stock-prediction-using-svm

January 14, 2024

Stock Prediction using SVM algorithm

Importing important libraries

[63]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import time
from multiprocessing import Pool
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

Tesla Dataset

In this project, I select tesla datasets which play a crucial role as they form the foundation for training and evaluating machine learning models i.e SVM algorithm 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

[64]:
```
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:

[65]:
```
data.shape
```

[65]:
```
(253, 7)
```

[66]:
```
data.columns
```

[66]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

[67]:
```
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
```

[68]:
```
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:

[69]:
```
features = ['Open', 'High', 'Low', 'Volume']
X = data[features]
Y = data['Close']
```

[70]:
```
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
```

Feature Scaling

[71]:
```
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

Feature Engineering

[72]:
```
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 Splitting and Model Training

[73]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
↪random_state=42, shuffle=True)

def train_svr(X_train, y_train):
    svm_model = SVR(kernel='rbf', C=1.0, epsilon=0.1, cache_size=1000,␣
↪verbose=True)
    svm_model.fit(X_train, y_train)
    return svm_model
```

[74]:
```
start_time = time.time()
with Pool(processes=2) as pool:
    svm_models = pool.starmap(train_svr, [(X_train, y_train), (X_train,␣
↪y_train)])
```

```
[LibSVM][LibSVM]
```

[75]:
```
training_time = time.time() - start_time
```

[76]:
```
predictions = svm_models[0].predict(X_test)
mse = mean_squared_error(y_test, predictions)
```

[77]:
```
print(f"Training Time: {training_time} seconds")
print(f"Mean Squared Error: {mse}")
```

```
Training Time: 0.15067815780639648 seconds
Mean Squared Error: 758.6961191249621
```

[78]:
```
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: (202, 4)
X_test shape: (51, 4)
y_train shape: (202,)
y_test shape: (51,)
```

Model Evaluation

[79]:
```
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error (SVM): {mse}')
```

```
Mean Squared Error (SVM): 758.6961191249621
```

[80]:
```
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
```

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

```
Mean Absolute Error: 18.69
```

[81]:
```
#Calculate R-squared
r2 = r2_score(y_test, predictions)
print(f'R-squared: {r2:.2f}')
```

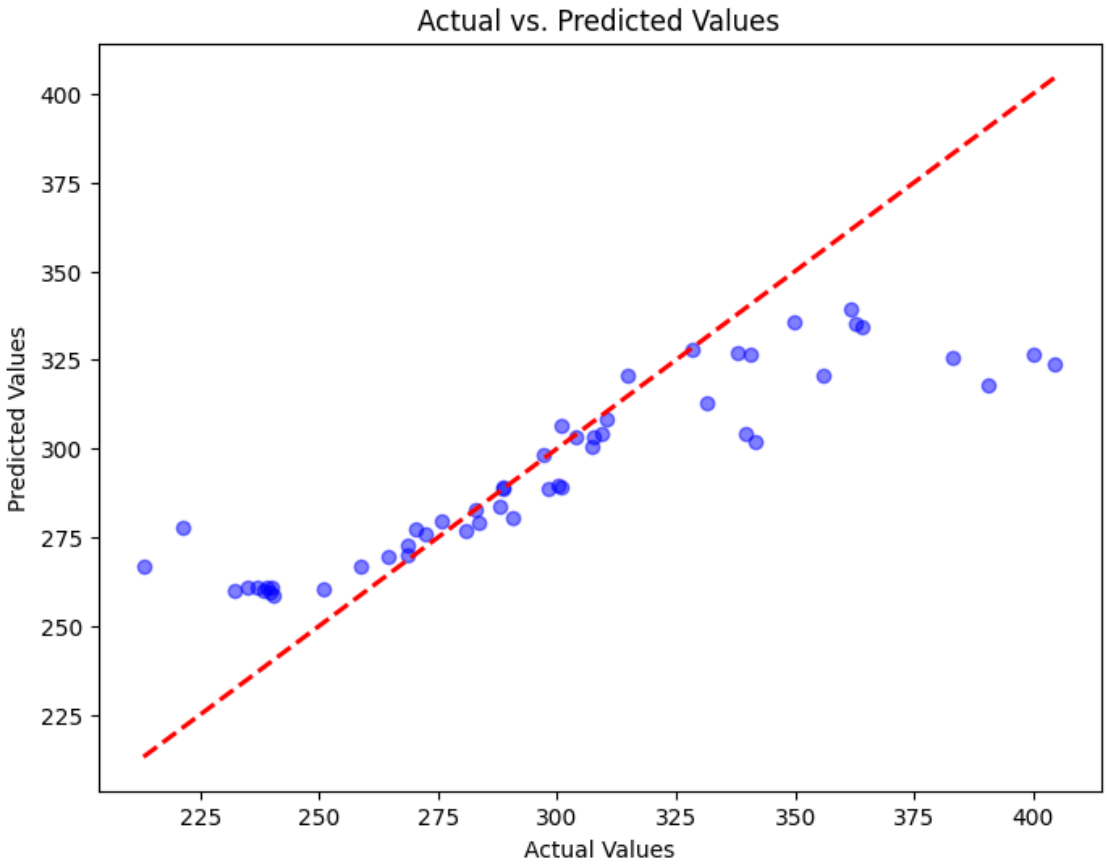
```
R-squared: 0.67
```

[82]:
```
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 00:00:00.000000208    280.899994       276.968878
1  1970-01-01 00:00:00.000000006    264.536682       269.628800
2  1970-01-01 00:00:00.000000079    314.633331       320.764456
3  1970-01-01 00:00:00.000000204    272.243347       276.064098
4  1970-01-01 00:00:00.000000117    290.533325       280.671102
5  1970-01-01 00:00:00.000000184    235.070007       260.784042
6  1970-01-01 00:00:00.000000200    240.546661       258.760436
7  1970-01-01 00:00:00.000000198    238.313339       259.806363
8  1970-01-01 00:00:00.000000009    268.573334       270.001718
9  1970-01-01 00:00:00.000000030    355.983337       320.408482
10 1970-01-01 00:00:00.000000251    287.809998       283.453148
11 1970-01-01 00:00:00.000000219    300.029999       289.756951
12 1970-01-01 00:00:00.000000227    297.096680       298.110346
13 1970-01-01 00:00:00.000000222    303.996674       303.210025
14 1970-01-01 00:00:00.000000136    340.790009       326.468016
15 1970-01-01 00:00:00.000000068    362.706665       335.177461
16 1970-01-01 00:00:00.000000197    237.039993       260.764006
17 1970-01-01 00:00:00.000000015    288.600006       289.107445
18 1970-01-01 00:00:00.000000096    307.476654       300.641992
19 1970-01-01 00:00:00.000000024    390.666656       317.925716
20 1970-01-01 00:00:00.000000232    275.609985       279.553877
21 1970-01-01 00:00:00.000000019    339.476654       303.955819
22 1970-01-01 00:00:00.000000120    331.326660       312.924591
23 1970-01-01 00:00:00.000000152    288.549988       288.778143
24 1970-01-01 00:00:00.000000033    337.796661       326.943155
25 1970-01-01 00:00:00.000000124    363.946655       334.358790
26 1970-01-01 00:00:00.000000250    282.940002       282.583703
27 1970-01-01 00:00:00.000000206    258.859985       266.954922
28 1970-01-01 00:00:00.000000010    270.359985       277.404612
```

```
29 1970-01-01 00:00:00.000000180    213.100006       266.770850
30 1970-01-01 00:00:00.000000173    238.886673       260.724093
31 1970-01-01 00:00:00.000000097    307.796661       303.085761
32 1970-01-01 00:00:00.000000148    300.980011       289.006776
33 1970-01-01 00:00:00.000000246    300.799988       306.283624
34 1970-01-01 00:00:00.000000220    309.320007       304.050477
35 1970-01-01 00:00:00.000000025    404.619995       323.609472
36 1970-01-01 00:00:00.000000086    310.416656       308.092633
37 1970-01-01 00:00:00.000000018    341.619995       301.879690
38 1970-01-01 00:00:00.000000075    349.869995       335.509276
39 1970-01-01 00:00:00.000000137    328.333344       327.961539
40 1970-01-01 00:00:00.000000194    250.763336       260.463192
41 1970-01-01 00:00:00.000000175    239.706665       259.685779
42 1970-01-01 00:00:00.000000236    283.700012       279.288371
43 1970-01-01 00:00:00.000000162    221.300003       277.588495
44 1970-01-01 00:00:00.000000045    361.533325       339.438307
45 1970-01-01 00:00:00.000000066    399.926666       326.724534
46 1970-01-01 00:00:00.000000016    298.000000       288.680514
47 1970-01-01 00:00:00.000000067    383.196655       325.638763
48 1970-01-01 00:00:00.000000205    268.433319       272.568208
49 1970-01-01 00:00:00.000000199    240.066666       260.782796
50 1970-01-01 00:00:00.000000176    232.229996       259.743927
```

[83]:
```
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()
```



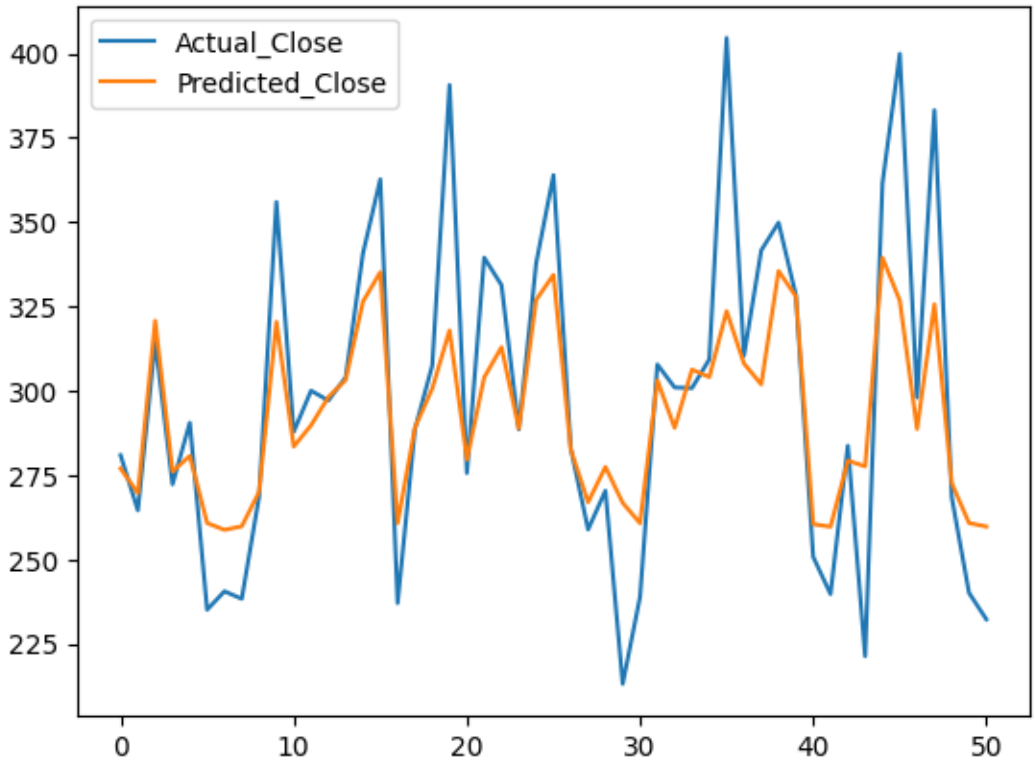
[84]:
```
df_results = pd.DataFrame({'Date': pd.to_datetime(y_test.index,␣
 ↪format='%Y-%m-%d'),
                           'Actual_Close': y_test.values,
                           'Predicted_Close': predictions})

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

```
                          Date  Actual_Close  Predicted_Close
0  1970-01-01 00:00:00.000000208    280.899994       276.968878
1  1970-01-01 00:00:00.000000006    264.536682       269.628800
2  1970-01-01 00:00:00.000000079    314.633331       320.764456
3  1970-01-01 00:00:00.000000204    272.243347       276.064098
4  1970-01-01 00:00:00.000000117    290.533325       280.671102
5  1970-01-01 00:00:00.000000184    235.070007       260.784042
6  1970-01-01 00:00:00.000000200    240.546661       258.760436
7  1970-01-01 00:00:00.000000198    238.313339       259.806363
8  1970-01-01 00:00:00.000000009    268.573334       270.001718
9  1970-01-01 00:00:00.000000030    355.983337       320.408482
10 1970-01-01 00:00:00.000000251    287.809998       283.453148
```

11	1970-01-01 00:00:00.000000219	300.029999	289.756951
12	1970-01-01 00:00:00.000000227	297.096680	298.110346
13	1970-01-01 00:00:00.000000222	303.996674	303.210025
14	1970-01-01 00:00:00.000000136	340.790009	326.468016
15	1970-01-01 00:00:00.000000068	362.706665	335.177461
16	1970-01-01 00:00:00.000000197	237.039993	260.764006
17	1970-01-01 00:00:00.000000015	288.600006	289.107445
18	1970-01-01 00:00:00.000000096	307.476654	300.641992
19	1970-01-01 00:00:00.000000024	390.666656	317.925716
20	1970-01-01 00:00:00.000000232	275.609985	279.553877
21	1970-01-01 00:00:00.000000019	339.476654	303.955819
22	1970-01-01 00:00:00.000000120	331.326660	312.924591
23	1970-01-01 00:00:00.000000152	288.549988	288.778143
24	1970-01-01 00:00:00.000000033	337.796661	326.943155
25	1970-01-01 00:00:00.000000124	363.946655	334.358790
26	1970-01-01 00:00:00.000000250	282.940002	282.583703
27	1970-01-01 00:00:00.000000206	258.859985	266.954922
28	1970-01-01 00:00:00.000000010	270.359985	277.404612
29	1970-01-01 00:00:00.000000180	213.100006	266.770850
30	1970-01-01 00:00:00.000000173	238.886673	260.724093
31	1970-01-01 00:00:00.000000097	307.796661	303.085761
32	1970-01-01 00:00:00.000000148	300.980011	289.006776
33	1970-01-01 00:00:00.000000246	300.799988	306.283624
34	1970-01-01 00:00:00.000000220	309.320007	304.050477
35	1970-01-01 00:00:00.000000025	404.619995	323.609472
36	1970-01-01 00:00:00.000000086	310.416656	308.092633
37	1970-01-01 00:00:00.000000018	341.619995	301.879690
38	1970-01-01 00:00:00.000000075	349.869995	335.509276
39	1970-01-01 00:00:00.000000137	328.333344	327.961539
40	1970-01-01 00:00:00.000000194	250.763336	260.463192
41	1970-01-01 00:00:00.000000175	239.706665	259.685779
42	1970-01-01 00:00:00.000000236	283.700012	279.288371
43	1970-01-01 00:00:00.000000162	221.300003	277.588495
44	1970-01-01 00:00:00.000000045	361.533325	339.438307
45	1970-01-01 00:00:00.000000066	399.926666	326.724534
46	1970-01-01 00:00:00.000000016	298.000000	288.680514
47	1970-01-01 00:00:00.000000067	383.196655	325.638763
48	1970-01-01 00:00:00.000000205	268.433319	272.568208
49	1970-01-01 00:00:00.000000199	240.066666	260.782796
50	1970-01-01 00:00:00.000000176	232.229996	259.743927

[85]:
```
df_results[['Actual_Close', 'Predicted_Close']].plot()
```

[85]: <Axes: >



Bitcoin Dataset

In this project, I select bitcoins datasets which play a crucial role as they form the foundation for training and evaluating machine learning models i.e SVM algorithm 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

[86]:
```
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:

[87]:
```
data.shape
```

[87]:
```
(366, 7)
```

[88]:
```
data.columns
```

[88]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

[89]:
```
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
```

[90]:
```
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:

[91]:
```
features = ['Open', 'High', 'Low', 'Volume']
X = data[features]
Y = data['Close']
```

[92]:
```
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
```

Feature Scaling

[93]:
```
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

Feature Engineering

[94]:
```
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 Splitting and Model Training

[95]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
↪random_state=42, shuffle=True)

def train_svr(X_train, y_train):
    svm_model = SVR(kernel='rbf', C=1.0, epsilon=0.1, cache_size=1000,␣
↪verbose=True)
    svm_model.fit(X_train, y_train)
    return svm_model
```

[96]:
```
start_time = time.time()
with Pool(processes=2) as pool:
    svm_models = pool.starmap(train_svr, [(X_train, y_train), (X_train,␣
↪y_train)])
```

```
[LibSVM][LibSVM]
```

[97]:
```
training_time = time.time() - start_time
```

[98]:
```
predictions = svm_models[0].predict(X_test)
mse = mean_squared_error(y_test, predictions)
```

[99]:
```
print(f"Training Time: {training_time} seconds")
print(f"Mean Squared Error: {mse}")
```

```
Training Time: 0.1360001564025879 seconds
Mean Squared Error: 44671142.32717951
```

[100]:
```
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: (292, 4)
X_test shape: (74, 4)
y_train shape: (292,)
y_test shape: (74,)
```

Model Evaluation

[101]:
```
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error (SVM): {mse}')
```

```
Mean Squared Error (SVM): 44671142.32717951
```

[102]:
```
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error: {mae:.2f}')
```

```
Mean Absolute Error: 4881.29
```

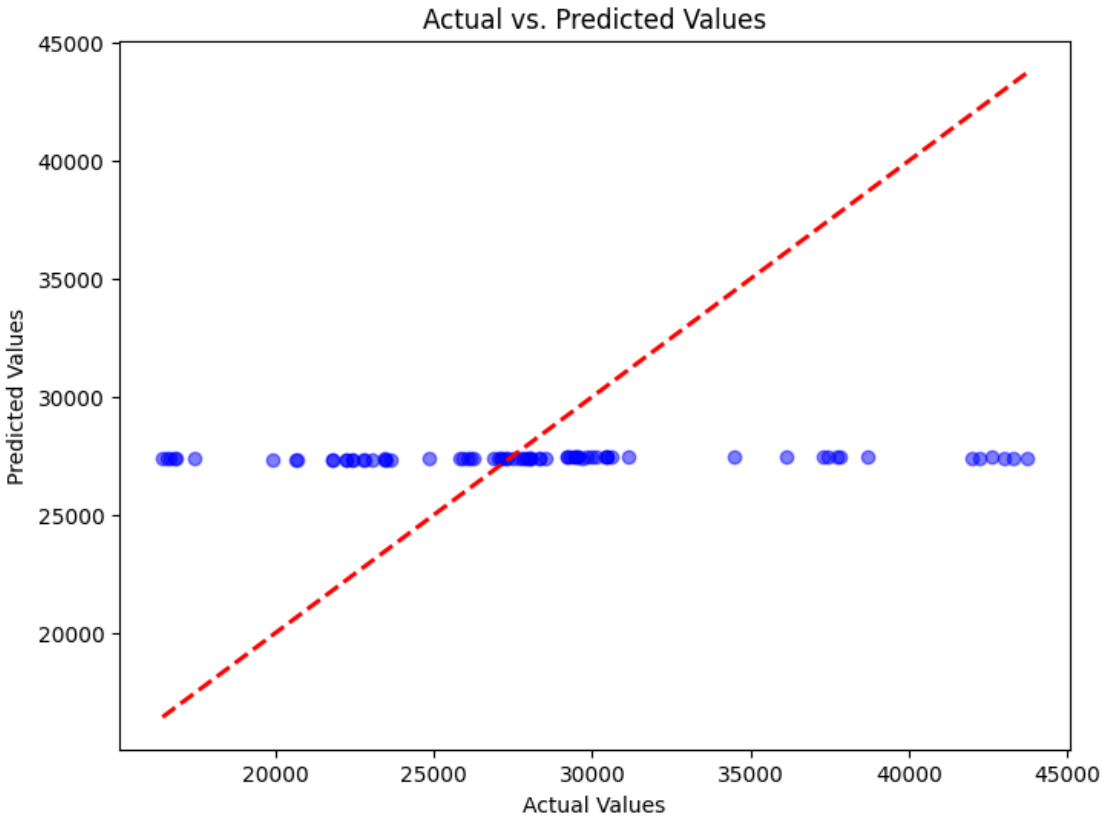
[103]:
```
# Calculate R-squared
r2 = r2_score(y_test, predictions)
print(f'R-squared: {r2:.2f}')
```

```
R-squared: -0.00
```

[104]:
```
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 00:00:00.000000193  30477.251953     27442.844825
1  1970-01-01 00:00:00.000000033  22777.625000     27368.300436
2  1970-01-01 00:00:00.000000015  16679.857422     27376.036081
3  1970-01-01 00:00:00.000000310  34502.820313     27460.821279
4  1970-01-01 00:00:00.000000057  22220.804688     27357.961106
..                           …           …              …
69 1970-01-01 00:00:00.000000082  20632.410156     27364.959171
70 1970-01-01 00:00:00.000000094  28333.972656     27410.685092
71 1970-01-01 00:00:00.000000192  30445.351563     27449.598969
72 1970-01-01 00:00:00.000000307  29993.896484     27443.419774
73 1970-01-01 00:00:00.000000211  29856.562500     27445.610153

[74 rows x 3 columns]
```

[105]:
```
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()
```



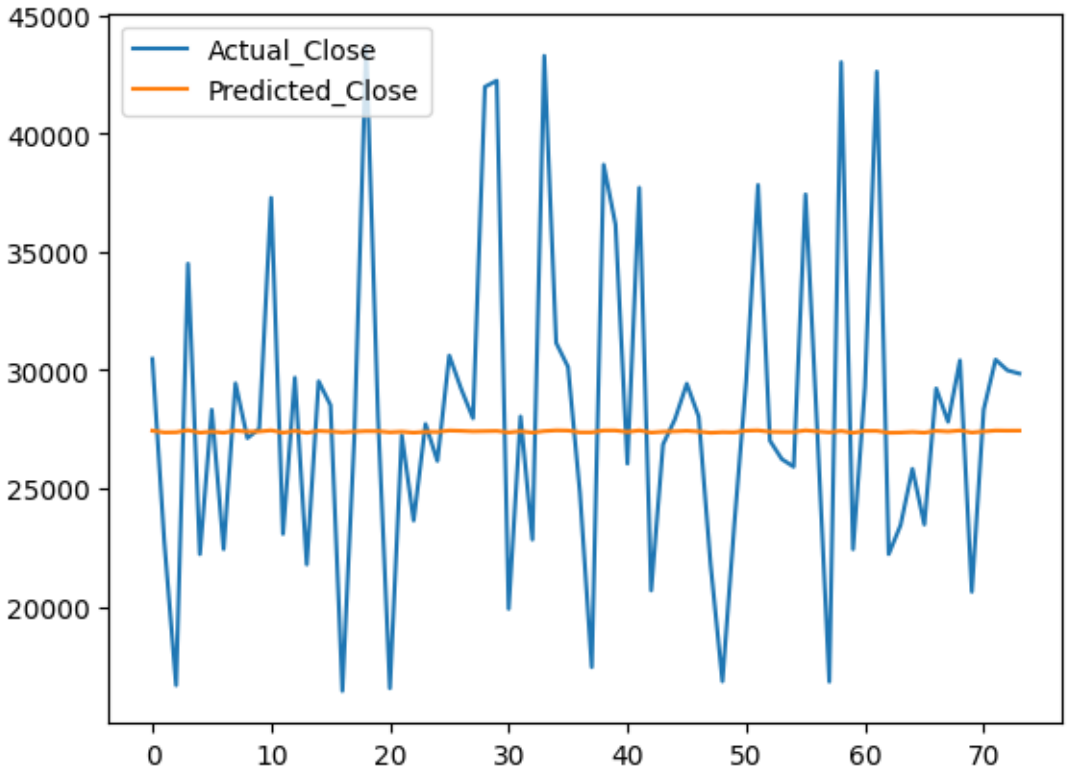
[106]:
```
df_results = pd.DataFrame({'Date': pd.to_datetime(y_test.index,␣
 ↪format='%Y-%m-%d'),
                           'Actual_Close': y_test.values,
                           'Predicted_Close': predictions})

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

```
                          Date  Actual_Close  Predicted_Close
0  1970-01-01 00:00:00.000000193  30477.251953     27442.844825
1  1970-01-01 00:00:00.000000033  22777.625000     27368.300436
2  1970-01-01 00:00:00.000000015  16679.857422     27376.036081
3  1970-01-01 00:00:00.000000310  34502.820313     27460.821279
4  1970-01-01 00:00:00.000000057  22220.804688     27357.961106
..                           …           …              …
69 1970-01-01 00:00:00.000000082  20632.410156     27364.959171
70 1970-01-01 00:00:00.000000094  28333.972656     27410.685092
71 1970-01-01 00:00:00.000000192  30445.351563     27449.598969
72 1970-01-01 00:00:00.000000307  29993.896484     27443.419774
73 1970-01-01 00:00:00.000000211  29856.562500     27445.610153
```

```
[74 rows x 3 columns]
```

[107]:
```
df_results[['Actual_Close', 'Predicted_Close']].plot()
```

[107]: <Axes: >



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 SVM algorithm 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

[108]:
```
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:

[109]:
```
data.shape
```

[109]:
```
(366, 7)
```

[110]:
```
data.columns
```

[110]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

[111]:
```
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
```

[112]:
```
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:

[113]:
```
features = ['Open', 'High', 'Low', 'Volume']
X = data[features]
Y = data['Close']
```

[114]:
```
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
```

Feature Scaling

[115]:
```
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

Feature Engineering

[116]:
```
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 Splitting and Model Training

[117]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=42, shuffle=True)

def train_svr(X_train, y_train):
    svm_model = SVR(kernel='rbf', C=1.0, epsilon=0.1, cache_size=1000,␣
 ↪verbose=True)
    svm_model.fit(X_train, y_train)
    return svm_model
```

[118]:
```
start_time = time.time()
with Pool(processes=2) as pool:
    svm_models = pool.starmap(train_svr, [(X_train, y_train), (X_train,␣
 ↪y_train)])
```

```
[LibSVM][LibSVM]
```

[119]:
```
training_time = time.time() - start_time
```

[120]:
```
predictions = svm_models[0].predict(X_test)
mse = mean_squared_error(y_test, predictions)
```

[121]:
```
print(f"Training Time: {training_time} seconds")
print(f"Mean Squared Error: {mse}")
```

```
Training Time: 0.11734175682067871 seconds
Mean Squared Error: 75.97710964120385
```

[122]:
```
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: (292, 4)
X_test shape: (74, 4)
y_train shape: (292,)
y_test shape: (74,)
```

Model Evaluation

[123]:
```
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error (SVM): {mse}')
```

```
Mean Squared Error (SVM): 75.97710964120385
```

[124]:
```
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error: {mae:.2f}')
```

```
Mean Absolute Error: 5.57
```

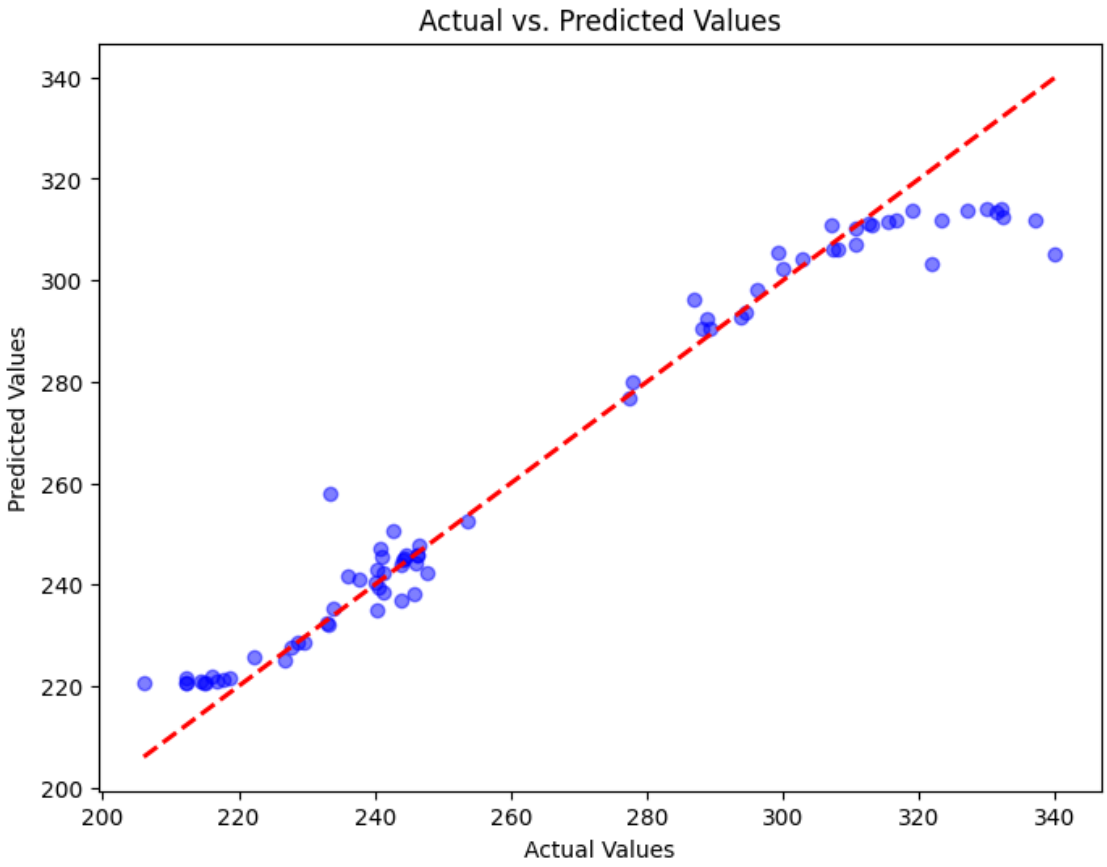
[125]:
```
# Calculate R-squared
r2 = r2_score(y_test, predictions)
print(f'R-squared: {r2:.2f}')
```

```
R-squared: 0.95
```

[126]:
```
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 00:00:00.000000193    240.369781       234.874175
1  1970-01-01 00:00:00.000000033    299.261169       305.567563
2  1970-01-01 00:00:00.000000015    246.133362       245.668141
3  1970-01-01 00:00:00.000000310    222.179932       225.682246
4  1970-01-01 00:00:00.000000057    296.120361       298.203134
..                           …           …              …
69 1970-01-01 00:00:00.000000082    277.961426       279.913027
70 1970-01-01 00:00:00.000000094    329.837006       313.967795
71 1970-01-01 00:00:00.000000192    233.232452       232.083448
72 1970-01-01 00:00:00.000000307    217.747375       221.375590
73 1970-01-01 00:00:00.000000211    240.264938       242.893292

[74 rows x 3 columns]
```

[127]:
```
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()
```



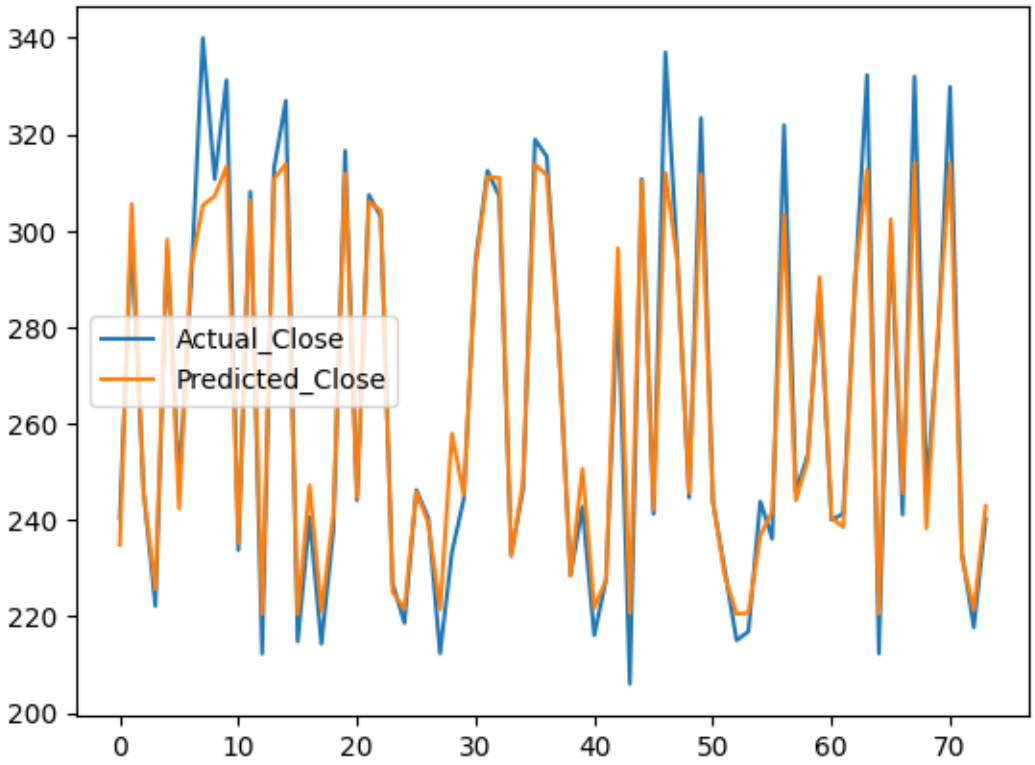
[128]:
```
df_results = pd.DataFrame({'Date': pd.to_datetime(y_test.index,␣
 ↪format='%Y-%m-%d'),
                           'Actual_Close': y_test.values,
                           'Predicted_Close': predictions})

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

```
                          Date  Actual_Close  Predicted_Close
0  1970-01-01 00:00:00.000000193    240.369781       234.874175
1  1970-01-01 00:00:00.000000033    299.261169       305.567563
2  1970-01-01 00:00:00.000000015    246.133362       245.668141
3  1970-01-01 00:00:00.000000310    222.179932       225.682246
4  1970-01-01 00:00:00.000000057    296.120361       298.203134
..                           …           …              …
69 1970-01-01 00:00:00.000000082    277.961426       279.913027
70 1970-01-01 00:00:00.000000094    329.837006       313.967795
71 1970-01-01 00:00:00.000000192    233.232452       232.083448
72 1970-01-01 00:00:00.000000307    217.747375       221.375590
73 1970-01-01 00:00:00.000000211    240.264938       242.893292
```

```
[74 rows x 3 columns]
```

[129]:
```
df_results[['Actual_Close', 'Predicted_Close']].plot()
```

[129]: <Axes: >



Netflix Dataset

In this project, I select netflix datasets which play a crucial role as they form the foundation for training and evaluating machine learning models i.e SVM algorithm 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

[130]:
```
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:

[131]:
```
data.shape
```

[131]:
```
(1009, 7)
```

[132]:
```
data.columns
```

[132]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

[133]:
```
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
```

[134]:
```
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:

[135]:
```
features = ['Open', 'High', 'Low', 'Volume']
X = data[features]
Y = data['Close']
```

[136]:
```
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
```

Feature Scaling

[137]:
```
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

Feature Engineering

[138]:
```
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 Splitting and Model Training

[139]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=42, shuffle=True)

def train_svr(X_train, y_train):
    svm_model = SVR(kernel='rbf', C=1.0, epsilon=0.1, cache_size=1000,␣
 ↪verbose=True)
    svm_model.fit(X_train, y_train)
    return svm_model
```

[140]:
```
start_time = time.time()
with Pool(processes=2) as pool:
    svm_models = pool.starmap(train_svr, [(X_train, y_train), (X_train,␣
 ↪y_train)])
```

```
[LibSVM][LibSVM]
```

[141]:
```
training_time = time.time() - start_time
```

[142]:
```
predictions = svm_models[0].predict(X_test)
mse = mean_squared_error(y_test, predictions)
```

[143]:
```
print(f"Training Time: {training_time} seconds")
print(f"Mean Squared Error: {mse}")
```

```
Training Time: 0.15522027015686035 seconds
Mean Squared Error: 1288.8355653921178
```

[144]:
```
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: (807, 4)
X_test shape: (202, 4)
y_train shape: (807,)
y_test shape: (202,)
```

Model Evaluation

[145]:
```
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error (SVM): {mse}')
```

```
Mean Squared Error (SVM): 1288.8355653921178
```

[146]:
```
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error: {mae:.2f}')
```

```
Mean Absolute Error: 16.89
```

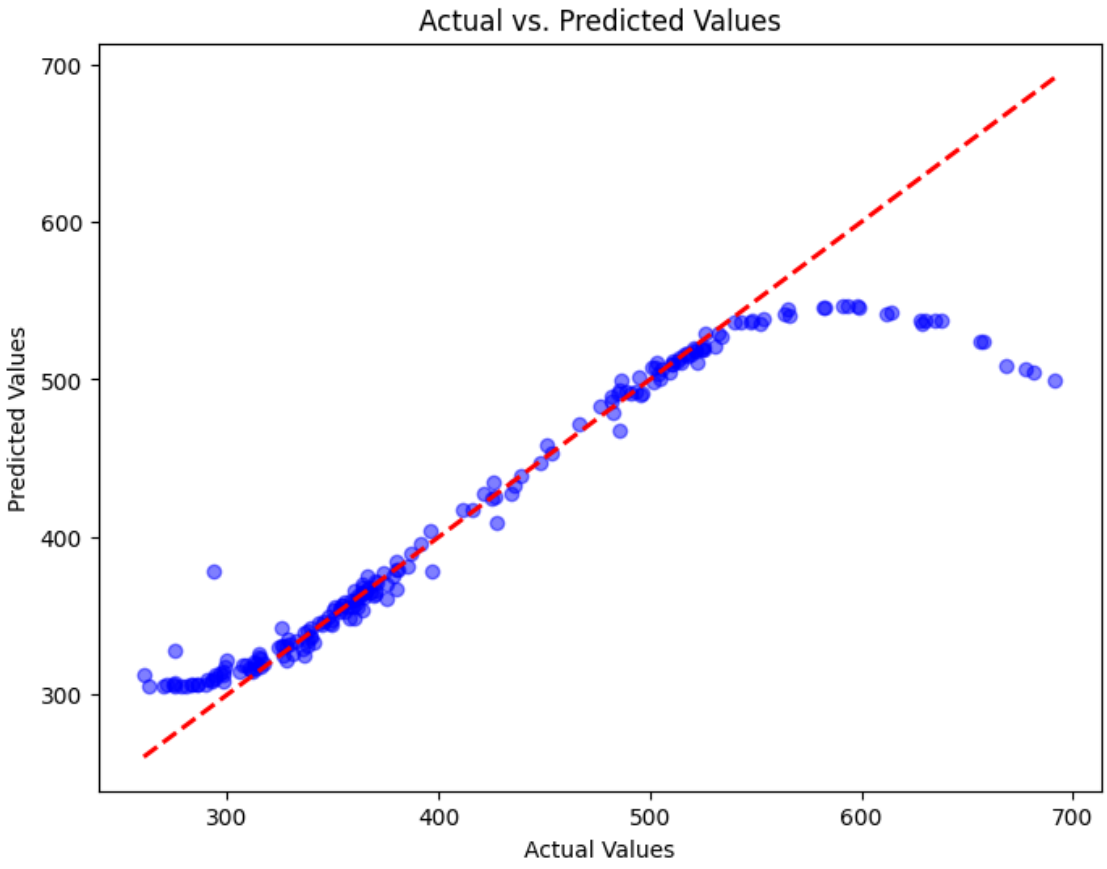
[147]:
```
# Calculate R-squared
r2 = r2_score(y_test, predictions)
print(f'R-squared: {r2:.2f}')
```

```
R-squared: 0.89
```

[148]:
```
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 00:00:00.000000628    509.640015       504.259031
1   1970-01-01 00:00:00.000000631    494.730011       501.734465
2   1970-01-01 00:00:00.000000741    500.859985       508.107198
3   1970-01-01 00:00:00.000000514    380.070007       384.082229
4   1970-01-01 00:00:00.000000365    315.100006       326.071684
..                            …           …              …
197 1970-01-01 00:00:00.000000780    518.020020       514.749704
198 1970-01-01 00:00:00.000000334    355.730011       352.663698
199 1970-01-01 00:00:00.000000210    275.329987       307.043317
200 1970-01-01 00:00:00.000000350    370.019989       365.658609
201 1970-01-01 00:00:00.000000078    349.730011       347.910883

[202 rows x 3 columns]
```

[149]:
```
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()
```



[150]:
```
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 00:00:00.000000628    509.640015       504.259031
1   1970-01-01 00:00:00.000000631    494.730011       501.734465
2   1970-01-01 00:00:00.000000741    500.859985       508.107198
3   1970-01-01 00:00:00.000000514    380.070007       384.082229
4   1970-01-01 00:00:00.000000365    315.100006       326.071684
..                            …           …              …
197 1970-01-01 00:00:00.000000780    518.020020       514.749704
198 1970-01-01 00:00:00.000000334    355.730011       352.663698
199 1970-01-01 00:00:00.000000210    275.329987       307.043317
200 1970-01-01 00:00:00.000000350    370.019989       365.658609
201 1970-01-01 00:00:00.000000078    349.730011       347.910883
```

```
[202 rows x 3 columns]
```

[151]:
```
df_results[['Actual_Close', 'Predicted_Close']].plot()
```

[151]: <Axes: >