Datasets

[ ] ↳ *9 cells hidden*

Combining both datasets

[ ] ↳ *3 cells hidden*

Classification on Price Prediction based on sentiment

```
import pandas as pd
import numpy as np

# Define custom bin edges based on quantiles
bin_edges = np.linspace(df_score['compound'].min(), df_score['compound'].max(), num=6)  # Adjust the 'num' parameter as needed

# Define labels
labels = ['Extreme Negative', 'Negative', 'Neutral', 'Positive', 'Extreme Positive']

# Calculate average compound score for each sentiment level
sentiment_scores = []
for i in range(len(bin_edges)-1):
    lower_bound = bin_edges[i]
    upper_bound = bin_edges[i+1]
    scores_in_range = df_score[(df_score['compound'] >= lower_bound) & (df_score['compound'] < upper_bound)]['compound']
    sentiment_scores.append(scores_in_range.mean())

# Map sentiment levels to numerical values with scores
sentiment_mapping = {label: score for label, score in zip(labels, sentiment_scores)}
df_score['sentiment_score'] = df_score['sentiment_level'].map(sentiment_mapping)

# Save the updated dataframe as a new CSV file
df_score.to_csv('updated_sentiment_data.csv', index=False)

```

```
# Create a copy of the  bitcoin tweets DataFrame
df_tweets = pd.read_csv('/content/updated_sentiment_data.csv')
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 ð□□»ð□□□Crypto ...	2011-01-11 01:37:00	42.0	22.0	

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

```
print(tweets_df.columns)

```

```
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')
```

```
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
```

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, classification_report
from scipy.sparse import hstack

# Feature Extraction: Unigrams
unigram_vectorizer = CountVectorizer(ngram_range=(1, 1))
unigram_features = unigram_vectorizer.fit_transform(tweets_df['text'])

# Feature Extraction: Bigrams
bigram_vectorizer = CountVectorizer(ngram_range=(2, 2))
bigram_features = bigram_vectorizer.fit_transform(tweets_df['text'])

# Combining Features
combined_features = hstack([unigram_features, bigram_features])

# Additional Input Features
additional_features = tweets_df[['compound', 'score', 'polarity', 'subjectivity', 'sentiment_score']].values

# Concatenate Additional Features with Combined Features
X = hstack([combined_features, additional_features])

# Target Variable
y = tweets_df['close']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



```

```
import numpy as np

# Print the first 10 rows of the term frequency matrix
print(combined_features[:10].toarray())
```

```
[[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]]
```

```
from sklearn.metrics import mean_squared_error, accuracy_score, precision_score, recall_score, f1_score
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from scipy.sparse import hstack
```

Linear Regression

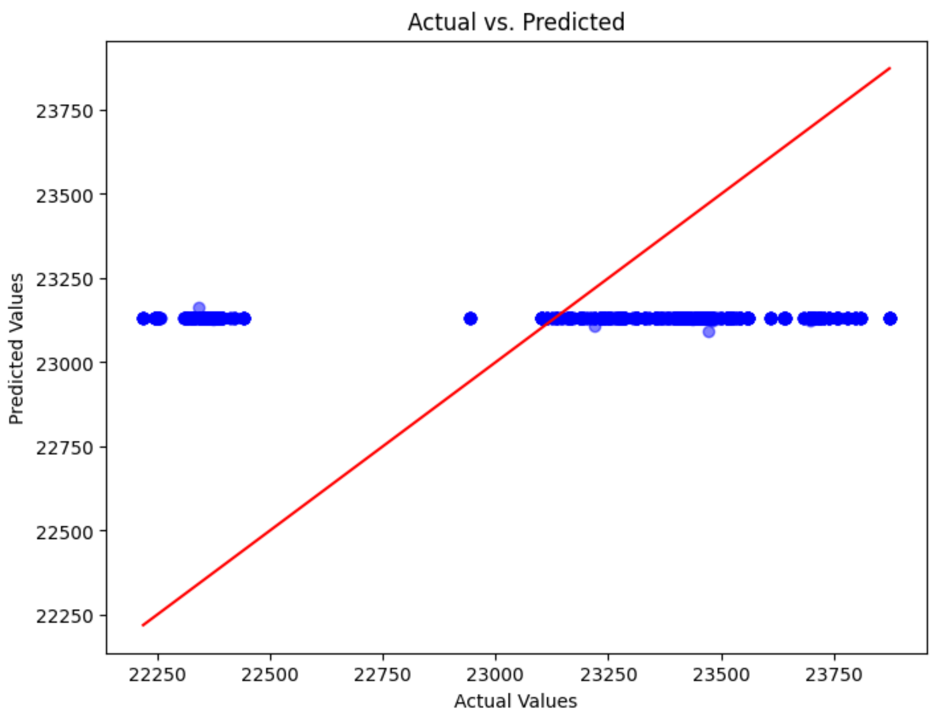
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score, precision_score, recall_score, f1_score

# Train the linear regression model
model = LinearRegression()
```

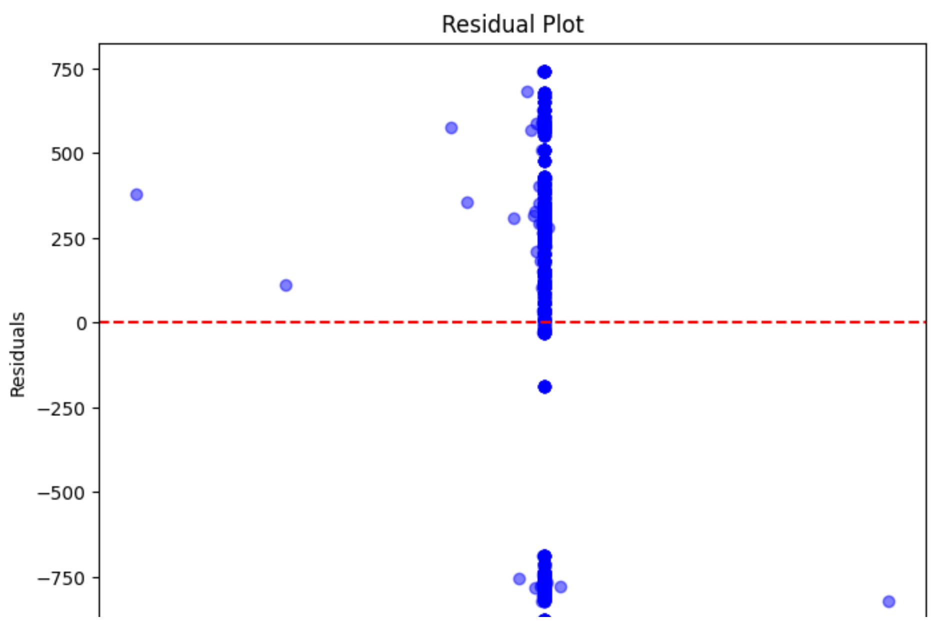
```

model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)

# Use the trained model for future predictions
new_tweet = ["New tweet about Bitcoin"]
new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new
predicted_close = model.predict(new_tweet_features)

print("Predicted Close Price:", predicted_close)
```

```
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]
```

```
import matplotlib.pyplot as plt
import numpy as np

# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted')
plt.show()

# Residual plot
```

```
plt.figure(figsize=(8, 6))
residuals = y_test - y_pred
plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()

```





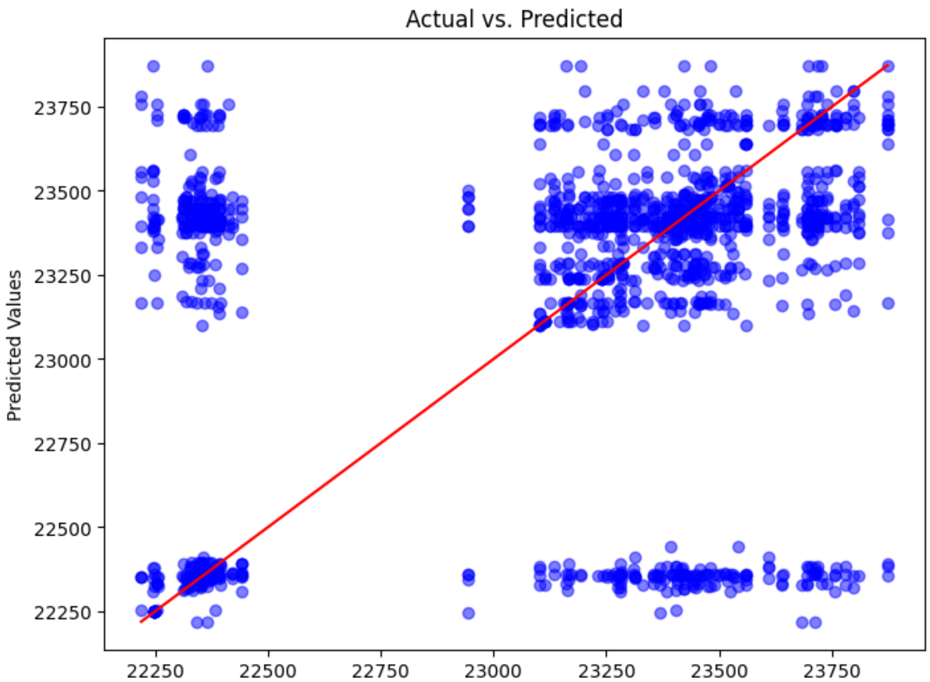
Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Train the decision tree regressor model
model = DecisionTreeRegressor()


```

```
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Model:", model)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)

# Use the trained model for future predictions
new_tweet = ["New tweet about Bitcoin"]
new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new
predicted_close = model.predict(new_tweet_features)

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]
```

```
import matplotlib.pyplot as plt
import numpy as np

# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted')
plt.show()

# Residual plot
plt.figure(figsize=(8, 6))
residuals = y_test - y_pred
plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```



Random Forest Regressor

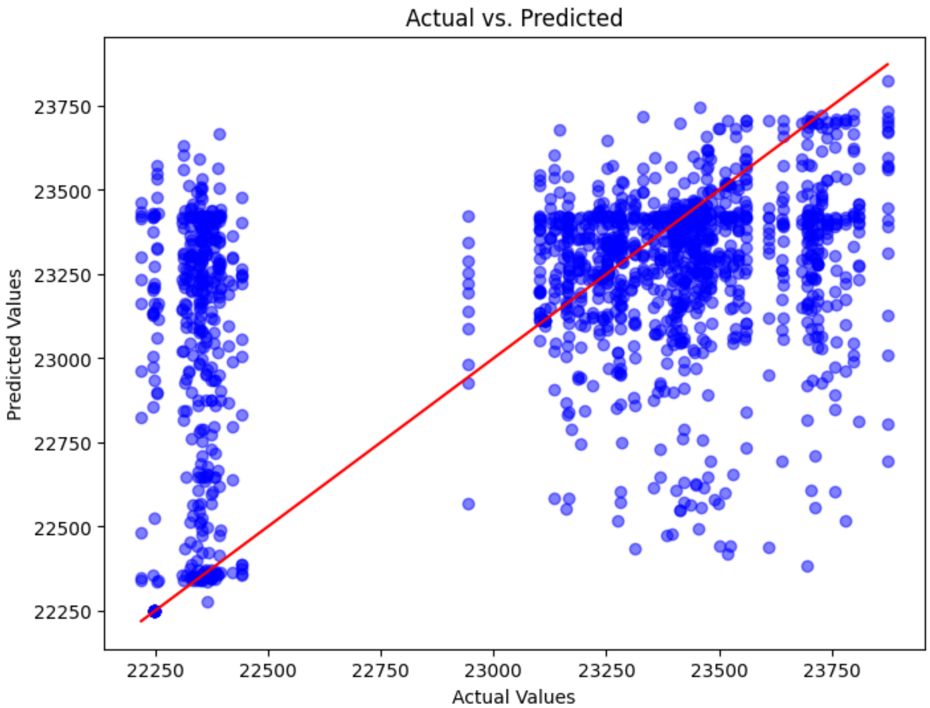
```
from sklearn.ensemble import RandomForestRegressor
# Rest of the code is the same as above
model = RandomForestRegressor()


```

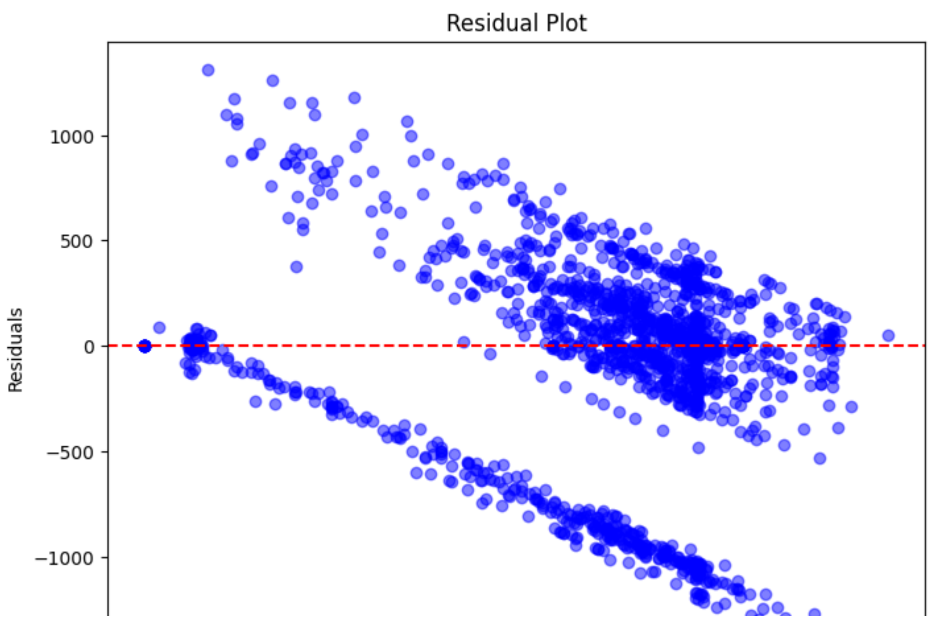
```
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Model:", model)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)

# Use the trained model for future predictions
new_tweet = ["New tweet about Bitcoin"]
new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new
predicted_close = model.predict(new_tweet_features)

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]
```

```
import matplotlib.pyplot as plt
import numpy as np

# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.xlabel('Actual Values')
```

```
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted')
plt.show()

# Residual plot
plt.figure(figsize=(8, 6))
residuals = y_test - y_pred
plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```





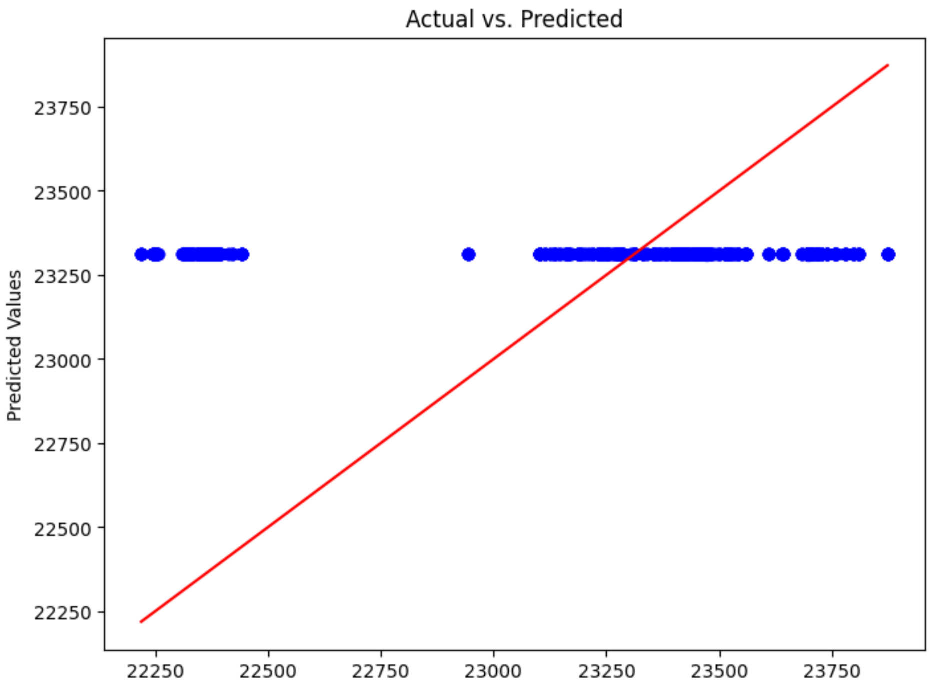
Support Vector Regressor

```
from sklearn.svm import SVR

model = SVR()
```

```
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Model:", model)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)

# Use the trained model for future predictions
new_tweet = ["New tweet about Bitcoin"]
new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new
predicted_close = model.predict(new_tweet_features)

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]
```

```
import matplotlib.pyplot as plt
import numpy as np

# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted')
plt.show()

# Residual plot
plt.figure(figsize=(8, 6))
residuals = y_test - y_pred
plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```



Gradient Boosting Regressor

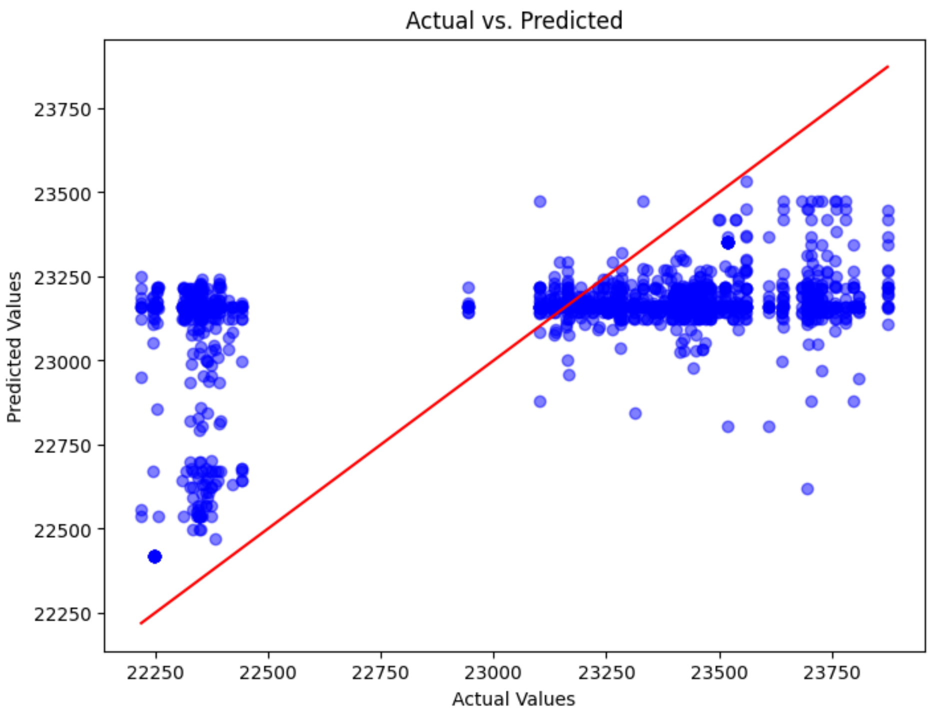
```
from sklearn.ensemble import GradientBoostingRegressor

model = GradientBoostingRegressor()
```

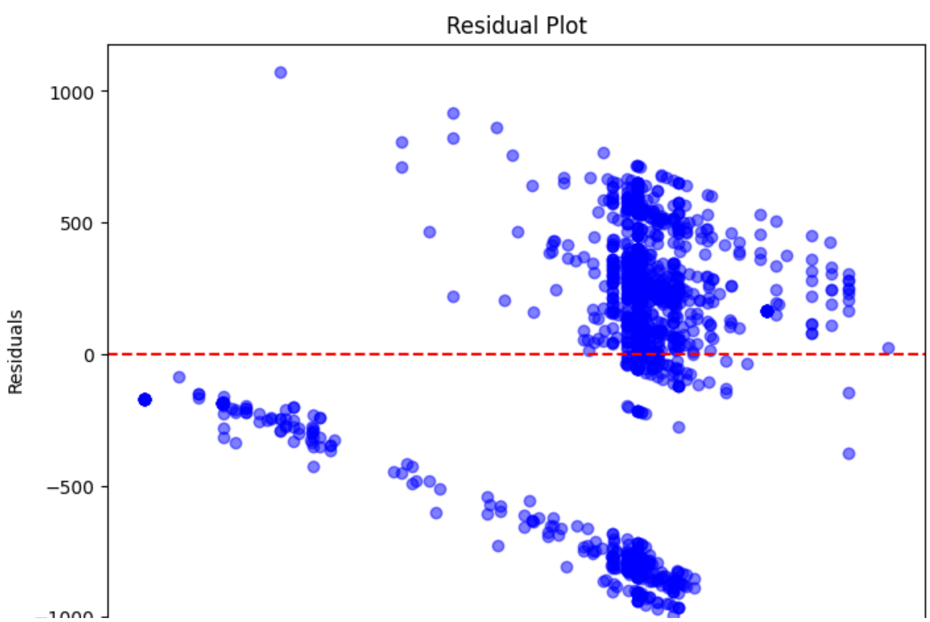
```
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Model:", model)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)

# Use the trained model for future predictions
new_tweet = ["New tweet about Bitcoin"]
new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new
predicted_close = model.predict(new_tweet_features)

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]
```

```
import matplotlib.pyplot as plt
import numpy as np

# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted')
```

```
plt.show()

# Residual plot
plt.figure(figsize=(8, 6))
residuals = y_test - y_pred
plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```





Neural Network Regressor (MLP)

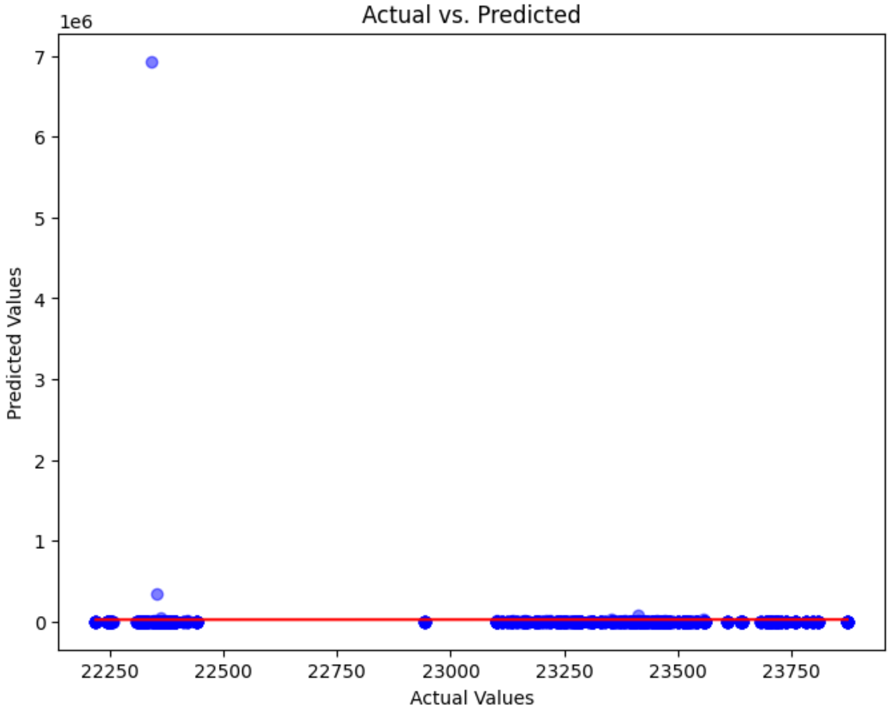
```
from sklearn.neural_network import MLPRegressor

model = MLPRegressor()
```

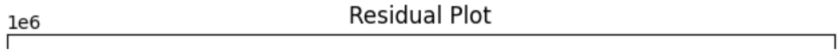
```
model.fit(X_train, y_train)

```

```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Model:", model)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)

# Use the trained model for future predictions
new_tweet = ["New tweet about Bitcoin"]
new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:len(new
predicted_close = model.predict(new_tweet_features)

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]
```

```
import matplotlib.pyplot as plt
import numpy as np

# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted')
plt.show()

# Residual plot
plt.figure(figsize=(8, 6))
residuals = y_test - y_pred
plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
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*

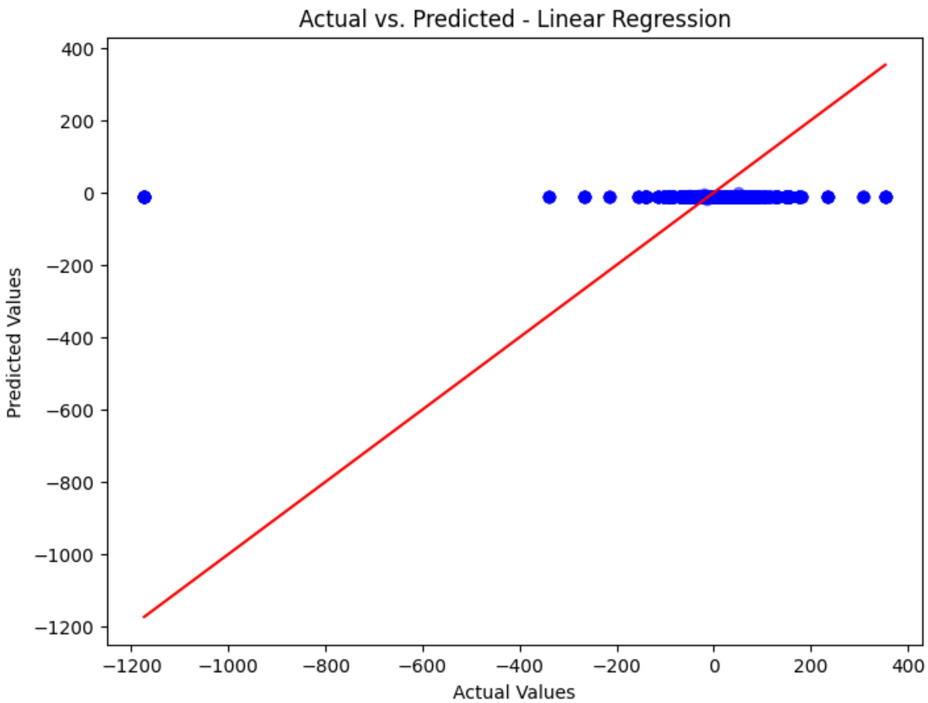
All Models Together for comparison with price_delta

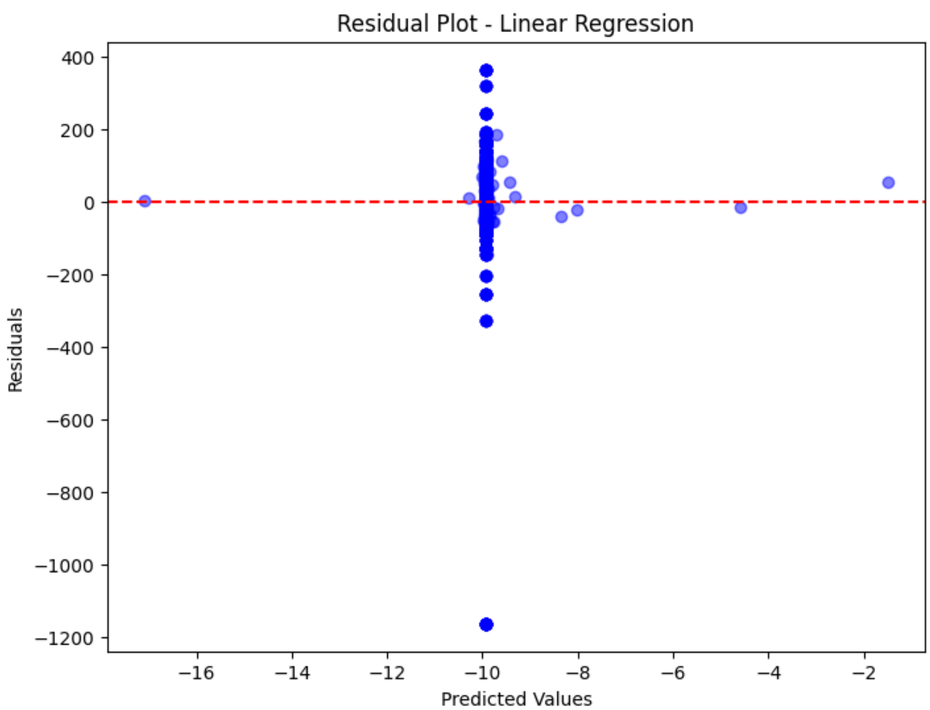
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, classification_report
from scipy.sparse import hstack

# Feature Extraction: Unigrams
unigram_vectorizer = CountVectorizer(ngram_range=(1, 1))
unigram_features = unigram_vectorizer.fit_transform(tweets_df['text'])

# Feature Extraction: Bigrams
bigram_vectorizer = CountVectorizer(ngram_range=(2, 2))
bigram_features = bigram_vectorizer.fit_transform(tweets_df['text'])

# Combining Features
combined_features = hstack([unigram_features, bigram_features])

# Additional Input Features
additional_features = tweets_df[['compound', 'score', 'polarity', 'subjectivity', 'sentiment_score']].values

# Concatenate Additional Features with Combined Features
```

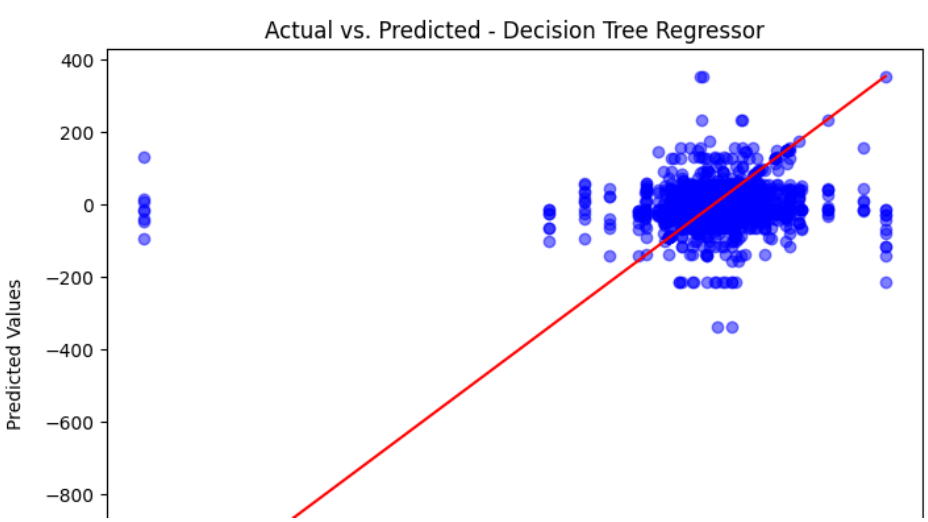
```
X = hstack([combined_features, additional_features])

# Target Variable
y = tweets_df['price_delta']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



```

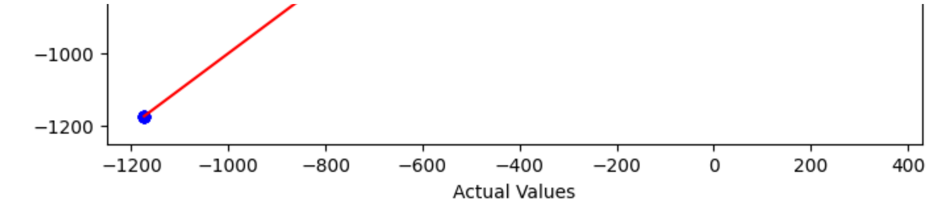
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from scipy.sparse import hstack
import matplotlib.pyplot as plt
import numpy as np

# Define the models
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree Regressor": DecisionTreeRegressor(),
    "Random Forest Regressor": RandomForestRegressor(),
    "Support Vector Regressor": SVR(),
    "Gradient Boosting Regressor": GradientBoostingRegressor(),
    "Neural Network Regressor": MLPRegressor()
}

# Initialize an empty dictionary to store the results
results = {}

# Iterate over the models
for model_name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mse)

    # Store the results in the dictionary
    results[model_name] = {
        "Mean Squared Error": mse,
        "R-squared": r2,
        "Mean Absolute Error": mae,
        "Root Mean Squared Error": rmse
    }

    # Use the trained model for future predictions
    new_tweet = ["New tweet about Bitcoin"]
    new_tweet_features = hstack([unigram_vectorizer.transform(new_tweet), bigram_vectorizer.transform(new_tweet), additional_features[:ler
    predicted_close = model.predict(new_tweet_features)

    results[model_name]["Predicted Close Price"] = predicted_close

    # Scatter plot
    plt.figure(figsize=(8, 6))
    plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title(f'Actual vs. Predicted - {model_name}')
    plt.show()

    # Residual plot
    plt.figure(figsize=(8, 6))
```

```
    residuals = y_test - y_pred
    plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Residual Plot - {model_name}')
    plt.show()

# Convert the results to a pandas DataFrame for tabular representation
results_df = pd.DataFrame.from_dict(results, orient="index")

# Print the results
print(results_df)

```

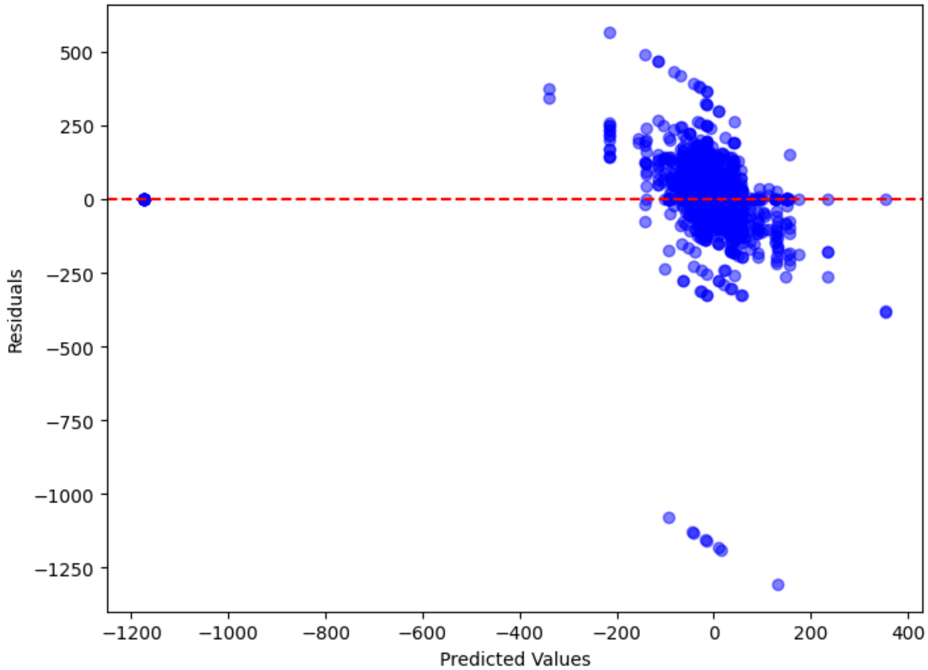

Actual vs. Predicted - Linear Regression
Predicted Values
Actual Values


Residual Plot - Linear Regression
Residuals
Predicted Values

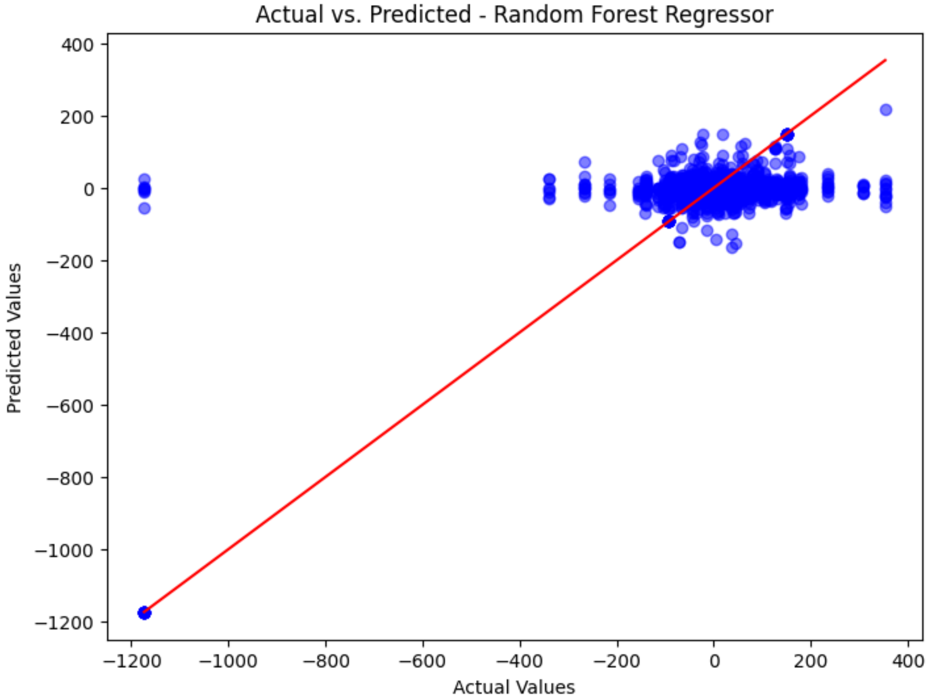

Actual vs. Predicted - Decision Tree Regressor
Predicted Values



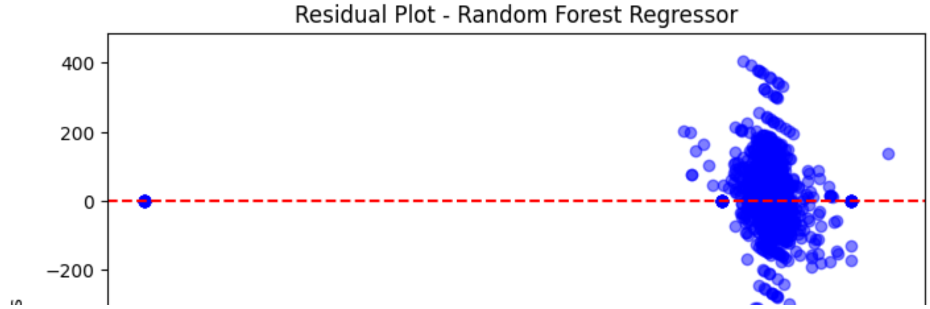
Residual Plot - Decision Tree Regressor

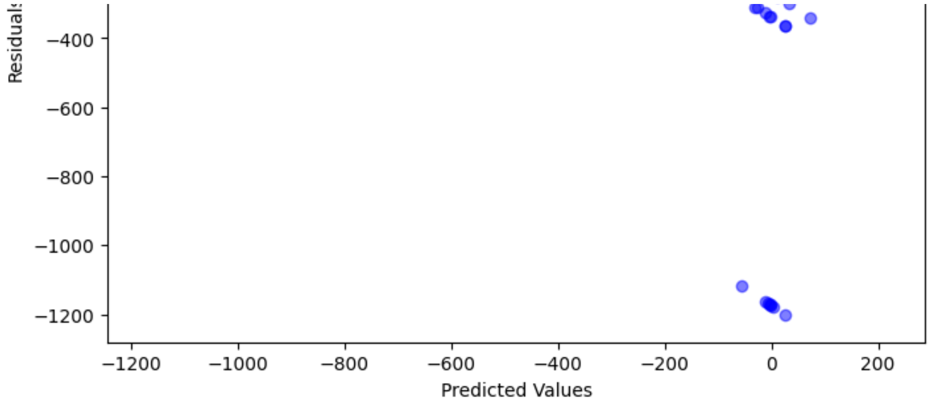


Actual vs. Predicted - Random Forest Regressor

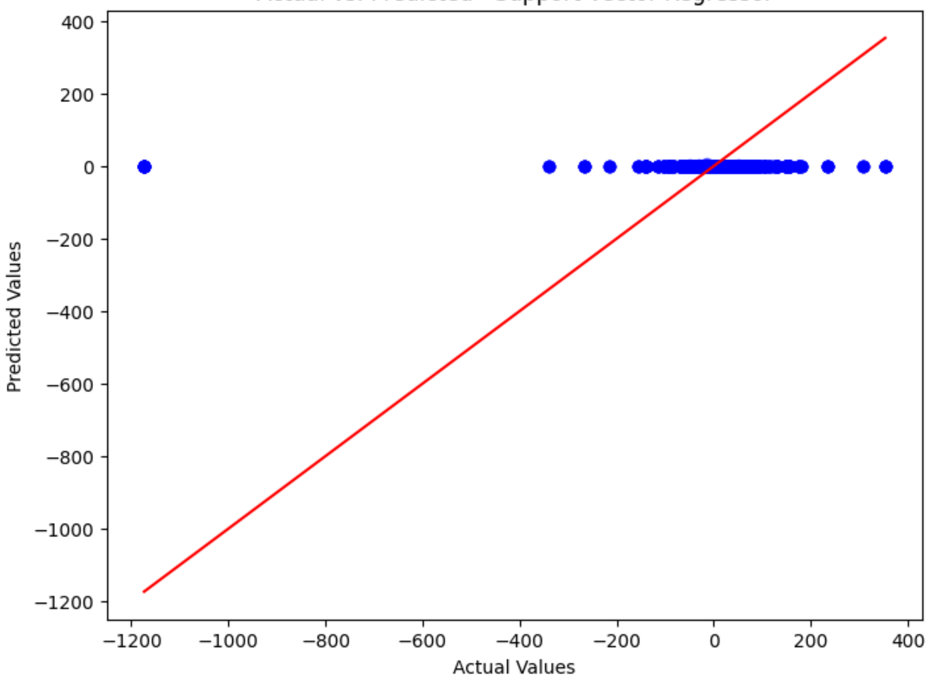


Residual Plot - Random Forest Regressor

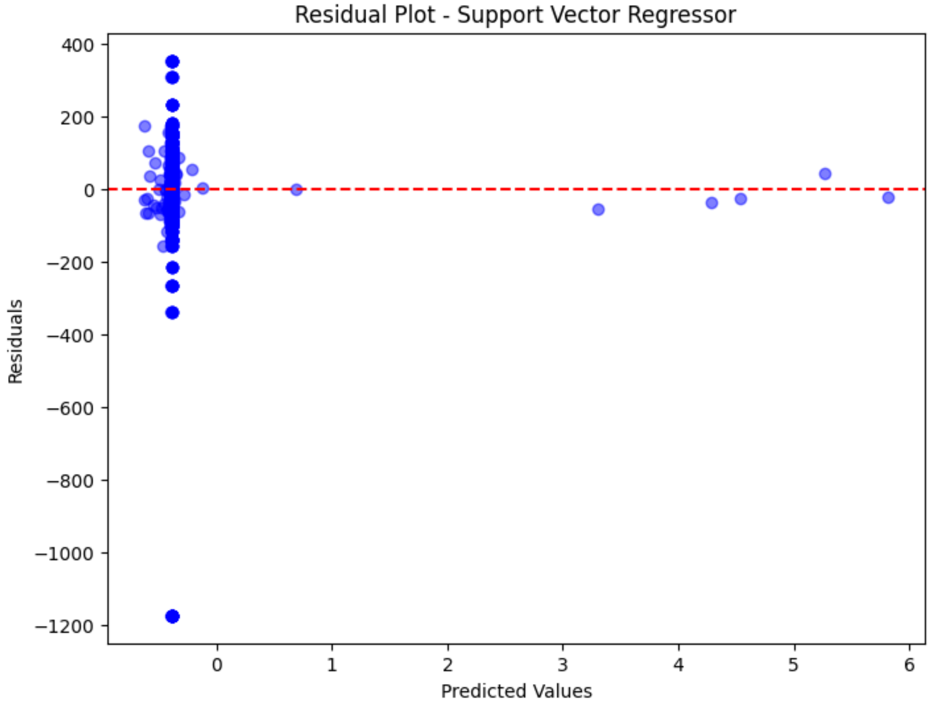




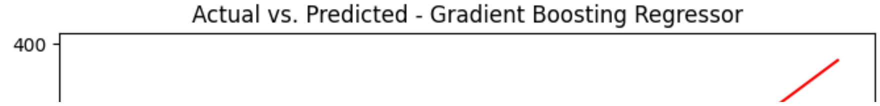
Actual vs. Predicted - Support Vector Regressor

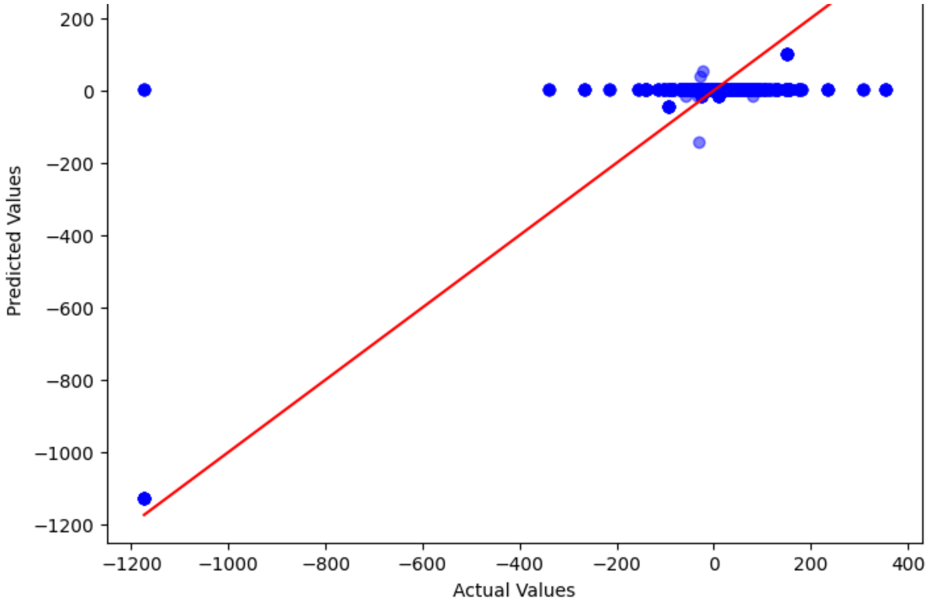


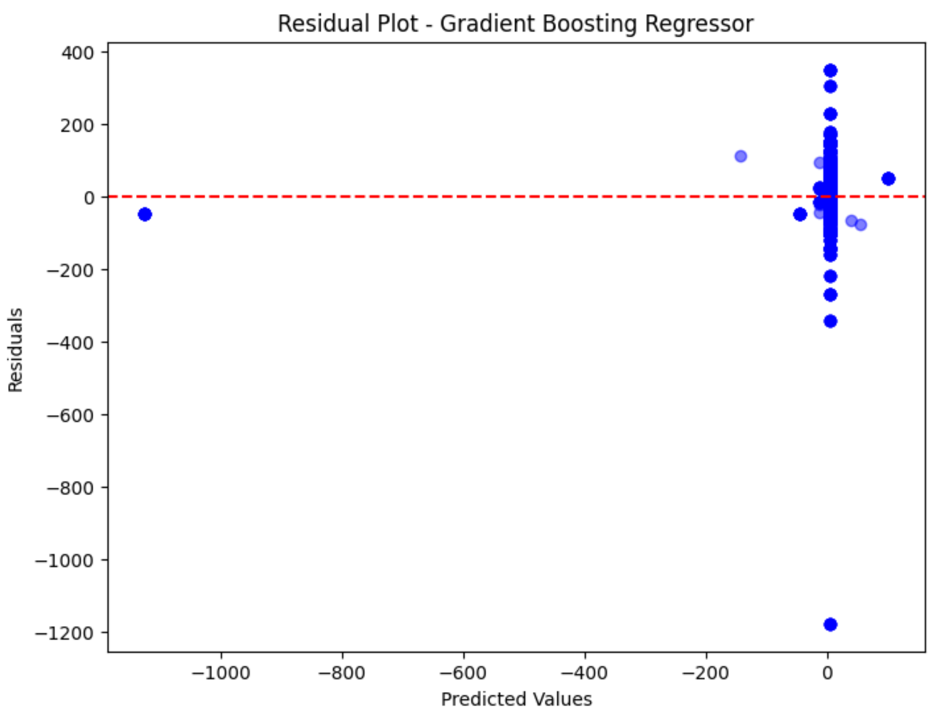
Residual Plot - Support Vector Regressor

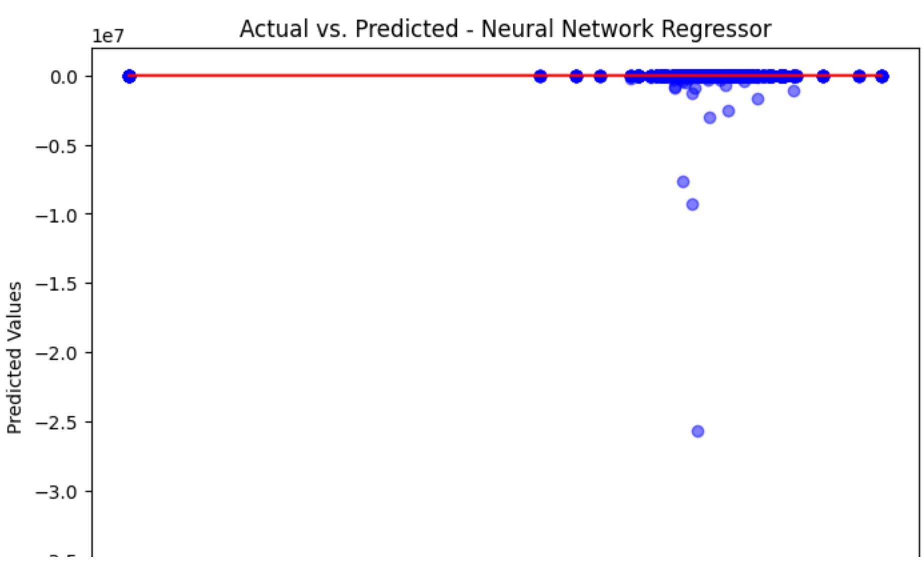


Actual vs. Predicted - Gradient Boosting Regressor










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

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

Actual Values

```

# Transpose the DataFrame
transposed_df = results_df.transpose()

# Print the transposed DataFrame
print(transposed_df)
```

Cross Validation for price_delta

```

```

Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score, precision_score, recall_score, f1_score

# Train the linear regression model
model = LinearRegression()

```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error


# Perform cross-validation
cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')

# Convert the negative mean squared error scores to positive
mse_scores = -cv_scores

# Calculate the mean and standard deviation of the MSE scores
mean_mse = np.mean(mse_scores)
std_mse = np.std(mse_scores)

# Print the mean and standard deviation of the MSE scores
print("Mean MSE:", mean_mse)
print("Std MSE:", std_mse)

```

```
Mean MSE: 26602.32728304749
Std MSE: 34442.19984498466
```

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Train the decision tree regressor model
model = DecisionTreeRegressor()


```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error


# Perform cross-validation
cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')

# Convert the negative mean squared error scores to positive
mse_scores = -cv_scores

```

```
# Calculate the mean and standard deviation of the MSE scores
mean_mse = np.mean(mse_scores)
std_mse = np.std(mse_scores)

# Print the mean and standard deviation of the MSE scores
print("Mean MSE:", mean_mse)
print("Std MSE:", std_mse)

```

```
Mean MSE: 29497.64889650693
Std MSE: 34444.122541768076
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor()
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error


# Perform cross-validation
cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')

# Convert the negative mean squared error scores to positive
mse_scores = -cv_scores

# Calculate the mean and standard deviation of the MSE scores
mean_mse = np.mean(mse_scores)
std_mse = np.std(mse_scores)

# Print the mean and standard deviation of the MSE scores
print("Mean MSE:", mean_mse)
print("Std MSE:", std_mse)

```

```
Mean MSE: 26945.52137651674
Std MSE: 34605.309807612946
```

Support Vector Regressor

```
from sklearn.svm import SVR

model = SVR()
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error


# Perform cross-validation
cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')

# Convert the negative mean squared error scores to positive
mse_scores = -cv_scores

# Calculate the mean and standard deviation of the MSE scores
mean_mse = np.mean(mse_scores)
std_mse = np.std(mse_scores)

# Print the mean and standard deviation of the MSE scores
print("Mean MSE:", mean_mse)
print("Std MSE:", std_mse)

```

```
Mean MSE: 26258.128645849814
Std MSE: 34451.38200920118
```

Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor

model = GradientBoostingRegressor()
```

```

```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error


# Perform cross-validation
cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')

# Convert the negative mean squared error scores to positive
mse_scores = -cv_scores

# Calculate the mean and standard deviation of the MSE scores
mean_mse = np.mean(mse_scores)
std_mse = np.std(mse_scores)

# Print the mean and standard deviation of the MSE scores
print("Mean MSE:", mean_mse)
print("Std MSE:", std_mse)

```

```
Mean MSE: 26563.170954889654
Std MSE: 34572.294054287326
```

Neural Network Regressor (MLP)

```
from sklearn.neural_network import MLPRegressor

model = MLPRegressor()
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error


# Perform cross-validation
cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')

# Convert the negative mean squared error scores to positive
mse_scores = -cv_scores

# Calculate the mean and standard deviation of the MSE scores
mean_mse = np.mean(mse_scores)
std_mse = np.std(mse_scores)

# Print the mean and standard deviation of the MSE scores
print("Mean MSE:", mean_mse)
print("Std MSE:", std_mse)

```

```
Mean MSE: 182165942647766.62
Std MSE: 344463338514648.1
```

Hyperparameter Tuning for Price_delta

```
#Hyperparameter Tuning
```

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, classification_report
from scipy.sparse import hstack

# Feature Extraction: Unigrams
unigram_vectorizer = CountVectorizer(ngram_range=(1, 1))
unigram_features = unigram_vectorizer.fit_transform(tweets_df['text'])

# Feature Extraction: Bigrams
bigram_vectorizer = CountVectorizer(ngram_range=(2, 2))
bigram_features = bigram_vectorizer.fit_transform(tweets_df['text'])

# Combining Features
combined_features = hstack([unigram_features, bigram_features])

# Additional Input Features
additional_features = tweets_df[['compound', 'score', 'polarity', 'subjectivity', 'sentiment_score']].values

# Concatenate Additional Features with Combined Features
X = hstack([combined_features, additional_features])

# Target Variable
y = tweets_df['close']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



```

```
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Define the models
models = {
    "Decision Tree Regressor": DecisionTreeRegressor(),
    "Random Forest Regressor": RandomForestRegressor(),
    "Support Vector Regressor": SVR(),
    "Gradient Boosting Regressor": GradientBoostingRegressor(),
    "Neural Network Regressor": MLPRegressor(),
    "Linear Regression": LinearRegression()
}

# Define the parameter grids for hyperparameter tuning
param_grids = {
    "Decision Tree Regressor": {
        "max_depth": [3, 5, 7],
        "min_samples_split": [2, 5, 10],
        "min_samples_leaf": [1, 3, 5]
    },
    "Random Forest Regressor": {
        "n_estimators": [50, 100, 200],
        "max_depth": [3, 5, 7],
        "min_samples_split": [2, 5, 10],
        "min_samples_leaf": [1, 3, 5]
    },
    "Support Vector Regressor": {
        "C": [0.1, 1, 10],
        "epsilon": [0.1, 0.01, 0.001]
    },
    "Gradient Boosting Regressor": {
        "n_estimators": [50, 100, 200],
        "learning_rate": [0.1, 0.01, 0.001],
        "max_depth": [3, 5, 7]
    },
    "Neural Network Regressor": {
        "hidden_layer_sizes": [(100,), (100, 50), (200, 100)],
        "alpha": [0.1, 0.01, 0.001]
    },
    "Linear Regression": {}
}

# Initialize an empty dictionary to store the results
```

```
results = {}

# Iterate over the models
for model_name, model in models.items():
    print("Model:", model_name)

    # Perform feature selection
    feature_selector = SelectKBest()
    X_selected = feature_selector.fit_transform(X, y)

    # Perform grid search cross-validation
    param_grid = param_grids[model_name]
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error')
    grid_search.fit(X_selected, y)

    # Get the best model and its corresponding hyperparameters
    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_

    # Train the best model on the entire dataset
    best_model.fit(X_selected, y)

    # Make predictions on the test set
    X_test_selected = feature_selector.transform(X_test)
    y_pred = best_model.predict(X_test_selected)

    # Evaluate the best model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mse)

    # Store the results in the dictionary
    results[model_name] = {
        "Best Model": best_model,
        "Best Parameters": best_params,
        "Mean Squared Error": mse,
        "R-squared": r2,
        "Mean Absolute Error": mae,
        "Root Mean Squared Error": rmse
    }

    # Print the results
    print("Best Model:", best_model)
    print("Best Parameters:", best_params)
    print("Mean Squared Error:", mse)
    print("R-squared:", r2)
    print("Mean Absolute Error:", mae)
    print("Root Mean Squared Error:", rmse)
    print()

```

```
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 Optimize
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimize
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimize
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimize
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimize
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimize
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimize
  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