Datasets

```
#Price
```

```
import pandas as pd

# URL to the raw CSV file
url = 'https://raw.githubusercontent.com/Amarpreet3/CIND-820-CAPSTONE/main/Sentimental%20Analysis/BitcoinPricePreprocessed.csv'

# Read the CSV file from the URL
crypto_usd = pd.read_csv(url)

# Display the first few rows of the data
print(crypto_usd.head())


```

```
                  time     close      high       low      open  volumefrom  \
0  2023-02-19 13:00:00  24682.03  24715.82  24682.03  24707.39      903.97
1  2023-02-19 14:00:00  24765.79  24792.85  24679.21  24682.03     1220.29
2  2023-02-19 15:00:00  24928.21  25022.49  24751.96  24765.79     5074.50
3  2023-02-19 16:00:00  24786.44  25175.28  24704.53  24928.21     7094.72
4  2023-02-19 17:00:00  24364.95  24806.64  24346.17  24786.44     6896.84

       volumeto        Date      Time        volume     marketcap  price_delta
0  2.233594e+07  2023-02-19  13:00:00  2.233504e+07  5.512964e+11          NaN
1  3.020300e+07  2023-02-19  14:00:00  3.020178e+07  7.480012e+11        83.76
2  1.263085e+08  2023-02-19  15:00:00  1.263034e+08  3.148644e+12       162.42
3  1.770671e+08  2023-02-19  16:00:00  1.770600e+08  4.388863e+12      -141.77
4  1.693379e+08  2023-02-19  17:00:00  1.693310e+08  4.125910e+12      -421.49
```

```
import pandas as pd

file_urls = [
    'https://github.com/Amarpreet3/CIND-820-CAPSTONE/raw/main/Sentimental%20Analysis/BitcoinTweetsPreprocessed_1.csv',
    'https://github.com/Amarpreet3/CIND-820-CAPSTONE/raw/main/Sentimental%20Analysis/BitcoinTweetsPreprocessed_2.csv',
    'https://github.com/Amarpreet3/CIND-820-CAPSTONE/raw/main/Sentimental%20Analysis/BitcoinTweetsPreprocessed_3.csv',
    'https://github.com/Amarpreet3/CIND-820-CAPSTONE/raw/main/Sentimental%20Analysis/BitcoinTweetsPreprocessed_4.csv',
    'https://github.com/Amarpreet3/CIND-820-CAPSTONE/raw/main/Sentimental%20Analysis/BitcoinTweetsPreprocessed_5.csv',
    'https://github.com/Amarpreet3/CIND-820-CAPSTONE/raw/main/Sentimental%20Analysis/BitcoinTweetsPreprocessed_6.csv'
]

dfs = []

for url in file_urls:
    # Read the CSV file
    df = pd.read_csv(url)

    # Append the DataFrame to the list
    dfs.append(df)

# Combine all DataFrames into a single DataFrame
combined_df = pd.concat(dfs)

# Display the first few rows of the combined DataFrame
print(combined_df.head())

```

```
            user_name  user_location  \
0                 Irk  Vancouver, WA
1         Xiang Zhang            NaN
2              Rhizoo            NaN
3        Hari Marquez  Las Vegas, NV
4  Bitcoin Candle Bot         Brazil

                                    user_description         user_created  \
0  Irk started investing in the stock market in 1...  2018-08-11 03:17:00
1  Professional Software Engineer ð⍰⍰»ð⍰⍰⍰Crypto ...  2011-01-11 01:37:00
2  researcher. local maxima dunningâ⍰⍰kruger spec...  2019-04-03 18:09:00
3  Donâ⍰⍰t trust, verify. #Bitcoin | El Salvador ...  2014-01-17 23:04:00
4  Robot that posts the closure of the bitcoin da...  2021-01-06 01:36:00

   user_followers  user_friends  user_favourites  user_verified  \
0           116.0           8.0           4580.0          False
```

```
1          42.0         22.0            5.0         False
2         778.0        627.0        32005.0         False
3         222.0        521.0        13052.0         False
4          40.0          4.0            1.0         False

                  date                                              text  \
0  2023-02-25 23:59:00  bitcoin btc rest crypto ye bitcoin cryptocurr ...
1  2023-02-25 23:59:00  retriev invest fund current ongo tidexcoin kic...
2  2023-02-25 23:59:00  bull save monthli thread today good shit bitco...
3  2023-02-25 23:59:00         el salvador shape futur bitcoin membvk32cn
4  2023-02-25 23:59:00  candl day 25022023 close open 2319406 high 232...

                                          hashtags              source  \
0           ['Bitcoin', 'crypto', 'NeedsMoreCrash']     Twitter Web App
1  ['Tidexcoin', 'Kicurrency', 'LMY', 'GMK', 'SYR...  Twitter for iPhone
2                                      ['bitcoin']     Twitter Web App
3                                      ['Bitcoin']     Twitter Web App
4               ['Bitcoin', 'Candle', 'BearMarket']  Bitcoin Candle Bot

   is_retweet  compound         score sentiment_level  polarity  subjectivity
0         0.0   -0.4019 -2.154092e+05        Negative  0.000000      0.000000
1         0.0    0.0000  0.000000e+00         Neutral  0.000000      0.400000
2         0.0    0.3612  9.005682e+06        Positive  0.250000      0.700000
3         0.0    0.0000  0.000000e+00         Neutral  0.000000      0.000000
4         0.0   -0.2732 -2.240240e+01        Negative  0.053333      0.446667
```

```
tweets = combined_df.copy()
```

```
tweets.head()
```

	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	
2	Rhizoo	NaN	researcher. local maxima dunningâ□□kruger spec...	2019-04-03 18:09:00	778.0	627.0	
3	Hari Marquez	Las Vegas, NV	Donâ□□t trust, verify. #Bitcoin | El Salvador ...	2014-01-17 23:04:00	222.0	521.0	
4	Bitcoin Candle Bot	Brazil	Robot that posts the closure of the bitcoin da...	2021-01-06 01:36:00	40.0	4.0	

```
print(tweets.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'],
      dtype='object')
```

```
import pandas as pd


# Check the shape of the dataset
print("Shape of the dataset:", tweets.shape)

# Check the size of the dataset
```

```
print("Size of the dataset (number of elements):",tweets.size)

```

```
Shape of the dataset: (167652, 18)
Size of the dataset (number of elements): 3017736
```

```
import pandas as pd
import os


# Check the shape of the data
print("Shape of the data:", tweets.shape)


```

```
Shape of the data: (167652, 18)
```

```
label_counts = tweets['sentiment_level'].value_counts()
print(label_counts)
```

```
Neutral             93169
Positive            35921
Extreme Positive    17343
Negative            15903
Extreme Negative     5316
Name: sentiment_level, dtype: int64
```

Combining both datasets

```
# Create a copy of the bitcoin price DataFrame
crypto_usd.head(2)
```

	time	close	high	low	open	volumefrom	volumeto	Date	Time	
0	2023-02-19 13:00:00	24682.03	24715.82	24682.03	24707.39	903.97	22335943.28	2023-02-19	13:00:00	22

```
print(crypto_usd.columns)
```

```
Index(['time', 'close', 'high', 'low', 'open', 'volumefrom', 'volumeto',
       'Date', 'Time', 'volume', 'marketcap', 'price_delta'],
      dtype='object')
```

```
df_score = tweets.copy()
```

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
```

```python
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)



```

```python
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]]
```

```python
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
import time
```

Linear Regression

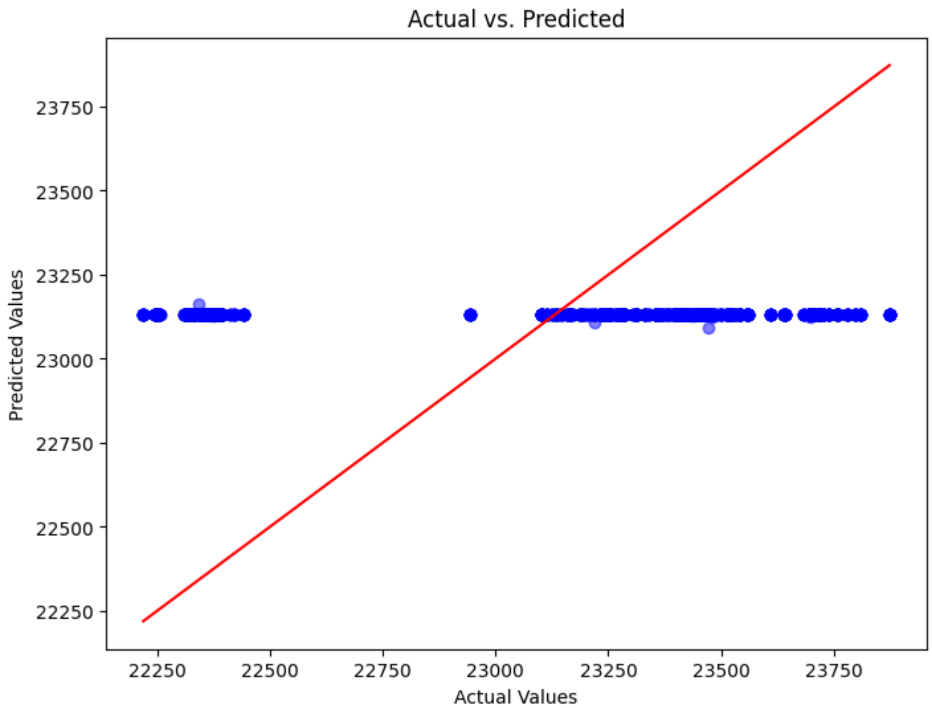
```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score, precision_score, recall_score, f1_score
start_time = time.time()
# Train the linear regression model
model = LinearRegression()

```

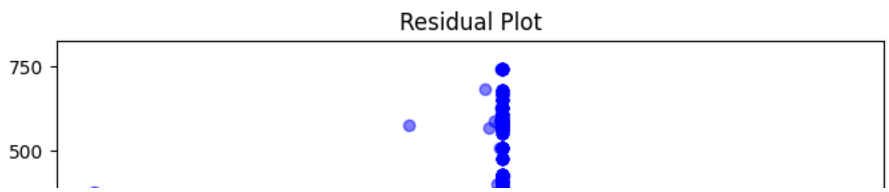
```python

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)
end_time = time.time()
# Calculate the execution time
execution_time = end_time - start_time

# Print the execution time
print(f"Execution time: {execution_time} seconds")
```

```
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]
Execution time: 0.04199504852294922 seconds
```

```
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

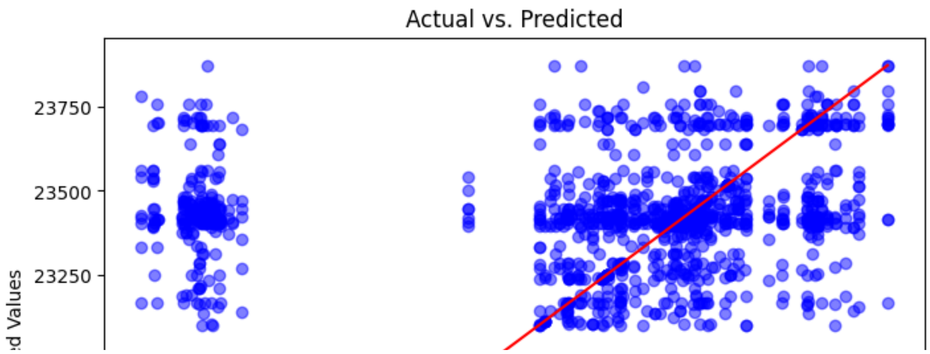
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
start_time = time.time()
# Train the decision tree regressor model
model = DecisionTreeRegressor()


```

```
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)
end_time = time.time()
# Calculate the execution time
execution_time = end_time - start_time

```

```
# Print the execution time
print(f"Execution time: {execution_time} seconds")
```

```
Model: DecisionTreeRegressor()
Mean Squared Error: 330171.6101172941
R-squared: -0.25269425158975833
Mean Absolute Error: 372.79546518987394
Root Mean Squared Error: 574.6056126747233
Predicted Close Price: [23457.06]
Execution time: 7.175631284713745 seconds
```

```
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

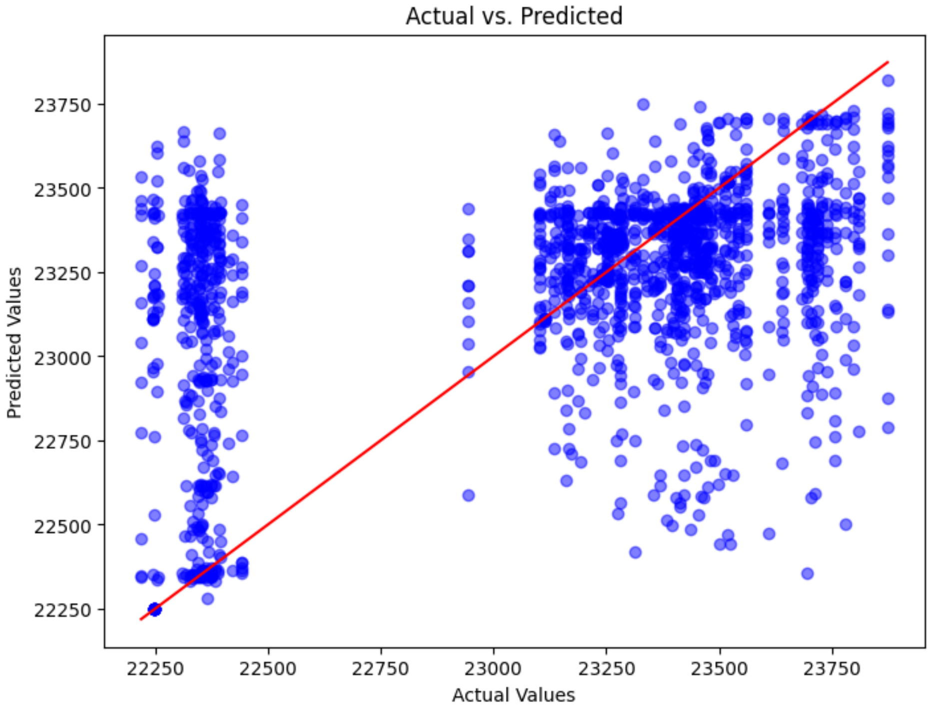
```
from sklearn.ensemble import RandomForestRegressor
start_time = time.time()
# Rest of the code is the same as above
model = RandomForestRegressor()


```

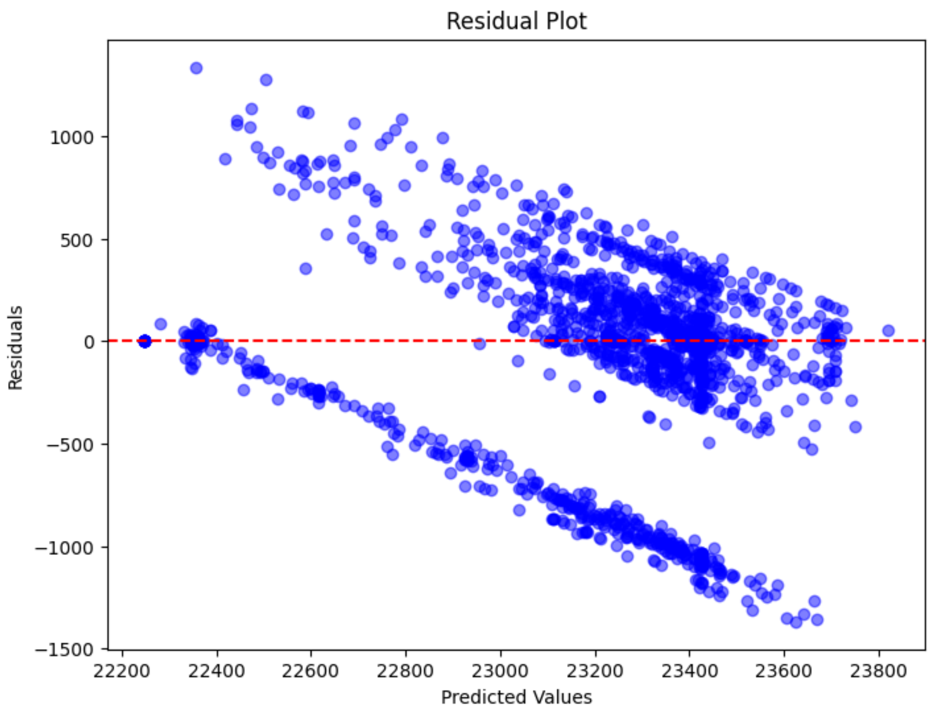
```
model.fit(X_train, y_train)

# 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)
end_time = time.time()
# Calculate the execution time
execution_time = end_time - start_time

# Print the execution time
print(f"Execution time: {execution_time} seconds")
```

```
Model: RandomForestRegressor()
Mean Squared Error: 229402.74962978673
R-squared: 0.12962987442188278
Mean Absolute Error: 332.16736972573915
Root Mean Squared Error: 478.96007101822875
Predicted Close Price: [22660.31346667]
Execution time: 433.4518885612488 seconds
```

```
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()
```





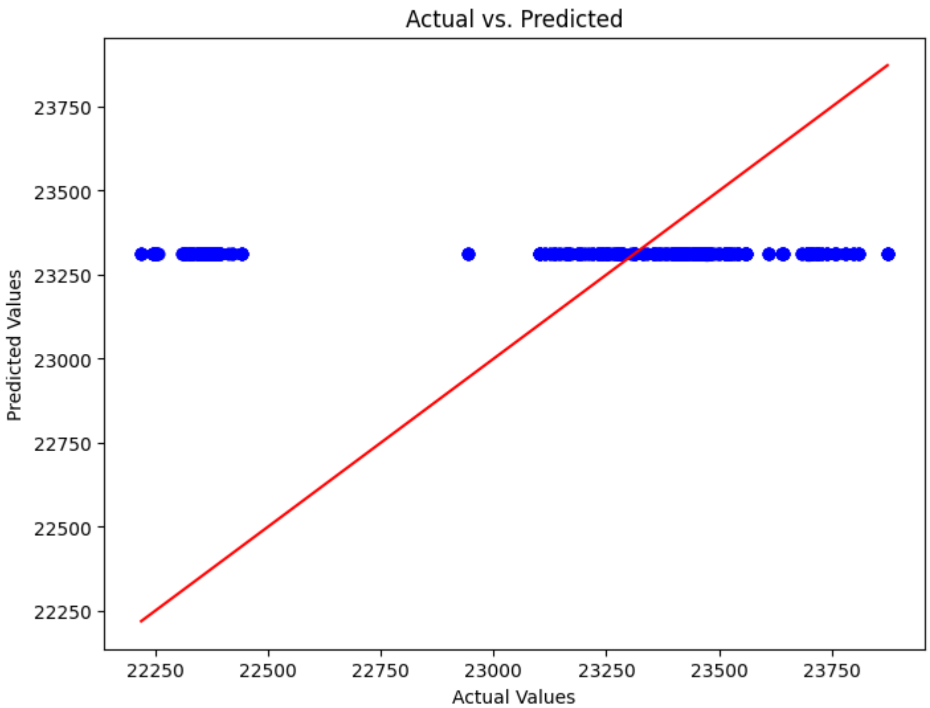
Support Vector Regressor

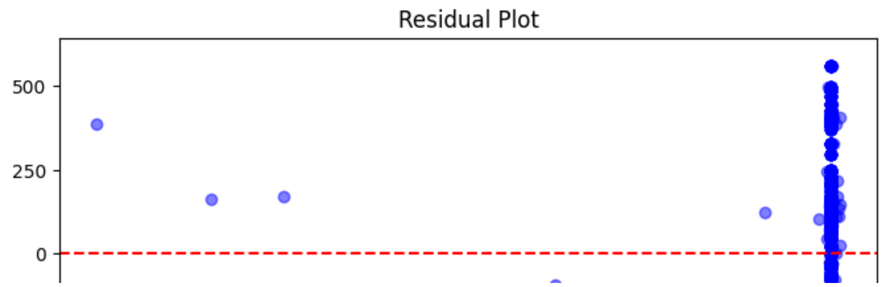
```
from sklearn.svm import SVR
start_time = time.time()
model = SVR()
```

```
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)
end_time = time.time()
# Calculate the execution time
execution_time = end_time - start_time

# Print the execution time
print(f"Execution time: {execution_time} seconds")
```

```
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]
Execution time: 11.594075202941895 seconds
```

```
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

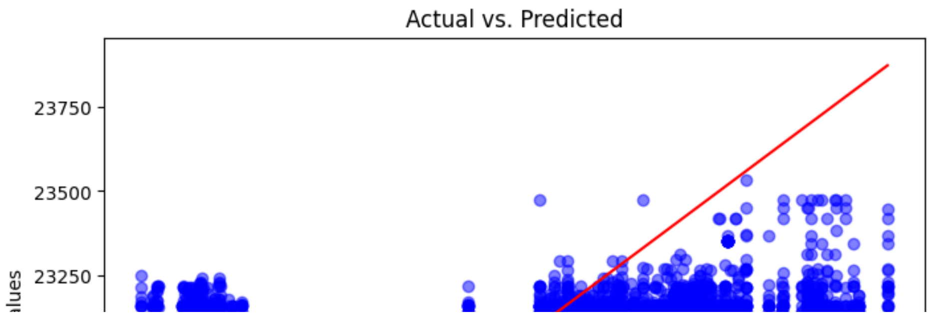
```
from sklearn.ensemble import GradientBoostingRegressor
start_time = time.time()
model = GradientBoostingRegressor()
```

```
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)
end_time = time.time()
# Calculate the execution time
execution_time = end_time - start_time

# Print the execution time
print(f"Execution time: {execution_time} seconds")
```

```
print(f'Execution time: {execution_time} seconds')
```

```
Model: GradientBoostingRegressor()
Mean Squared Error: 218755.55046442378
R-squared: 0.17002609499712606
Mean Absolute Error: 378.03401143678497
Root Mean Squared Error: 467.71310700516375
Predicted Close Price: [23144.45192473]
Execution time: 40.16045880317688 seconds
```

```
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)

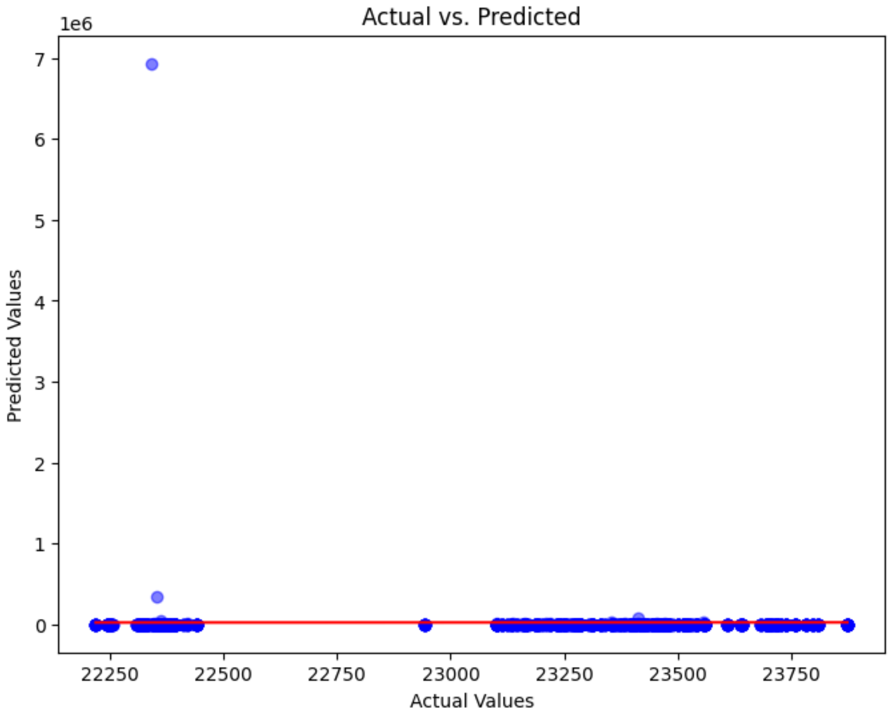
```
from sklearn.neural_network import MLPRegressor
start_time = time.time()
model = MLPRegressor()
```

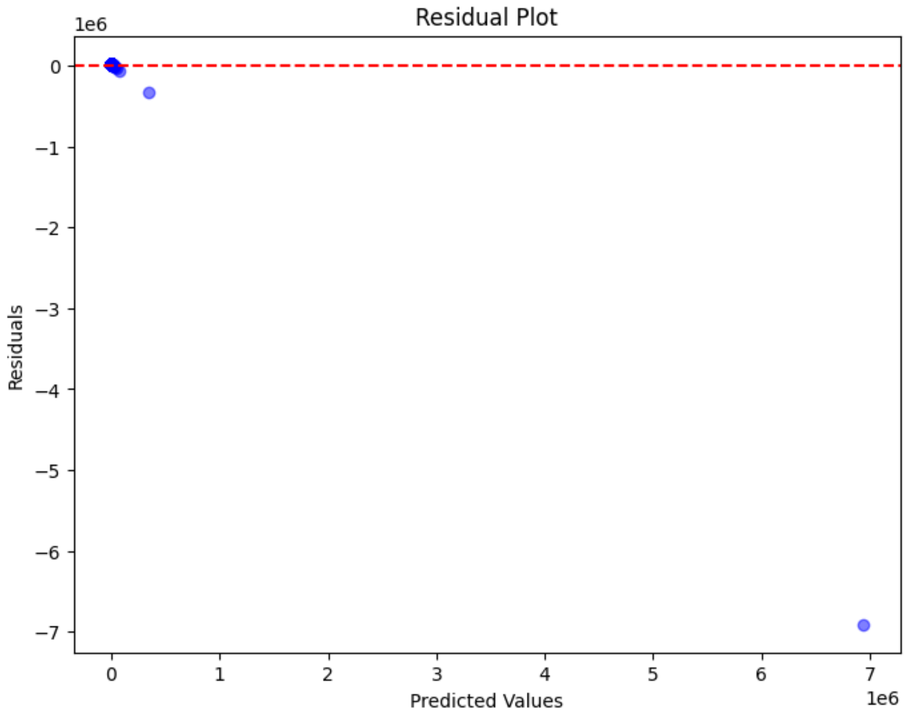
```
model.fit(X_train, y_train)

# 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)
end_time = time.time()
# Calculate the execution time
execution_time = end_time - start_time

# Print the execution time
print(f"Execution time: {execution_time} seconds")
```

```
Model: MLPRegressor()
Mean Squared Error: 312875279304.78375
R-squared: -1187069.7590831518
Mean Absolute Error: 51811.649418078436
Root Mean Squared Error: 559352.5536768235
Predicted Close Price: [2.23164465]
Execution time: 144.90826535224915 seconds
```

```
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

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: 369821.0979640644
Std MSE: 306106.77056563296
```

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: 465009.6395099891
Std MSE: 335780.0275937437
```

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: 358518.1748665846
Std MSE: 372697.4355903579
```

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: 347812.1745007644
Std MSE: 401289.31804972445
```

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: 348887.6854683334
Std MSE: 310402.66705840186
```

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: 31205126756566.926
Std MSE: 47041070509094.12
```

Hyperparameter Tuning for Price

```
#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)
```

```python
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()

```

```
Model: Decision Tree Regressor
Best Model: DecisionTreeRegressor(max_depth=7, min_samples_leaf=3, min_samples_split=5)
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 3, 'min_samples_split': 5}
Mean Squared Error: 249886.37207620614
R-squared: 0.05191357385549278
Mean Absolute Error: 413.05794247618445
Root Mean Squared Error: 499.8863591619661

Model: Random Forest Regressor
Best Model: RandomForestRegressor(max_depth=7, min_samples_split=5, n_estimators=50)
Best Parameters: {'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 50}
Mean Squared Error: 249886.91762665412
R-squared: 0.05191150399882072
Mean Absolute Error: 412.8730317998603
Root Mean Squared Error: 499.886904836138

Model: Support Vector Regressor
Best Model: SVR(C=0.1, epsilon=0.001)
Best Parameters: {'C': 0.1, 'epsilon': 0.001}
Mean Squared Error: 296035.0250691949
R-squared: -0.12317765310490625
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}
Mean Squared Error: 249884.83969784752
R-squared: 0.05191938780648764
Mean Absolute Error: 413.1128964010035
Root Mean Squared Error: 499.88482643289694

Model: Neural Network Regressor
/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(
/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(
```

```
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: 31205126756566.926
Std MSE: 47041070509094.12
```

All Models Together for comparison with price [close]

```
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 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

# 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

# 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)

```

```
                             Mean Squared Error     R-squared  \
Linear Regression                  2.636422e+05 -2.768400e-04
Decision Tree Regressor            3.345694e+05 -2.693799e-01
Random Forest Regressor            2.315119e+05  1.216275e-01
Support Vector Regressor           2.972390e+05 -1.277455e-01
Gradient Boosting Regressor        2.188733e+05  1.695795e-01
Neural Network Regressor           4.803038e+11 -1.822305e+06

                             Mean Absolute Error  Root Mean Squared Error  \
Linear Regression                     434.524224               513.460960
Decision Tree Regressor               374.074494               578.419766
Random Forest Regressor               334.207529               481.156859
Support Vector Regressor              399.947267               545.196259
Gradient Boosting Regressor           378.111052               467.838936
Neural Network Regressor            58711.619235            693039.555859

                            Predicted Close Price
Linear Regression            [23131.47449878233]
Decision Tree Regressor               [23447.51]
Random Forest Regressor     [22580.549866666664]
Support Vector Regressor    [23312.329858272555]
Gradient Boosting Regressor  [23144.45192472882]
Neural Network Regressor     [4.349573192223386]
```

```

# Transpose the DataFrame
transposed_df = results_df.transpose()

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

```
                         Linear Regression Decision Tree Regressor  \
Mean Squared Error            263642.157208          334569.425348
R-squared                         -0.000277               -0.26938
Mean Absolute Error              434.524224             374.074494
Root Mean Squared Error           513.46096             578.419766
Predicted Close Price    [23131.47449878233]             [23447.51]

                      Random Forest Regressor Support Vector Regressor  \
Mean Squared Error              231511.922529            297238.961051
R-squared                            0.121628                -0.127745
Mean Absolute Error                334.207529               399.947267
Root Mean Squared Error            481.156859               545.196259
Predicted Close Price      [22580.549866666664]     [23312.329858272555]

                      Gradient Boosting Regressor Neural Network Regressor
Mean Squared Error                  218873.270426      480303825985.190186
R-squared                                0.169579         -1822305.410943
Mean Absolute Error                    378.111052             58711.619235
Root Mean Squared Error                467.838936            693039.555859
Predicted Close Price          [23144.45192472882]      [4.349573192223386]
```

```
#Linear Regression and Decision Tree Regressor have low R-squared values and high mean squared error, indicating poor fit and high predict
#Random Forest Regressor and Gradient Boosting Regressor show relatively better performance with higher R-squared values, lower mean squar
#The Neural Network Regressor seems to have highly inaccurate predictions, as indicated by the extremely high mean squared error and mean
```

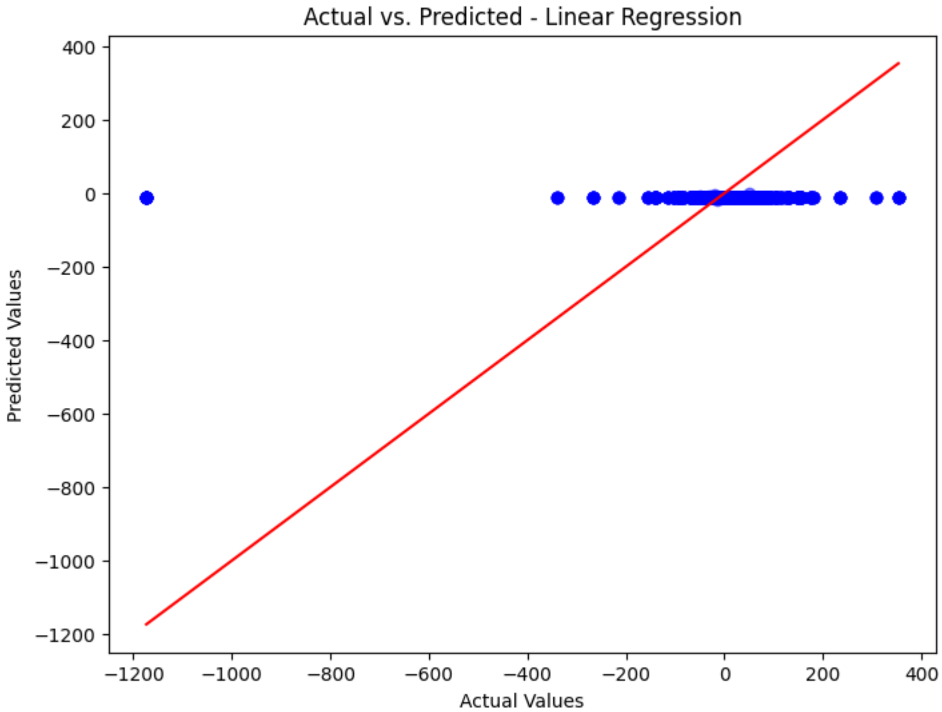
```
#Based on the provided evaluation metrics, the Random Forest Regressor and Gradient Boosting Regressor perform relatively better compared
```

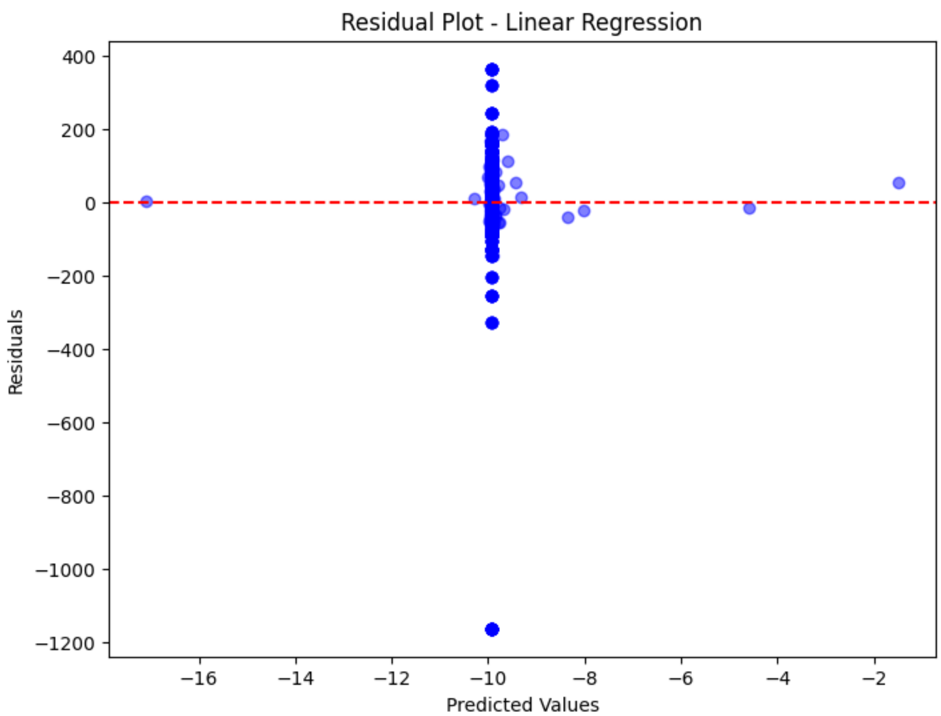
All Models Together for comparison with price_delta

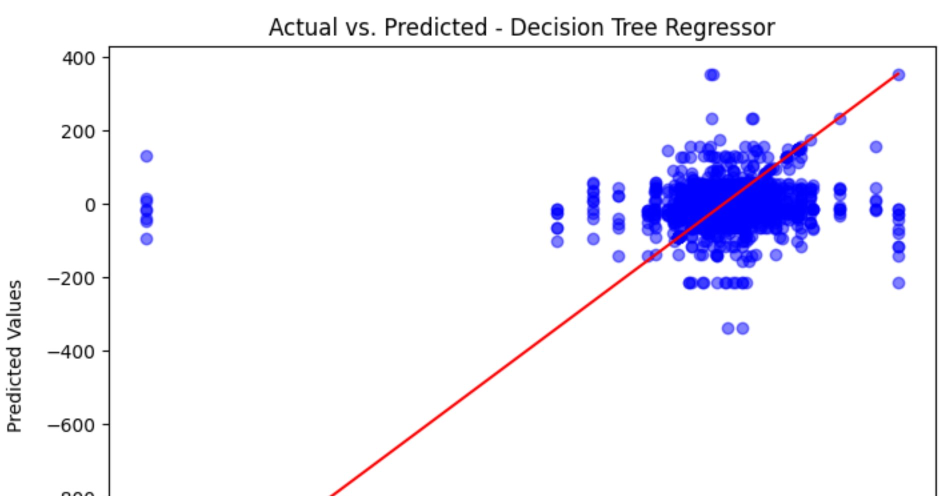
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, classification_report
from scipy.sparse import hstack

# 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)



```

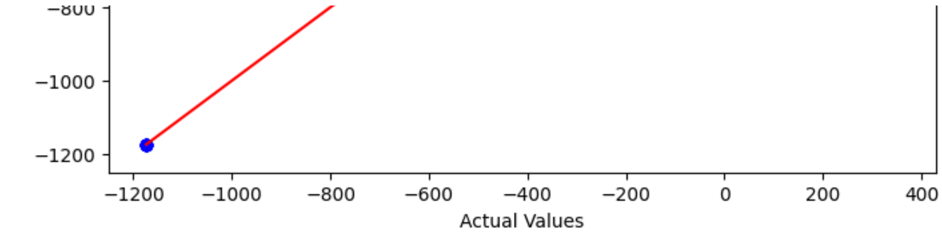
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
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
```

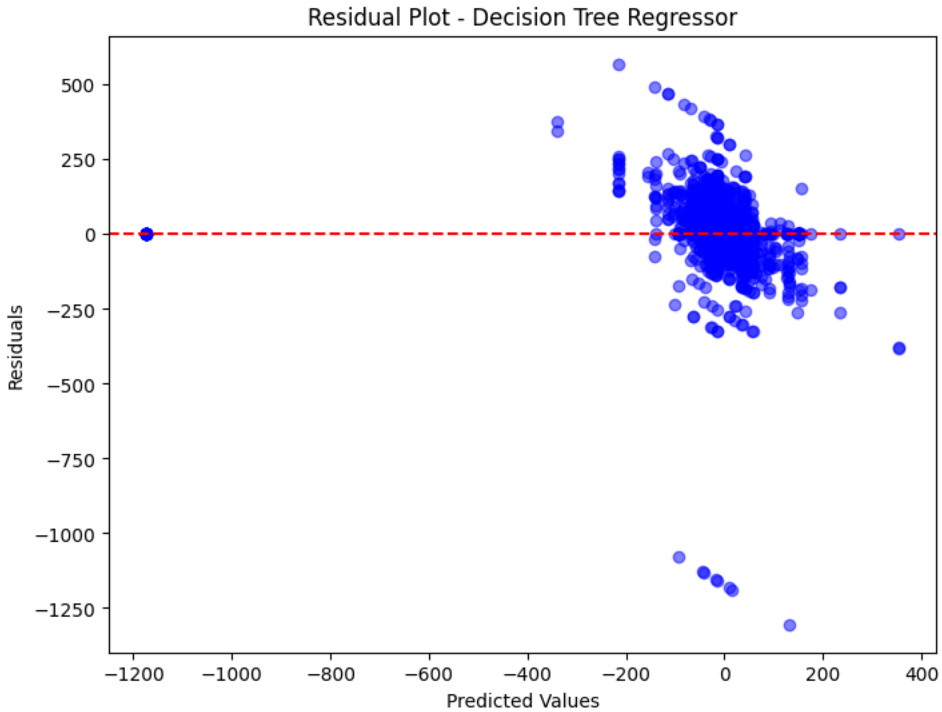
```

# 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)

```

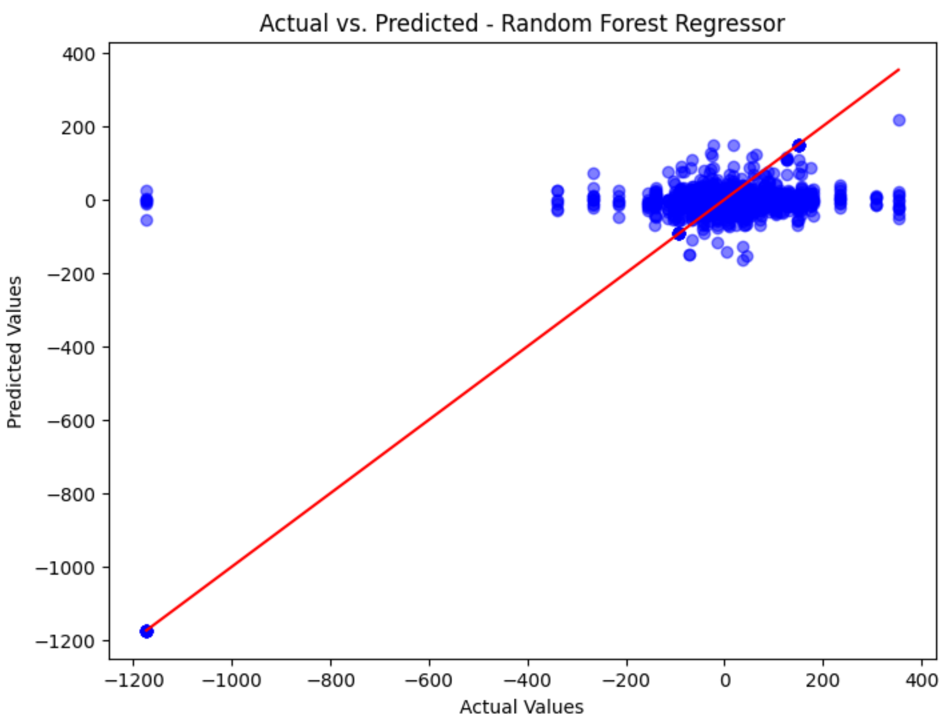

Actual vs. Predicted - Linear Regression
Predicted Values
Actual Values
400
200
0
−200
−400
−600
−800
−1000
−1200
−1200
−1000
−800
−600
−400
−200
0
200
400

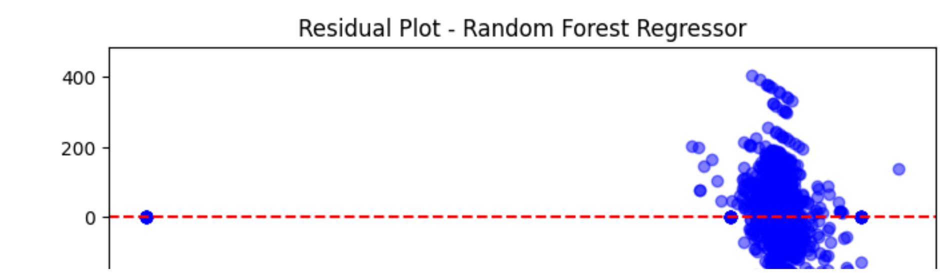

Residual Plot - Linear Regression
Residuals
Predicted Values
400
200
0
−200
−400
−600
−800
−1000
−1200
−16
−14
−12
−10
−8
−6
−4
−2

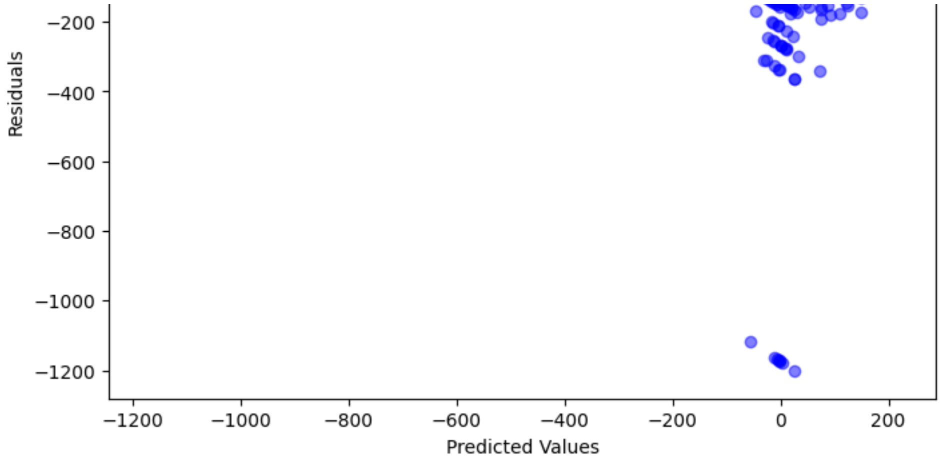

Actual vs. Predicted - Decision Tree Regressor
Predicted Values
400
200
0
−200
−400
−600

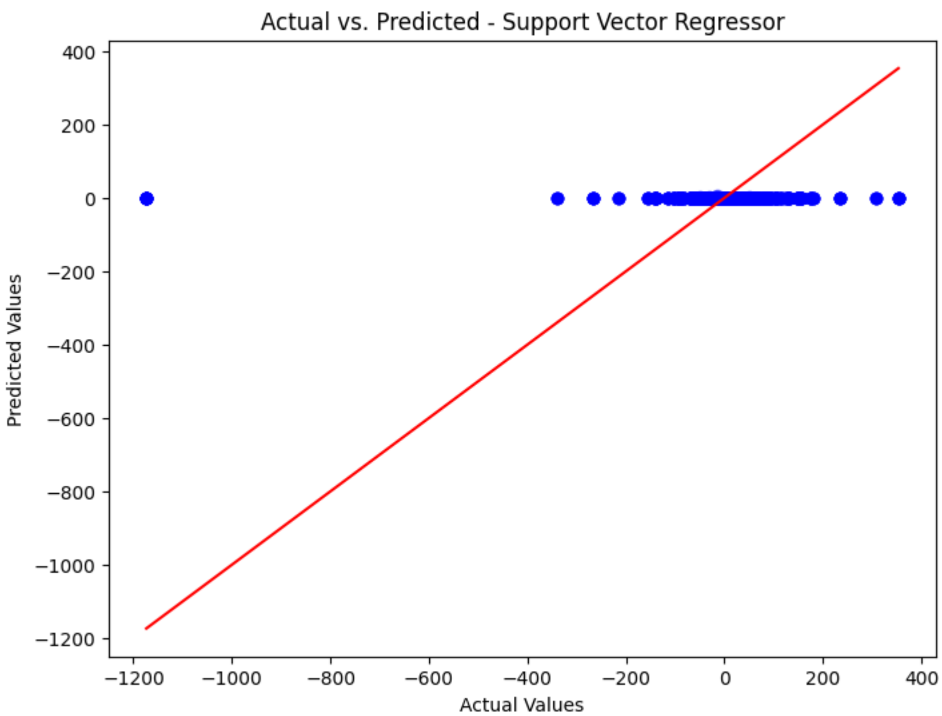

−800
−1000
−1200
−1200 −1000 −800 −600 −400 −200 0 200 400
Actual Values

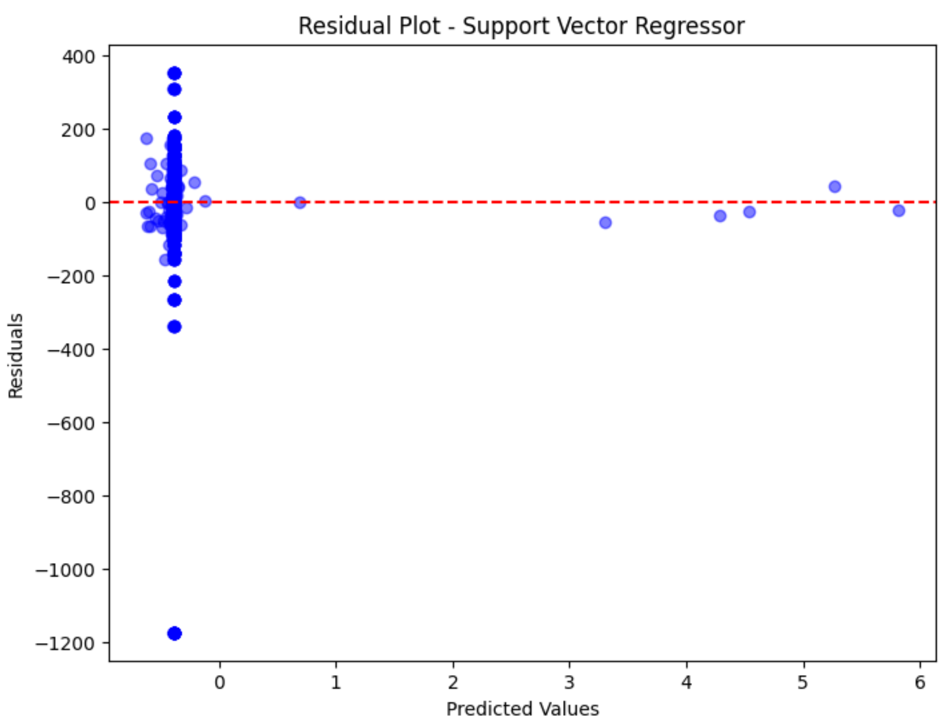

Residual Plot - Decision Tree Regressor
Residuals
500
250
0
−250
−500
−750
−1000
−1250
−1200 −1000 −800 −600 −400 −200 0 200 400
Predicted Values


Actual vs. Predicted - Random Forest Regressor
Predicted Values
400
200
0
−200
−400
−600
−800
−1000
−1200
−1200 −1000 −800 −600 −400 −200 0 200 400
Actual Values

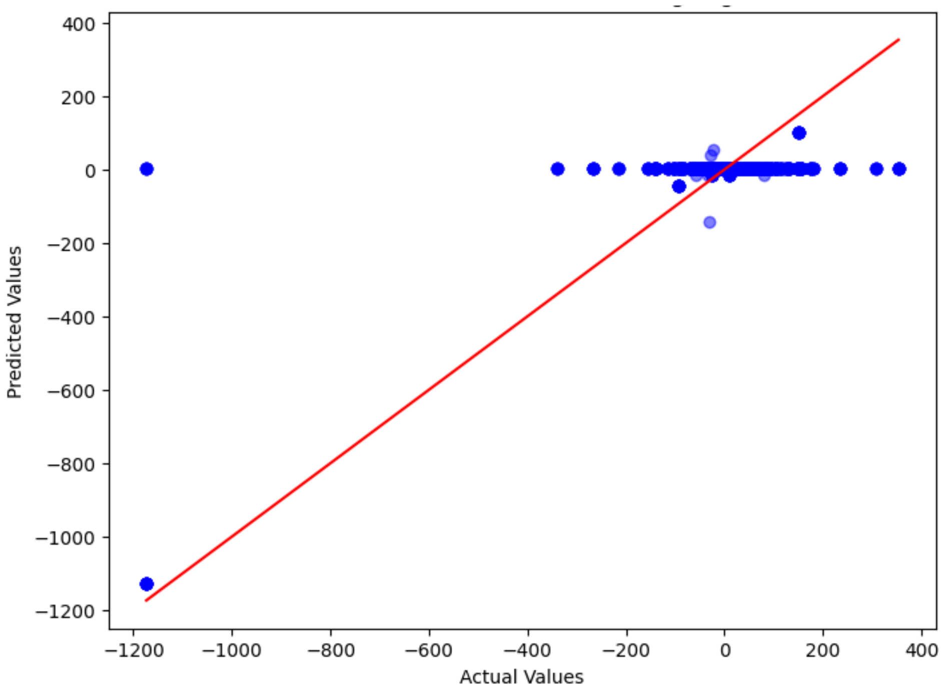

Residual Plot - Random Forest Regressor
400
200
0

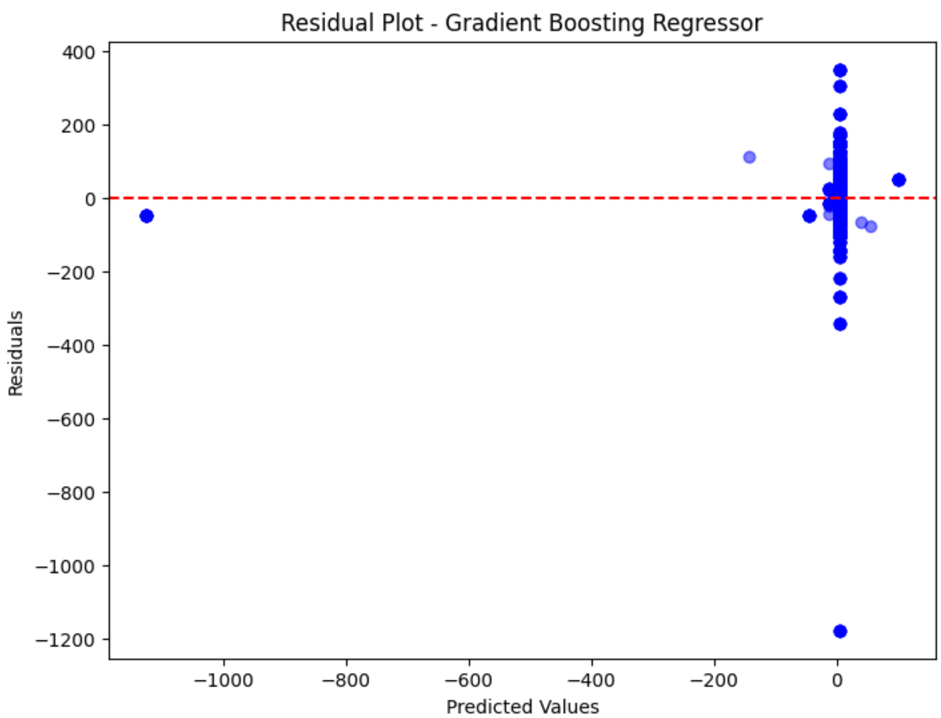


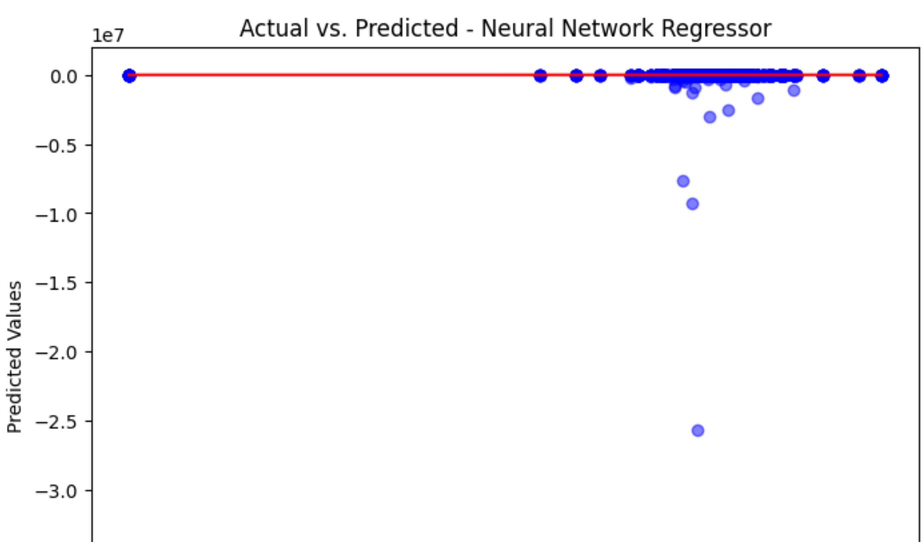


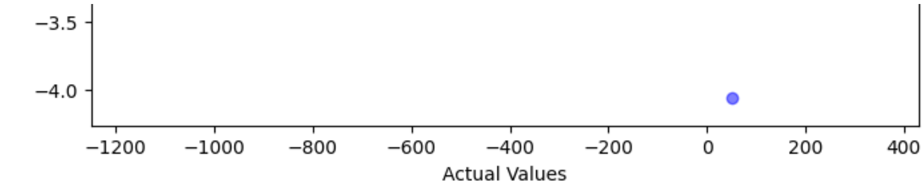


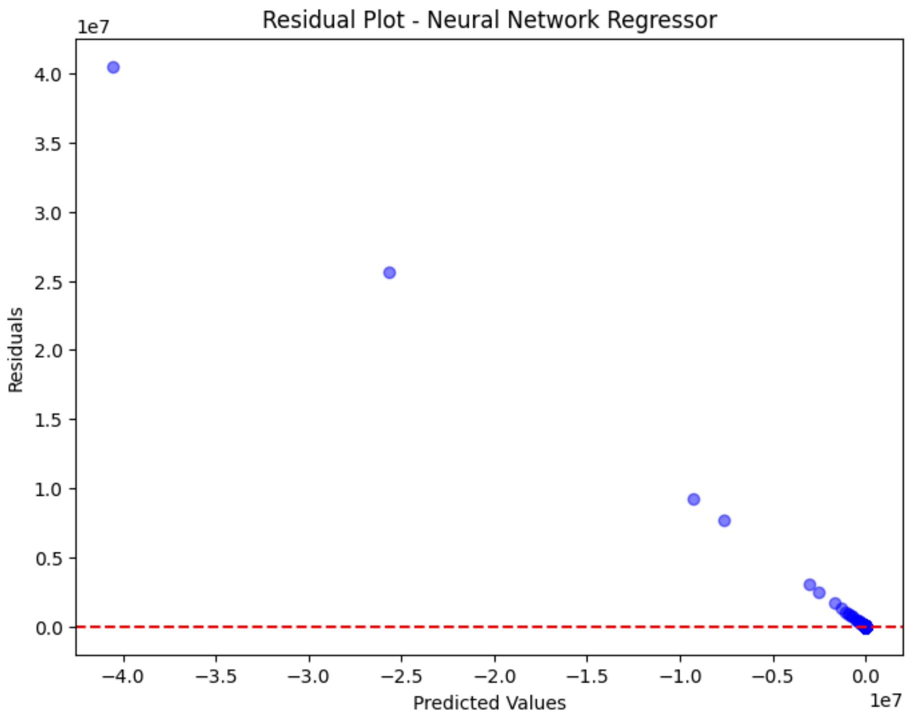
Actual vs. Predicted - Gradient Boosting Regressor


Predicted Values
Actual Values


Residual Plot - Gradient Boosting Regressor
Residuals
Predicted Values


Actual vs. Predicted - Neural Network Regressor
1e7
Predicted Values






```
                             Mean Squared Error     R-squared  \
Linear Regression                  2.719533e+04  1.690131e-05
Decision Tree Regressor            1.629985e+04  4.006479e-01
Random Forest Regressor            1.426430e+04  4.754961e-01
Support Vector Regressor           2.728427e+04 -3.253527e-03
Gradient Boosting Regressor        1.402917e+04  4.841418e-01
Neural Network Regressor           1.561907e+12 -5.743195e+07

                             Mean Absolute Error  Root Mean Squared Error  \
Linear Regression                      77.295484             1.649101e+02
Decision Tree Regressor                72.397503             1.276709e+02
Random Forest Regressor                65.215510             1.194332e+02
Support Vector Regressor               76.810227             1.651795e+02
Gradient Boosting Regressor            64.989379             1.184448e+02
Neural Network Regressor            64229.120665             1.249763e+06
```

```
#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 th
```

```
Gradient Boosting Regressor    [3.6645414199153072]
```

```

# 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()
```