Activity based

Project Report on

AI Business Intelligence

Project Phase - II

Submitted to Vishwakarma University, Pune

Under the Initiative of

Contemporary Curriculum, Pedagogy, and Practice (C2P2)

By

Shravan Sudhir Meshram

SRN No : 202101425

Roll No : 31

Div : E

Third Year Engineering

Faculty In charge:- Prof. Moumita Pal

Department of Computer Engineering

Faculty of Science and Technology

Academic Year

2023-2024 Term-II

Business Intelligence : Phase I

Project Name : Social Media Sentiment Analysis

Introduction:

Social Media Sentiment Analysis, as it provides a direct window into the thoughts and feelings of customers, stakeholders, and the general public. By integrating sentiment analysis into BI processes, organizations can enhance their decision-making processes, improve customer relations, and stay ahead of market trends. Social Media Sentiment Analysis within the BI framework involves the systematic extraction and interpretation of sentiments embedded in the vast sea of user-generated content across social media channels. This process empowers BI professionals to gauge public perception, track brand sentiment, and identify emerging trends with unprecedented granularity.

Social Media Sentiment Analysis is a pivotal component in the realm of Business Intelligence, offering a unique lens into the sentiments expressed across digital platforms. By employing advanced natural language processing and machine learning techniques, BI professionals can decipher the positive, negative, or neutral tones in user-generated content.

Problem Statement

Understanding customer sentiments on social media platforms is crucial for businesses to manage their online reputation, identify areas for improvement, and tailor marketing strategies. This project focuses on developing a Business Intelligence (BI) solution for Social Media Sentiment Analysis, extracting insights from social media data

Objective

The primary objective is to design and implement a BI system that integrates with social media data sources, performs sentiment analysis, and provides actionable insights into customer opinions, trends, and sentiment shifts.

Existing Work Done:

In this project, we undertake exploratory data analysis (EDA) on the Social Media Sentiment dataset. The process includes:

1. **Data Collection:** Collect social media data from various platforms, including customer reviews, comments, and mentions.

2. **Data Preprocessing:** Preprocess the social media data to handle noise, irrelevant information, and ensure data consistency. Clean text data, handle emotions, and address any data quality issues that may affect sentiment analysis.

3. **Feature Engineering:** Identify and engineer features that contribute to sentiment analysis. This may include sentiment scores, sentiment trends over time, and the identification of key topics or keywords associated with positive or negative sentiments.

4. **BI Dashboard Development**: Design and implement a user-friendly BI dashboard that visualizes key sentiment analysis metrics. Include components for monitoring overall sentiment trends, identifying sentiment influencers, and assessing the impact of marketing campaigns.

Further Assessments:

5. **Sentiment Score Calculation:** Implement features for calculating sentiment scores from social media text data. Utilize natural language processing (NLP) techniques to assess the polarity of customer opinions.

6. **Trend Analysis:** Develop features for trend analysis, tracking sentiment changes over time. Identify patterns, spikes, or dips in sentiment that may coincide with specific events, product launches, or marketing efforts.

7. **Influencer Identification:** Integrate features for identifying social media influencers who impact sentiment. This includes recognizing individuals or accounts whose opinions carry significant weight within the online community.

8. **Brand Mention Analysis:** Analyze brand mentions within the BI system. Understand the context and sentiment associated with brand mentions to assess the overall perception of the brand in the social media landscape.

Execution Part:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
from imblearn.over_sampling import RandomOverSampler
import matplotlib.pyplot as plt
```

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.linear_model import LogisticRegression
```

This code snippet seems to be a Python script for performing text classification using Support Vector Machine (SVM) and Logistic Regression models. Let me break down each part of the code and explain its purpose:

1. import pandas as pd: This imports the pandas library, which is commonly used for data manipulation and analysis in Python. It's often used for handling tabular data.
2. from sklearn.model_selection import train_test_split: This imports the train_test_split function from the model_selection module of scikit-learn (sklearn). This function is used to split data into training and testing sets.
3. from sklearn.svm import SVC: This imports the Support Vector Classifier (SVC) class from the svm module of sklearn. SVC is a type of SVM algorithm used for classification tasks.
4. from sklearn.metrics import accuracy_score, classification_report: This imports evaluation metrics such as accuracy_score and classification_report from the metrics module of sklearn. These metrics are commonly used to evaluate the performance of classification models.
5. from sklearn.feature_extraction.text import TfidfVectorizer: This imports the TfidfVectorizer class from the feature_extraction.text module of sklearn. TfidfVectorizer is used to convert a collection of raw documents into a matrix of TF-IDF features.
6. from imblearn.over_sampling import RandomOverSampler: This imports the RandomOverSampler class from the imbalanced-learn library (imblearn). RandomOverSampler is a technique used to handle class imbalance by randomly duplicating instances from the minority class(es) until the class distribution is balanced.
7. import matplotlib.pyplot as plt: This imports the pyplot module from the matplotlib library, which is commonly used for data visualization in Python.
8. from nltk.sentiment.vader import SentimentIntensityAnalyzer: This imports the SentimentIntensityAnalyzer class from the vader module of the Natural Language Toolkit (NLTK). VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool specifically designed for analyzing sentiments expressed in text.
9. from sklearn.linear_model import LogisticRegression: This imports the Logistic Regression classifier from the linear_model module of sklearn. Logistic Regression is a linear model commonly used for binary classification tasks.

```
# Initialize the VADER sentiment analyzer
sia = SentimentIntensityAnalyzer()

# Function to get sentiment score for each text
def get_sentiment_score(Text):
    sentiment = sia.polarity_scores(Text)['compound']
    return sentiment

# Apply the sentiment analysis function to each text in the dataset
df['sentiment_score'] = df['Text'].apply(get_sentiment_score)

# Write the updated dataset with sentiment scores to a new CSV file
df.to_csv('sentiment_scores_dataset.csv', index=False)
df
```

1. Initialization: It initializes the VADER sentiment analyzer.
2. Sentiment Score Calculation Function: It defines a function (get_sentiment_score) to calculate sentiment scores for each text using VADER.

3. Application of Sentiment Analysis Function: It applies the sentiment analysis function to each text in the DataFrame and stores the sentiment scores in a new column ('sentiment_score').
4. Exporting Results: It exports the updated DataFrame, including the sentiment scores, to a CSV file named 'sentiment_scores_dataset.csv'.
5. Display: It displays the updated DataFrame showing the original text data along with the newly added sentiment scores.

```
# Initialize the VADER sentiment analyzer
sia = SentimentIntensityAnalyzer()

# Function to classify sentiment into positive, negative, or neutral
def classify_sentiment(score):
    if score > 0.05:
        return 'Positive'
    elif score < -0.05:
        return 'Negative'
    else:
        return 'Neutral'

# Function to get sentiment score for each text
def get_sentiment_score(text):
    sentiment = sia.polarity_scores(text)['compound']
    return sentiment

# Apply the sentiment analysis function to each text in the dataset
df['Sentiment_score'] = df['Text'].apply(get_sentiment_score)

# Classify sentiment into categories
df['Sentiment'] = df['Sentiment_score'].apply(classify_sentiment)
df
```

This code segment conducts sentiment analysis on text data using the VADER sentiment analyzer.

1. Initialization:
- Initialize the VADER sentiment analyzer.
2. Sentiment Classification Function:
- Define a function (classify_sentiment) to classify sentiment scores into categories: Positive, Negative, or Neutral.
- Scores above 0.05 are classified as Positive, scores below -0.05 as Negative, and the rest as Neutral.
3. Sentiment Scoring Function:
- Define a function (get_sentiment_score) to compute sentiment scores for each text using VADER's polarity_scores method.
4. Apply Sentiment Analysis:
- Apply the sentiment analysis function to each text in the DataFrame, storing the sentiment scores in a new column ('Sentiment_score').
5. Sentiment Classification:
- Classify sentiment scores into categories (Positive, Negative, or Neutral) using the defined classification function.
- The result is stored in a new column ('Sentiment') in the DataFrame.
6. Summary:

- The DataFrame df now contains two additional columns: 'Sentiment_score' and 'Sentiment', representing sentiment scores and classified sentiments, respectively.

```
# Splliting the Dataset into Training and Testing
features = ['Text','Hashtags','Likes','Retweets','Country']
target = 'sentiment_score'
df = df.dropna(subset=features + [target])
X = df[features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, 
random_state=20)
```

1. Feature Selection:
- The variable features contains a list of feature columns including 'Text', 'Hashtags', 'Likes', 'Retweets', and 'Country'. These features will be used for training the machine learning model.
2. Target Variable:
- The variable target represents the target variable, which is 'sentiment_score'. This is the variable we want to predict using the features.
3. Handling Missing Values:
- The code drops any rows with missing values in the selected features (features) or the target variable ('sentiment_score').
4. Splitting the Dataset:
- The train_test_split function from scikit-learn is used to split the dataset into training and testing sets.
- X contains the selected features, while y contains the target variable.
- The parameter test_size=0.2 specifies that 20% of the data will be used for testing, and the remaining 80% will be used for training.
- random_state=20 ensures reproducibility by fixing the random seed for the data splitting process.
5. Result:
- The resulting variables X_train, X_test, y_train, and y_test contain the training and testing sets of features and target variable, respectively.

```
# Check the unique values in y_train
print("Unique labels in y_train:", y_train.unique())

# If the labels are continuous, you might need to convert them to discrete classes.
# For example, if your labels are floats, you can convert them to integers:
y_train = y_train.astype(int)
y_test = y_test.astype(int)
```

1. Check Unique Values in y_train:
- The unique() method is used to print the unique labels in the y_train variable.
2. Conversion of Continuous Labels:
- The comment suggests that if the labels in y_train are continuous (e.g., floats), they might need to be converted to discrete classes. This conversion is often necessary for classification tasks where the model predicts discrete categories.
3. Conversion to Integers:
- The code then converts the labels in both y_train and y_test to integers using the astype(int) method. This ensures that the labels are represented as discrete classes rather than continuous values.

```
# Convert sentiment scores to sentiment labels (positive, negative, neutral)
def get_sentiment_label(score):
    if score > 0:
        return 'positive'
    elif score < 0:
        return 'negative'
    else:
        return 'neutral'

# Assuming 'sentiment_score' is the column containing sentiment scores
y = df['sentiment_score']
y_multiclass = y.apply(get_sentiment_label)

# Perform TF-IDF vectorization on the text data
tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform(df['Text'])

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

# Oversample the minority classes
oversampler = RandomOverSampler(random_state=42)
X_train_resampled, y_train_resampled = oversampler.fit_resample(X_train, y_train)

# Initialize and train the logistic regression model for multi-class classification
model = LogisticRegression(multi_class='auto', max_iter=1000)
model.fit(X_train_resampled, y_train_resampled)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nClassification Report:")
print(report)
```

1. Sentiment Score to Label Conversion:
- Defines a function (get_sentiment_label) to convert sentiment scores to sentiment labels (positive, negative, neutral).
- Applies this function to the 'sentiment_score' column of the DataFrame df, storing the result in y_multiclass.
2. TF-IDF Vectorization:
- Uses TF-IDF vectorization (TfidfVectorizer) to convert text data into numerical features (X_tfidf).
3. Data Splitting:
- Splits the data into training and testing sets (X_train, X_test, y_train, y_test) using train_test_split.
4. Oversampling:
- Uses RandomOverSampler to oversample the minority classes in the training data to handle class imbalance.

5. Model Training:
- Initializes and trains a logistic regression model (LogisticRegression) for multi-class classification using the resampled training data.
6. Prediction:
- Makes predictions (y_pred) using the trained model on the test data.
7. Model Evaluation:
- Computes accuracy and generates a classification report to evaluate the model's performance on the test data.
8. Print Results:
- Prints the accuracy and classification report

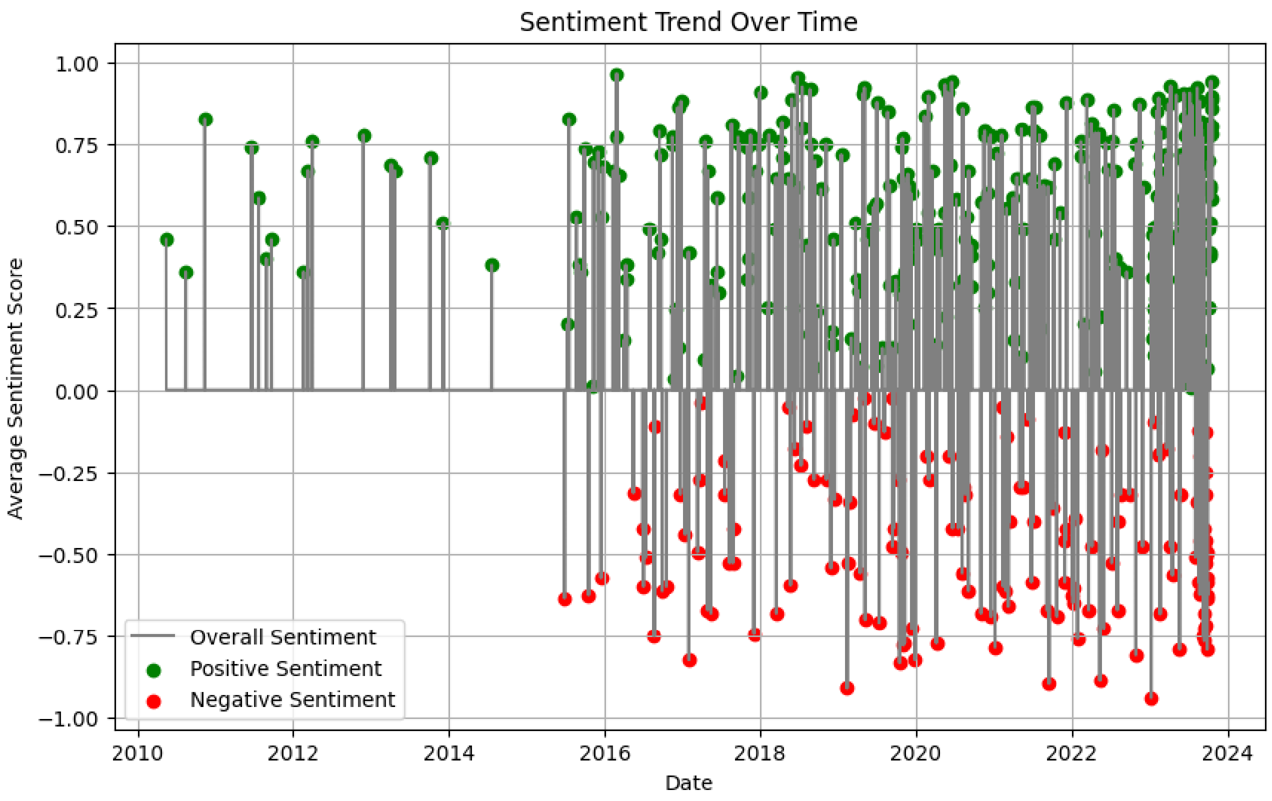
```python
# Convert sentiment scores to sentiment labels (positive, negative, neutral)
def get_sentiment_label(score):
    if score > 0:
        return 'positive'
    elif score < 0:
        return 'negative'
    else:
        return 'neutral'

# Assuming 'sentiment_score' is the column containing sentiment scores
y = df['sentiment_score']
y_multiclass = y.apply(get_sentiment_label)

# Perform TF-IDF vectorization on the text data
tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform(df['Text'])

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

# Oversample the minority classes
oversampler = RandomOverSampler(random_state=42)
X_train_resampled, y_train_resampled = oversampler.fit_resample(X_train, y_train)

# Initialize and train the SVM model for multi-class classification
model = SVC(kernel='linear')
model.fit(X_train_resampled, y_train_resampled)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nClassification Report:")
print(report)
```

1. Sentiment Score to Label Conversion:
- Defines a function get_sentiment_label to convert sentiment scores to sentiment labels (positive, negative, neutral).
- Applies this function to the 'sentiment_score' column of the DataFrame df, storing the result in y_multiclass.
2. TF-IDF Vectorization:
- Uses TF-IDF vectorization (TfidfVectorizer) to convert text data into numerical features (X_tfidf).
3. Data Splitting:
- Splits the data into training and testing sets (X_train, X_test, y_train, y_test) using train_test_split.
4. Oversampling:
- Uses RandomOverSampler to oversample the minority classes in the training data to handle class imbalance.
5. Model Training:
- Initializes and trains a Support Vector Machine (SVM) model (SVC) for multi-class classification using the resampled training data.
6. Prediction:
- Makes predictions (y_pred) using the trained SVM model on the test data.
7. Model Evaluation:
- Computes accuracy and generates a classification report to evaluate the model's performance on the test data.
8. Print Results:
- Prints the accuracy and classification report.

```
# Convert the timestamp column to datetime
df['Timestamp'] = pd.to_datetime(df['Timestamp'], format='%d-%m-%Y %H:%M')

# Initialize the VADER sentiment analyzer
sia = SentimentIntensityAnalyzer()


# Define the time window for trend analysis (e.g., daily, weekly)
time_window = 'D'  # 'D' for daily, 'W' for weekly, 'M' for monthly, etc.

# Aggregate sentiment scores within each time window
df.set_index('Timestamp', inplace=True)
sentiment_trend = df['Sentiment_score'].resample(time_window).mean().fillna(0)

# Separate positive and negative sentiment scores
positive_sentiment = sentiment_trend[sentiment_trend > 0]
negative_sentiment = sentiment_trend[sentiment_trend < 0]

# Plot the trend in sentiment scores over time
plt.figure(figsize=(10, 6))
plt.plot(sentiment_trend.index, sentiment_trend.values, color='gray', 
label='Overall Sentiment')
plt.scatter(positive_sentiment.index, positive_sentiment.values, color='green', 
label='Positive Sentiment')
```

```
plt.scatter(negative_sentiment.index, negative_sentiment.values, color='red',
label='Negative Sentiment')
plt.title('Sentiment Trend Over Time')
plt.xlabel('Date')
plt.ylabel('Average Sentiment Score')
plt.legend()
plt.grid(True)
plt.savefig('sentiment_trend.png')  # Save the plot as an image
plt.show()
```

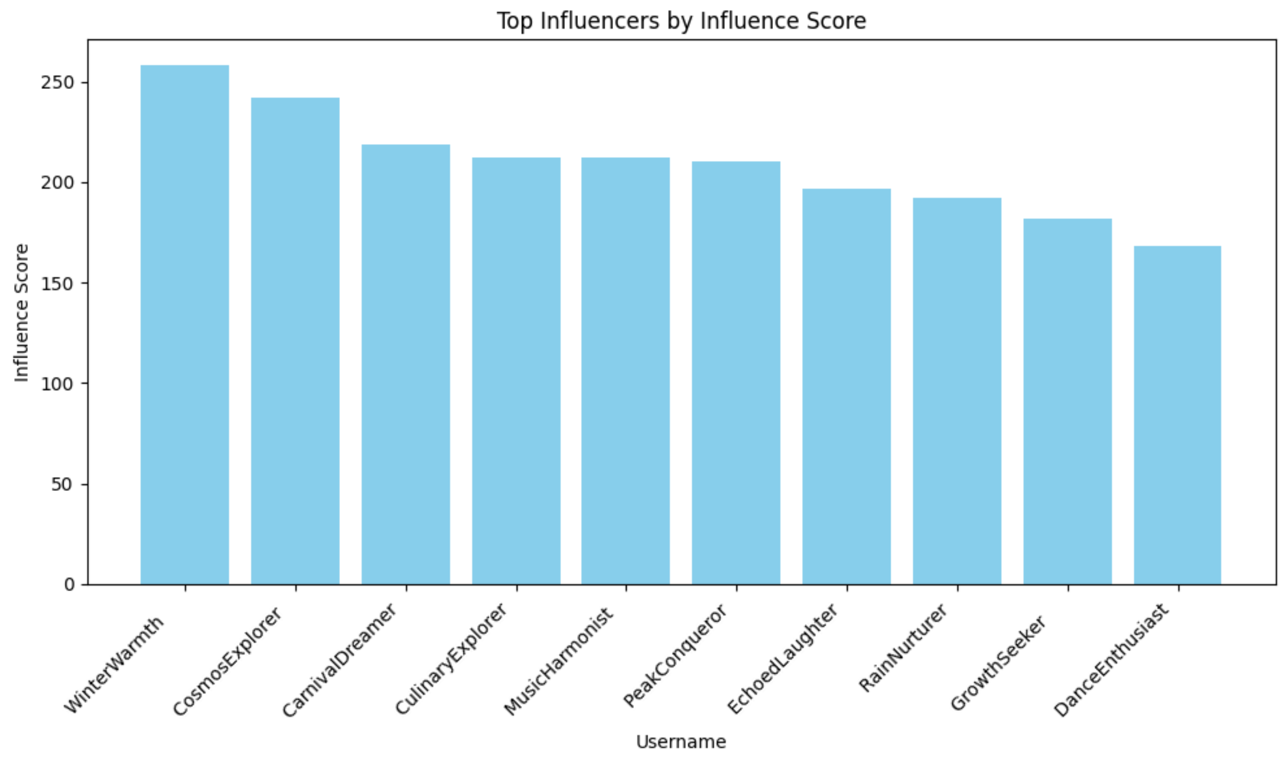
1. Convert Timestamp to Datetime:
- Converts the 'Timestamp' column to datetime format using the pd.to_datetime() function. The specified format is '%d-%m-%Y %H:%M'.
2. Initialize Sentiment Analyzer:
- Initializes the VADER sentiment analyzer (sia = SentimentIntensityAnalyzer()).
3. Define Time Window:
- Specifies the time window for trend analysis, e.g., daily ('D'), weekly ('W'), monthly ('M'), etc.
4. Aggregate Sentiment Scores:
- Sets the 'Timestamp' column as the index of the DataFrame.
- Resamples sentiment scores ('Sentiment_score') within each time window and calculates the mean. Missing values are filled with 0.
- This creates a time series of sentiment trend.
5. Separate Positive and Negative Sentiment:
- Separates positive and negative sentiment scores from the aggregated trend based on their sign.
6. Plot Sentiment Trend:
- Plots the overall sentiment trend over time.
- Overlays scatter plots of positive and negative sentiment on the same graph.
- Adds titles, labels, legends, grid, and saves the plot as an image named 'sentiment_trend.png'.
- Finally, displays the plot using plt.show().



1. Aggregate Metrics:
- Groups the DataFrame df by the 'Username' column.
- Aggregates metrics including total retweets ('Retweets'), total likes ('Likes'), and frequency of mentions ('SR.No' which is renamed to 'Mentions').
- Calculates the sum of retweets and likes, and counts the frequency of mentions for each influencer.
2. Rank Influencers:
- Computes an influence score for each influencer by summing up retweets, likes, and mentions.
- Sorts the influencers based on their influence score in descending order.
3. Visualize Influence Metrics:
- Plots a bar chart showing the top 10 influencers by their influence score.
- The x-axis represents the usernames of influencers, and the y-axis represents the influence score.
- Adds titles, labels, and rotates the x-axis labels for better readability.
4. Display Plot:
- Displays the bar chart using plt.show()

```
# Aggregate metrics: Total retweets, total likes, and frequency of mentions
influence_metrics = df.groupby('Username').agg({
    'Retweets': 'sum',
    'Likes': 'sum',
    'SR.No': 'count'  # Frequency of mentions
}).rename(columns={'SR.No': 'Mentions'}).reset_index()

# Rank influencers based on aggregated metrics
influence_metrics['Influence_score'] = influence_metrics['Retweets'] +
influence_metrics['Likes'] + influence_metrics['Mentions']
influence_metrics.sort_values(by='Influence_score', ascending=False, inplace=True)
```

```
# Visualize influence metrics
plt.figure(figsize=(10, 6))
plt.bar(influence_metrics['Username'][:10],
influence_metrics['Influence_score'][:10], color='skyblue')
plt.title('Top Influencers by Influence Score')
plt.xlabel('Username')
plt.ylabel('Influence Score')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



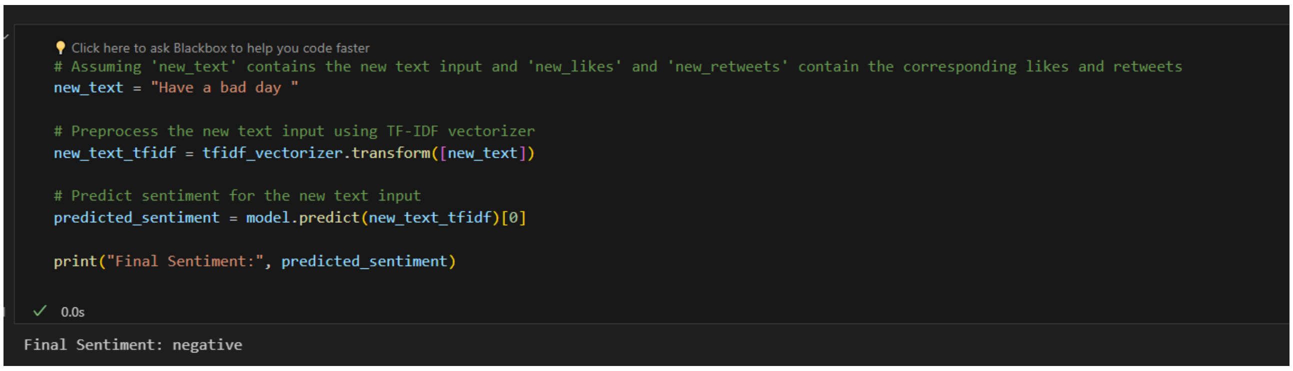
```
# Assuming 'new_text' contains the new text input and 'new_likes' and
'new_retweets' contain the corresponding likes and retweets
new_text = "Have a bad day "

# Preprocess the new text input using TF-IDF vectorizer
new_text_tfidf = tfidf_vectorizer.transform([new_text])

# Predict sentiment for the new text input
predicted_sentiment = model.predict(new_text_tfidf)[0]

print("Final Sentiment:", predicted_sentiment)
```

1. New Text Input:

- The variable new_text contains the new text input for which sentiment prediction is required.

2. Preprocessing with TF-IDF Vectorizer:
- Uses the pre-trained TF-IDF vectorizer (tfidf_vectorizer) to preprocess the new text input.
- Transforms the new text input into TF-IDF vector representation (new_text_tfidf).
3. Sentiment Prediction:
- Utilizes the pre-trained machine learning model (model) to predict the sentiment for the new text input.
- Predicts the sentiment label (predicted_sentiment) based on the TF-IDF vectorized representation of the new text input.
4. Print Result:
- Prints the predicted sentiment for the new text input.

Output:



```python
# Assuming 'new_text' contains the new text input and 'new_likes' and 'new_retweets' contain the corresponding likes and retweets
new_text = "Have a bad day "

# Preprocess the new text input using TF-IDF vectorizer
new_text_tfidf = tfidf_vectorizer.transform([new_text])

# Predict sentiment for the new text input
predicted_sentiment = model.predict(new_text_tfidf)[0]

print("Final Sentiment:", predicted_sentiment)
```

```
Final Sentiment: negative
```