

▸ Datasets

[]↳ 9 cells hidden

▸ Combining both datasets

[]↳ 3 cells hidden

▾ Classification on Price Prediction based on sentiment

```
1 import pandas as pd
2 import numpy as np
3
4 # Define custom bin edges based on quantiles
5 bin_edges = np.linspace(df_score['compound'].min(), df_score['compound'].max(), num=6) # Adjust the 'num' parameter as needed
6
7 # Define labels
8 labels = ['Extreme Negative', 'Negative', 'Neutral', 'Positive', 'Extreme Positive']
9
10 # Calculate average compound score for each sentiment level
11 sentiment_scores = []
12 for i in range(len(bin_edges)-1):
13     lower_bound = bin_edges[i]
14     upper_bound = bin_edges[i+1]
15     scores_in_range = df_score[(df_score['compound'] >= lower_bound) & (df_score['compound'] < upper_bound)]['compound']
16     sentiment_scores.append(scores_in_range.mean())
17
18 # Map sentiment levels to numerical values with scores
19 sentiment_mapping = {label: score for label, score in zip(labels, sentiment_scores)}
20 df_score['sentiment_score'] = df_score['sentiment_level'].map(sentiment_mapping)
21
22 # Save the updated dataframe as a new CSV file
23 df_score.to_csv('updated_sentiment_data.csv', index=False)
24
```

```
1 # Create a copy of the bitcoin tweets DataFrame
2 df_tweets = pd.read_csv('/content/updated_sentiment_data.csv')
3 df_tweets.head(2)
```

	user_name	user_location	user_description	user_created	user_followers	user_friends	u:
0	Irk	Vancouver, WA	Irk started investing in the stock market in 1...	2018-08-11 03:17:00	116.0	8.0	
1	Xiang Zhang	NaN	Professional Software Engineer ðŸ’Ž»ðŸ’ŽCrypto ...	2011-01-11 01:37:00	42.0	22.0	



```
1 # Merge the tweet data with the Bitcoin price data
2 tweets_df = pd.merge(df_tweets, crypto_usd, left_on='date', right_on='time', how='inner')

1 print(tweets_df.columns)
2
```

```
Index(['user_name', 'user_location', 'user_description', 'user_created',
      'user_followers', 'user_friends', 'user_favourites', 'user_verified',
      'date', 'text', 'hashtags', 'source', 'is_retweet', 'compound', 'score',
```

```
'sentiment_level', 'polarity', 'subjectivity', 'sentiment_score',
'time', 'close', 'high', 'low', 'open', 'volumefrom', 'volumeto',
'Date', 'Time', 'volume', 'marketcap', 'price_delta'],
dtype='object')
```

```
1 tweets_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7898 entries, 0 to 7897
Data columns (total 31 columns):
#   Column                Non-Null Count  Dtype
---  -
0   user_name              7898 non-null   object
1   user_location          3898 non-null   object
2   user_description       7620 non-null   object
3   user_created           7898 non-null   object
4   user_followers         7898 non-null   float64
5   user_friends           7898 non-null   float64
6   user_favourites        7898 non-null   float64
7   user_verified          7898 non-null   bool
8   date                   7898 non-null   object
9   text                   7898 non-null   object
10  hashtags               7891 non-null   object
11  source                 7891 non-null   object
12  is_retweet             7891 non-null   float64
13  compound               7898 non-null   float64
14  score                  7898 non-null   float64
15  sentiment_level        7898 non-null   object
16  polarity               7898 non-null   float64
17  subjectivity           7898 non-null   float64
18  sentiment_score        7898 non-null   float64
19  time                   7898 non-null   object
20  close                  7898 non-null   float64
21  high                   7898 non-null   float64
22  low                    7898 non-null   float64
23  open                   7898 non-null   float64
24  volumefrom             7898 non-null   float64
25  volumeto               7898 non-null   float64
26  Date                   7898 non-null   object
27  Time                   7898 non-null   object
28  volume                 7898 non-null   float64
29  marketcap              7898 non-null   float64
30  price_delta            7898 non-null   float64
dtypes: bool(1), float64(18), object(12)
memory usage: 1.9+ MB
```

```
1 import pandas as pd
2 from sklearn.feature_extraction.text import CountVectorizer
3 from sklearn.model_selection import train_test_split
4 from sklearn.linear_model import LinearRegression
5 from sklearn.metrics import mean_squared_error, classification_report
6 from scipy.sparse import hstack
7
8 # Feature Extraction: Unigrams
9 unigram_vectorizer = CountVectorizer(ngram_range=(1, 1))
10 unigram_features = unigram_vectorizer.fit_transform(tweets_df['text'])
11
12 # Feature Extraction: Bigrams
13 bigram_vectorizer = CountVectorizer(ngram_range=(2, 2))
14 bigram_features = bigram_vectorizer.fit_transform(tweets_df['text'])
15
16 # Combining Features
17 combined_features = hstack([unigram_features, bigram_features])
18
19 # Additional Input Features
20 additional_features = tweets_df[['compound', 'score', 'polarity', 'subjectivity', 'sentiment_score']].values
21
22 # Concatenate Additional Features with Combined Features
23 X = hstack([combined_features, additional_features])
24
25 # Target Variable
26 y = tweets_df['close']
27
28 # Split the data into training and testing sets
29 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
30
31
32
```

```

1 import numpy as np
2
3 # Print the first 10 rows of the term frequency matrix
4 print(combined_features[:10].toarray())
5
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]

1 from sklearn.metrics import mean_squared_error, accuracy_score, precision_score, recall_score, f1_score
2 import numpy as np
3 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
4 from scipy.sparse import hstack
5

```

▼ Linear Regression

```

1 from sklearn.linear_model import LinearRegression
2 from sklearn.metrics import mean_squared_error, accuracy_score, precision_score, recall_score, f1_score
3
4 # Train the linear regression model
5 model = LinearRegression()
6
1
2 model.fit(X_train, y_train)
3
4 # Make predictions on the test set
5 y_pred = model.predict(X_test)
6
7 # Evaluate the model
8 mse = mean_squared_error(y_test, y_pred)
9 r2 = r2_score(y_test, y_pred)
10 mae = mean_absolute_error(y_test, y_pred)
11 rmse = np.sqrt(mse)
12
13 print("Mean Squared Error:", mse)
14 print("R-squared:", r2)
15 print("Mean Absolute Error:", mae)
16 print("Root Mean Squared Error:", rmse)
17
18 # Use the trained model for future predictions
19 new_tweet = ["New tweet about Bitcoin"]
20 new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new_tweet)]])
21 predicted_close = model.predict(new_tweet_features)
22
23 print("Predicted Close Price:", predicted_close)
24

Mean Squared Error: 263642.15720830334
R-squared: -0.00027683995676786033
Mean Absolute Error: 434.52422429092474
Root Mean Squared Error: 513.460959770364
Predicted Close Price: [23131.47449878]

```

```

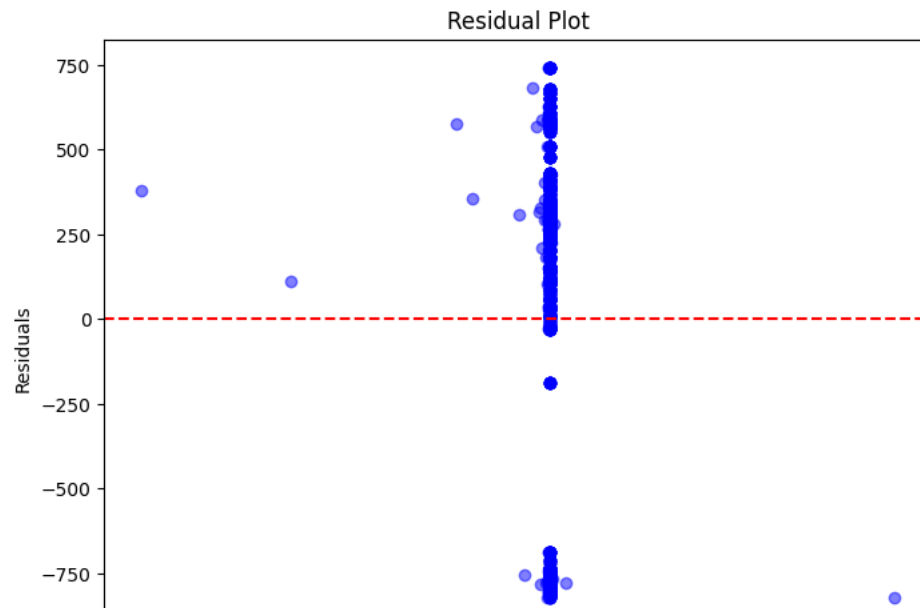
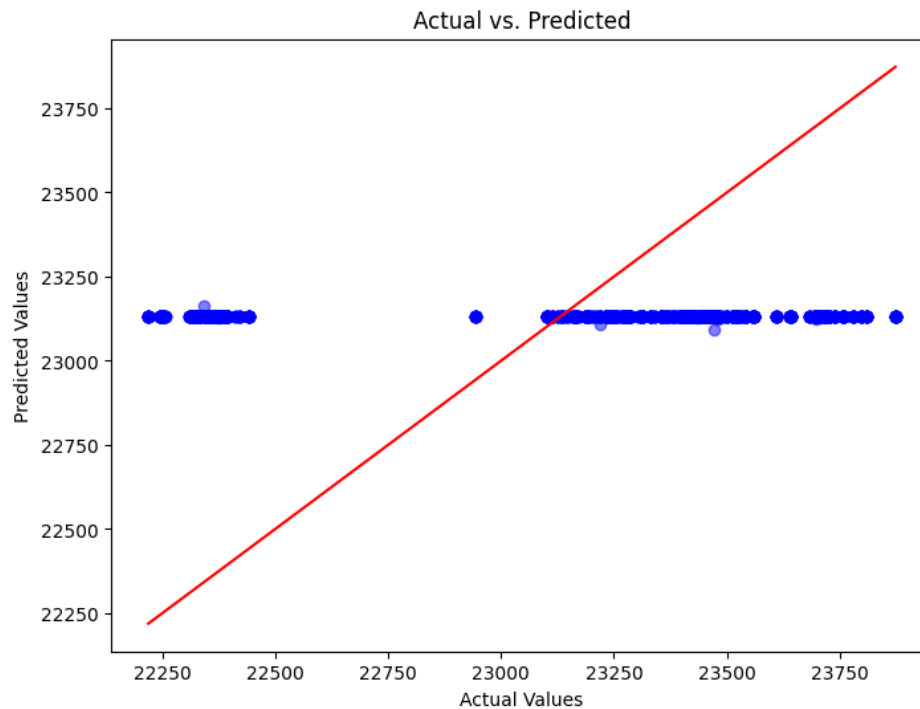
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Scatter plot
5 plt.figure(figsize=(8, 6))
6 plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
7 plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
8 plt.xlabel('Actual Values')
9 plt.ylabel('Predicted Values')
10 plt.title('Actual vs. Predicted')
11 plt.show()
12
13 # Residual plot

```

```

14 plt.figure(figsize=(8, 6))
15 residuals = y_test - y_pred
16 plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
17 plt.axhline(y=0, color='red', linestyle='--')
18 plt.xlabel('Predicted Values')
19 plt.ylabel('Residuals')
20 plt.title('Residual Plot')
21 plt.show()
22

```



▼ Decision Tree Regressor

```

1 from sklearn.tree import DecisionTreeRegressor
2 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
3
4 # Train the decision tree regressor model
5 model = DecisionTreeRegressor()
6
7

```

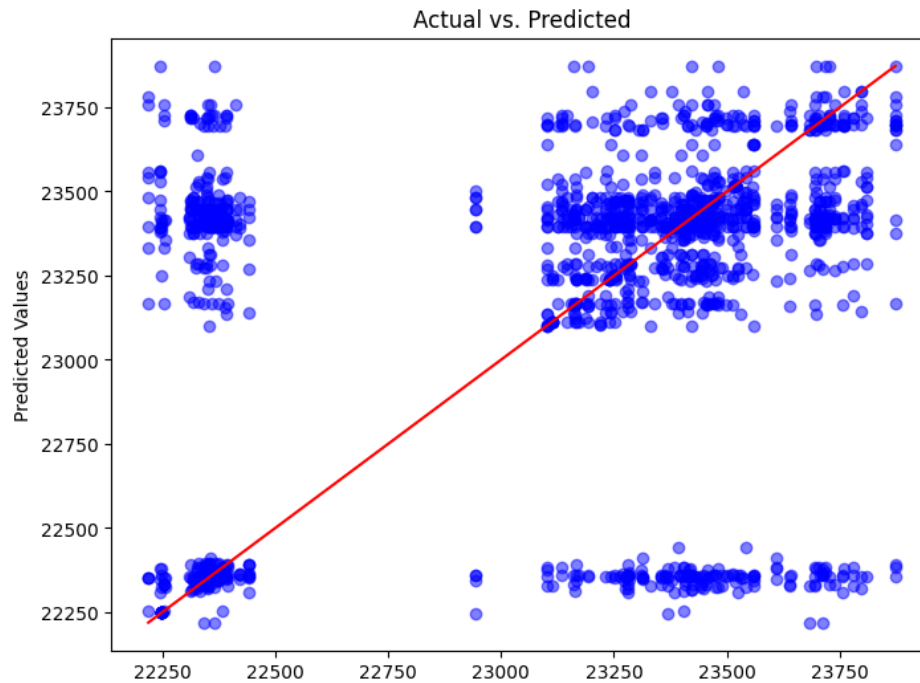
```

1 model.fit(X_train, y_train)
2
3 # Make predictions on the test set
4 y_pred = model.predict(X_test)
5
6 # Evaluate the model
7 mse = mean_squared_error(y_test, y_pred)
8 r2 = r2_score(y_test, y_pred)
9 mae = mean_absolute_error(y_test, y_pred)
10 rmse = np.sqrt(mse)
11 print("Model:", model)
12 print("Mean Squared Error:", mse)
13 print("R-squared:", r2)
14 print("Mean Absolute Error:", mae)
15 print("Root Mean Squared Error:", rmse)
16
17 # Use the trained model for future predictions
18 new_tweet = ["New tweet about Bitcoin"]
19 new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new_tweet)]])
20 predicted_close = model.predict(new_tweet_features)
21
22 print("Predicted Close Price:", predicted_close)

Model: DecisionTreeRegressor()
Mean Squared Error: 342639.9980948259
R-squared: -0.3000001902817502
Mean Absolute Error: 380.0892753164562
Root Mean Squared Error: 585.3545917602645
Predicted Close Price: [23447.51]

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Scatter plot
5 plt.figure(figsize=(8, 6))
6 plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
7 plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
8 plt.xlabel('Actual Values')
9 plt.ylabel('Predicted Values')
10 plt.title('Actual vs. Predicted')
11 plt.show()
12
13 # Residual plot
14 plt.figure(figsize=(8, 6))
15 residuals = y_test - y_pred
16 plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
17 plt.axhline(y=0, color='red', linestyle='--')
18 plt.xlabel('Predicted Values')
19 plt.ylabel('Residuals')
20 plt.title('Residual Plot')
21 plt.show()

```



▼ Random Forest Regressor

```

1500 |
1 from sklearn.ensemble import RandomForestRegressor
2 # Rest of the code is the same as above
3 model = RandomForestRegressor()
4
5
600 |
1 model.fit(X_train, y_train)
2
3 # Make predictions on the test set
4 y_pred = model.predict(X_test)
5
6 # Evaluate the model
7 mse = mean_squared_error(y_test, y_pred)
8 r2 = r2_score(y_test, y_pred)
9 mae = mean_absolute_error(y_test, y_pred)
10 rmse = np.sqrt(mse)
11 print("Model:", model)
12 print("Mean Squared Error:", mse)
13 print("R-squared:", r2)
14 print("Mean Absolute Error:", mae)
15 print("Root Mean Squared Error:", rmse)
16
17 # Use the trained model for future predictions
18 new_tweet = ["New tweet about Bitcoin"]
19 new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new_tweet)]])
20 predicted_close = model.predict(new_tweet_features)
21
22 print("Predicted Close Price:", predicted_close)

Model: RandomForestRegressor()
Mean Squared Error: 230637.58379911553
R-squared: 0.12494482695509634
Mean Absolute Error: 333.74744294567586
Root Mean Squared Error: 480.24741935705964
Predicted Close Price: [22601.664225]

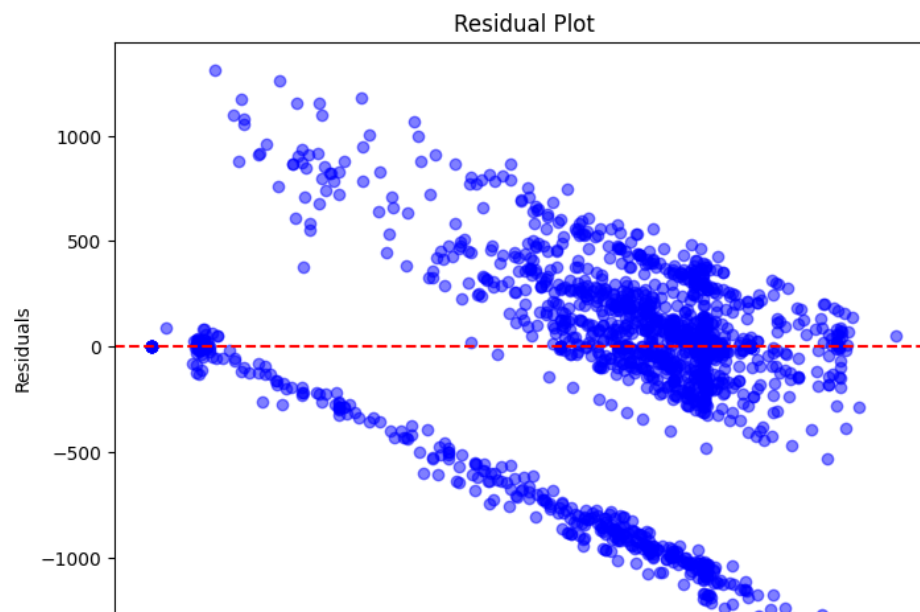
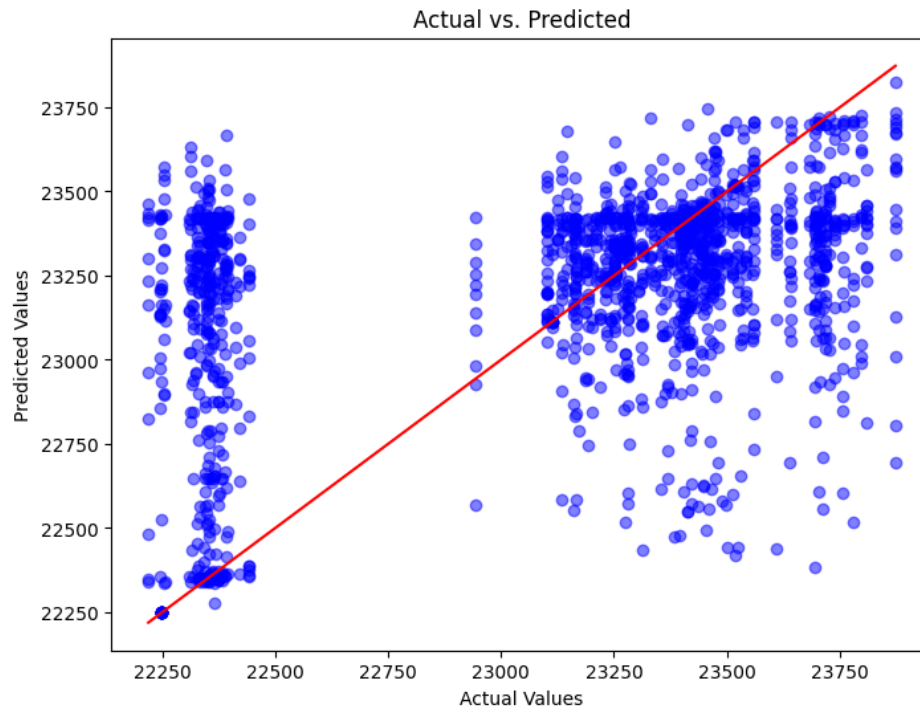
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Scatter plot
5 plt.figure(figsize=(8, 6))
6 plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
7 plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
8 plt.xlabel('Actual Values')

```

```

9 plt.ylabel('Predicted Values')
10 plt.title('Actual vs. Predicted')
11 plt.show()
12
13 # Residual plot
14 plt.figure(figsize=(8, 6))
15 residuals = y_test - y_pred
16 plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
17 plt.axhline(y=0, color='red', linestyle='--')
18 plt.xlabel('Predicted Values')
19 plt.ylabel('Residuals')
20 plt.title('Residual Plot')
21 plt.show()

```



▼ Support Vector Regressor

```

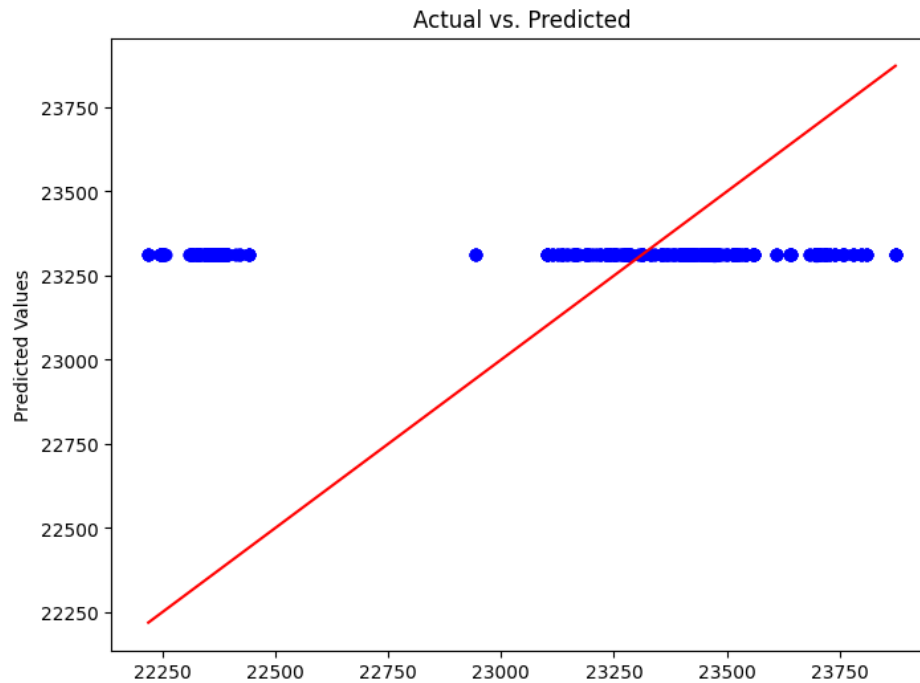
1 from sklearn.svm import SVR
2
3 model = SVR()

```

```
1 model.fit(X_train, y_train)
2
3 # Make predictions on the test set
4 y_pred = model.predict(X_test)
5
6 # Evaluate the model
7 mse = mean_squared_error(y_test, y_pred)
8 r2 = r2_score(y_test, y_pred)
9 mae = mean_absolute_error(y_test, y_pred)
10 rmse = np.sqrt(mse)
11 print("Model:", model)
12 print("Mean Squared Error:", mse)
13 print("R-squared:", r2)
14 print("Mean Absolute Error:", mae)
15 print("Root Mean Squared Error:", rmse)
16
17 # Use the trained model for future predictions
18 new_tweet = ["New tweet about Bitcoin"]
19 new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new_tweet)]])
20 predicted_close = model.predict(new_tweet_features)
21
22 print("Predicted Close Price:", predicted_close)

Model: SVR()
Mean Squared Error: 297238.96105083235
R-squared: -0.12774547067996522
Mean Absolute Error: 399.9472665866779
Root Mean Squared Error: 545.1962592047311
Predicted Close Price: [23312.32985827]

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Scatter plot
5 plt.figure(figsize=(8, 6))
6 plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
7 plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
8 plt.xlabel('Actual Values')
9 plt.ylabel('Predicted Values')
10 plt.title('Actual vs. Predicted')
11 plt.show()
12
13 # Residual plot
14 plt.figure(figsize=(8, 6))
15 residuals = y_test - y_pred
16 plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
17 plt.axhline(y=0, color='red', linestyle='--')
18 plt.xlabel('Predicted Values')
19 plt.ylabel('Residuals')
20 plt.title('Residual Plot')
21 plt.show()
```

▼ Gradient Boosting Regressor

```

1 from sklearn.ensemble import GradientBoostingRegressor
2
3 model = GradientBoostingRegressor()
4
5
6
7
8
9
10
11
12
13
14
15
16
17 # Use the trained model for future predictions
18 new_tweet = ["New tweet about Bitcoin"]
19 new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new_tweet)]])
20 predicted_close = model.predict(new_tweet_features)
21
22 print("Predicted Close Price:", predicted_close)

```

Model: GradientBoostingRegressor()
Mean Squared Error: 218693.7380274719
R-squared: 0.17026061571930096
Mean Absolute Error: 378.0107561190105
Root Mean Squared Error: 467.6470229002553
Predicted Close Price: [23144.45192473]

```

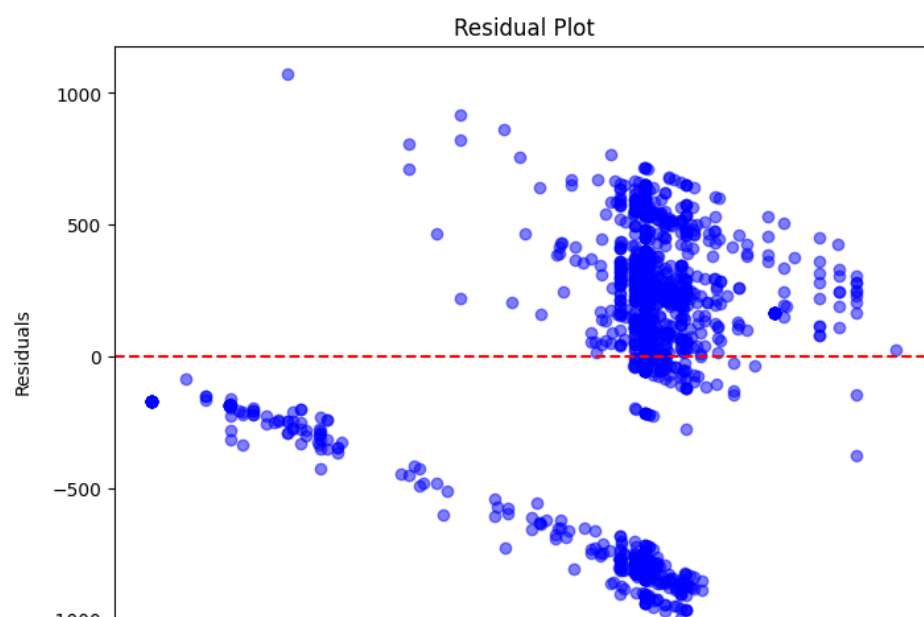
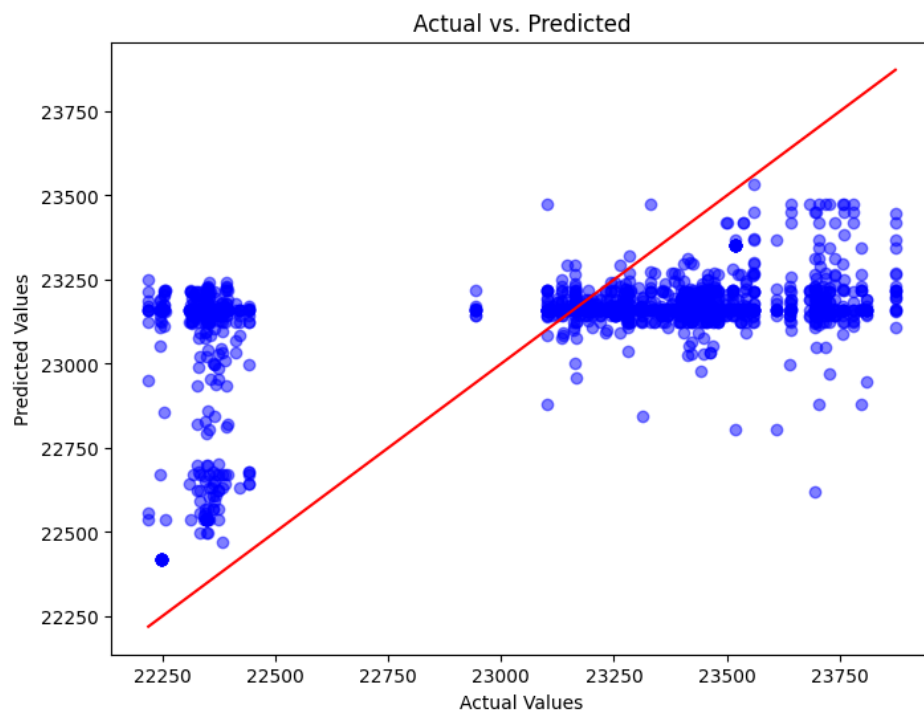
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Scatter plot
5 plt.figure(figsize=(8, 6))
6 plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
7 plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
8 plt.xlabel('Actual Values')
9 plt.ylabel('Predicted Values')
10 plt.title('Actual vs. Predicted')

```

```

11 plt.show()
12
13 # Residual plot
14 plt.figure(figsize=(8, 6))
15 residuals = y_test - y_pred
16 plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
17 plt.axhline(y=0, color='red', linestyle='--')
18 plt.xlabel('Predicted Values')
19 plt.ylabel('Residuals')
20 plt.title('Residual Plot')
21 plt.show()

```



▼ Neural Network Regressor (MLP)

```

1 from sklearn.neural_network import MLPRegressor
2
3 model = MLPRegressor()

1 model.fit(X_train, y_train)
2

```

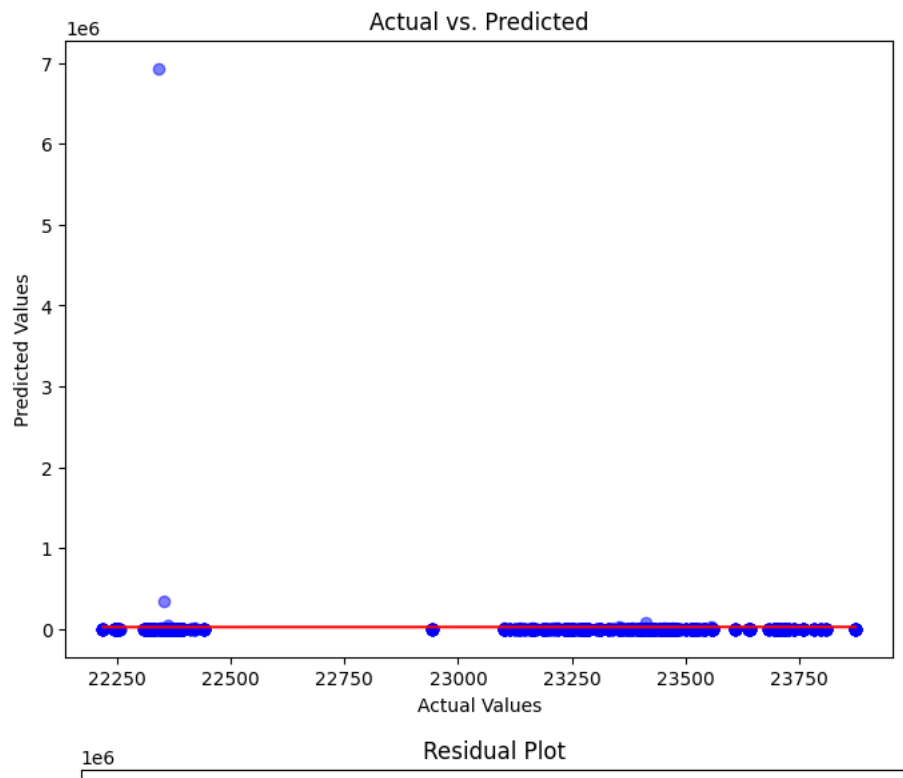
```

3 # Make predictions on the test set
4 y_pred = model.predict(X_test)
5
6 # Evaluate the model
7 mse = mean_squared_error(y_test, y_pred)
8 r2 = r2_score(y_test, y_pred)
9 mae = mean_absolute_error(y_test, y_pred)
10 rmse = np.sqrt(mse)
11 print("Model:", model)
12 print("Mean Squared Error:", mse)
13 print("R-squared:", r2)
14 print("Mean Absolute Error:", mae)
15 print("Root Mean Squared Error:", rmse)
16
17 # Use the trained model for future predictions
18 new_tweet = ["New tweet about Bitcoin"]
19 new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new_tweet)]])
20 predicted_close = model.predict(new_tweet_features)
21
22 print("Predicted Close Price:", predicted_close)

Model: MLPRegressor()
Mean Squared Error: 30822089613.234673
R-squared: -116940.17028040953
Mean Absolute Error: 27528.286373381194
Root Mean Squared Error: 175562.2100944126
Predicted Close Price: [3.93983228]

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Scatter plot
5 plt.figure(figsize=(8, 6))
6 plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
7 plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
8 plt.xlabel('Actual Values')
9 plt.ylabel('Predicted Values')
10 plt.title('Actual vs. Predicted')
11 plt.show()
12
13 # Residual plot
14 plt.figure(figsize=(8, 6))
15 residuals = y_test - y_pred
16 plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
17 plt.axhline(y=0, color='red', linestyle='--')
18 plt.xlabel('Predicted Values')
19 plt.ylabel('Residuals')
20 plt.title('Residual Plot')
21 plt.show()

```



▸ Cross Validation of Models

```
[ ] ↳ 19 cells hidden
```

▸ Hyperparameter Tuning for Price

```
[ ] ↳ 4 cells hidden
```

▸ All Models Together for comparison with price [close]

```
[ ] ↳ 5 cells hidden
```

▸ All Models Together for comparison with price_delta

```
↳ ]
1 import pandas as pd
2 from sklearn.feature_extraction.text import CountVectorizer
3 from sklearn.model_selection import train_test_split
4 from sklearn.linear_model import LinearRegression
5 from sklearn.metrics import mean_squared_error, classification_report
6 from scipy.sparse import hstack
7
8 # Feature Extraction: Unigrams
9 unigram_vectorizer = CountVectorizer(ngram_range=(1, 1))
10 unigram_features = unigram_vectorizer.fit_transform(tweets_df['text'])
11
12 # Feature Extraction: Bigrams
13 bigram_vectorizer = CountVectorizer(ngram_range=(2, 2))
14 bigram_features = bigram_vectorizer.fit_transform(tweets_df['text'])
15
16 # Combining Features
17 combined_features = hstack([unigram_features, bigram_features])
18
19 # Additional Input Features
20 additional_features = tweets_df[['compound', 'score', 'polarity', 'subjectivity', 'sentiment_score']].values
21
22 # Concatenate Additional Features with Combined Features
```

```

23 X = hstack([combined_features, additional_features])
24
25 # Target Variable
26 y = tweets_df['price_delta']
27
28 # Split the data into training and testing sets
29 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

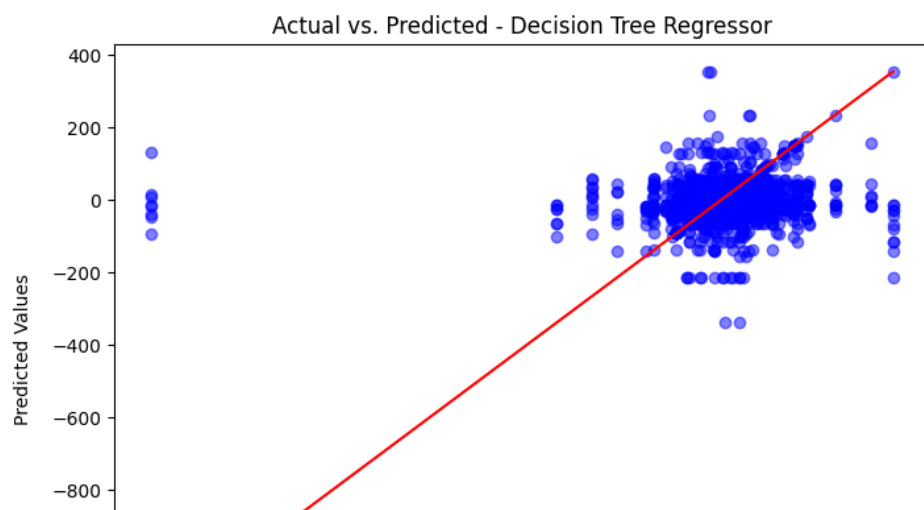
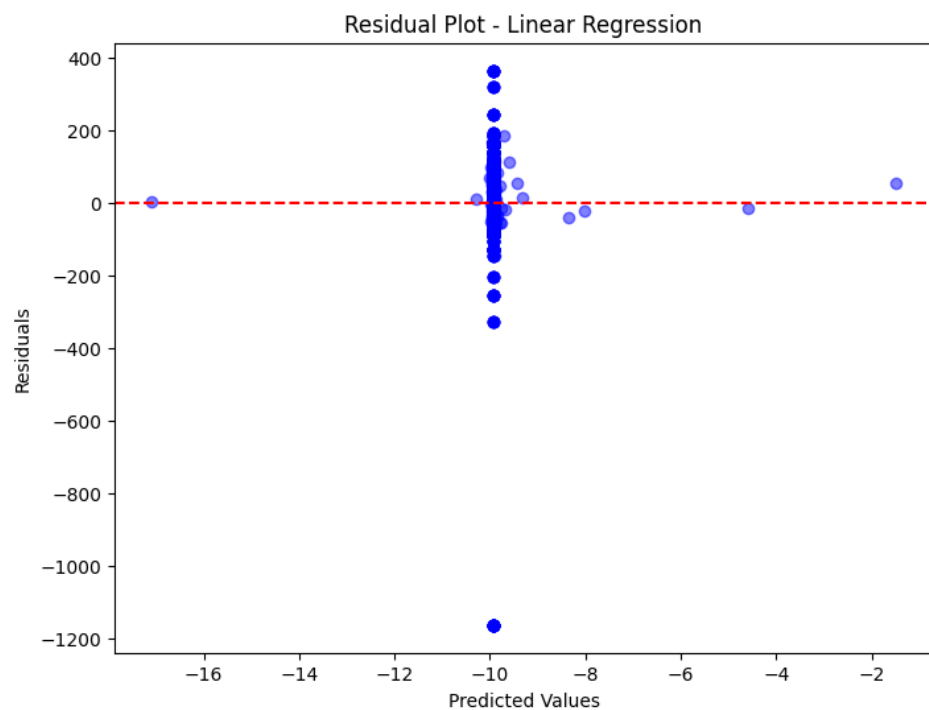
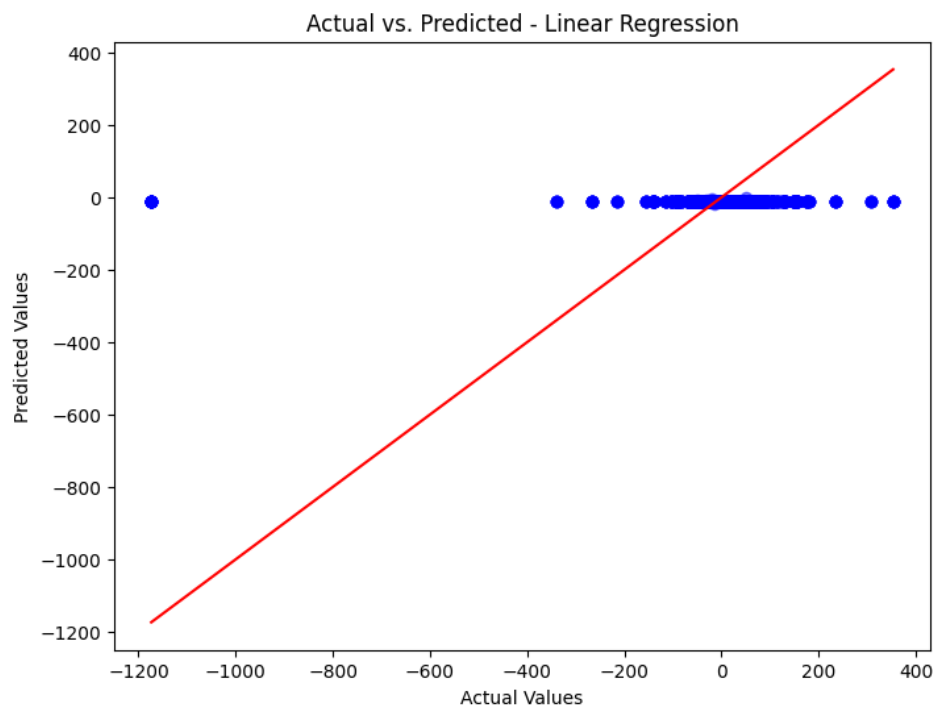
```

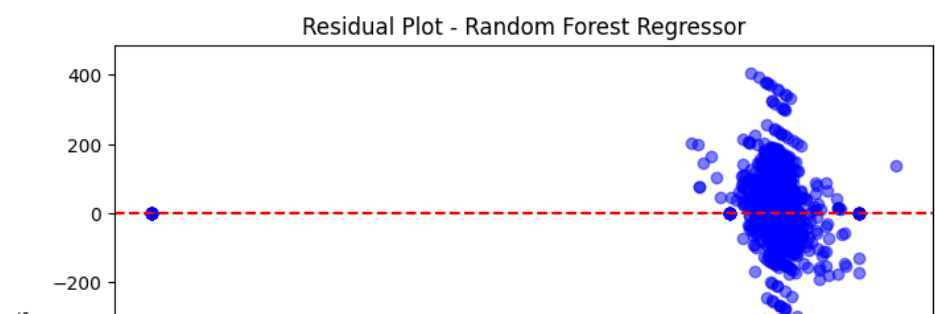
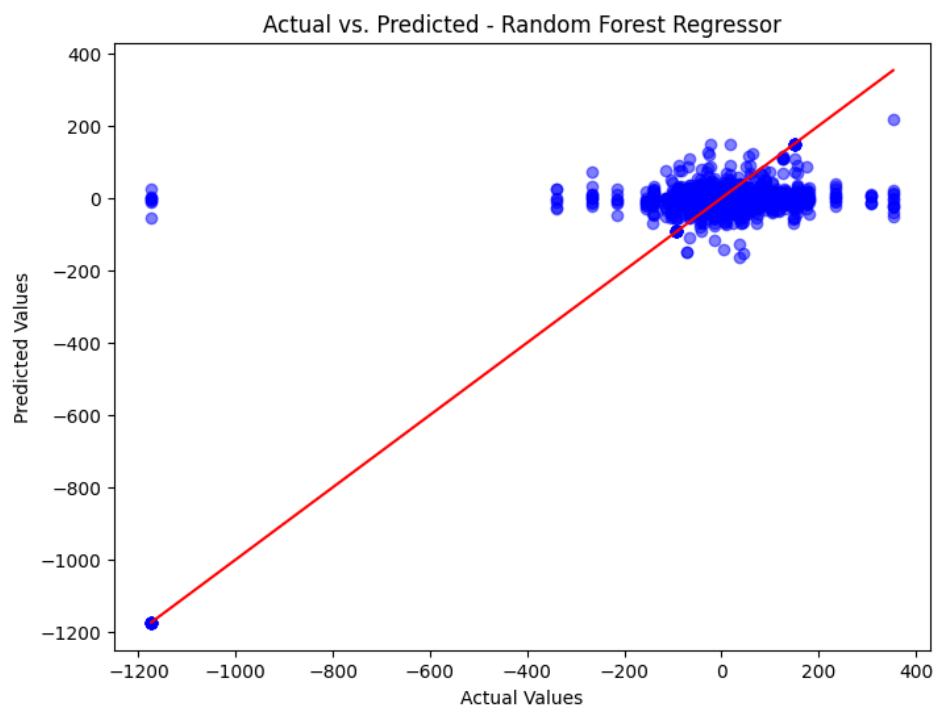
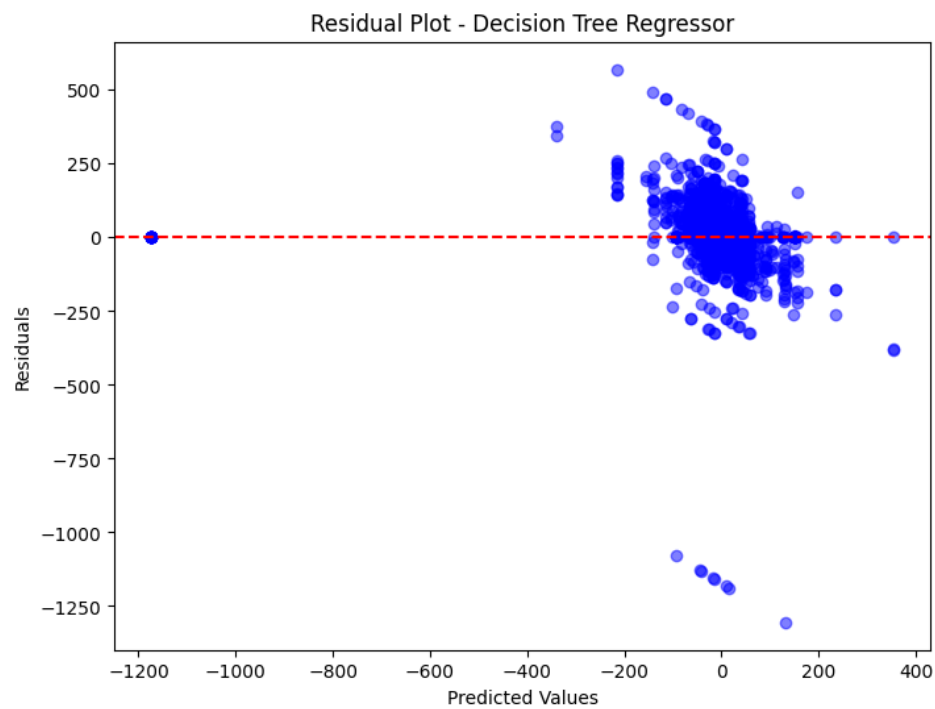
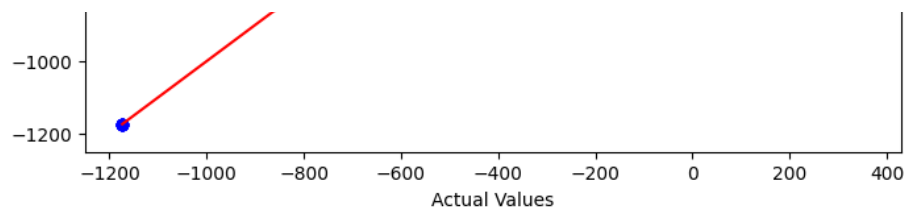
```

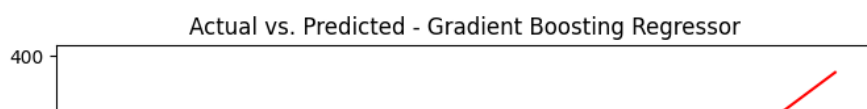
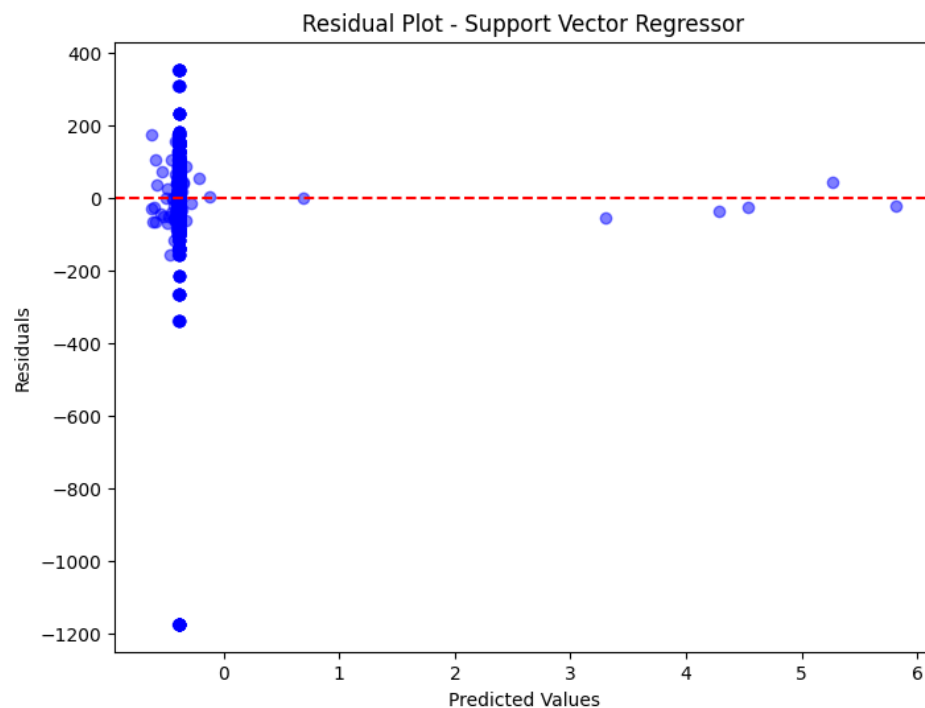
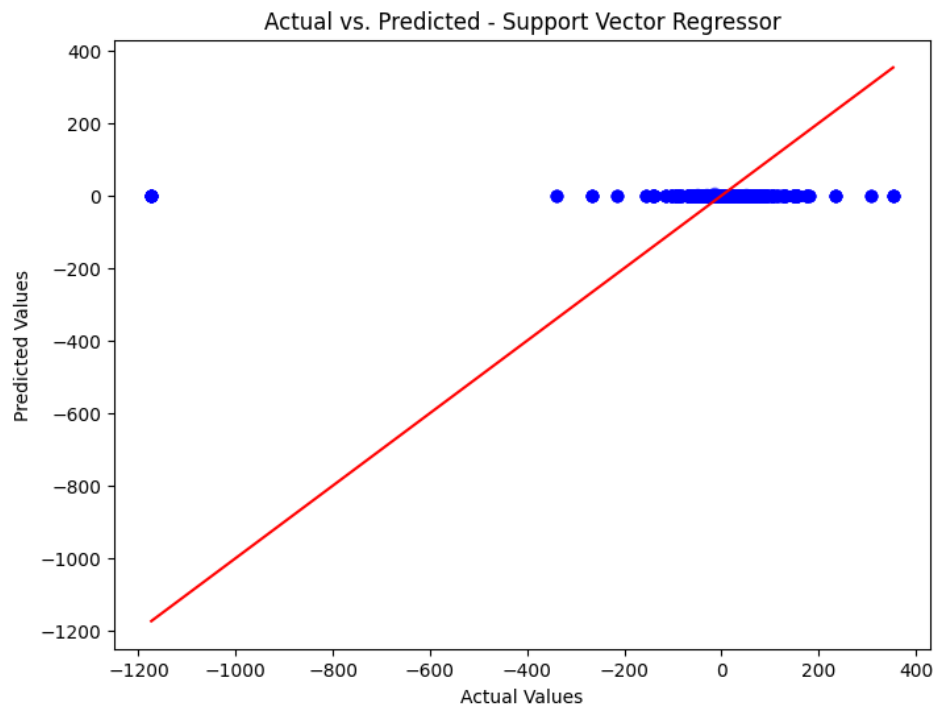
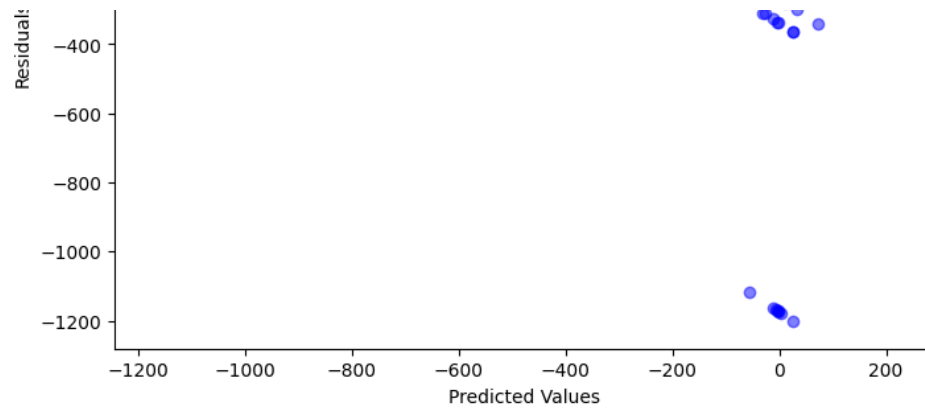
1 import pandas as pd
2 from sklearn.linear_model import LinearRegression
3 from sklearn.tree import DecisionTreeRegressor
4 from sklearn.ensemble import RandomForestRegressor
5 from sklearn.svm import SVR
6 from sklearn.ensemble import GradientBoostingRegressor
7 from sklearn.neural_network import MLPRegressor
8 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
9 from scipy.sparse import hstack
10 import matplotlib.pyplot as plt
11 import numpy as np
12
13 # Define the models
14 models = {
15     "Linear Regression": LinearRegression(),
16     "Decision Tree Regressor": DecisionTreeRegressor(),
17     "Random Forest Regressor": RandomForestRegressor(),
18     "Support Vector Regressor": SVR(),
19     "Gradient Boosting Regressor": GradientBoostingRegressor(),
20     "Neural Network Regressor": MLPRegressor()
21 }
22
23 # Initialize an empty dictionary to store the results
24 results = {}
25
26 # Iterate over the models
27 for model_name, model in models.items():
28     # Train the model
29     model.fit(X_train, y_train)
30
31     # Make predictions on the test set
32     y_pred = model.predict(X_test)
33
34     # Evaluate the model
35     mse = mean_squared_error(y_test, y_pred)
36     r2 = r2_score(y_test, y_pred)
37     mae = mean_absolute_error(y_test, y_pred)
38     rmse = np.sqrt(mse)
39
40     # Store the results in the dictionary
41     results[model_name] = {
42         "Mean Squared Error": mse,
43         "R-squared": r2,
44         "Mean Absolute Error": mae,
45         "Root Mean Squared Error": rmse
46     }
47
48     # Use the trained model for future predictions
49     new_tweet = ["New tweet about Bitcoin"]
50     new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:1]
51     predicted_close = model.predict(new_tweet_features)
52
53     results[model_name]["Predicted Close Price"] = predicted_close
54
55     # Scatter plot
56     plt.figure(figsize=(8, 6))
57     plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
58     plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
59     plt.xlabel('Actual Values')
60     plt.ylabel('Predicted Values')
61     plt.title(f'Actual vs. Predicted - {model_name}')
62     plt.show()
63
64     # Residual plot
65     plt.figure(figsize=(8, 6))

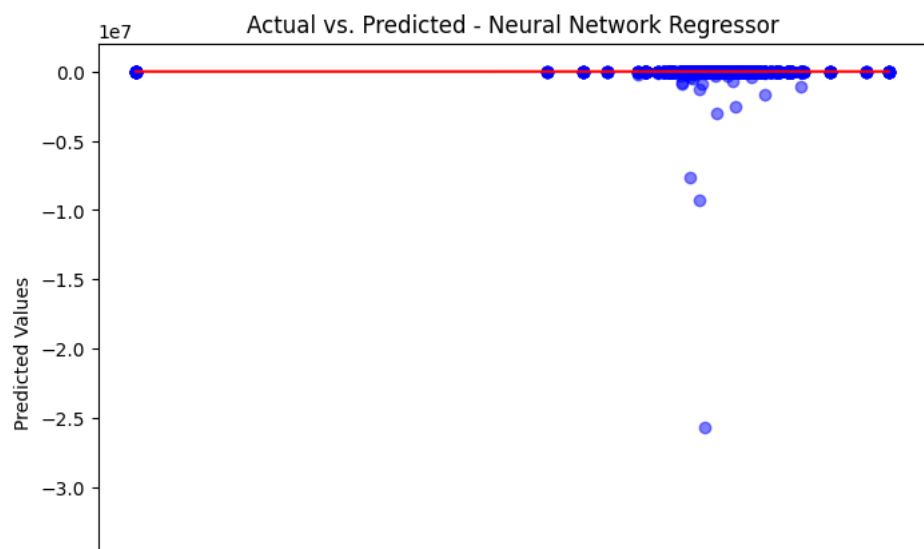
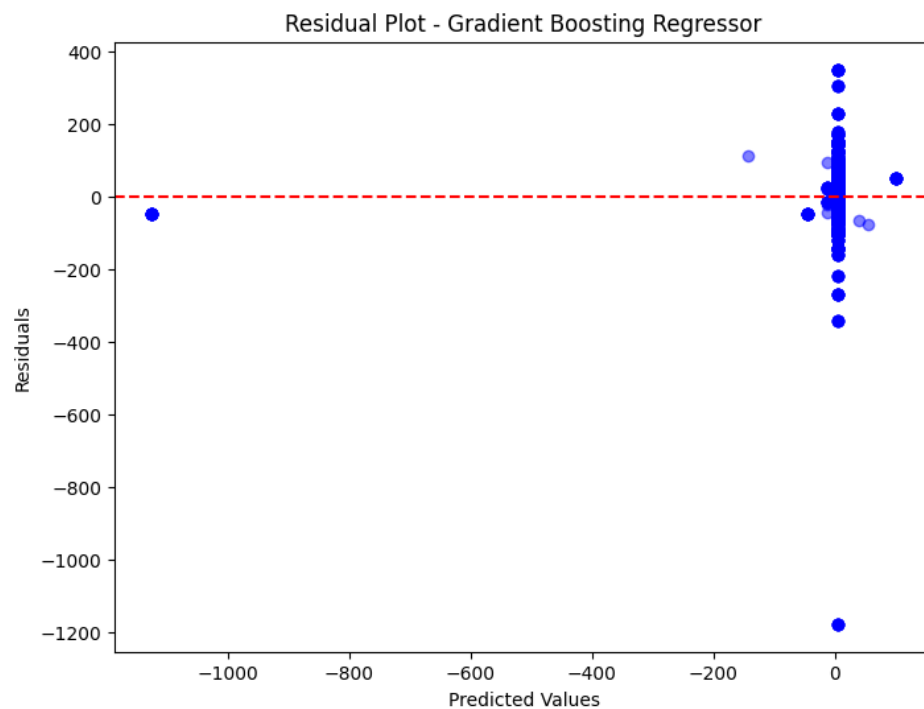
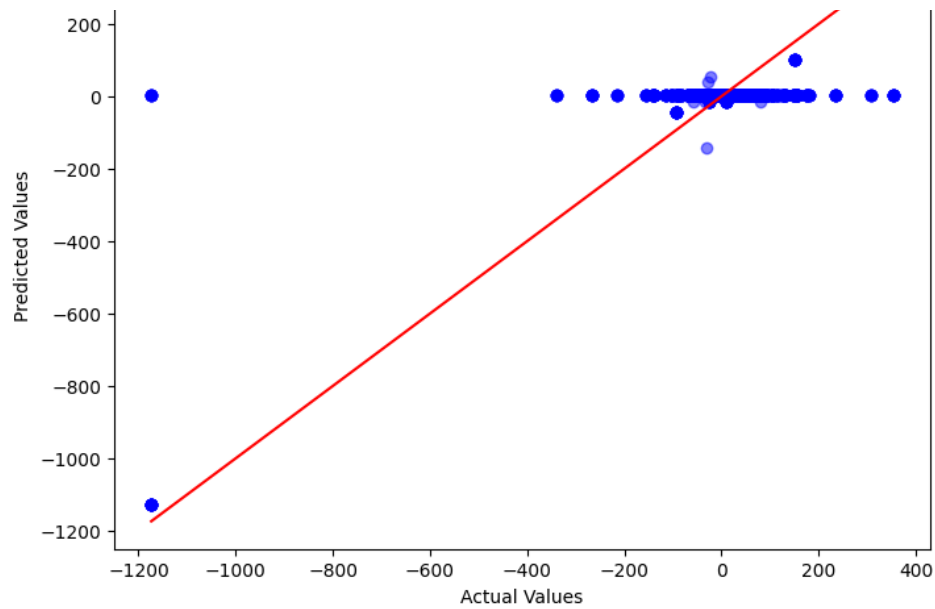
```

```
66     residuals = y_test - y_pred
67     plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
68     plt.axhline(y=0, color='red', linestyle='--')
69     plt.xlabel('Predicted Values')
70     plt.ylabel('Residuals')
71     plt.title(f'Residual Plot - {model_name}')
72     plt.show()
73
74 # Convert the results to a pandas DataFrame for tabular representation
75 results_df = pd.DataFrame.from_dict(results, orient="index")
76
77 # Print the results
78 print(results_df)
79
```









1 #The Decision Tree Regressor, Random Forest Regressor, and Gradient Boosting Regressor perform relatively better with higher R-squared val
2 #The Neural Network Regressor has a significantly negative R-squared value and extremely high mean squared error and mean absolute error,

```
3 #Among the three better-performing models, the Random Forest Regressor has the lowest mean squared error and mean absolute error, followed
4 #Based on these metrics, both the Random Forest Regressor and Gradient Boosting Regressor show promise in predicting price change, with th
```

Actual Values

```
1
2 # Transpose the DataFrame
3 transposed_df = results_df.transpose()
4
5 # Print the transposed DataFrame
6 print(transposed_df)
```

▼ Cross Validation for price_delta

```
1
2
```

▼ Linear Regression

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.metrics import mean_squared_error, accuracy_score, precision_score, recall_score, f1_score
3
4 # Train the linear regression model
5 model = LinearRegression()
6
```

```
1 from sklearn.tree import DecisionTreeRegressor
2 from sklearn.model_selection import cross_val_score
3 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
4
5
6 # Perform cross-validation
7 cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
8
9 # Convert the negative mean squared error scores to positive
10 mse_scores = -cv_scores
11
12 # Calculate the mean and standard deviation of the MSE scores
13 mean_mse = np.mean(mse_scores)
14 std_mse = np.std(mse_scores)
15
16 # Print the mean and standard deviation of the MSE scores
17 print("Mean MSE:", mean_mse)
18 print("Std MSE:", std_mse)
19
```

Mean MSE: 26602.32728304749

Std MSE: 34442.19984498466

▼ Decision Tree Regressor

```
1 from sklearn.tree import DecisionTreeRegressor
2 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
3
4 # Train the decision tree regressor model
5 model = DecisionTreeRegressor()
6
7
```

```
1 from sklearn.tree import DecisionTreeRegressor
2 from sklearn.model_selection import cross_val_score
3 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
4
5
6 # Perform cross-validation
7 cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
8
9 # Convert the negative mean squared error scores to positive
10 mse_scores = -cv_scores
11
```

```

12 # Calculate the mean and standard deviation of the MSE scores
13 mean_mse = np.mean(mse_scores)
14 std_mse = np.std(mse_scores)
15
16 # Print the mean and standard deviation of the MSE scores
17 print("Mean MSE:", mean_mse)
18 print("Std MSE:", std_mse)
19

```

```

Mean MSE: 29497.64889650693
Std MSE: 34444.122541768076

```

▼ Random Forest Regressor

```

1 from sklearn.ensemble import RandomForestRegressor
2
3 model = RandomForestRegressor()

1 from sklearn.tree import DecisionTreeRegressor
2 from sklearn.model_selection import cross_val_score
3 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
4
5
6 # Perform cross-validation
7 cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
8
9 # Convert the negative mean squared error scores to positive
10 mse_scores = -cv_scores
11
12 # Calculate the mean and standard deviation of the MSE scores
13 mean_mse = np.mean(mse_scores)
14 std_mse = np.std(mse_scores)
15
16 # Print the mean and standard deviation of the MSE scores
17 print("Mean MSE:", mean_mse)
18 print("Std MSE:", std_mse)
19

```

```

Mean MSE: 26945.52137651674
Std MSE: 34605.309807612946

```

▼ Support Vector Regressor

```

1 from sklearn.svm import SVR
2
3 model = SVR()

1 from sklearn.tree import DecisionTreeRegressor
2 from sklearn.model_selection import cross_val_score
3 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
4
5
6 # Perform cross-validation
7 cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
8
9 # Convert the negative mean squared error scores to positive
10 mse_scores = -cv_scores
11
12 # Calculate the mean and standard deviation of the MSE scores
13 mean_mse = np.mean(mse_scores)
14 std_mse = np.std(mse_scores)
15
16 # Print the mean and standard deviation of the MSE scores
17 print("Mean MSE:", mean_mse)
18 print("Std MSE:", std_mse)
19

```

```

Mean MSE: 26258.128645849814
Std MSE: 34451.38200920118

```

▼ Gradient Boosting Regressor

```

1 from sklearn.ensemble import GradientBoostingRegressor
2
3 model = GradientBoostingRegressor()

1

1 from sklearn.tree import DecisionTreeRegressor
2 from sklearn.model_selection import cross_val_score
3 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
4
5
6 # Perform cross-validation
7 cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
8
9 # Convert the negative mean squared error scores to positive
10 mse_scores = -cv_scores
11
12 # Calculate the mean and standard deviation of the MSE scores
13 mean_mse = np.mean(mse_scores)
14 std_mse = np.std(mse_scores)
15
16 # Print the mean and standard deviation of the MSE scores
17 print("Mean MSE:", mean_mse)
18 print("Std MSE:", std_mse)
19

Mean MSE: 26563.170954889654
Std MSE: 34572.294054287326

```

▼ Neural Network Regressor (MLP)

```

1 from sklearn.neural_network import MLPRegressor
2
3 model = MLPRegressor()

1 from sklearn.tree import DecisionTreeRegressor
2 from sklearn.model_selection import cross_val_score
3 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
4
5
6 # Perform cross-validation
7 cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
8
9 # Convert the negative mean squared error scores to positive
10 mse_scores = -cv_scores
11
12 # Calculate the mean and standard deviation of the MSE scores
13 mean_mse = np.mean(mse_scores)
14 std_mse = np.std(mse_scores)
15
16 # Print the mean and standard deviation of the MSE scores
17 print("Mean MSE:", mean_mse)
18 print("Std MSE:", std_mse)
19

Mean MSE: 182165942647766.62
Std MSE: 344463338514648.1

```

▼ Hyperparameter Tuning for Price_delta

```

1 #Hyperparameter Tuning

1 import pandas as pd
2 from sklearn.feature_extraction.text import CountVectorizer

```

```
3 from sklearn.model_selection import train_test_split
4 from sklearn.linear_model import LinearRegression
5 from sklearn.metrics import mean_squared_error, classification_report
6 from scipy.sparse import hstack
7
8 # Feature Extraction: Unigrams
9 unigram_vectorizer = CountVectorizer(ngram_range=(1, 1))
10 unigram_features = unigram_vectorizer.fit_transform(tweets_df['text'])
11
12 # Feature Extraction: Bigrams
13 bigram_vectorizer = CountVectorizer(ngram_range=(2, 2))
14 bigram_features = bigram_vectorizer.fit_transform(tweets_df['text'])
15
16 # Combining Features
17 combined_features = hstack([unigram_features, bigram_features])
18
19 # Additional Input Features
20 additional_features = tweets_df[['compound', 'score', 'polarity', 'subjectivity', 'sentiment_score']].values
21
22 # Concatenate Additional Features with Combined Features
23 X = hstack([combined_features, additional_features])
24
25 # Target Variable
26 y = tweets_df['close']
27
28 # Split the data into training and testing sets
29 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
46 results = {}
47
48 # Iterate over the models
49 for model_name, model in models.items():
50     print("Model:", model_name)
51
52     # Perform feature selection
53     feature_selector = SelectKBest()
54     X_selected = feature_selector.fit_transform(X, y)
55
56     # Perform grid search cross-validation
57     param_grid = param_grids[model_name]
58     grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error')
59     grid_search.fit(X_selected, y)
60
61     # Get the best model and its corresponding hyperparameters
62     best_model = grid_search.best_estimator_
63     best_params = grid_search.best_params_
64
65     # Train the best model on the entire dataset
66     best_model.fit(X_selected, y)
67
68     # Make predictions on the test set
69     X_test_selected = feature_selector.transform(X_test)
70     y_pred = best_model.predict(X_test_selected)
71
72     # Evaluate the best model
73     mse = mean_squared_error(y_test, y_pred)
74     r2 = r2_score(y_test, y_pred)
75     mae = mean_absolute_error(y_test, y_pred)
76     rmse = np.sqrt(mse)
77
78     # Store the results in the dictionary
79     results[model_name] = {
80         "Best Model": best_model,
81         "Best Parameters": best_params,
82         "Mean Squared Error": mse,
83         "R-squared": r2,
84         "Mean Absolute Error": mae,
85         "Root Mean Squared Error": rmse
86     }
87
88     # Print the results
89     print("Best Model:", best_model)
90     print("Best Parameters:", best_params)
91     print("Mean Squared Error:", mse)
92     print("R-squared:", r2)
93     print("Mean Absolute Error:", mae)
94     print("Root Mean Squared Error:", rmse)
95     print()
96
```

Mean Absolute Error: 399.4075468116839
Root Mean Squared Error: 544.091008076034

Model: Gradient Boosting Regressor
Best Model: GradientBoostingRegressor(n_estimators=50)
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}

```
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimiz
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimiz
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimiz
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimiz
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimiz
warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimiz
warnings.warn(
Best Model: MLPRegressor(alpha=0.1, hidden_layer_sizes=(200, 100))
Best Parameters: {'alpha': 0.1, 'hidden_layer_sizes': (200, 100)}
Mean Squared Error: 249884.8371738839
R-squared: 0.051919397382582666
Mean Absolute Error: 413.3885683452907
Root Mean Squared Error: 499.8848239083518

Model: Linear Regression
Best Model: LinearRegression()
Best Parameters: {}
Mean Squared Error: 249886.37207620472
R-squared: 0.05191357385549822
Mean Absolute Error: 413.0579424764286
Root Mean Squared Error: 499.88635916196466
```

✓ 12m 39s completed at 12:30 PM

