Datasets

[ ] ↳ 9 cells hidden

Classification Modeling on Sentiment Prediction

```
# 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')
```

```
# Create a copy of the  bitcoin tweets DataFrame
df_tweets = tweets.copy()
df_tweets.head(2)
```

	user_name	user_location	user_description	user_created	user_followers	user_friends	us
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', 'time', 'close', 'high',
       'low', 'open', 'volumefrom', 'volumeto', 'Date', 'Time', 'volume',
       'marketcap', 'price_delta'],
      dtype='object')
```

Feature Extraction

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
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])

# Perform sentiment analysis
X = combined_features
y = tweets_df['sentiment_level']

# 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.feature_extraction.text import CountVectorizer: This line imports the CountVectorizer class from the Scikit-learn library. C
```

```
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]]
```

```
import numpy as np

matrix = unigram_features[:].toarray()  # Select the desired subset of rows

value_counts = {}
for value in range(14):
    count = np.count_nonzero(matrix == value)
    value_counts[value] = count

# Print the value counts
for value, count in value_counts.items():
    print("Count of", value, ":", count)
```

```
Count of 0 : 189953938
Count of 1 : 116293
Count of 2 : 8897
Count of 3 : 1795
Count of 4 : 142
Count of 5 : 39
Count of 6 : 55
Count of 7 : 1
Count of 8 : 5
Count of 9 : 0
Count of 10 : 0
Count of 11 : 0
Count of 12 : 0
Count of 13 : 0
```

```
import numpy as np

matrix = bigram_features[:].toarray()  # Select the desired subset of rows

value_counts = {}
for value in range(14):
    count = np.count_nonzero(matrix == value)
    value_counts[value] = count

# Print the value counts
for value, count in value_counts.items():
    print("Count of", value, ":", count)
```

```
Count of 0 : 516760987
Count of 1 : 130418
Count of 2 : 1014
Count of 3 : 83
Count of 4 : 4
Count of 5 : 1
Count of 6 : 0
Count of 7 : 0
Count of 8 : 0
Count of 9 : 0
Count of 10 : 0
Count of 11 : 0
Count of 12 : 1
Count of 13 : 0
```

```
import numpy as np

matrix = combined_features[:].toarray()  # Select the desired subset of rows

value_counts = {}
for value in range(14):
    count = np.count_nonzero(matrix == value)
    value_counts[value] = count

# Print the value counts
for value, count in value_counts.items():
    print("Count of", value, ":", count)

```

```
Count of 0 : 706714925
Count of 1 : 246711
Count of 2 : 9911
Count of 3 : 1878
Count of 4 : 146
Count of 5 : 40
Count of 6 : 55
Count of 7 : 1
Count of 8 : 5
Count of 9 : 0
Count of 10 : 0
Count of 11 : 0
Count of 12 : 1
Count of 13 : 0
```

Naive_bayes

```
import time
```

```
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
start_time = time.time()
# Train a classification model (e.g., Naive Bayes)
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

# Predict sentiment labels for test data
y_pred = classifier.predict(X_test)

# Evaluate the model using additional metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)

# 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)])
predicted_sentiment = classifier.predict(new_tweet_features)
#Classification Report
```

```
print("Predicted sentiment:", predicted_sentiment)
print(classification_report(y_test, y_pred))
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")
```

```
Accuracy: 0.7879746835443038
Precision: 0.8031951087547753
Recall: 0.7879746835443038
F1-Score: 0.791715079259514
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support

Extreme Negative       0.93      0.71      0.80        55
Extreme Positive       0.50      0.69      0.58       127
        Negative       0.83      0.61      0.71       157
         Neutral       0.88      0.86      0.87       908
        Positive       0.67      0.74      0.70       333

        accuracy                           0.79      1580
       macro avg       0.76      0.72      0.73      1580
    weighted avg       0.80      0.79      0.79      1580

Execution time: 0.47232484817504883 seconds
```

```
#The Naive Bayes classifier achieved an accuracy of 0.7879746835443038 and a precision of 0.8031951087547753 for sentiment analysis on the
```

Support Vector Machines (SVM)

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
start_time = time.time()
try:
    # 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)

    # Train a linear SVM classifier
    classifier = LinearSVC()
    classifier.fit(X_train, y_train)

    # Evaluate the model using additional metrics
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-Score:", f1)

    # 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)])
    predicted_sentiment = classifier.predict(new_tweet_features)
    print("Predicted sentiment:", predicted_sentiment)

except Exception as e:
    print("An error occurred:", str(e))
#Classification Report
print(classification_report(y_test, y_pred))
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")
```

```
Accuracy: 0.8645569620253165
Precision: 0.8642113824767497
Recall: 0.8645569620253165
F1-Score: 0.859632616414193
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support

Extreme Negative       0.95      0.76      0.85        55
Extreme Positive       0.86      0.65      0.74       127
        Negative       0.89      0.70      0.78       157
         Neutral       0.87      0.97      0.92       908
        Positive       0.82      0.74      0.78       333

        accuracy                           0.86      1580
       macro avg       0.88      0.77      0.81      1580
    weighted avg       0.86      0.86      0.86      1580

Execution time: 3.031984806060791 seconds
```

```
#The SVM classifier achieved an accuracy of 0.8645569620253165 and a precision of 0.8642113824767497 for sentiment analysis on the tweet c
```

Random Forest

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
start_time = time.time()
# Feature Extraction: Unigrams
vectorizer = CountVectorizer(ngram_range=(1, 1))
X = vectorizer.fit_transform(tweets_df['text'])
y = tweets_df['sentiment_level']

try:
    # 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)

    # Train a linear Random Forest classifier
    classifier = RandomForestClassifier()
    classifier.fit(X_train, y_train)

    # Evaluate the model using additional metrics
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-Score:", f1)

    # Use the trained model for future predictions
    new_tweet = ["New tweet about Bitcoin"]
    new_tweet_features = vectorizer.transform(new_tweet)
    predicted_sentiment = classifier.predict(new_tweet_features)
    print("Predicted sentiment:", predicted_sentiment)

except Exception as e:
    print("An error occurred:", str(e))
#Classification Report
print(classification_report(y_test, y_pred))
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")
```

```
Accuracy: 0.8582278481012658
Precision: 0.8634838727696282
Recall: 0.8582278481012658
F1-Score: 0.8504686275310068
```

```
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support

Extreme Negative       1.00      0.75      0.85        55
Extreme Positive       0.92      0.52      0.66       127
        Negative       0.92      0.66      0.77       157
         Neutral       0.85      0.98      0.91       908
        Positive       0.83      0.75      0.79       333

        accuracy                           0.86      1580
       macro avg       0.90      0.73      0.80      1580
    weighted avg       0.86      0.86      0.85      1580

Execution time: 18.978980779647827 seconds
```

```
#The Random Forest classifier achieved an accuracy of 0.870253164556962 and a precision of 0.8670628029958947 for sentiment analysis on th
```

Logistic Regression

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
start_time = time.time()
# Assuming you have tweets_df with the appropriate 'text' and 'sentiment_level' columns

# Feature Extraction: Unigrams
vectorizer = CountVectorizer(ngram_range=(1, 1))
X = vectorizer.fit_transform(tweets_df['text'])
y = tweets_df['sentiment_level']

try:
    # 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)

    # Train a logistic regression classifier with increased max_iter
    classifier = LogisticRegression(max_iter=1000)
    classifier.fit(X_train, y_train)

    # Evaluate the model using additional metrics
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-Score:", f1)

    # Use the trained model for future predictions
    new_tweet = ["New tweet about Bitcoin"]
    new_tweet_features = vectorizer.transform(new_tweet)
    predicted_sentiment = classifier.predict(new_tweet_features)
    print("Predicted sentiment:", predicted_sentiment)

except Exception as e:
    print("An error occurred:", str(e))
#Classification Report
print(classification_report(y_test, y_pred))
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")
```

```
Accuracy: 0.859493670886076
Precision: 0.8596313804881421
Recall: 0.859493670886076
F1-Score: 0.8545443425051455
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support
```

```
Extreme Negative       1.00      0.73      0.84        55
Extreme Positive       0.86      0.64      0.73       127
        Negative       0.86      0.69      0.76       157
         Neutral       0.87      0.97      0.91       908
        Positive       0.82      0.75      0.78       333

        accuracy                           0.86      1580
       macro avg       0.88      0.75      0.81      1580
    weighted avg       0.86      0.86      0.85      1580

Execution time: 12.210850715637207 seconds
```

```
#The Logistic Regression classifier achieved an accuracy of 0.859493670886076 and a precision of 0.8596313804881421 for sentiment analysis
```

Gradient Boosting

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
start_time = time.time()
# Assuming you have tweets_df with the appropriate 'text' and 'sentiment_level' columns

# Feature Extraction: Unigrams
vectorizer = CountVectorizer(ngram_range=(1, 1))
X = vectorizer.fit_transform(tweets_df['text'])
y = tweets_df['sentiment_level']

try:
    # 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)

    # Train a Gradient Boosting classifier
    classifier = GradientBoostingClassifier()
    classifier.fit(X_train, y_train)

    # Evaluate the model using additional metrics
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-Score:", f1)

    # Use the trained model for future predictions
    new_tweet = ["New tweet about Bitcoin"]
    new_tweet_features = vectorizer.transform(new_tweet)
    predicted_sentiment = classifier.predict(new_tweet_features)
    print("Predicted sentiment:", predicted_sentiment)

except Exception as e:
    print("An error occurred:", str(e))
#Classification Report
print(classification_report(y_test, y_pred))
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")
```

```
Accuracy: 0.8455696202531645
Precision: 0.8532373257556967
Recall: 0.8455696202531645
F1-Score: 0.8363030504432803
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support

Extreme Negative       0.91      0.76      0.83        55
```

```
Extreme Positive       0.94      0.57      0.71       127
        Negative       0.92      0.62      0.74       157
         Neutral       0.83      0.99      0.90       908
        Positive       0.85      0.67      0.75       333

        accuracy                           0.85      1580
       macro avg       0.89      0.72      0.79      1580
    weighted avg       0.85      0.85      0.84      1580

Execution time: 121.13087058067322 seconds
```

```
#The Gradient Boosting classifier achieved an accuracy of 0.8468354430379746 and a precision of 0.8543684218834472 for sentiment analysis
```

Cross Validation of Models

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# Define the models
models = [
    ("Naive Bayes", MultinomialNB()),
    ("Support Vector Machine", SVC()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Gradient Boosting", GradientBoostingClassifier())
]

# Perform cross-validation and evaluation for each model
for model_name, model in models:
    # Perform cross-validation
    scores = cross_val_score(model, X_train, y_train, cv=5)
    mean_score = scores.mean()

    # Fit the model on the entire training set
    model.fit(X_train, y_train)

    # Evaluate the model on the test set
    accuracy = model.score(X_test, y_test)

    # Print the results
    print("Model:", model_name)
    print("Cross-Validation Mean Score:", mean_score)
    print("Accuracy:", accuracy)
    print()

```

```
Model: Naive Bayes
Cross-Validation Mean Score: 0.7910727171592651
Accuracy: 0.7879746835443038
```

Cross Validation

```python
#cross-validation for the models using scikit-learn's cross_val_score function
```

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from scipy.sparse import hstack

```

```python
#Naive Bayes
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Train and evaluate Naive Bayes
naive_bayes = MultinomialNB()
naive_bayes_scores = cross_val_score(naive_bayes, X_train, y_train, cv=5)
print("Naive Bayes Cross-Validation Scores:", naive_bayes_scores.mean())
naive_bayes.fit(X_train, y_train)
naive_bayes_accuracy = naive_bayes.score(X_test, y_test)
print("Naive Bayes Accuracy:", naive_bayes_accuracy)
# Predict sentiment labels for test data
y_pred = naive_bayes.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
Naive Bayes Cross-Validation Scores: 0.7888584042414585
Naive Bayes Accuracy: 0.7943037974683544
                  precision    recall  f1-score   support

Extreme Negative       0.94      0.58      0.72        55
Extreme Positive       0.60      0.57      0.59       127
        Negative       0.85      0.58      0.69       157
         Neutral       0.84      0.92      0.88       908
        Positive       0.69      0.68      0.69       333

        accuracy                           0.79      1580
       macro avg       0.79      0.67      0.71      1580
    weighted avg       0.79      0.79      0.79      1580
```

```
#SVM
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Train and evaluate SVM
svm = LinearSVC()
svm_scores = cross_val_score(svm, X_train, y_train, cv=5)
print("SVM Cross-Validation Scores:", svm_scores.mean())
svm.fit(X_train, y_train)
svm_accuracy = svm.score(X_test, y_test)
print("SVM Accuracy:", svm_accuracy)
# Predict sentiment labels for test data
y_pred = svm.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
SVM Cross-Validation Scores: 0.8630914439199415
SVM Accuracy: 0.870253164556962
                  precision    recall  f1-score   support

Extreme Negative       0.89      0.76      0.82        55
Extreme Positive       0.79      0.70      0.74       127
        Negative       0.81      0.71      0.76       157
         Neutral       0.90      0.96      0.93       908
        Positive       0.83      0.77      0.80       333

        accuracy                           0.87      1580
       macro avg       0.84      0.78      0.81      1580
    weighted avg       0.87      0.87      0.87      1580
```

```
#Random Forest
```

```

# Train Random Forest classifier
random_forest = RandomForestClassifier(n_estimators=100, n_jobs=-1)
random_forest.fit(X_train, y_train)

# Evaluate Random Forest
random_forest_scores = cross_val_score(random_forest, X_train, y_train, cv=5)
random_forest_mean_score = random_forest_scores.mean()
```

```python

random_forest_accuracy = random_forest.score(X_test, y_test)

# Print results
print("Random Forest Cross-Validation Mean Score:", random_forest_mean_score)
print("Random Forest Accuracy:", random_forest_accuracy)
# Predict sentiment labels for test data
y_pred = random_forest.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))

```

```
Random Forest Cross-Validation Mean Score: 0.8553332681880594
Random Forest Accuracy: 0.8645569620253165
                  precision    recall  f1-score   support

Extreme Negative       1.00      0.75      0.85        55
Extreme Positive       0.96      0.52      0.67       127
        Negative       0.94      0.66      0.78       157
         Neutral       0.85      0.99      0.92       908
        Positive       0.83      0.78      0.80       333

        accuracy                           0.86      1580
       macro avg       0.92      0.74      0.80      1580
    weighted avg       0.87      0.86      0.86      1580
```

```python
#Logistic Regression
```

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# Train and evaluate Logistic Regression
logistic_regression = LogisticRegression(max_iter=1000)
logistic_regression_scores = cross_val_score(logistic_regression, X_train, y_train, cv=5)
logistic_regression_mean_score = logistic_regression_scores.mean()
logistic_regression.fit(X_train, y_train)
logistic_regression_accuracy = logistic_regression.score(X_test, y_test)
print("Logistic Regression Cross-Validation Mean Score:", logistic_regression_mean_score)
print("Logistic Regression Accuracy:", logistic_regression_accuracy)
# Predict sentiment labels for test data
y_pred = logistic_regression.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
Logistic Regression Cross-Validation Mean Score: 0.8429882387724625
Logistic Regression Accuracy: 0.8537974683544304
                  precision    recall  f1-score   support

Extreme Negative       0.98      0.75      0.85        55
Extreme Positive       0.90      0.55      0.68       127
        Negative       0.92      0.69      0.79       157
         Neutral       0.84      0.98      0.91       908
        Positive       0.83      0.73      0.78       333

        accuracy                           0.85      1580
       macro avg       0.89      0.74      0.80      1580
    weighted avg       0.86      0.85      0.85      1580
```

```python
#Gradient Boosting
```

```python
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import cross_val_score

# Train and evaluate Gradient Boosting Classifier
gradient_boosting = GradientBoostingClassifier()
gradient_boosting_scores = cross_val_score(gradient_boosting, X_train, y_train, cv=3)  # Adjust cv parameter as needed
gradient_boosting_mean_score = gradient_boosting_scores.mean()

gradient_boosting.fit(X_train, y_train)
gradient_boosting_accuracy = gradient_boosting.score(X_test, y_test)

print("Gradient Boosting Cross-Validation Mean Score:", gradient_boosting_mean_score)
```

```
print("Gradient Boosting Accuracy:", gradient_boosting_accuracy)
# Predict sentiment labels for test data
y_pred = gradient_boosting.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
Gradient Boosting Cross-Validation Mean Score: 0.8421968977524533
Gradient Boosting Accuracy: 0.8436708860759494
                  precision    recall  f1-score   support

Extreme Negative       0.89      0.76      0.82        55
Extreme Positive       0.93      0.58      0.71       127
        Negative       0.93      0.61      0.74       157
         Neutral       0.82      0.99      0.90       908
        Positive       0.86      0.66      0.75       333

        accuracy                           0.84      1580
       macro avg       0.89      0.72      0.78      1580
    weighted avg       0.85      0.84      0.83      1580
```

Hyperparameter Tuning

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
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])

# Perform sentiment analysis
X = combined_features
y = tweets_df['sentiment_level']

# Apply feature selection
k = 1000  # Number of top features to select
feature_selector = SelectKBest(chi2, k=k)
X_selected = feature_selector.fit_transform(X, y)

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

# Define the models and their respective hyperparameter grids
models = [
    ("Naive Bayes", MultinomialNB(), {'alpha': [0.1, 1.0, 10.0]}),
    ("Support Vector Machine", SVC(), {'C': [0.1, 1.0, 10.0]}),
    ("Random Forest", RandomForestClassifier(), {'n_estimators': [100, 200, 300]}),
    ("Logistic Regression", LogisticRegression(), {'C': [0.1, 1.0, 10.0]}),
    ("Gradient Boosting", GradientBoostingClassifier(), {'n_estimators': [100, 200, 300]})
]

# Perform cross-validation and evaluation for each model
for model_name, model, param_grid in models:
    # Perform hyperparameter tuning using GridSearchCV
    grid_search = GridSearchCV(model, param_grid, cv=5)
    grid_search.fit(X_train, y_train)

    # Get the best model and its parameters
    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_
```

```

    # Perform cross-validation with the best model
    cross_val_scores = cross_val_score(best_model, X_train, y_train, cv=5)

    # Fit the best model on the entire training set
    best_model.fit(X_train, y_train)

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

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Print the results
    print("Model:", model_name)
    print("Best Parameters:", best_params)
    print("Cross-Validation Accuracy:", cross_val_scores.mean())
    print("Accuracy:", accuracy)
    print()

```

```
Model: Naive Bayes
Best Parameters: {'alpha': 1.0}
Cross-Validation Accuracy: 0.7736638954869359
Accuracy: 0.7639240506329114

Model: Support Vector Machine
Best Parameters: {'C': 10.0}
Cross-Validation Accuracy: 0.8809735710634715
Accuracy: 0.8848101265822785

Model: Random Forest
Best Parameters: {'n_estimators': 300}
Cross-Validation Accuracy: 0.8649886747446806
Accuracy: 0.8613924050632912

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Model TPOT

```
# Assuming you have the 'data1' and 'data2' DataFrames
data1 = crypto_usd.copy()
data2 = tweets.copy()
# Merge the two DataFrames based on 'time' and 'date' columns
merge = pd.merge(data1, data2, left_on='time', right_on='date')

# Drop the duplicate 'date' column
merge.drop('date', axis=1, inplace=True)

# Display the merged DataFrame
print(merge)

```

```
                     time     close      high       low      open  volumefrom  \
0     2023-02-25 21:00:00  22944.16  22960.69  22863.96  22921.71     1331.05
1     2023-02-25 21:00:00  22944.16  22960.69  22863.96  22921.71     1331.05
2     2023-02-25 21:00:00  22944.16  22960.69  22863.96  22921.71     1331.05
3     2023-02-25 21:00:00  22944.16  22960.69  22863.96  22921.71     1331.05
4     2023-02-25 21:00:00  22944.16  22960.69  22863.96  22921.71     1331.05
...                   ...       ...       ...       ...       ...         ...
7893  2023-03-04 23:00:00  22351.08  22352.28  22302.56  22311.46      476.12
7894  2023-03-04 23:00:00  22351.08  22352.28  22302.56  22311.46      476.12
7895  2023-03-04 23:00:00  22351.08  22352.28  22302.56  22311.46      476.12
7896  2023-03-04 23:00:00  22351.08  22352.28  22302.56  22311.46      476.12
7897  2023-03-04 23:00:00  22351.08  22352.28  22302.56  22311.46      476.12

         volumeto        Date      Time       volume  ...  user_verified  \
0     30505954.61  2023-02-25  21:00:00  30504623.56  ...          False
1     30505954.61  2023-02-25  21:00:00  30504623.56  ...          False
2     30505954.61  2023-02-25  21:00:00  30504623.56  ...          False
3     30505954.61  2023-02-25  21:00:00  30504623.56  ...          False
4     30505954.61  2023-02-25  21:00:00  30504623.56  ...          False
...           ...         ...       ...          ...  ...            ...
7893  10632637.83  2023-03-04  23:00:00  10632161.71  ...          False
7894  10632637.83  2023-03-04  23:00:00  10632161.71  ...          False
7895  10632637.83  2023-03-04  23:00:00  10632161.71  ...          False
7896  10632637.83  2023-03-04  23:00:00  10632161.71  ...          False
7897  10632637.83  2023-03-04  23:00:00  10632161.71  ...          False

                                                   text  \
0     ethereum price updat eth 157128 usd bitcoin 00...
1                      bitcoin 1month predict tuhgbqklxn
2     btcusdt 15m volum spike btc btc bitcoin ucl5iaaq4
3     lð���k take time think littlebit person load a...
4     ð���ð��¸ sat 25 feb 2023 210035 gmt top 10 btc...
...                                                 ...
7893  usd racist built colonist slaver paid btc bc e...
7894  everris rise everrisev3 everrevok defi crypto ...
7895  ð��� parti time ð��� ð��� 10000 x1 megapr ð�ª©...
7896  strategi 5010hl1h atr20d 92138 04 mar 2023 230...
7897  complet variou task hh8vl67nz5 claim slm token...

                                               hashtags               source  \
0     ['Ethereum', 'ETH', 'Bitcoin', 'BTC', 'altcoin...      Twitter Web App
1                                           ['Bitcoin']         predictCCbot
2                                    ['BTC', 'Bitcoin']       JumpLineAlerts
3     ['GGA', 'cryptocurrency', 'Bitcoin', 'bnb', 'T...  Twitter for Android
4                                           ['bitcoin']              eht10c
...                                                 ...                  ...
7893                                            ['BTC']  Twitter for Android
7894  ['EverRise', 'EverRiseV3', 'EverRevoke', 'DeFi...  EverRiseTwitterBot1
7895  ['btc', 'eth', 'xrp', 'doge', 'shiba', 'lto', ...      Twitter Web App
7896                                  ['BTC', 'BitMEX']      system'cRe5520'
7897  ['SLMGames', 'SLM', 'Web3', 'BTC', 'ETH', 'BSC...           TweetDeck

      is_retweet compound         score   sentiment_level  polarity  \
0            0.0   0.0000  0.000000e+00           Neutral  0.000000
1            0.0   0.0000  0.000000e+00           Neutral  0.000000
2            0.0   0.0000  0.000000e+00           Neutral  0.000000
3            0.0  -0.3089 -9.666133e+05          Negative -0.041667
4            0.0   0.2023  7.485100e+00          Positive  0.500000
```

```
merge.head()
```

	time	close	high	low	open	volumefrom	volumeto	Date	Time	
0	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
1	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
2	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
3	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
4	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30

5 rows × 29 columns

```
1 merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7898 entries, 0 to 7897
Data columns (total 29 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   time              7898 non-null   object
 1   close             7898 non-null   float64
 2   high              7898 non-null   float64
 3   low               7898 non-null   float64
 4   open              7898 non-null   float64
 5   volumefrom        7898 non-null   float64
 6   volumeto          7898 non-null   float64
 7   Date              7898 non-null   object
 8   Time              7898 non-null   object
 9   volume            7898 non-null   float64
 10  marketcap         7898 non-null   float64
 11  price_delta       7898 non-null   float64
 12  user_name         7898 non-null   object
 13  user_location     3898 non-null   object
 14  user_description  7620 non-null   object
 15  user_created      7898 non-null   object
 16  user_followers    7898 non-null   float64
 17  user_friends      7898 non-null   float64
 18  user_favourites   7898 non-null   float64
 19  user_verified     7898 non-null   bool
 20  text              7898 non-null   object
 21  hashtags          7891 non-null   object
 22  source            7891 non-null   object
 23  is_retweet        7891 non-null   float64
 24  compound          7898 non-null   float64
 25  score             7898 non-null   float64
 26  sentiment_level   7898 non-null   object
 27  polarity          7898 non-null   float64
 28  subjectivity      7898 non-null   float64
dtypes: bool(1), float64(17), object(11)
memory usage: 1.8+ MB
```

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

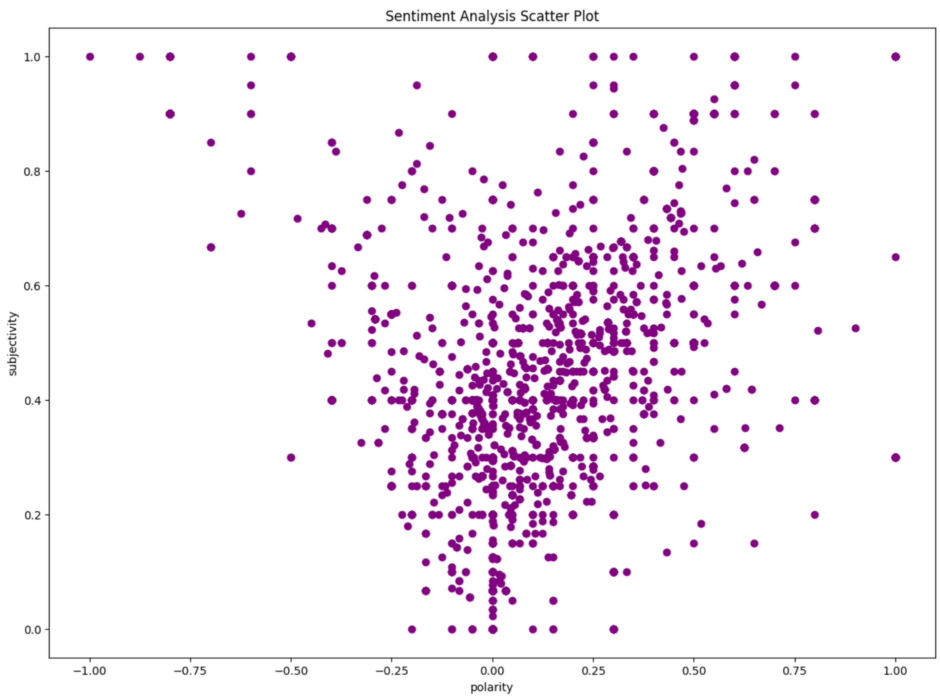
```
Neutral             93169
Positive            35921
Extreme Positive    17343
Negative            15903
```

```
Extreme Negative     5316
Name: sentiment_level, dtype: int64
```

```
import matplotlib.pyplot as plt
# scatter plot to show the subjectivity and the polarity
plt.figure(figsize=(14,10))

for i in range(merge.shape[0]):
    plt.scatter(merge["polarity"].iloc[[i]].values[0], merge["subjectivity"].iloc[[i]].values[0], color="Purple")

plt.title("Sentiment Analysis Scatter Plot")
plt.xlabel('polarity')
plt.ylabel('subjectivity')
plt.show()
```




```
#Creating Target Column
```

```
price_indicator = [merge.close[0] - merge['open'][0]]
for i in range(99):
    price_indicator.append(merge.close[i+1] - merge.close[i])
#price_indicator
```

```
merge['price_indicator'] = 0
for i in range(len(price_indicator)):
    merge['price_indicator'][i] = price_indicator[i]

merge.head()
```

```
<ipython-input-18-8f90b0759c32>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
  merge['price_indicator'][i] = price_indicator[i]
```

	time	close	high	low	open	volumefrom	volumeto	Date	Time	
0	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
1	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
2	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
3	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
4	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30

5 rows × 30 columns

```
merge['target'] = 0
for i in range(100):
    if merge.price_indicator[i] > 0:
        merge['target'][i] = 1

# 0 - price down
# 1 - price up

merge.head()
```

```
<ipython-input-19-e0c87f2219f9>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
  merge['target'][i] = 1
```

	time	close	high	low	open	volumefrom	volumeto	Date	Time	
0	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
1	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
2	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
3	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30
4	2023-02-25 21:00:00	22944.16	22960.69	22863.96	22921.71	1331.05	30505954.61	2023-02-25	21:00:00	30

5 rows × 31 columns

```
keep_columns = ['open', 'high', 'low', 'close', 'volume','polarity','subjectivity','compound','score','price_indicator','target']
df = merge[keep_columns]
df.head()
```

	open	high	low	close	volume	polarity	subjectivity	compound	
0	22921.71	22960.69	22863.96	22944.16	30504623.56	0.000000	0.250000	0.0000	
1	22921.71	22960.69	22863.96	22944.16	30504623.56	0.000000	0.000000	0.0000	
2	22921.71	22960.69	22863.96	22944.16	30504623.56	0.000000	0.000000	0.0000	
3	22921.71	22960.69	22863.96	22944.16	30504623.56	-0.041667	0.458333	-0.3089	-9666
4	22921.71	22960.69	22863.96	22944.16	30504623.56	0.500000	0.500000	0.2023	

```
#Model Building
```

```
import numpy as np
#Create the feature data set
X = df
X = np.array(X.drop(['target'],1))
#Create the target data set
y = np.array(df['target'])
```

```
<ipython-input-22-63d9de6a3c5f>:4: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument
  X = np.array(X.drop(['target'],1))
```

```
from sklearn.model_selection import train_test_split
#Split the data into 80% training and 20% testing data sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 0)
```

```
!pip install tpot

```

```
Collecting tpot
  Downloading TPOT-0.12.0-py3-none-any.whl (87 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 87.4/87.4 kB 5.8 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/python3.10/dist-packages (from tpot) (1.22.4)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from tpot) (1.10.1)
Requirement already satisfied: scikit-learn>=0.22.0 in /usr/local/lib/python3.10/dist-packages (from tpot) (1.2.2)
Collecting deap>=1.2 (from tpot)
  Downloading deap-1.3.3-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (139 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.9/139.9 kB 14.0 MB/s eta 0:00:00
Collecting update-checker>=0.16 (from tpot)
  Downloading update_checker-0.18.0-py3-none-any.whl (7.0 kB)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/dist-packages (from tpot) (4.65.0)
Collecting stopit>=1.1.1 (from tpot)
  Downloading stopit-1.1.2.tar.gz (18 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.10/dist-packages (from tpot) (1.5.3)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from tpot) (1.2.0)
Requirement already satisfied: xgboost>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tpot) (1.7.6)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->tpot) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->tpot) (2022.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22.0->tpot) (3.1.0
Requirement already satisfied: requests>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from update-checker>=0.16->tpot) (2.27.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas>=0.24.2->tpot)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.3.0->update-checker>=
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.3.0->update-checker>=0.1
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests>=2.3.0->update-check
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.3.0->update-checker>=0.16->tpo
Building wheels for collected packages: stopit
  Building wheel for stopit (setup.py) ... done
  Created wheel for stopit: filename=stopit-1.1.2-py3-none-any.whl size=11938 sha256=a56fda5b968cc0cd8d28799e3e03a41bd1d28bf77cd34f2460
  Stored in directory: /root/.cache/pip/wheels/af/f9/87/bf5b3d565c2a007b4dae9d8142dccc85a9f164e517062dd519
Successfully built stopit
Installing collected packages: stopit, deap, update-checker, tpot
Successfully installed deap-1.3.3 stopit-1.1.2 tpot-0.12.0 update-checker-0.18.0
```

```
from tpot import TPOTClassifier
from sklearn.metrics import confusion_matrix,accuracy_score,roc_auc_score
```

```
from sklearn.metrics import roc_auc_score
from tpot import TPOTClassifier
import numpy as np

```

```
from sklearn.metrics import roc_auc_score
from tpot import TPOTClassifier
import numpy as np

# Instantiate TPOTClassifier
tpot = TPOTClassifier(
    generations=5,
    population_size=20,
    verbosity=2,
    scoring='roc_auc',
    random_state=42,
    disable_update_check=True,
    config_dict='TPOT light'
)

# Convert X_train and y_train to NumPy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)

# Ensure that there are at least two classes in y_train
if len(np.unique(y_train)) < 2:
    raise ValueError("At least two classes are required in y_train for ROC AUC score calculation.")

try:
    # Fit TPOTClassifier
    tpot.fit(X_train, y_train)

    # AUC score for tpot model
    X_test = np.array(X_test)  # Assuming you have X_test data
    y_test = np.array(y_test)  # Assuming you have y_test data

    # Ensure that there are at least two classes in y_test
    if len(np.unique(y_test)) < 2:
        raise ValueError("At least two classes are required in y_test for ROC AUC score calculation.")

    tpot_auc_score = roc_auc_score(y_test, tpot.predict_proba(X_test)[:, 1])
    print(f'\nAUC score: {tpot_auc_score:.4f}')

    # Print best pipeline steps
    print('\nBest pipeline steps:')
    for idx, (name, transform) in enumerate(tpot.fitted_pipeline_.steps, start=1):
        print(f'{idx}. {transform}')

except ValueError as e:
    print("Error:", str(e))

```

Optimization Progress: 33% 40/120 [00:11<00:29, 2.76pipeline/s]

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
Error: Only one class present in y_true. ROC AUC score is not defined in that case.
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
```

```
# Instantiate TPOTClassifier
tpot = TPOTClassifier(
    generations=5, #number of iterations to run ; pipeline optimisation process ; by default value is 100
    population_size=20, #number of individuals to retrain in the genetic programing popluation in every generation, by default value is 1
    verbosity=2, #it will state how much info TPOT will communicate while it is running
    scoring='roc_auc', #use to evaluate the quality of given pipeline
    random_state=42,
    disable_update_check=True,
    config_dict='TPOT light'
)
tpot.fit(X_train, y_train)

# AUC score for tpot model
tpot_auc_score = roc_auc_score(y_test, tpot.predict_proba(X_test)[:, 1])
print(f'\nAUC score: {tpot_auc_score:.4f}')

# Print best pipeline steps
print('\nBest pipeline steps:', end='\n')
for idx, (name, transform) in enumerate(tpot.fitted_pipeline_.steps, start=1):
    # Print idx and transform
    print(f'{idx}. {transform}')
```

Optimization Progress: 33% 40/120 [00:06<00:21, 3.71pipeline/s]

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning:
  warnings.warn(
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/tpot/base.py in fit(self, features, target,
sample_weight, groups)
    816                 warnings.simplefilter("ignore")
--> 817                 self._pop, _ = eaMuPlusLambda(
    818                     population=self._pop,

                                26 frames

IndexError: tuple index out of range

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_ranking.py in
_binary_roc_auc_score(y_true, y_score, sample_weight, max_fpr)
    337     """Binary roc auc score."""
    338     if len(np.unique(y_true)) != 2:
--> 339         raise ValueError(
    340             "Only one class present in y_true. ROC AUC score "
    341             "is not defined in that case."

ValueError: Only one class present in y_true. ROC AUC score is not defined in that case.
```

SEARCH STACK OVERFLOW

```
tpot.fitted_pipeline_
```

Model 1: Decision tree classifier

```
from sklearn.tree import DecisionTreeClassifier

```

```
clf = DecisionTreeClassifier(criterion='entropy', max_depth=8,
                                        min_samples_leaf=10,
                                        min_samples_split=6,
                                        random_state=42)
clf.fit(X_train,y_train)
```

```
                              DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=8, min_samples_leaf=10,
                       min_samples_split=6, random_state=42)
```

```
y_predicted = clf.predict(X_test)
```

```
y_predicted
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```

print( classification_report(y_test, y_predicted) )
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1580

    accuracy                           1.00      1580
   macro avg       1.00      1.00      1.00      1580
weighted avg       1.00      1.00      1.00      1580
```

```
accuracy_score(y_test,y_predicted)*100
```

```
100.0
```

```
#Creating Pipeline to see which model has more accuracy
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
pipeline_lr = Pipeline([('scaler1',StandardScaler()),
                       ('pca1',PCA(n_components=2)),
                       ('lr_classifier',LogisticRegression(random_state=0))])
```

```
pipeline_dt = Pipeline([('scaler2',StandardScaler()),
                       ('pca2',PCA(n_components=2)),
                       ('dt_classifier',DecisionTreeClassifier())])
```

```
pipeline_randomforest = Pipeline([('scaler3',StandardScaler()),
                       ('pca3',PCA(n_components=2)),
                       ('rf_classifier',RandomForestClassifier())])
```

```
pipeline = [pipeline_lr,pipeline_dt,pipeline_randomforest]
```

```
best_accuracy=0.0
best_classifier=0
best_pipeline=""
```

```
pipe_dict = {0:'Logistic Regression', 1:'Decision Tree', 2:'RandomForest'}

for pipe in pipeline:
    pipe.fit(X_train,y_train)

```

```
for i,model in enumerate(pipeline):
    print("{}Test Accuracy: {}".format(pipe_dict[i],model.score(X_test,y_test)))
```

```
Logistic RegressionTest Accuracy: 1.0
Decision TreeTest Accuracy: 0.9987341772151899
RandomForestTest Accuracy: 1.0
```

2m 1s completed at 7:51 AM