prediction-using-linear-regression

January 14, 2024

Stock Market Price Prediction using Linear Regression

Importing important libraries

```
[198]: 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 Linear Regression 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

```
[199]: 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:

```
[200]:  data.shape
```

```
[200]:  (253, 7)
```

```
[201]:  data.columns
```

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

```
[202]:  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
```

[203]: `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:

[204]:
```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

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

```
[205]:            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

```python
[206]: 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()
```

4

```
data
```

```
[206]:            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

```
[207]: 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
```

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

```
[209]: 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,)
```

```
[210]: model = LinearRegression()
```

Model Evaluation

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

```
[211]: array([348.1907123 , 270.78589377, 258.2871296 , 357.59659895,
              279.27878679, 232.44825668, 243.89604557, 233.94078808,
              290.25163363, 280.28747999, 275.72083563, 297.92993748,
              267.29696863, 299.93555655, 346.43900509, 300.04113158,
              237.05777353, 346.45178154, 237.21013379, 384.71536719,
              285.90976854, 396.92573935, 298.65227684, 353.44704664,
              234.70683464, 353.46778476, 336.01322822, 287.73846273,
              349.5052757 , 366.24638617, 273.32053511, 375.05202409,
              297.49041901, 223.6441999 , 351.79650452, 266.92148547,
              367.43888429, 240.28344645, 213.94777134, 219.42979926,
              335.18694721, 273.10481998, 273.43962405, 225.56079037,
              250.1817186 , 349.97138204, 234.94368095, 368.3420379 ,
              230.51596643, 380.71677181])
```

```
[212]: print('Model Coefficients: \n', model.coef_)
       print('Model Intercept: \n', model.intercept_)
```

```
Model Coefficients:
 [-9.38906926e-02  8.13059532e+01  1.04863781e+02  4.73729074e-08
  1.20016289e+00  1.48664618e-01  6.73318266e+05]
Model Intercept:
 194.69111786880944
```

```
[213]: print("Performance (R^2): ", model.score(X_train, y_train))
```

```
Performance (R^2):  0.995311871841997
```

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

Mean Squared Error: 13.267188620192698

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

Mean Absolute Error: 2.69

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

R-squared: 1.00

```
[217]: 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)
```
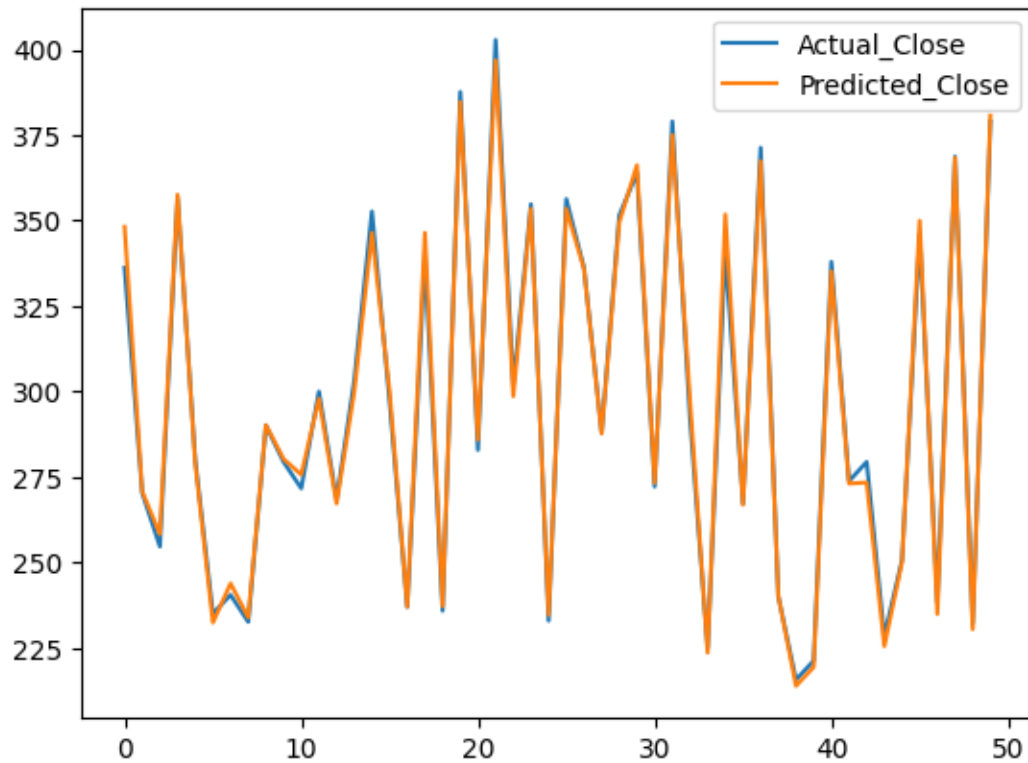
```
                             Date  Actual_Close  Predicted_Close
    0   1970-01-01 00:00:00.000000141    336.260010        348.190712
    1   1970-01-01 00:00:00.000000010    270.359985        270.785894
    2   1970-01-01 00:00:00.000000101    254.679993        258.287130
    3   1970-01-01 00:00:00.000000064    356.779999        357.596599
    4   1970-01-01 00:00:00.000000116    280.076660        279.278787
    5   1970-01-01 00:00:00.000000184    235.070007        232.448257
    6   1970-01-01 00:00:00.000000200    240.546661        243.896046
    7   1970-01-01 00:00:00.000000187    232.663330        233.940788
    8   1970-01-01 00:00:00.000000013    290.036682        290.251634
    9   1970-01-01 00:00:00.000000108    279.429993        280.287480
    10  1970-01-01 00:00:00.000000203    271.706665        275.720836
    11  1970-01-01 00:00:00.000000219    300.029999        297.929937
    12  1970-01-01 00:00:00.000000205    268.433319        267.296969
    13  1970-01-01 00:00:00.000000243    303.350006        299.935557
    14  1970-01-01 00:00:00.000000071    352.706665        346.439005
    15  1970-01-01 00:00:00.000000227    297.096680        300.041132
    16  1970-01-01 00:00:00.000000197    237.039993        237.057774
    17  1970-01-01 00:00:00.000000019    339.476654        346.451782
    18  1970-01-01 00:00:00.000000166    235.910004        237.210134
    19  1970-01-01 00:00:00.000000028    387.646667        384.715367
    20  1970-01-01 00:00:00.000000250    282.940002        285.909769
    21  1970-01-01 00:00:00.000000023    402.863342        396.925739
```
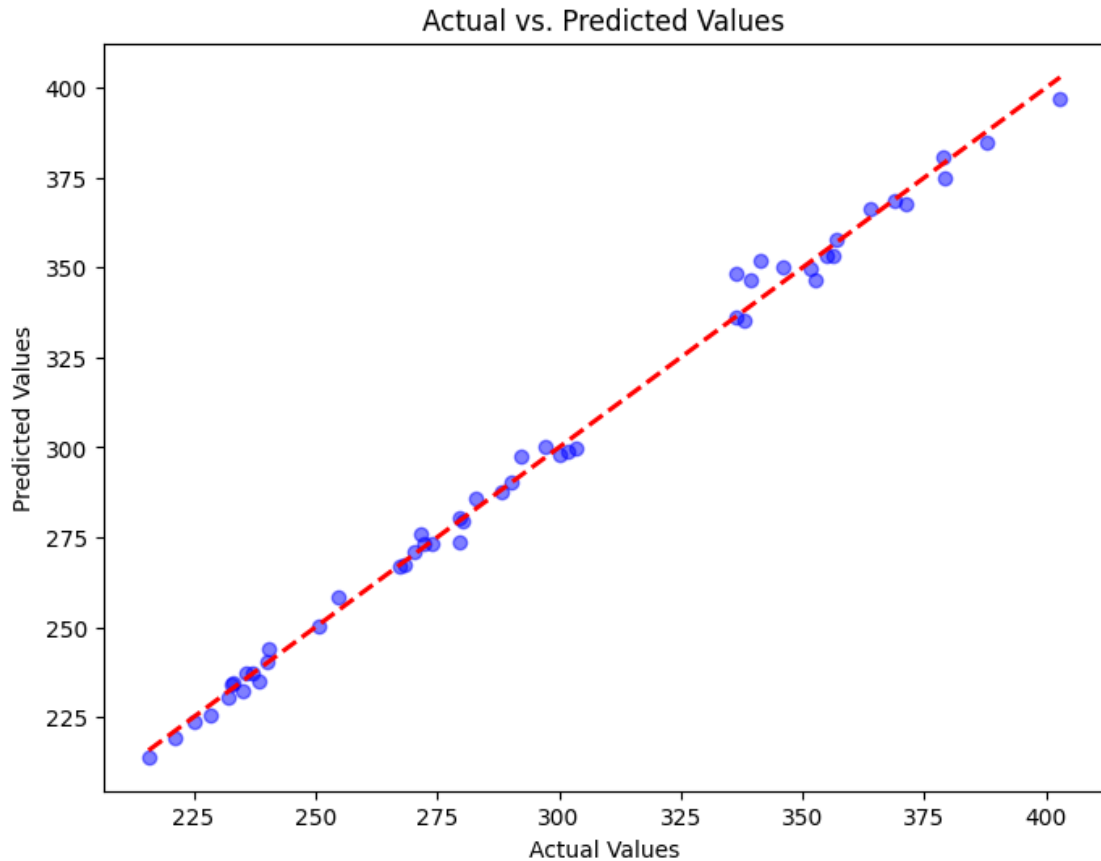
```
22 1970-01-01 00:00:00.000000118     301.796661        298.652277
23 1970-01-01 00:00:00.000000072     354.799988        353.447047
24 1970-01-01 00:00:00.000000179     233.000000        234.706835
25 1970-01-01 00:00:00.000000049     356.320007        353.467785
26 1970-01-01 00:00:00.000000059     336.290009        336.013228
27 1970-01-01 00:00:00.000000014     288.089996        287.738463
28 1970-01-01 00:00:00.000000034     351.576660        349.505276
29 1970-01-01 00:00:00.000000124     363.946655        366.246386
30 1970-01-01 00:00:00.000000204     272.243347        273.320535
31 1970-01-01 00:00:00.000000037     379.019989        375.052024
32 1970-01-01 00:00:00.000000144     292.140015        297.490419
33 1970-01-01 00:00:00.000000163     224.966660        223.644200
34 1970-01-01 00:00:00.000000029     341.166656        351.796505
35 1970-01-01 00:00:00.000000115     267.296661        266.921485
36 1970-01-01 00:00:00.000000022     371.333344        367.438884
37 1970-01-01 00:00:00.000000199     240.066666        240.283446
38 1970-01-01 00:00:00.000000177     215.736664        213.947771
39 1970-01-01 00:00:00.000000162     221.300003        219.429799
40 1970-01-01 00:00:00.000000122     337.973328        335.186947
41 1970-01-01 00:00:00.000000100     273.843323        273.104820
42 1970-01-01 00:00:00.000000112     279.433319        273.439624
43 1970-01-01 00:00:00.000000188     228.490005        225.560790
44 1970-01-01 00:00:00.000000194     250.763336        250.181719
45 1970-01-01 00:00:00.000000020     345.953339        349.971382
46 1970-01-01 00:00:00.000000198     238.313339        234.943681
47 1970-01-01 00:00:00.000000073     368.739990        368.342038
48 1970-01-01 00:00:00.000000176     232.229996        230.515966
49 1970-01-01 00:00:00.000000042     378.996674        380.716772
```

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

[218]: `<Axes: >`

```
[219]: 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

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 Linear Regression 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

```
[220]: 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:

[221]: `data.shape`

[221]: (1009, 7)

[222]: `data.columns`

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

[223]: `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
```

[224]: `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:

[225]:
```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

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

```
[226]:             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

```
[227]: 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
```

[227]:

	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

[228]:
```
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
```

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

```
[230]: 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,)
```

```
[231]: model = LinearRegression()
```

Model Evaluation

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

```
[232]: array([632.61270693, 481.22701004, 488.96217951, 379.51519084,
              325.43048915, 467.60677527, 469.4260359 , 339.58004573,
              365.80901347, 332.8271788 , 551.86449529, 316.11884671,
              550.43087097, 304.65217292, 357.26570826, 522.00343059,
              489.22474189, 617.68249918, 358.17573946, 405.6530356 ,
              622.8343465 , 500.62089829, 266.96400631, 325.45251539,
              449.8116748 , 360.56319167, 309.73479084, 364.42007048,
              289.9407526 , 519.00683043, 516.92836766, 546.16392241,
              393.86635033, 341.37678002, 548.02245115, 351.63110003,
              665.95930781, 270.38133503, 365.40245605, 379.31570665,
              346.50421287, 304.83343514, 354.77018472, 682.32915681,
              344.00749895, 350.06205724, 303.88920407, 633.02704431,
              506.45281172, 657.91366516, 484.29907276, 523.08546024,
              326.04245303, 501.55848007, 420.07251222, 538.03079431,
              293.45085522, 314.3414298 , 320.16545822, 320.77352429,
              516.89221532, 271.52433847, 294.70078812, 488.70022007,
              504.2886236 , 329.34670224, 294.18841274, 260.90249707,
              364.54061165, 512.37922972, 401.75248478, 245.69502106,
              284.91210442, 345.73623861, 530.29279267, 375.1937541 ,
              316.11608009, 636.14072487, 294.79207961, 433.63293478,
              487.56391465, 377.69700363, 328.54164434, 373.88336055,
              325.37596963, 584.64983007, 588.0353875 , 658.42451705,
              364.24367625, 494.35558164, 446.64560716, 580.00504592,
              517.44413147, 286.39982296, 556.28858562, 359.32330372,
              313.99179367, 361.19773028, 388.82588707, 326.32659024,
```

15

```
                 506.99875969, 413.40006017, 275.37503295, 494.94420718,
                 440.85413602, 361.4152411 , 514.83509353, 541.47742751,
                 626.81375952, 651.15172148, 570.24873001, 611.43675046,
                 489.50123985, 529.43853957, 376.66032123, 392.50708266,
                 538.85064091, 376.85083914, 369.78571894, 374.27504525,
                 359.68119478, 634.07723103, 304.32167852, 494.77735607,
                 362.56843888, 338.0602494 , 328.1632136 , 502.79941379,
                 300.40769419, 365.03058654, 502.46859876, 498.16933608,
                 534.22348099, 291.94641753, 426.45023122, 362.47674289,
                 310.51987974, 343.13138546, 317.7055651 , 310.35436881,
                 348.84098966, 307.48706182, 340.91174292, 290.69714933,
                 587.28059855, 587.43476253, 526.51086332, 265.91861117,
                 497.58407729, 380.87102632, 357.20559207, 514.50741972,
                 498.59721758, 352.29950593, 418.1054378 , 649.85105906,
                 678.92120886, 588.22147946, 375.48160695, 324.34905024,
                 343.33447469, 318.38599193, 442.0654303 , 378.71667665,
                 503.24605054, 518.70145686, 458.6848658 , 370.59809049,
                 511.29183769, 371.45529121, 354.19830378, 335.93302863,
                 349.51269275, 352.24633931, 368.02486293, 332.02833573,
                 507.95578024, 331.56858944, 362.66040307, 367.07714543,
                 367.1093099 , 269.28179763, 274.08336857, 485.38008772,
                 367.91091982, 279.67665211, 502.09191049, 367.24825804,
                 365.69476993, 630.8837519 , 248.97250873, 547.26269295,
                 465.25896479, 410.55543452, 310.31873081, 348.43532991,
                 264.83108993, 528.65136688, 352.5373946 , 265.84747282,
                 380.78917196])
```

[233]:
```python
print('Model Coefficients: \n', model.coef_)
print('Model Intercept: \n', model.intercept_)
```

```
Model Coefficients:
 [-2.43821406e-01  2.40274181e+02  2.51727632e+02  1.03764904e-07
  1.29322827e+00  1.55308327e-01  7.15429432e+03]
Model Intercept:
 261.1926335446415
```

[234]:
```python
print("Performance (R^2): ", model.score(X_train, y_train))
```

```
Performance (R^2):  0.9992609726796228
```

[235]:
```python
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 8.393052932903913
```

[236]:
```python
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
```

```
print(f'Mean Absolute Error: {mae:.2f}')
```

Mean Absolute Error: 2.20

[237]:
```
# Calculate R-squared
r2 = r2_score(y_test, predictions)
print(f'R-squared: {r2:.2f}')
```

R-squared: 1.00

[238]:
```
df_results = pd.DataFrame({'Date': pd.to_datetime(y_test.index,␣
 ↪format='%Y-%m-%d'),
                          'Actual_Close': y_test.values,
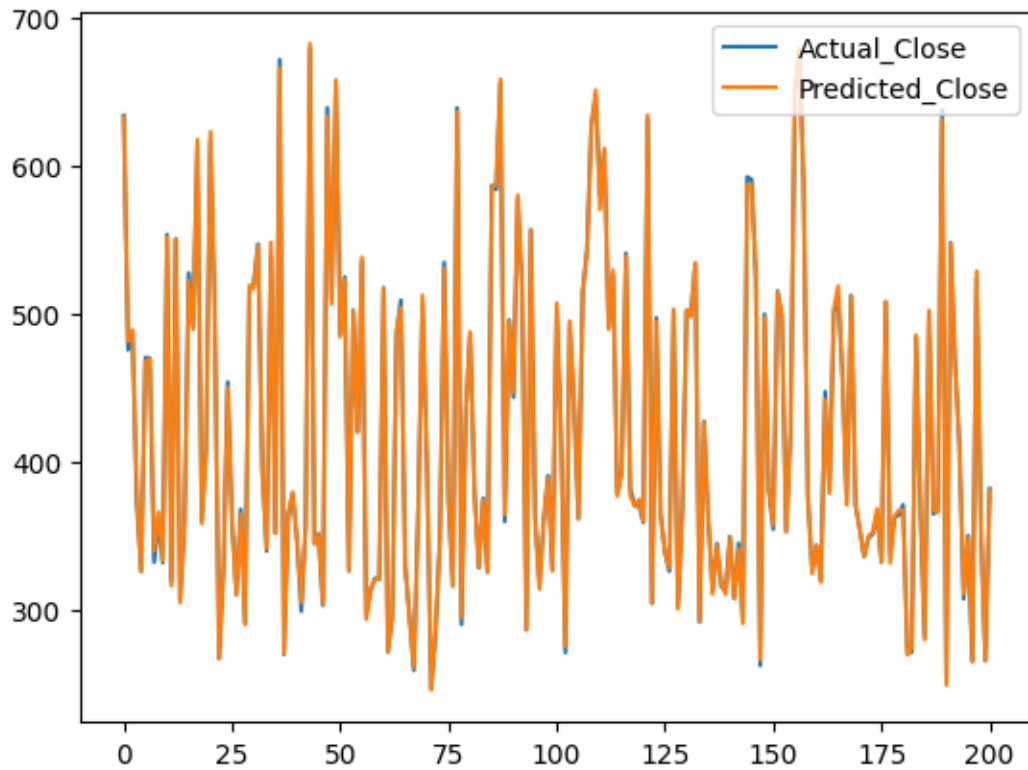                          'Predicted_Close': predictions})

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

```
                               Date   Actual_Close   Predicted_Close
0    1970-01-01 00:00:00.000000930    633.799988       632.612707
1    1970-01-01 00:00:00.000000634    475.470001       481.227010
2    1970-01-01 00:00:00.000000686    488.239990       488.962180
3    1970-01-01 00:00:00.000000518    371.709991       379.515191
4    1970-01-01 00:00:00.000000369    326.459991       325.430489
..                              …              …                …
196  1970-01-01 00:00:00.000000212    265.140015       264.831090
197  1970-01-01 00:00:00.000000782    524.030029       528.651367
198  1970-01-01 00:00:00.000000338    351.269989       352.537395
199  1970-01-01 00:00:00.000000214    265.320007       265.847473
200  1970-01-01 00:00:00.000000354    381.720001       380.789172

[201 rows x 3 columns]
```

[239]:
```
df_results[['Actual_Close', 'Predicted_Close']].plot()
```

[239]: <Axes: >

```
[240]: 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
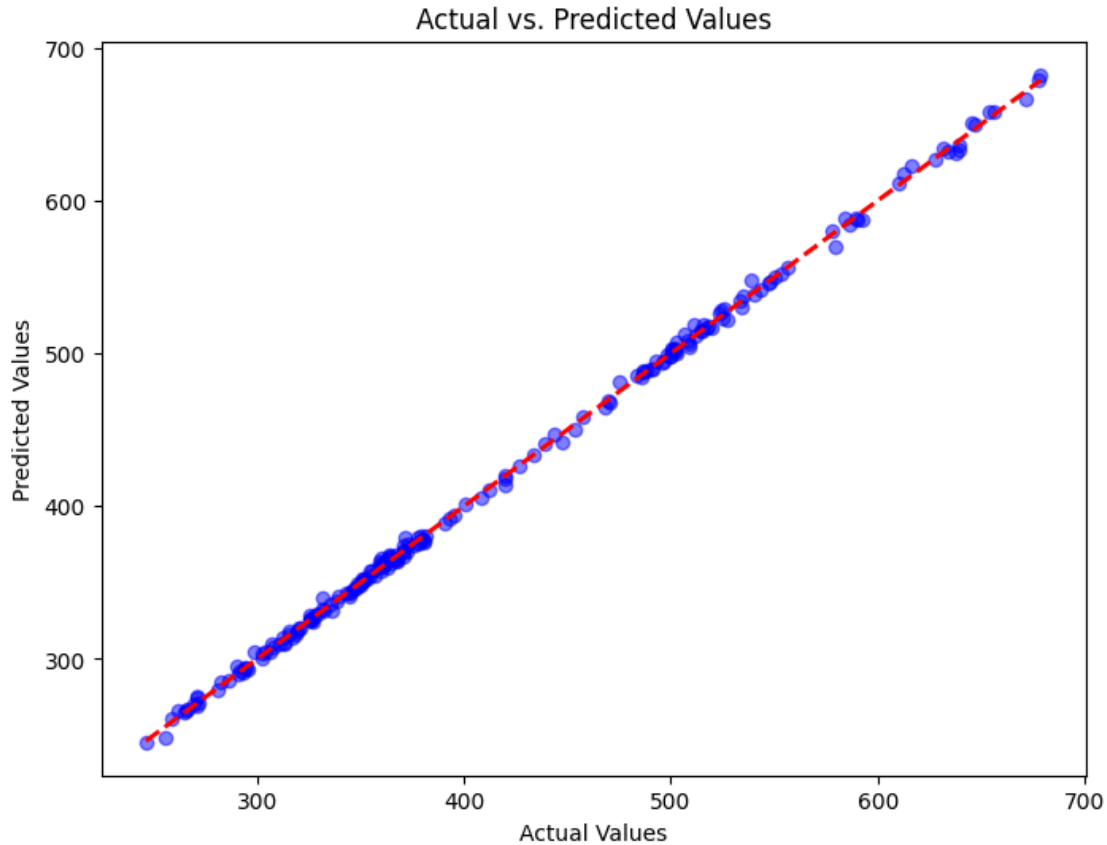
For Binance Dataset > In this project, I selected Binance dataset which play a crucial role as they form the foundation for training and evaluating machine learning models i.e Linear Regression for stock market price prediction. Dataset represents historical stock market data for different assets, and 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

```
[241]: 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
```

19

```
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:

[242]: `data.shape`

[242]: (366, 7)

[243]: `data.columns`

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

[244]: `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
```

[245]: `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:

[246]:
```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

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

```
[247]:            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

```
[248]: 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
```

```
[248]:          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

```
[249]: 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
```

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

```
[251]: X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
        ↪random_state=42)
```

```python
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,)
```

[252]:
```python
model = LinearRegression()
```

Model Evaluation

[253]:
```python
model.fit(X_train, y_train)
predictions = model.predict(X_test)
predictions
```

[253]:
```
array([305.45798154, 306.44999011, 261.38443134, 251.37371917,
       316.29684068, 245.30138127, 277.52421086, 322.04574845,
       215.15432438, 325.09661019, 217.20679535, 311.77399759,
       240.5078801 , 306.66873487, 312.92366454, 247.20745748,
       215.4010219 , 241.65905465, 242.55550747, 308.32330262,
       243.91878578, 246.71134523, 289.1035218 , 213.27036997,
       233.04487353, 238.66528041, 242.60737285, 260.52444922,
       310.65527831, 244.64075152, 299.99363272, 317.51200554,
       330.3236847 , 233.81694928, 304.80316657, 345.25594617,
       302.70689307, 304.81599577, 210.6496197 , 236.93539452,
       216.68900473, 227.94515144, 303.15804864, 209.64158587,
       322.81037735, 234.78068791, 328.70708937, 312.44642267,
       243.69627811, 325.04454434, 240.20043541, 311.87117871,
       215.44082462, 240.78321639, 239.38896467, 238.31794006,
       327.92837728, 244.36413165, 213.18134847, 276.40504854,
       233.4347253 , 214.49228236, 277.8077716 , 331.82971277,
       209.88546359, 287.83455155, 239.40419998, 331.86283221,
       216.24987892, 306.89129082, 312.9506727 , 227.04232899,
       218.5264309 ])
```

[254]:
```python
print('Model Coefficients: \n', model.coef_)
print('Model Intercept: \n', model.intercept_)
```

```
Model Coefficients:
 [7.87776562e-01 1.02282590e+01 1.53486627e+01 3.72532227e-10
 2.44372733e+00 2.60647753e-02 3.04111720e+05]
Model Intercept:
 37.899612517170965
```

[255]:
```python
print("Performance (R^2): ", model.score(X_train, y_train))
```

Performance (R^2):   0.9996484357297784

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

Mean Squared Error: 0.6873156526879616

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

Mean Absolute Error: 0.59

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

R-squared: 1.00

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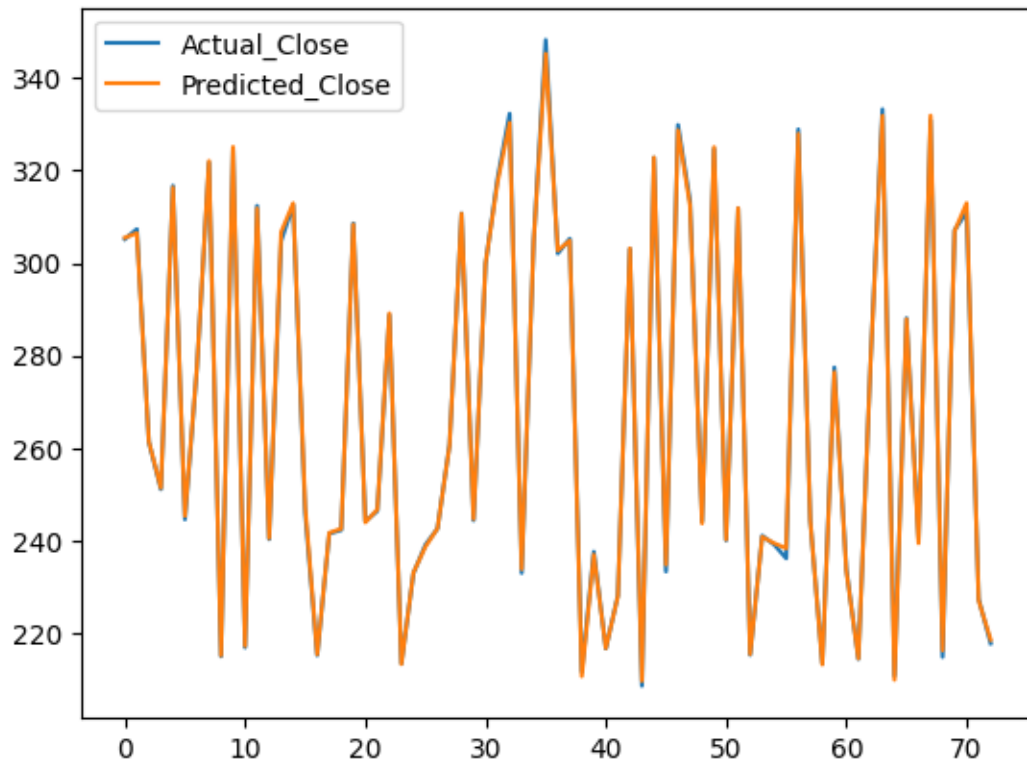                                  'Predicted_Close': predictions})

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

```
                              Date  Actual_Close  Predicted_Close
0   1970-01-01 00:00:00.000000167     305.156799       305.457982
1   1970-01-01 00:00:00.000000037     307.307648       306.449990
2   1970-01-01 00:00:00.000000019     261.282837       261.384431
3   1970-01-01 00:00:00.000000326     251.082367       251.373719
4   1970-01-01 00:00:00.000000061     316.682709       316.296841
..                            ...            ...              ...
68  1970-01-01 00:00:00.000000301     214.823959       216.249879
69  1970-01-01 00:00:00.000000086     307.124939       306.891291
70  1970-01-01 00:00:00.000000098     310.949127       312.950673
71  1970-01-01 00:00:00.000000345     227.342758       227.042329
72  1970-01-01 00:00:00.000000307     217.747375       218.526431

[73 rows x 3 columns]
```

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

```
[260]: <Axes: >
```

25

```
[261]: 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
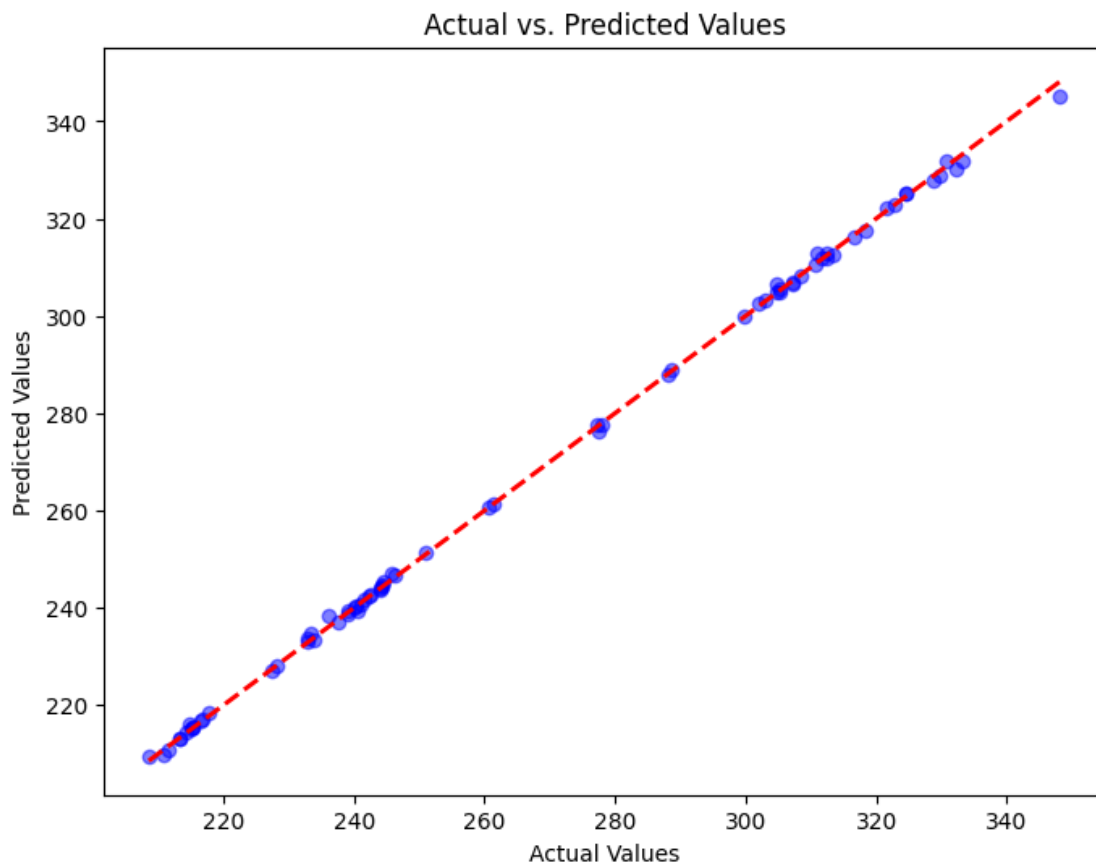
Bitcoin Dataset

In this project, I selected Bitcoin datasets which play a crucial role as they form the foundation for training and evaluating machine learning models i.e Linear Regression 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

```
[262]: 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:

[263]: `data.shape`

[263]: (366, 7)

[264]: `data.columns`

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

[265]: `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
```

[266]: `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:

[267]:
```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

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

```
[268]:            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

```
[269]:  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
```

[269]:

	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

[270]:
```
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
```

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

```
[272]: 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,)
```

```
[273]: y_test
```

```
[273]: 167     27119.066406
       37      23117.859375
       19      16955.078125
       326     37313.968750
       61      24641.277344
                  …
       301     28519.466797
       86      24375.960938
       98      27139.888672
       345     37858.492188
       307     29993.896484
       Name: Close, Length: 73, dtype: float64
```

```
[274]: model = LinearRegression()
```

Model Evaluation

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

```
[275]: array([27168.81618667, 23242.4045972 , 16908.9546495 , 37128.56257963,
              24596.45024889, 30711.40482712, 20108.81585305, 27374.01533076,
              26567.61872914, 29222.90583675, 26253.27876905, 23173.14769021,
              29445.71165637, 23636.88313986, 27679.60179647, 30508.930958  ,
              25792.62220786, 29082.32950352, 35695.22492157, 27741.80526729,
              16617.22036706, 31159.27937809, 22496.55255111, 27757.67959263,
              37815.38794398, 30511.461779  , 29367.25928414, 26507.64157129,
              27136.2214614 , 42299.07252993, 21125.74885449, 29563.43480724,
              23420.94314085, 43389.27107424, 26414.3372935 , 30351.94433431,
              23124.49729033, 21128.22967503, 28309.74710527, 25075.31181006,
```

```
            26232.54322186, 39841.63469332, 22616.67822997, 27431.64245117,
            30183.46184497, 41546.7047931 , 28392.50037347, 24619.45149591,
            16440.90970336, 22756.3971701 , 29315.37999311, 27731.45222389,
            26889.42477727, 29793.57394954, 26287.85268271, 37243.6598322 ,
            28024.97761452, 16904.74133938, 27397.83051304, 20038.08151563,
            37297.71279234, 26009.33650618, 20755.96990191, 23327.46007376,
            26522.56839146, 22434.37849572, 29218.54936174, 28506.85243214,
            28682.67349793, 24245.78178796, 27034.34622324, 37949.79879498,
            29961.94352127])
```

[276]:
```python
print('Model Coefficients: \n', model.coef_)
print('Model Intercept: \n', model.intercept_)
```

```
Model Coefficients:
 [ 4.33637933e-01  6.75946913e+03  7.03147763e+03 -7.20319804e-10
   1.95315431e+02  7.18247292e-02  3.96059121e-02]
Model Intercept:
 8132.503069674331
```

[277]:
```python
print("Performance (R^2): ", model.score(X_train, y_train))
```

```
Performance (R^2):  0.9996098832213572
```

[278]:
```python
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 9246.456983353508
```

[279]:
```python
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error: {mae:.2f}')
```

```
Mean Absolute Error: 68.14
```

[280]:
```python
# Calculate R-squared
r2 = r2_score(y_test, predictions)
print(f'R-squared: {r2:.2f}')
```

```
R-squared: 1.00
```

[281]:
```python
df_results = pd.DataFrame({'Date': pd.to_datetime(y_test.index,␣
 ↪format='%Y-%m-%d'),
                           'Actual_Close': y_test.values,
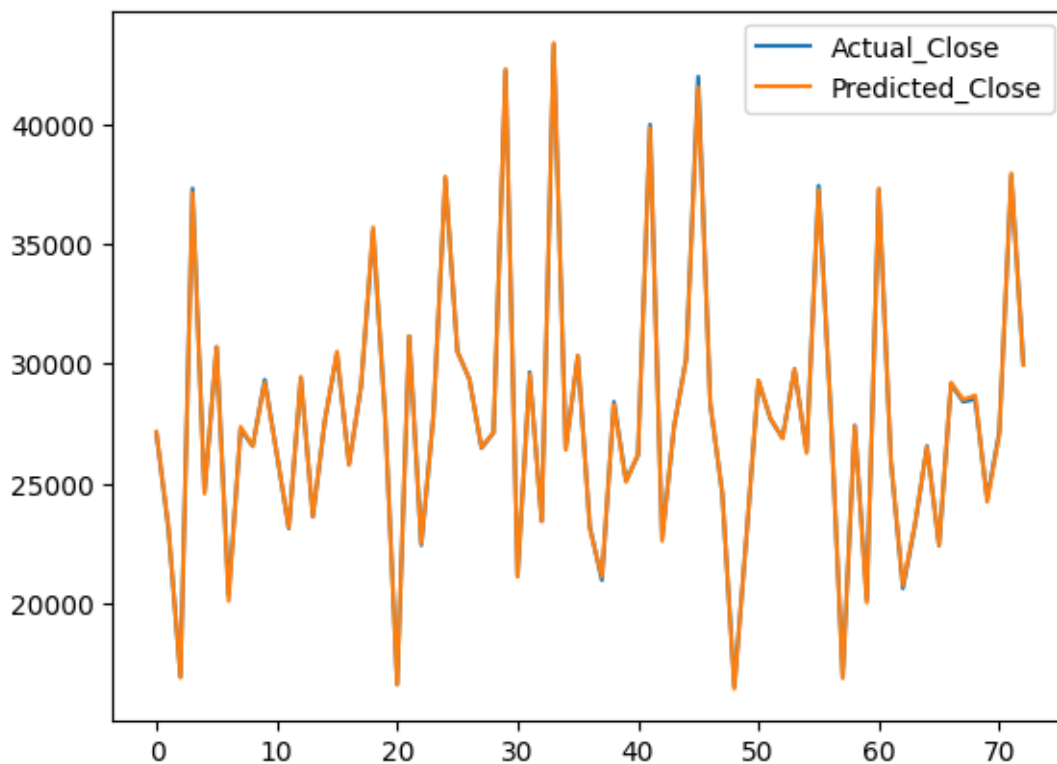                           'Predicted_Close': predictions})

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

```
                              Date   Actual_Close   Predicted_Close
0   1970-01-01 00:00:00.000000167   27119.066406      27168.816187
1   1970-01-01 00:00:00.000000037   23117.859375      23242.404597
2   1970-01-01 00:00:00.000000019   16955.078125      16908.954649
3   1970-01-01 00:00:00.000000326   37313.968750      37128.562580
4   1970-01-01 00:00:00.000000061   24641.277344      24596.450249
..                            ...            ...               ...
68  1970-01-01 00:00:00.000000301   28519.466797      28682.673498
69  1970-01-01 00:00:00.000000086   24375.960938      24245.781788
70  1970-01-01 00:00:00.000000098   27139.888672      27034.346223
71  1970-01-01 00:00:00.000000345   37858.492188      37949.798795
72  1970-01-01 00:00:00.000000307   29993.896484      29961.943521

[73 rows x 3 columns]
```

[282]:
```
df_results[['Actual_Close', 'Predicted_Close']].plot()
```

[282]: `<Axes: >`

[283]:
```
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