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	u
0	lrk	Vancouver, WA	lrk 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]]
```

Naive_bayes

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

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# 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))
```

```
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
```

```
#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

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))
```

```
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
```

```
#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.svm import LinearSVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score



# 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 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 = 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))
```

```
Accuracy: 0.870253164556962
Precision: 0.8670628029958947
Recall: 0.870253164556962
F1-Score: 0.8670875346312075
Predicted sentiment: ['Neutral']
                  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
```

```
#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

# 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))
```

```
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
```

```
#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

# 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))
```

```
Accuracy: 0.8468354430379746
Precision: 0.8543684218834472
Recall: 0.8468354430379746
F1-Score: 0.8375792142254445
Predicted sentiment: ['Neutral']
                  precision    recall  f1-score   support

Extreme Negative       0.90      0.78      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       0.86      0.68      0.76       333

        accuracy                           0.85      1580
       macro avg       0.89      0.73      0.79      1580
    weighted avg       0.85      0.85      0.84      1580
```

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

Cross Validation of Models

```
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

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

```
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

```

```
#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()

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
```

```
#Logistic Regression
```

```
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
```

```
#Gradient Boosting
```

```
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()

```

```
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(
Model: Logistic Regression
Best Parameters: {'C': 10.0}
Cross-Validation Accuracy: 0.8732179009190496
Accuracy: 0.8816455696202532

Model: Gradient Boosting
Best Parameters: {'n_estimators': 300}
Cross-Validation Accuracy: 0.8779677430670395
Accuracy: 0.8822784810126583
```

Model TPOT

[ ] ↳ *34 cells hidden*

✓ 0s completed at 4:20 PM