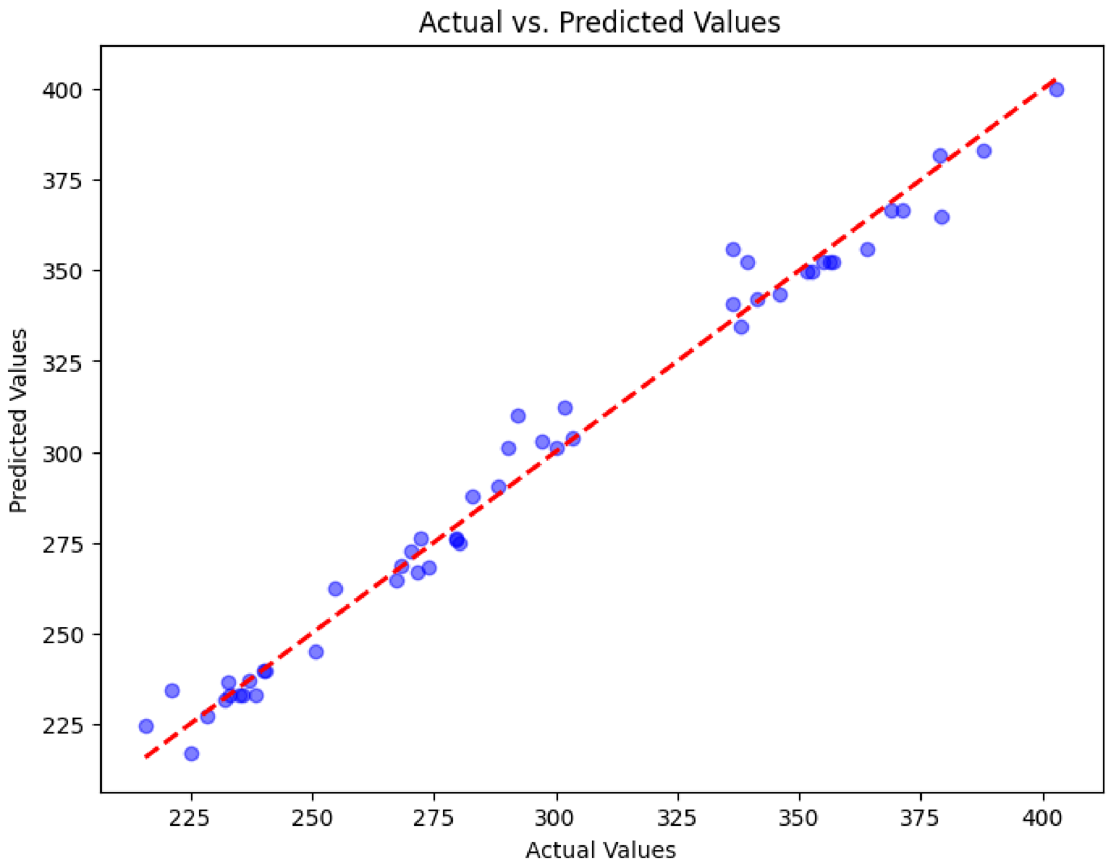
[246]:
```
df_results[['Actual_Close', 'Predicted_Close']].plot()
```

[246]: <Axes: >



[247]:

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, predictions, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],linestyle='--',␣
↪color='red', linewidth=2)
plt.title('Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



Netflix Dataset

In this project, I selected Netflix datasets which play a crucial role as they form the foundation for training and evaluating machine learning models i.e decision tree for stock market price prediction. Dataset represents historical stock market data for different assets, and understanding their characteristics is vital for building effective predictive models.

Features of dataset * Date: Essential for organizing data chronologically and identifying trends over time. * Open: The opening price of Datasets on a given day. * High: The highest price of Datasets on a given day. * Low: The lowest price of Datasets on a given day. * Close: The closing price of Datasets on a given day. * Adj Close: The adjusted closing price of Datasets on a given day, considering dividends, stock splits, etc. * Volume: The volume of Datasets traded on a given day.

Load the dataset

[248]:
```python
data = pd.read_csv('NFLX.csv')
print(data)
```

```
            Date        Open        High         Low       Close   Adj Close  \
0     2018-02-05  262.000000  267.899994  250.029999  254.259995  254.259995
1     2018-02-06  247.699997  266.700012  245.000000  265.720001  265.720001
2     2018-02-07  266.579987  272.450012  264.329987  264.559998  264.559998
3     2018-02-08  267.079987  267.619995  250.000000  250.100006  250.100006
4     2018-02-09  253.850006  255.800003  236.110001  249.470001  249.470001
…            …         …           …           …           …         …
1004  2022-01-31  401.970001  427.700012  398.200012  427.140015  427.140015
1005  2022-02-01  432.959991  458.480011  425.540009  457.130005  457.130005
1006  2022-02-02  448.250000  451.980011  426.480011  429.480011  429.480011
1007  2022-02-03  421.440002  429.260010  404.279999  405.600006  405.600006
1008  2022-02-04  407.309998  412.769989  396.640015  410.170013  410.170013

        Volume
0     11896100
1     12595800
2      8981500
3      9306700
4     16906900
…          …
1004  20047500
1005  22542300
1006  14346000
1007   9905200
1008   7782400

[1009 rows x 7 columns]
```

Quick peek at functions:

[249]:
```python
data.shape
```

[249]:
```
(1009, 7)
```

[250]:
```python
data.columns
```

[250]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

[251]:
```python
print(data.describe())
```

```
              Open         High          Low        Close    Adj Close  \
count  1009.000000  1009.000000  1009.000000  1009.000000  1009.000000
mean    419.059673   425.320703   412.374044   419.000733   419.000733
std     108.537532   109.262960   107.555867   108.289999   108.289999
min     233.919998   250.649994   231.229996   233.880005   233.880005
25%     331.489990   336.299988   326.000000   331.619995   331.619995
50%     377.769989   383.010010   370.880005   378.670013   378.670013
75%     509.130005   515.630005   502.529999   509.079987   509.079987
max     692.349976   700.989990   686.090027   691.690002   691.690002

             Volume
count  1.009000e+03
mean   7.570685e+06
std    5.465535e+06
min    1.144000e+06
25%    4.091900e+06
50%    5.934500e+06
75%    9.322400e+06
max    5.890430e+07
```

[252]:
```python
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1009 non-null   object
 1   Open       1009 non-null   float64
 2   High       1009 non-null   float64
 3   Low        1009 non-null   float64
 4   Close      1009 non-null   float64
 5   Adj Close  1009 non-null   float64
 6   Volume     1009 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 55.3+ KB
None
```

Data Preprocessing

Handling Missing Values:

[253]:
```python
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)
data = data.dropna()
print("Missing Values After Handling:\n", data.isnull().sum())
```

```
Missing Values:
 Date         0
```

```
Open          0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
Missing Values After Handling:
 Date          0
Open          0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
```

Feature Scaling

[254]:
```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data[['High']] = scaler.fit_transform(data[['High']])
data[['Low']] = scaler.fit_transform(data[['Low']])
data
```

[254]:
```
            Date        Open      High       Low       Close   Adj Close  \
0     2018-02-05  262.000000  0.038304  0.041331  254.259995  254.259995
1     2018-02-06  247.699997  0.035640  0.030273  265.720001  265.720001
2     2018-02-07  266.579987  0.048408  0.072770  264.559998  264.559998
3     2018-02-08  267.079987  0.037683  0.041265  250.100006  250.100006
4     2018-02-09  253.850006  0.011436  0.010729  249.470001  249.470001
…            …           …        …         …         …           …
1004  2022-01-31  401.970001  0.393147  0.367080  427.140015  427.140015
1005  2022-02-01  432.959991  0.461496  0.427186  457.130005  457.130005
1006  2022-02-02  448.250000  0.447062  0.429253  429.480011  429.480011
1007  2022-02-03  421.440002  0.396611  0.380447  405.600006  405.600006
1008  2022-02-04  407.309998  0.359995  0.363650  410.170013  410.170013

        Volume
0     11896100
1     12595800
2      8981500
3      9306700
4     16906900
…          …
1004  20047500
1005  22542300
1006  14346000
```

```
1007    9905200
1008    7782400

[1009 rows x 7 columns]
```

Feature Engineering

[255]:
```
data['DailyReturn'] = data['Adj Close'].pct_change() * 100
data['MovingAverage'] = data['Adj Close'].rolling(window=5).mean()
data['PriceToVolumeRatio'] = data['Adj Close']/data['Volume']
data=data.dropna()
data
```

[255]:
```
            Date        Open      High       Low       Close   Adj Close  \
4     2018-02-09  253.850006  0.011436  0.010729  249.470001  249.470001
5     2018-02-12  252.139999  0.018875  0.039067  257.950012  257.950012
6     2018-02-13  257.290009  0.023893  0.051598  258.269989  258.269989
7     2018-02-14  260.470001  0.042701  0.063976  266.000000  266.000000
8     2018-02-15  270.029999  0.066283  0.080025  280.269989  280.269989
…            …           …         …         …           …           …
1004  2022-01-31  401.970001  0.393147  0.367080  427.140015  427.140015
1005  2022-02-01  432.959991  0.461496  0.427186  457.130005  457.130005
1006  2022-02-02  448.250000  0.447062  0.429253  429.480011  429.480011
1007  2022-02-03  421.440002  0.396611  0.380447  405.600006  405.600006
1008  2022-02-04  407.309998  0.359995  0.363650  410.170013  410.170013

        Volume  DailyReturn  MovingAverage  PriceToVolumeRatio
4     16906900    -0.251901     256.822000            0.000015
5      8534900     3.399211     257.560004            0.000030
6      6855200     0.124046     256.070001            0.000038
7     10972000     2.992996     256.358002            0.000024
8     10759700     5.364658     262.391998            0.000026
…          …           …            …                  …
1004  20047500    11.130199     384.864007            0.000021
1005  22542300     7.021115     403.006006            0.000020
1006  14346000    -6.048606     416.962006            0.000030
1007   9905200    -5.560213     420.742004            0.000041
1008   7782400     1.126728     425.904010            0.000053

[1005 rows x 10 columns]
```

Data Splitting and Model Training

[256]:
```
features = ['Open', 'High', 'Low', 'Volume', 'DailyReturn', 'MovingAverage',␣
↪'PriceToVolumeRatio']
X = data[features]
Y = data['Close']
print(X.columns)
```

```python
print(Y.name)
```

```
Index(['Open', 'High', 'Low', 'Volume', 'DailyReturn', 'MovingAverage',
       'PriceToVolumeRatio'],
      dtype='object')
Close
```

[257]:
```python
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
```

[258]:
```python
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2,␣
  ↪random_state=42)
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (804, 7)
X_test shape: (201, 7)
y_train shape: (804,)
y_test shape: (201,)
```

[259]:
```python
model = DecisionTreeRegressor()
```

Model Evaluation

[260]:
```python
model.fit(X_train, y_train)
predictions = model.predict(X_test)
predictions
```

[260]:
```
array([629.76001 , 484.119995, 486.23999 , 368.769989, 320.799988,
       466.929993, 466.929993, 339.559998, 369.950012, 332.630005,
       553.330017, 316.480011, 547.580017, 307.350006, 358.      ,
       534.659973, 493.369995, 610.340027, 359.070007, 398.390015,
       611.659973, 496.950012, 274.459991, 329.540009, 449.869995,
       361.399994, 307.350006, 363.920013, 292.01001 , 517.919983,
       512.659973, 548.219971, 399.390015, 345.089996, 541.940002,
       349.730011, 665.640015, 262.799988, 364.130005, 379.23999 ,
       343.429993, 300.940002, 355.059998, 681.169983, 349.190002,
       351.350006, 298.5     , 629.76001 , 500.859985, 641.900024,
       480.23999 , 521.659973, 329.809998, 503.839996, 419.600006,
       539.440002, 294.339996, 312.279999, 318.450012, 315.      ,
       515.409973, 270.720001, 290.299988, 489.429993, 503.859985,
       328.899994, 294.179993, 258.269989, 367.320007, 513.469971,
       404.980011, 233.880005, 289.619995, 351.350006, 532.280029,
       368.769989, 317.5     , 632.659973, 293.970001, 429.480011,
       488.809998, 377.049988, 329.809998, 369.609985, 325.899994,
       592.640015, 586.72998 , 657.580017, 364.130005, 495.98999 ,
```

```
466.26001 , 575.429993, 520.650024, 288.75    , 552.780029,
369.029999, 315.      , 363.519989, 386.190002, 325.899994,
503.839996, 410.170013, 269.579987, 491.899994, 447.769989,
362.869995, 515.409973, 547.530029, 629.76001 , 641.900024,
573.140015, 602.440002, 490.649994, 530.869995, 374.230011,
386.700012, 539.440002, 379.23999 , 371.119995, 365.799988,
358.      , 632.659973, 302.859985, 503.380005, 361.450012,
337.48999 , 339.100006, 503.839996, 302.859985, 364.559998,
504.579987, 499.890015, 530.869995, 294.179993, 410.170013,
363.519989, 311.690002, 339.850006, 305.76001 , 315.25    ,
343.089996, 310.839996, 343.160004, 291.450012, 592.640015,
593.73999 , 520.700012, 268.149994, 494.730011, 380.399994,
356.559998, 518.909973, 503.380005, 346.459991, 418.649994,
641.900024, 681.169983, 597.98999 , 369.609985, 325.899994,
345.609985, 298.070007, 441.950012, 363.440002, 505.869995,
528.909973, 466.26001 , 374.130005, 510.399994, 367.320007,
352.600006, 337.48999 , 349.359985, 352.600006, 367.679993,
335.779999, 504.579987, 317.940002, 364.130005, 369.429993,
365.98999 , 270.600006, 266.690002, 480.670013, 369.670013,
278.140015, 500.859985, 368.48999 , 368.329987, 624.940002,
256.079987, 547.820007, 466.26001 , 413.549988, 310.619995,
352.600006, 275.329987, 529.559998, 349.730011, 266.769989,
379.23999 ])
```

[261]:

```python
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 30.48792478286902
```

[262]:

```python
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error: {mae:.2f}')
```

```
Mean Absolute Error: 4.03
```

[263]:

```python
# Calculate R-squared
r2 = r2_score(y_test, predictions)
print(f'R-squared: {r2:.2f}')
```

```
R-squared: 1.00
```

[264]:

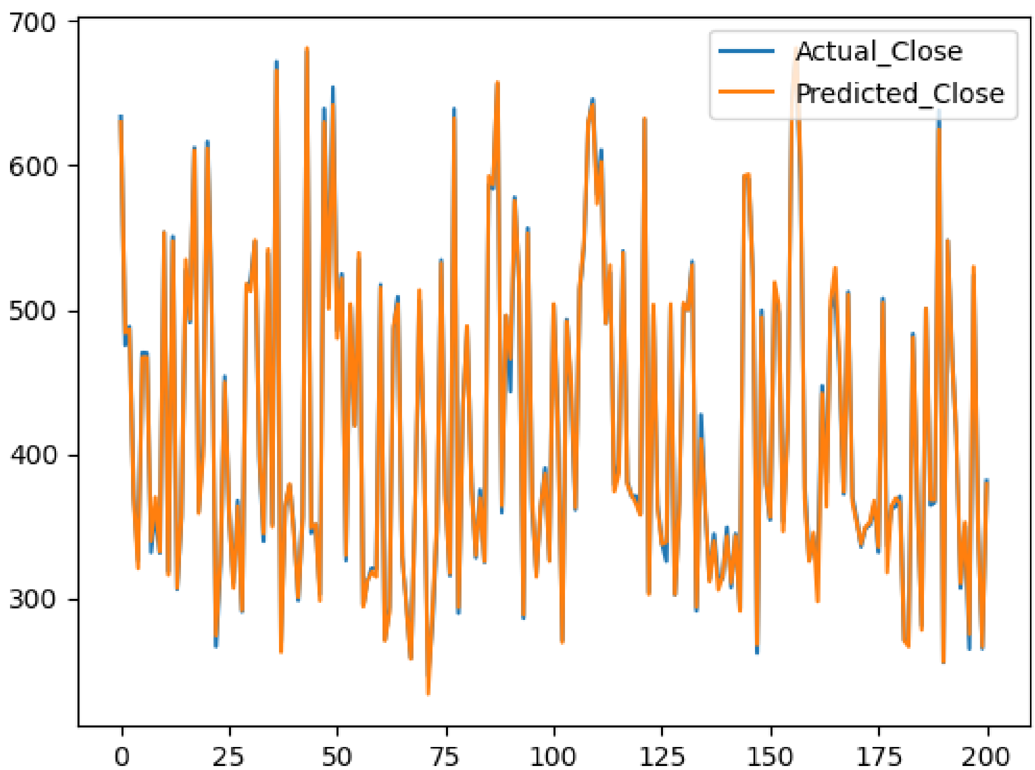
```python
df_results = pd.DataFrame({'Date': pd.to_datetime(y_test.index,␣
 ↪format='%Y-%m-%d'),
                           'Actual_Close': y_test.values,
                           'Predicted_Close': predictions})

# Display the new DataFrame
```

```
print(df_results)
```

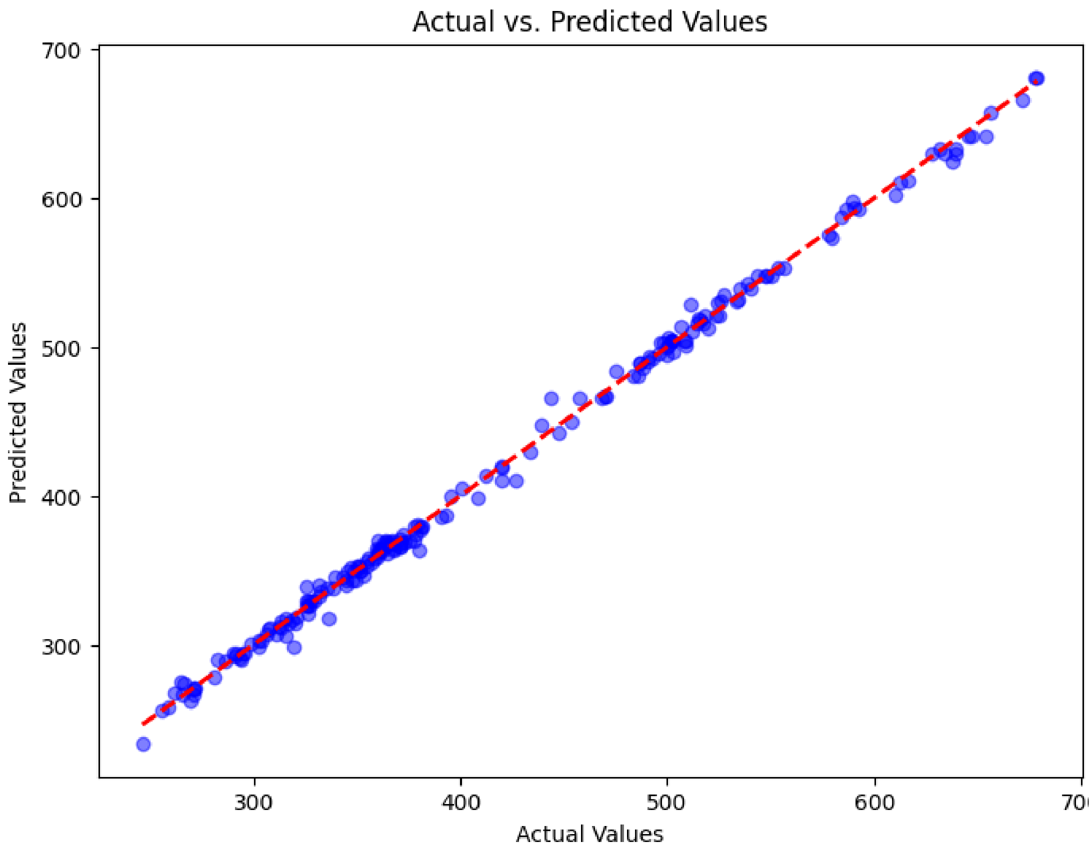
```
                                Date  Actual_Close  Predicted_Close
0    1970-01-01 00:00:00.000000930    633.799988       629.760010
1    1970-01-01 00:00:00.000000634    475.470001       484.119995
2    1970-01-01 00:00:00.000000686    488.239990       486.239990
3    1970-01-01 00:00:00.000000518    371.709991       368.769989
4    1970-01-01 00:00:00.000000369    326.459991       320.799988
..                             …           …              …
196  1970-01-01 00:00:00.000000212    265.140015       275.329987
197  1970-01-01 00:00:00.000000782    524.030029       529.559998
198  1970-01-01 00:00:00.000000338    351.269989       349.730011
199  1970-01-01 00:00:00.000000214    265.320007       266.769989
200  1970-01-01 00:00:00.000000354    381.720001       379.239990

[201 rows x 3 columns]
```

[265]:
```
df_results[['Actual_Close', 'Predicted_Close']].plot()
```

[265]: <Axes: >



[266]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, predictions, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],linestyle='--',␣
 ↪color='red', linewidth=2)
plt.title('Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



Bitcoin Dataset

In this project, I select Bitcoin datasets which play a crucial role as they form the foundation for training and evaluating machine learning models i.e decision tree for stock market price prediction. Dataset represents historical stock market data for different assets, and understanding their characteristics is vital for building effective predictive models.

Features of dataset * Date: Essential for organizing data chronologically and identifying trends over time. * Open: The opening price of Datasets on a given day. * High: The highest price of Datasets on a given day. * Low: The lowest price of Datasets on a given day. * Close: The closing price of Datasets on a given day. * Adj Close: The adjusted closing price of Datasets on a given day, considering dividends, stock splits, etc. * Volume: The volume of Datasets traded on a given day.

Load the dataset

[267]:
```
data = pd.read_csv('BTC-USD.csv')
print(data)
```

```
           Date          Open          High           Low         Close  \
0    2022-12-19  16759.041016  16807.527344  16398.136719  16439.679688
1    2022-12-20  16441.787109  17012.984375  16427.867188  16906.304688
2    2022-12-21  16904.527344  16916.800781  16755.912109  16817.535156
3    2022-12-22  16818.380859  16866.673828  16592.408203  16830.341797
4    2022-12-23  16829.644531  16905.218750  16794.458984  16796.953125
..          …             …             …             …             …
361  2023-12-15  43028.250000  43087.824219  41692.968750  41929.757813
362  2023-12-16  41937.742188  42664.945313  41723.113281  42240.117188
363  2023-12-17  42236.109375  42359.496094  41274.542969  41364.664063
364  2023-12-18  41348.203125  42720.296875  40530.257813  42623.539063
365  2023-12-19  42641.511719  43281.062500  41848.339844  42150.578125

        Adj Close       Volume
0    16439.679688  17221074814
1    16906.304688  22722096615
2    16817.535156  14882945045
3    16830.341797  16441573050
4    16796.953125  15329265213
..            …           …
361  41929.757813  19639442462
362  42240.117188  14386729590
363  41364.664063  16678702876
364  42623.539063  25224642008
365  42150.578125  25344405504

[366 rows x 7 columns]
```

Quick peek at functions:

[268]:
```
data.shape
```

[268]:
```
(366, 7)
```

[269]:
```
data.columns
```

[269]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

[270]:
```
print(data.describe())
```

```
              Open          High           Low         Close     Adj Close  \
count    366.000000    366.000000    366.000000    366.000000    366.000000
mean   27892.374450  28351.467635  27512.248116  27961.859242  27961.859242
std     5679.175786   5798.605380   5573.093646   5698.534708   5698.534708
min    16441.787109  16628.986328  16398.136719  16439.679688  16439.679688
25%    25614.489746  25957.333008  24999.646973  25754.951660  25754.951660
50%    27438.595703  27926.062500  26966.659179  27461.631836  27461.631836
75%    29913.611817  30364.928223  29675.398926  29975.025390  29975.025390
max    44180.019531  44705.515625  43627.597656  44166.601563  44166.601563

             Volume
count  3.660000e+02
mean   1.802335e+10
std    8.494735e+09
min    5.331173e+09
25%    1.200029e+10
50%    1.575343e+10
75%    2.262914e+10
max    5.462223e+10
```

[271]:
```python
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       366 non-null    object
 1   Open       366 non-null    float64
 2   High       366 non-null    float64
 3   Low        366 non-null    float64
 4   Close      366 non-null    float64
 5   Adj Close  366 non-null    float64
 6   Volume     366 non-null    int64
dtypes: float64(5), int64(1), object(1)
memory usage: 20.1+ KB
None
```

Data Preprocessing

Handling Missing Values:

[272]:
```python
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)
data = data.dropna()
print("Missing Values After Handling:\n", data.isnull().sum())
```

```
Missing Values:
 Date         0
```

```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
Missing Values After Handling:
 Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

Feature Scaling

[273]:
```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data[['High']] = scaler.fit_transform(data[['High']])
data[['Low']] = scaler.fit_transform(data[['Low']])
data
```

[273]:
```
           Date          Open      High       Low         Close     Adj Close  \
0    2022-12-19  16759.041016  0.006359  0.000000  16439.679688  16439.679688
1    2022-12-20  16441.787109  0.013677  0.001092  16906.304688  16906.304688
2    2022-12-21  16904.527344  0.010251  0.013139  16817.535156  16817.535156
3    2022-12-22  16818.380859  0.008466  0.007135  16830.341797  16830.341797
4    2022-12-23  16829.644531  0.009839  0.014555  16796.953125  16796.953125
..          …             …         …         …             …             …
361  2023-12-15  43028.250000  0.942383  0.928951  41929.757813  41929.757813
362  2023-12-16  41937.742188  0.927321  0.930058  42240.117188  42240.117188
363  2023-12-17  42236.109375  0.916442  0.913584  41364.664063  41364.664063
364  2023-12-18  41348.203125  0.929293  0.886250  42623.539063  42623.539063
365  2023-12-19  42641.511719  0.949265  0.934657  42150.578125  42150.578125

          Volume
0    17221074814
1    22722096615
2    14882945045
3    16441573050
4    15329265213
..           …
361  19639442462
362  14386729590
363  16678702876
```

```
364   25224642008
365   25344405504

[366 rows x 7 columns]
```

Feature Engineering

[274]:
```python
data['DailyReturn'] = data['Adj Close'].pct_change() * 100
data['MovingAverage'] = data['Adj Close'].rolling(window=5).mean()
data['PriceToVolumeRatio'] = data['Adj Close']/data['Volume']
data=data.dropna()
data
```

[274]:
```
           Date          Open      High       Low         Close     Adj Close  \
4    2022-12-23  16829.644531  0.009839  0.014555  16796.953125  16796.953125
5    2022-12-24  16796.976563  0.008396  0.014521  16847.755859  16847.755859
6    2022-12-25  16847.505859  0.008248  0.013115  16841.986328  16841.986328
7    2022-12-26  16842.250000  0.010369  0.015213  16919.804688  16919.804688
8    2022-12-27  16919.291016  0.011784  0.008959  16717.173828  16717.173828
..          …             …         …         …             …            …
361  2023-12-15  43028.250000  0.942383  0.928951  41929.757813  41929.757813
362  2023-12-16  41937.742188  0.927321  0.930058  42240.117188  42240.117188
363  2023-12-17  42236.109375  0.916442  0.913584  41364.664063  41364.664063
364  2023-12-18  41348.203125  0.929293  0.886250  42623.539063  42623.539063
365  2023-12-19  42641.511719  0.949265  0.934657  42150.578125  42150.578125

          Volume  DailyReturn  MovingAverage  PriceToVolumeRatio
4    15329265213    -0.198384   16758.162891            0.000001
5     9744636213     0.302452   16839.778125            0.000002
6    11656379938    -0.034245   16826.914453            0.000001
7    11886957804     0.462050   16847.368359            0.000001
8    15748580239    -1.197596   16824.734766            0.000001
..           …           …             …                  …
361  19639442462    -2.543268   42107.705469            0.000002
362  14386729590     0.740189   42306.962500            0.000003
363  16678702876    -2.072563   42289.850782            0.000002
364  25224642008     3.043358   42236.410157            0.000002
365  25344405504    -1.109624   42061.731250            0.000002

[362 rows x 10 columns]
```

Data Splitting and Model Training

[275]:
```python
features = ['Open', 'High', 'Low', 'Volume', 'DailyReturn', 'MovingAverage',␣
 ↪'PriceToVolumeRatio']
X = data[features]
Y = data['Close']
print(X.columns)
```

```
print(Y.name)
```

```
Index(['Open', 'High', 'Low', 'Volume', 'DailyReturn', 'MovingAverage',
       'PriceToVolumeRatio'],
      dtype='object')
Close
```

[276]:
```
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
```

[277]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2,␣
↪random_state=42)
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (289, 7)
X_test shape: (73, 7)
y_train shape: (289,)
y_test shape: (73,)
```

[278]:
```
model = DecisionTreeRegressor()
```

Model Evaluation

[279]:
```
model.fit(X_train, y_train)
predictions = model.predict(X_test)
predictions
```

[279]:
```
array([27132.007813, 22777.625   , 17091.144531, 36693.125   ,
       24327.642578, 30485.699219, 21651.183594, 27493.285156,
       26579.390625, 29178.679688, 25940.167969, 22934.431641,
       29397.714844, 23947.492188, 27591.384766, 30485.699219,
       25800.724609, 29170.347656, 34938.242188, 27530.785156,
       16547.496094, 30688.164063, 22353.349609, 27494.707031,
       37831.085938, 30391.646484, 29397.714844, 26608.693359,
       26930.638672, 42623.539063, 20688.78125 , 29473.787109,
       23475.466797, 43746.445313, 26539.673828, 30342.265625,
       23522.871094, 21086.792969, 28328.341797, 25576.394531,
       26106.150391, 39476.332031, 21819.039063, 27297.265625,
       30399.066406, 42623.539063, 28038.675781, 23947.492188,
       16717.173828, 22934.431641, 29429.591797, 27968.839844,
       27021.546875, 29792.015625, 25833.34375 , 37880.582031,
       28044.140625, 16836.736328, 27297.265625, 19909.574219,
       37476.957031, 26008.462891, 21086.792969, 22840.138672,
       26334.818359, 22219.769531, 29178.679688, 28033.5625  ,
       28033.5625  , 24436.353516, 27583.677734, 37796.792969,
```

```
29792.015625])
```

[280]:
```
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 113528.4115859737
```

[281]:
```
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error: {mae:.2f}')
```

```
Mean Absolute Error: 250.19
```

[282]:
```
# Calculate R-squared
r2 = r2_score(y_test, predictions)
print(f'R-squared: {r2:.2f}')
```

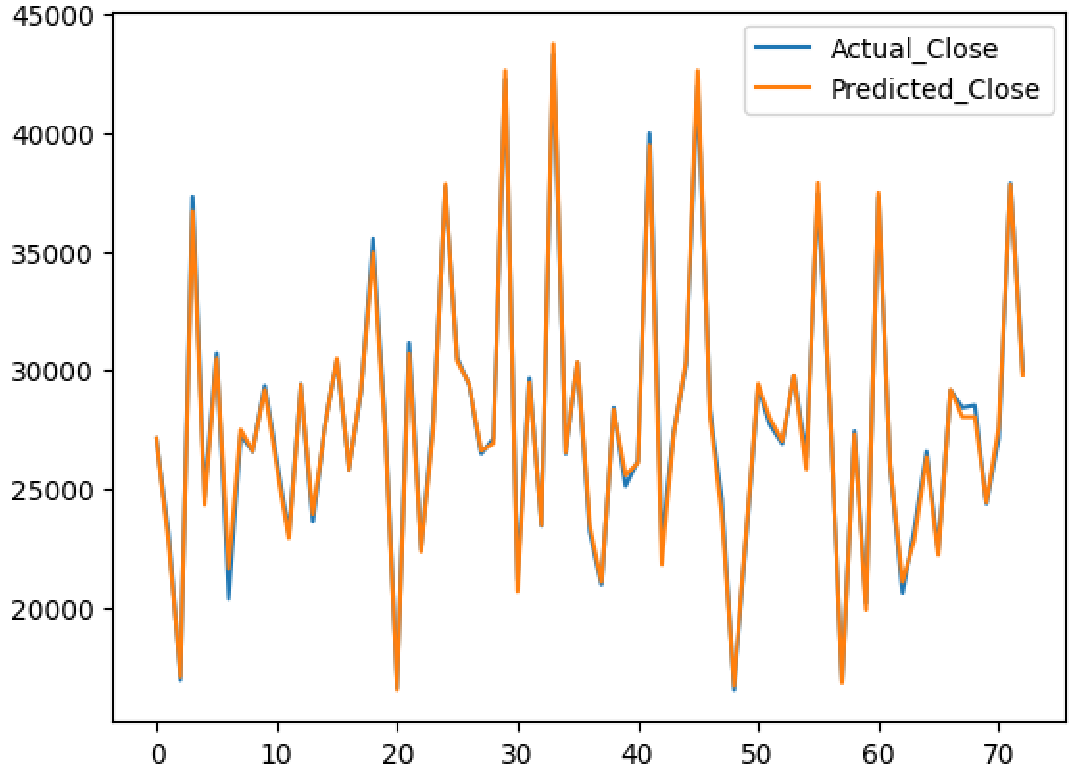
```
R-squared: 1.00
```

[283]:
```
df_results = pd.DataFrame({'Date': pd.to_datetime(y_test.index,␣
 ↪format='%Y-%m-%d'),
                           'Actual_Close': y_test.values,
                           'Predicted_Close': predictions})

# Display the new DataFrame
print(df_results)
```

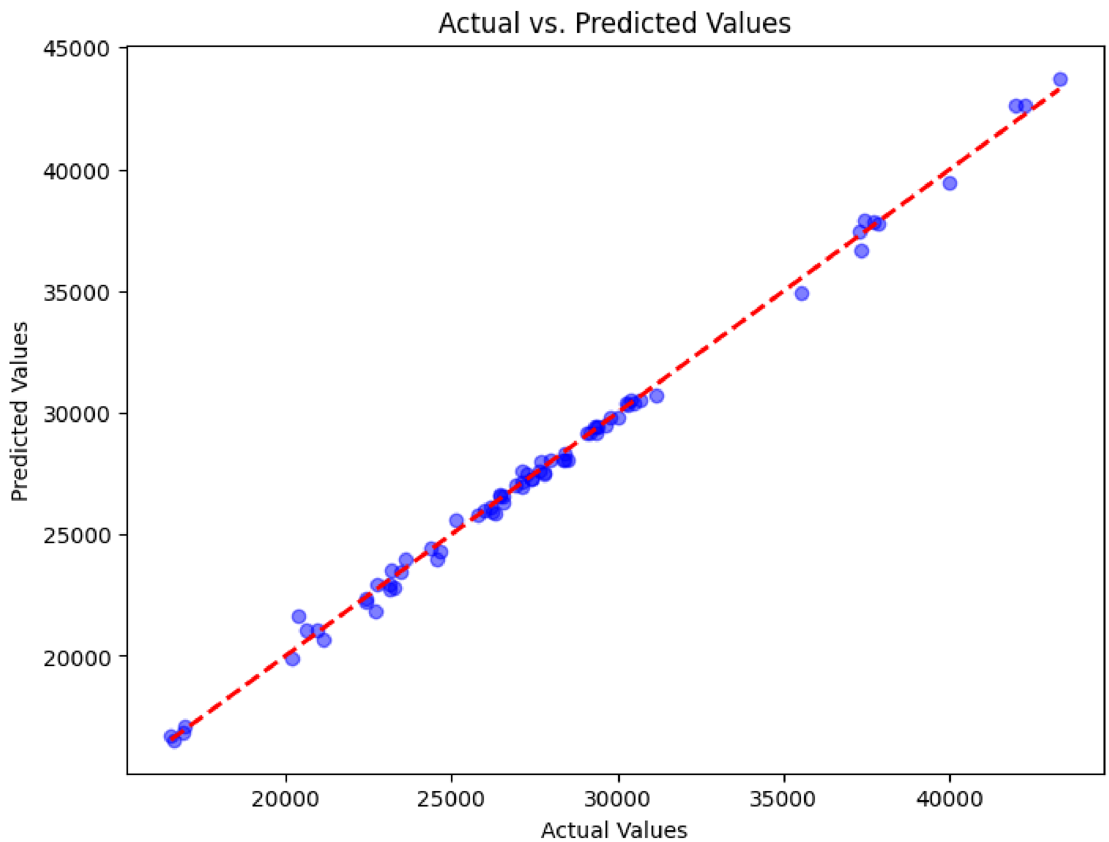
```
                            Date  Actual_Close  Predicted_Close
0  1970-01-01 00:00:00.000000167  27119.066406     27132.007813
1  1970-01-01 00:00:00.000000037  23117.859375     22777.625000
2  1970-01-01 00:00:00.000000019  16955.078125     17091.144531
3  1970-01-01 00:00:00.000000326  37313.968750     36693.125000
4  1970-01-01 00:00:00.000000061  24641.277344     24327.642578
..                           …             …                …
68 1970-01-01 00:00:00.000000301  28519.466797     28033.562500
69 1970-01-01 00:00:00.000000086  24375.960938     24436.353516
70 1970-01-01 00:00:00.000000098  27139.888672     27583.677734
71 1970-01-01 00:00:00.000000345  37858.492188     37796.792969
72 1970-01-01 00:00:00.000000307  29993.896484     29792.015625

[73 rows x 3 columns]
```

[284]:
```
df_results[['Actual_Close', 'Predicted_Close']].plot()
```

[284]: <Axes: >



[285]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, predictions, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],linestyle='--',␣
 ↪color='red', linewidth=2)
plt.title('Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



Binance Dataset

In this project, I select Binance datasets which play a crucial role as they form the foundation for training and evaluating machine learning models i.e decision tree for stock market price prediction. Dataset represents historical stock market data for different assets, and understanding their characteristics is vital for building effective predictive models.

Features of dataset * Date: Essential for organizing data chronologically and identifying trends over time. * Open: The opening price of Datasets on a given day. * High: The highest price of Datasets on a given day. * Low: The lowest price of Datasets on a given day. * Close: The closing price of Datasets on a given day. * Adj Close: The adjusted closing price of Datasets on a given day, considering dividends, stock splits, etc. * Volume: The volume of Datasets traded on a given day.

Load the dataset

[286]:
```
data = pd.read_csv('BNB-USD.csv')
print(data)
```

```
         Date        Open        High         Low       Close   Adj Close  \
0  2022-12-19  251.242676  252.933014  238.650787  240.657806  240.657806
```

```
1     2022-12-20  240.668228  252.628662  239.801437  251.744537  251.744537
2     2022-12-21  251.694321  251.694321  245.757248  246.046982  246.046982
3     2022-12-22  246.068329  248.032028  240.483200  245.890625  245.890625
4     2022-12-23  245.894135  248.274719  244.452942  246.148178  246.148178
..           …           …           …           …           …           …
361   2023-12-15  253.517441  253.549713  243.867371  244.898438  244.898438
362   2023-12-16  244.896423  248.086380  243.450653  244.350967  244.350967
363   2023-12-17  244.350708  244.432175  239.230637  239.308289  239.308289
364   2023-12-18  239.247147  241.348434  232.752808  241.348434  241.348434
365   2023-12-19  241.347687  253.778625  241.347687  253.105240  253.105240

         Volume
0     751196285
1     667866377
2     479296549
3     543367184
4     388929772
..          …
361   769388533
362   651447427
363   650163942
364   871708609
365  1226686976

[366 rows x 7 columns]
```

Quick peek at functions:

[287]:
```python
data.shape
```

[287]:
```
(366, 7)
```

[288]:
```python
data.columns
```

[288]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

[289]:
```python
print(data.describe())
```

```
             Open        High         Low       Close   Adj Close  \
count  366.000000  366.000000  366.000000  366.000000  366.000000
mean   265.041061  269.089091  261.058403  265.037719  265.037719
std     41.667309   42.656486   40.717153   41.663551   41.663551
min    205.225800  206.659103  203.655441  205.229416  205.229416
25%    231.900402  236.389728  228.605893  231.913357  231.913357
50%    246.355537  251.505004  242.926544  246.388756  246.388756
75%    308.557312  313.169899  304.356903  308.555268  308.555268
max    348.175751  350.072296  338.260620  348.220917  348.220917
```

```
             Volume
count  3.660000e+02
mean   5.566887e+08
std    2.657211e+08
min    2.038465e+08
25%    3.765235e+08
50%    4.849198e+08
75%    6.678265e+08
max    2.480554e+09
```

[290]:
```python
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       366 non-null    object
 1   Open       366 non-null    float64
 2   High       366 non-null    float64
 3   Low        366 non-null    float64
 4   Close      366 non-null    float64
 5   Adj Close  366 non-null    float64
 6   Volume     366 non-null    int64
dtypes: float64(5), int64(1), object(1)
memory usage: 20.1+ KB
None
```

Data Preprocessing

Handling Missing Values:

[291]:
```python
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)
data = data.dropna()
print("Missing Values After Handling:\n", data.isnull().sum())
```

```
Missing Values:
 Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
Missing Values After Handling:
 Date         0
```

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

Feature Scaling

[292]:
```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data[['High']] = scaler.fit_transform(data[['High']])
data[['Low']] = scaler.fit_transform(data[['Low']])
data
```

[292]:
```
           Date        Open      High       Low       Close   Adj Close  \
0    2022-12-19  251.242676  0.322661  0.259985  240.657806  240.657806
1    2022-12-20  240.668228  0.320539  0.268533  251.744537  251.744537
2    2022-12-21  251.694321  0.314024  0.312780  246.046982  246.046982
3    2022-12-22  246.068329  0.288488  0.273598  245.890625  245.890625
4    2022-12-23  245.894135  0.290180  0.303090  246.148178  246.148178
..          …           …         …         …           …           …
361  2023-12-15  253.517441  0.326962  0.298740  244.898438  244.898438
362  2023-12-16  244.896423  0.288867  0.295644  244.350967  244.350967
363  2023-12-17  244.350708  0.263386  0.264293  239.308289  239.308289
364  2023-12-18  239.247147  0.241884  0.216168  241.348434  241.348434
365  2023-12-19  241.347687  0.328558  0.280021  253.105240  253.105240

         Volume
0     751196285
1     667866377
2     479296549
3     543367184
4     388929772
..          …
361   769388533
362   651447427
363   650163942
364   871708609
365  1226686976

[366 rows x 7 columns]
```

Feature Engineering

[293]:
```python
data['DailyReturn'] = data['Adj Close'].pct_change() * 100
data['MovingAverage'] = data['Adj Close'].rolling(window=5).mean()
```

```
data['PriceToVolumeRatio'] = data['Adj Close']/data['Volume']
data=data.dropna()
data
```

[293]:

```
           Date        Open      High       Low       Close   Adj Close  \
4    2022-12-23  245.894135  0.290180  0.303090  246.148178  246.148178
5    2022-12-24  246.151642  0.275586  0.299376  244.635529  244.635529
6    2022-12-25  244.636398  0.272443  0.282642  243.141495  243.141495
7    2022-12-26  243.147934  0.264528  0.286534  244.198288  244.198288
8    2022-12-27  244.202652  0.286031  0.289043  246.596680  246.596680
..          …           …         …         …           …           …
361  2023-12-15  253.517441  0.326962  0.298740  244.898438  244.898438
362  2023-12-16  244.896423  0.288867  0.295644  244.350967  244.350967
363  2023-12-17  244.350708  0.263386  0.264293  239.308289  239.308289
364  2023-12-18  239.247147  0.241884  0.216168  241.348434  241.348434
365  2023-12-19  241.347687  0.328558  0.280021  253.105240  253.105240

         Volume  DailyReturn  MovingAverage  PriceToVolumeRatio
4     388929772     0.104743     246.097626        6.328859e-07
5     280627376    -0.614528     246.893170        8.717451e-07
6     298063868    -0.610718     245.172562        8.157362e-07
7     276115280     0.434641     244.802823        8.844070e-07
8     391342277     0.982149     244.944034        6.301304e-07
..          …            …              …                   …
361   769388533    -3.408825     250.355765        3.183027e-07
362   651447427    -0.223550     249.941785        3.750893e-07
363   650163942    -2.063703     246.904447        3.680738e-07
364   871708609     0.852517     244.689468        2.768682e-07
365  1226686976     4.871300     244.602274        2.063324e-07

[362 rows x 10 columns]
```

Data Splitting and Model Training

[294]:

```
features = ['Open', 'High', 'Low', 'Volume', 'DailyReturn', 'MovingAverage',␣
 ↪'PriceToVolumeRatio']
X = data[features]
Y = data['Close']
print(X.columns)
print(Y.name)
```

```
Index(['Open', 'High', 'Low', 'Volume', 'DailyReturn', 'MovingAverage',
       'PriceToVolumeRatio'],
      dtype='object')
Close
```

[295]:
```
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
```

[296]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2,␣
↪random_state=42)
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (289, 7)
X_test shape: (73, 7)
y_train shape: (289,)
y_test shape: (73,)
```

[297]:
```
model = DecisionTreeRegressor()
```

Model Evaluation

[298]:
```
model.fit(X_train, y_train)
predictions = model.predict(X_test)
predictions
```

[298]:
```
array([304.953278, 308.912201, 272.701538, 251.421616, 317.040009,
       247.75943 , 277.289642, 323.349792, 216.465775, 321.887726,
       216.465775, 309.245636, 237.808563, 323.349792, 313.928619,
       243.890533, 215.134659, 242.733139, 239.089081, 307.695831,
       244.898438, 246.349197, 290.282837, 212.282837, 234.439941,
       239.706894, 242.798599, 259.867157, 308.964294, 246.148178,
       304.973175, 314.063171, 331.995087, 229.424408, 305.954132,
       336.978241, 306.866699, 308.912201, 210.679672, 238.946213,
       214.363327, 229.300842, 309.245636, 210.99437 , 323.349792,
       229.300842, 326.212891, 315.377441, 242.65683 , 321.952698,
       241.977036, 314.083527, 215.075989, 241.797501, 241.348434,
       229.692963, 321.611053, 243.141495, 214.448547, 277.289642,
       236.280685, 214.305664, 277.289642, 337.645538, 210.638947,
       289.343689, 238.946213, 328.724213, 212.378448, 307.068878,
       318.953766, 229.300842, 214.821304])
```

[299]:
```
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 15.029912793055043
```

[300]:
```
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error: {mae:.2f}')
```

```
Mean Absolute Error: 2.48
```

[301]:
```
# Calculate R-squared
r2 = r2_score(y_test, predictions)
print(f'R-squared: {r2:.2f}')
```

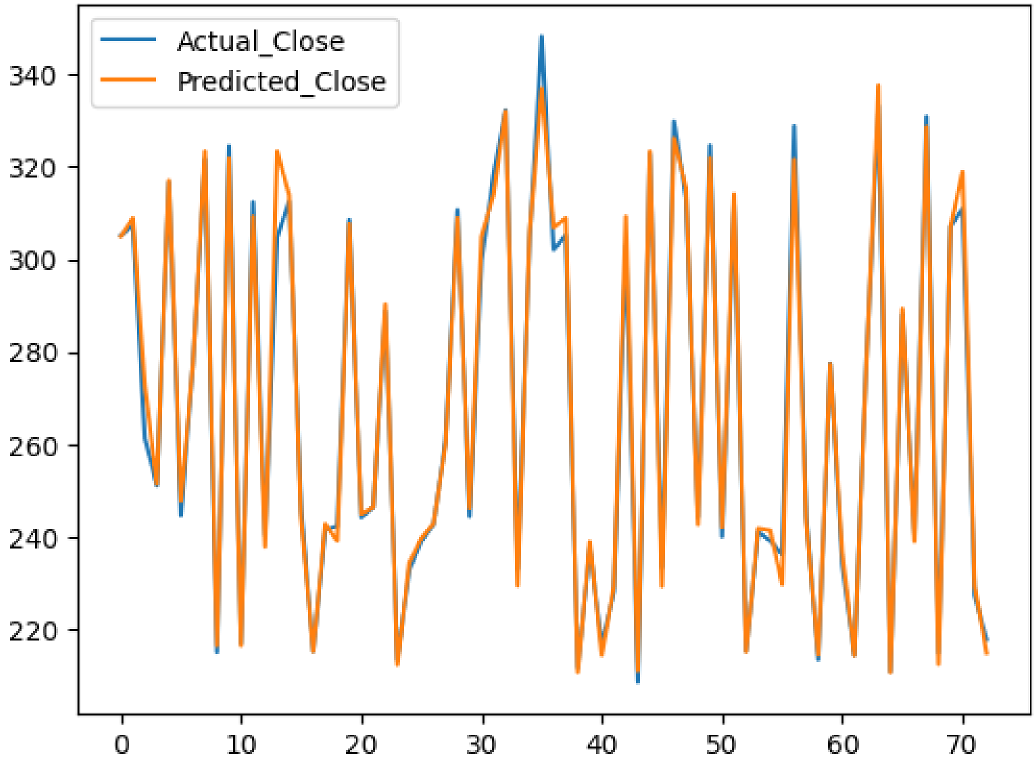
```
R-squared: 0.99
```

[302]:
```
df_results = pd.DataFrame({'Date': pd.to_datetime(y_test.index,␣
 ↪format='%Y-%m-%d'),
                           'Actual_Close': y_test.values,
                           'Predicted_Close': predictions})

# Display the new DataFrame
print(df_results)
```

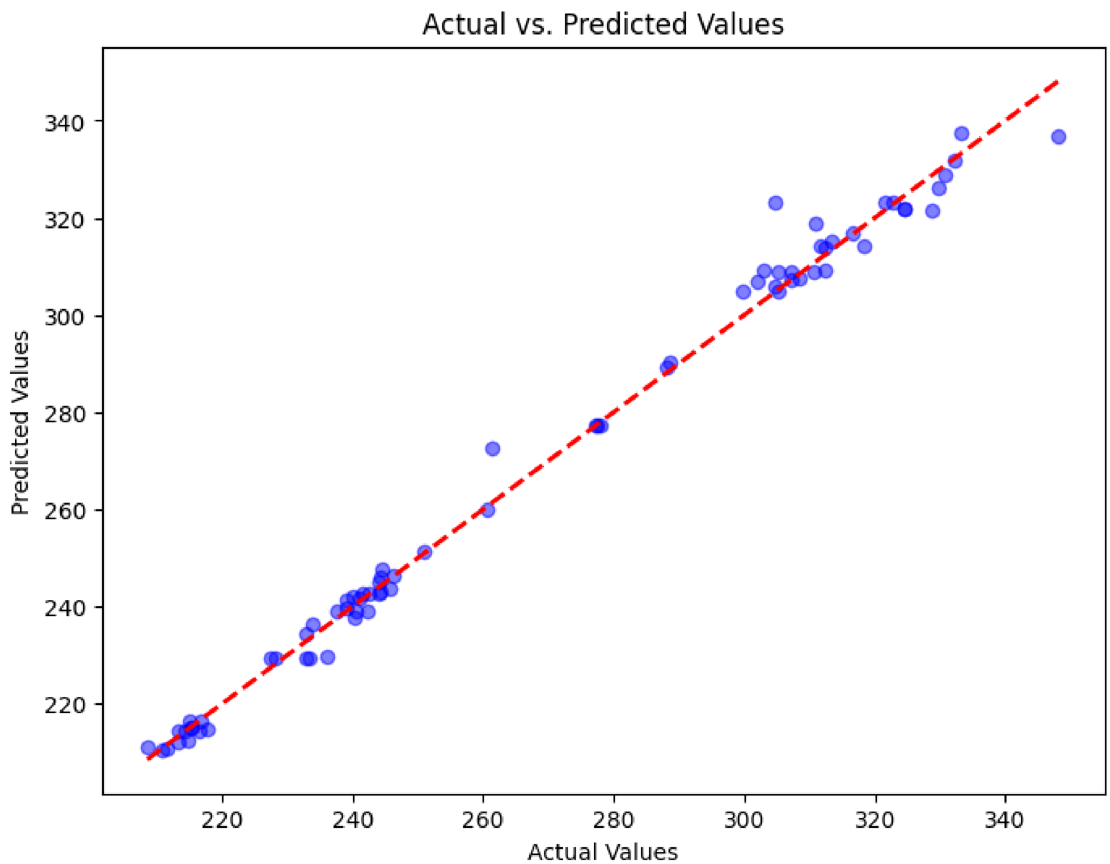
```
                            Date  Actual_Close  Predicted_Close
0  1970-01-01 00:00:00.000000167    305.156799       304.953278
1  1970-01-01 00:00:00.000000037    307.307648       308.912201
2  1970-01-01 00:00:00.000000019    261.282837       272.701538
3  1970-01-01 00:00:00.000000326    251.082367       251.421616
4  1970-01-01 00:00:00.000000061    316.682709       317.040009
..                           …           …              …
68 1970-01-01 00:00:00.000000301    214.823959       212.378448
69 1970-01-01 00:00:00.000000086    307.124939       307.068878
70 1970-01-01 00:00:00.000000098    310.949127       318.953766
71 1970-01-01 00:00:00.000000345    227.342758       229.300842
72 1970-01-01 00:00:00.000000307    217.747375       214.821304

[73 rows x 3 columns]
```

[303]:
```
df_results[['Actual_Close', 'Predicted_Close']].plot()
```

[303]: <Axes: >



[304]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, predictions, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],linestyle='--',␣
 ↪color='red', linewidth=2)
plt.title('Actual vs. Predicted Values')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```


Actual vs. Predicted Values
340
320
300
280
260
240
220
Predicted Values
220
240
260
280
300
320
340
Actual Values