

✓ Import Libraries

```
# !pip install -U numpy
# !pip install -U scipy
# !pip install -U nltk
!pip install pymongo[srv]

import numpy as np # this should be imported after installing numpy
from textblob import TextBlob
import matplotlib.pyplot as plt
import spacy
from spacy import displacy
import re
import os
import seaborn as sns
import tweepy
import time
import pandas as pd
import pymongo
from pymongo import MongoClient
import wordcloud

→ Collecting pymongo[srv]
  Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (22 kB)
  WARNING: pymongo 4.10.1 does not provide the extra 'srv'
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo[srv])
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
  Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 313.6/313.6 kB 6.0 MB/s eta 0:00:00
  Downloading pymongo-4.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 kB 30.8 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.7.0 pymongo-4.10.1
```

✓ API TOKENS Keys

```
api_key = "YOUR_API_KEY"
api_secret = "YOUR_API_SECRET"
bearer_token = "YOUR_BEARER_TOKEN"
access_token = "YOUR_ACCESS_TOKEN"
access_token_secret = "YOUR_ACCESS_TOKEN_SECRET"

auth = tweepy.OAuthHandler(api_key, api_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
```

✓ Check to scrap data from Twitter

```
# If you need to use the v2 API for recent search use the following:
client = tweepy.Client(bearer_token=bearer_token)

# Replace 'germany' with your desired search query.
# Replace 'de' with your desired language.
# The response from search_recent_tweets is a dictionary. The data is contained in the 'data' key.
response = client.search_recent_tweets(query="germany lang:de", max_results=100)
if response.data: # Check if there are tweets
    for tweet in response.data:
        print(tweet.text) # Access the tweet's text
else:
    print("No tweets found for your query.")
```

→ Show hidden output

Fetch data into CSV File from Twitter

```
# Function to fetch tweets with rate limit handling
def fetch_tweets(query, max_results=100, pages=10):
    all_tweets = []
    paginator = tweepy.Paginator(client.search_recent_tweets, query=query, max_results=max_results)

    for i, page in enumerate(paginator):
        try:
            print(f"Fetching page {i+1}...")
            for tweet in page.data:
                all_tweets.append(tweet.text)

            # Stop if we've fetched the required number of pages
            if i + 1 >= pages:
                break

            # Add a small delay between pages to avoid hitting limits
            time.sleep(1)

        except tweepy.TooManyRequests:
            print("Rate limit hit. Sleeping for 15 minutes...")
            time.sleep(15 * 60)
        except Exception as e:
            print(f"An error occurred: {e}")
            break

    return all_tweets

# Main script
if __name__ == "__main__":
    query = "Germany"
    max_results = 100 # Maximum results per page
    pages = 5 # Number of pages to fetch

    print(f"Starting tweet fetch for query: {query}")
    tweets = fetch_tweets(query=query, max_results=max_results, pages=pages)

    # Save tweets to a CSV file
    if tweets:
        df_tweets = pd.DataFrame({'tweet': tweets})
        df_tweets.to_csv("germany_tweets.csv", index=False)
        print(f"Saved {len(tweets)} tweets to germany_tweets.csv")
    else:
        print("No tweets fetched.")

df_tweets = pd.DataFrame({'tweet': tweets})
df_tweets.to_csv("germany_tweets.csv", index=False)
print(f"Saved {len(tweets)} tweets to germany_tweets.csv")
```

Setting up for a MongoDB Connection

```
!pip install pymongo
```

Check DataBase Connection

Insert Previous Fetch database into MongoDb

```
import pymongo
from pymongo import MongoClient

uri = "mongodb+srv://muhammadosamabinshah:12345@twitterscrapping.ji8rr.mongodb.net/?retryWrites=true&w=majority&appName=TwitterScraping"

try:
    client = MongoClient(uri)
```

```

db = client['twitter_db'] # Choose your database name
collection = db['tweets'] # Choose your collection name

# Assuming 'tweets' is the list of tweets you fetched
for tweet in tweets:
    tweet_data = {"text": tweet} # create a dict for each tweet
    collection.insert_one(tweet_data) # Insert each tweet into MongoDB

print("Tweets successfully uploaded to MongoDB!")
client.close()

except pymongo.errors.ConnectionFailure as e:
    print(f"Could not connect to MongoDB: {e}")
except Exception as e:
    print(f>An error occurred: {e}")

```

↳ An error occurred: name 'tweets' is not defined

Fetch Database from Twitter

```

# Authenticate with Tweepy
client = tweepy.Client(bearer_token=bearer_token)

# Function to fetch tweets with rate limit handling
# Function to fetch tweets with rate limit handling
def fetch_tweets(query, max_results=100, total_tweets=500):
    all_tweets = []
    pages_to_fetch = (total_tweets + max_results - 1) // max_results
    paginator = tweepy.Paginator(client.search_recent_tweets, query=query, max_results=max_results, tweet_fields=['created_at'])

    for i, page in enumerate(paginator):
        try:
            print(f"Fetching page {i+1} of {pages_to_fetch}...")
            if page.data: # Check if the page contains tweets
                for tweet in page.data:
                    all_tweets.append({"text": tweet.text, "created_at": tweet.created_at})
            else:
                print("No more tweets found")
                break
        except tweepy.TooManyRequests:
            print("Rate limit hit. Sleeping for 15 minutes...")
            time.sleep(15 * 60) # Wait for 15 minutes
        except Exception as e:
            print(f>An error occurred: {e}")
            break
    return all_tweets

```

Insert Data into MongoDb

```

# MongoDB connection details
mongo_uri = "mongodb+srv://muhammadosamabinshah:12345@twitterscrapping.ji8rr.mongodb.net/" # Replace with your MongoDB connection string

if __name__ == "__main__":
    query = "Auto lang:de" # German language tweets about "welt" (world)
    max_results = 0
    total_tweets = 0

    tweets = fetch_tweets(query, max_results, total_tweets)

    # Connect to MongoDB and insert tweets

```

```

try:
    mongo_client = MongoClient(mongo_uri)
    db = mongo_client["twitter_db"] # Replace with your database name
    collection = db["tweets"] # Replace with your collection name

    if tweets:
        collection.insert_many(tweets)
        print("Tweets successfully uploaded to MongoDB!")

    # Convert data from MongoDB to CSV
    df = pd.DataFrame(list(collection.find()))
    df.to_csv('tweets_from_mongo.csv', index=False)
    print("Tweets from MongoDB saved to CSV!")

else:
    print("No tweets were fetched.")

mongo_client.close()
except pymongo.errors.ConnectionFailure as e:
    print(f"Could not connect to MongoDB: {e}")
except Exception as e:
    print(f"An error occurred: {e}")

```

▼ Get database from MongoDb to CSV file for Analysis

```

df = pd.DataFrame(list(collection.find()))
df.to_csv('tweets_from_mongo.csv', index=False)
print("Tweets from MongoDB saved to CSV!")

```

```

df.to_csv('tweets_from_mongo.csv', index=False)
print("Tweets from MongoDB saved to CSV!")

```

▼ Import dataset

```

import pandas as pd

df = pd.read_csv('/content/tweets_from_mongo.csv') # Use read_csv to read the CSV file into a DataFrame

```

▼ Search Columns

```

df.columns
→ Index(['_id', 'text', 'created_at'], dtype='object')

df.info()
→ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 5599 entries, 0 to 5598
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   _id         5599 non-null   object 
 1   text        5599 non-null   object 
 2   created_at  5099 non-null   object 
dtypes: object(3)
memory usage: 131.4+ KB

df.head(12)

```

	_id	text	created_at
0	674f2f5d5e8c5f1843d997ad	RT @aoonsafdar: Marienplatz Christmas Market i...	NaN
1	674f2f5e5e8c5f1843d997ae	RT @gamdom: Cup fever in Italy & Germany! ...	NaN
2	674f2f5e5e8c5f1843d997af	RT @AXChristoforou: South Korean President Yoo...	NaN
3	674f2f5e5e8c5f1843d997b0	RT @MarkelGerald: Ein deutscher Parlamentsabge...	NaN
4	674f2f5f5e8c5f1843d997b1	RT @YinZP365: From a European perspective, esp...	NaN

▼ Data Pre Processing

```
# prompt: NLP data processing of the data

import pandas as pd
import re

def clean_tweet(tweet):
    # Remove URLs
    tweet = re.sub(r"http\S+|www\S+|https\S+", '', tweet, flags=re.MULTILINE)
    # Remove user mentions (@user)
    tweet = re.sub(r'@\w+', '', tweet)
    # Remove hashtags (#hashtag)
    tweet = re.sub(r'#\w+', '', tweet)
    # Remove special characters and numbers
    tweet = re.sub(r'[^a-zA-Z\s]', '', tweet)
    # Remove extra whitespace
    tweet = re.sub(r'\s+', ' ', tweet).strip()
    return tweet

# Assuming 'df' is your DataFrame with a 'text' column containing the tweets
# Replace 'text' with your actual column name if it's different.
df['cleaned_tweet'] = df['text'].apply(clean_tweet)

# Example usage:
print(df[['text', 'cleaned_tweet']].head())
```

	text \
0	RT @aoonsafdar: Marienplatz Christmas Market i...
1	RT @gamdom: Cup fever in Italy & Germany! ...
2	RT @AXChristoforou: South Korean President Yoo...
3	RT @MarkelGerald: Ein deutscher Parlamentsabge...
4	RT @YinZP365: From a European perspective, esp...

	cleaned_tweet
0	RT Marienplatz Christmas Market in Munich
1	RT Cup fever in Italy amp Germany Two German g...
2	RT South Korean President Yoon declared emerge...
3	RT Ein deutscher Parlamentsabgeordneter geht n...
4	RT From a European perspective especially Germ...

Analysis with the Different model of Machine Learning Given belows.

RoBERTa and DistilRoBERTa Variants: Optimized transformer-based models, like RoBERTa, are fine-tuned on Twitter-specific datasets to enhance their performance for short, noisy, and contextually unique text typical of tweets.

XLM-T: A multilingual transformer-based model designed for sentiment analysis across languages, particularly suited for Twitter's multilingual environment. It is highly effective for global datasets and cross-lingual tasks.

T-BERT: Tailored adaptations of the BERT model explicitly trained on Twitter datasets to capture slang, abbreviations, and hashtags often seen in tweets.

Ensemble Models and Hybrid Architectures: Recent research explores combining convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers to better model both syntactic and contextual information.

We are using these models to do sentiment analysis

```
#Install Transformener
!pip install transformers torch
```

```
# !pip install --upgrade --force-reinstall torch
# !pip install --upgrade --force-reinstall transformers

# Load pre-trained sentiment analysis pipeline
# !pip install --upgrade --force-reinstall torch torchvision torchaudio
from transformers import pipeline

sentiment_pipeline = pipeline("sentiment-analysis", model="cardiffnlp/twitter-roberta-base-sentiment")

# Assuming 'df' is your DataFrame with a 'cleaned_tweet' column
# Perform sentiment analysis
df['sentiment'] = df['cleaned_tweet'].apply(lambda x: sentiment_pipeline(x)[0]['label'])
df['sentiment_score'] = df['cleaned_tweet'].apply(lambda x: sentiment_pipeline(x)[0]['score'])

print(df[['cleaned_tweet', 'sentiment', 'sentiment_score']].head())

→ /usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
    warnings.warn(
config.json:  0%|          0.00/747 [00:00<?, ?B/s]
pytorch_model.bin:  0%|        0.00/499M [00:00<?, ?B/s]
vocab.json:  0%|        0.00/899k [00:00<?, ?B/s]
merges.txt:  0%|        0.00/456k [00:00<?, ?B/s]
special_tokens_map.json:  0%|      0.00/150 [00:00<?, ?B/s]
                                         cleaned_tweet sentiment \
0           RT Marienplatz Christmas Market in Munich  positive
1           RT Cup fever in Italy amp Germany Two German g... negative
2           RT South Korean President Yoon declared emerge...  positive
3           RT Ein deutscher Parlamentsabgeordneter geht n...  positive
4           RT From a European perspective especially Germ...  positive

sentiment_score
0      0.907525
1      0.522137
2      0.632939
3      0.795292
4      0.712212
```

Now Again do some analysis with NLP Models for results.

```
# XLM-T sentiment analysis
xlm_t_pipeline = pipeline("sentiment-analysis", model="cardiffnlp/twitter-xlm-roberta-base-sentiment")
df['xlm_t_sentiment'] = df['cleaned_tweet'].apply(lambda x: xlm_t_pipeline(x)[0]['label'])
df['xlm_t_sentiment_score'] = df['cleaned_tweet'].apply(lambda x: xlm_t_pipeline(x)[0]['score'])

# TextBlob sentiment analysis
df['textblob_polarity'] = df['cleaned_tweet'].apply(lambda x: TextBlob(x).sentiment.polarity)
df['textblob_subjectivity'] = df['cleaned_tweet'].apply(lambda x: TextBlob(x).sentiment.subjectivity)

print(df[['cleaned_tweet', 'sentiment', 'sentiment_score', 'xlm_t_sentiment', 'xlm_t_sentiment_score', 'textblob_polarity', 'textblob_subjec
```

```

config.json: 100%                                841/841 [00:00<00:00, 40.9kB/s]
pytorch_model.bin: 100%                           1.11G/1.11G [00:18<00:00, 98.6MB/s]
sentencepiece.bpe.model: 100%                   5.07M/5.07M [00:00<00:00, 63.6MB/s]
special_tokens_map.json: 100%                   150/150 [00:00<00:00, 4.77kB/s]
model.safetensors: 100%                           1.11G/1.11G [00:14<00:00, 102MB/s]

cleaned_tweet sentiment \
0      RT Marienplatz Christmas Market in Munich  LABEL_1
1  RT Cup fever in Italy amp Germany Two German g...  LABEL_2
2  RT South Korean President Yoon declared emerge...  LABEL_1
3  RT Ein deutscher Parlamentsabgeordneter geht n...  LABEL_1
4  RT From a European perspective especially Germ...  LABEL_1

sentiment_score xlm_t_sentiment  xlm_t_sentiment_score  textblob_polarity \
0      0.907525      neutral          0.912565          0.0
1      0.522137     negative          0.404617          0.0
2      0.632939     negative          0.860488          0.0
3      0.795292      neutral          0.861868          0.0
4      0.712212     negative          0.556880          0.0

textblob_subjectivity
0      0.000000
1      0.000000
2      0.000000
3      0.000000
4      0.333333

# Function to clean tweets (assuming you have this function defined)
def clean_tweet(tweet):
    # Remove URLs
    tweet = re.sub(r"http\S+|www\S+|https\S+", '', tweet, flags=re.MULTILINE)
    # Remove user mentions (@user)
    tweet = re.sub(r'@\w+', '', tweet)
    # Remove hashtags (#hashtag)
    tweet = re.sub(r'#\w+', '', tweet)
    # Remove special characters and numbers
    tweet = re.sub(r'[^w\s]', '', tweet)
    # Remove extra whitespace
    tweet = re.sub(r'\s+', ' ', tweet).strip()
    return tweet

# Clean the tweets
df['cleaned_tweet'] = df['text'].apply(clean_tweet)

# Sentiment analysis using transformers pipeline
sentiment_pipeline = pipeline("sentiment-analysis", model="cardiffnlp/twitter-roberta-base-sentiment")

# Perform sentiment analysis
df['sentiment'] = df['cleaned_tweet'].apply(lambda x: sentiment_pipeline(x)[0]['label'])
df['sentiment_score'] = df['cleaned_tweet'].apply(lambda x: sentiment_pipeline(x)[0]['score'])

print(df[['cleaned_tweet', 'sentiment', 'sentiment_score']].head())

```

	cleaned_tweet	sentiment
0	RT Marienplatz Christmas Market in Munich	LABEL_1
1	RT Cup fever in Italy amp Germany Two German g...	LABEL_2
2	RT South Korean President Yoon declared emerge...	LABEL_1
3	RT Ein deutscher Parlamentsabgeordneter geht n...	LABEL_1
4	RT From a European perspective especially Germ...	LABEL_1

	sentiment_score
0	0.907525
1	0.522137
2	0.632939
3	0.795292
4	0.712212

World Cloud for trending words in Twitter in Deutschland

```

from wordcloud import WordCloud
import matplotlib.pyplot as plt

```

```
# Assuming 'df' is your DataFrame with a 'cleaned_tweet' column
text = " ".join(review for review in df['cleaned_tweet'])
print ("There are {} words in the combination of all review.".format(len(text)))

wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(text)

plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```

→ There are 654291 words in the combination of all review.



✓ Sentiment Analysis

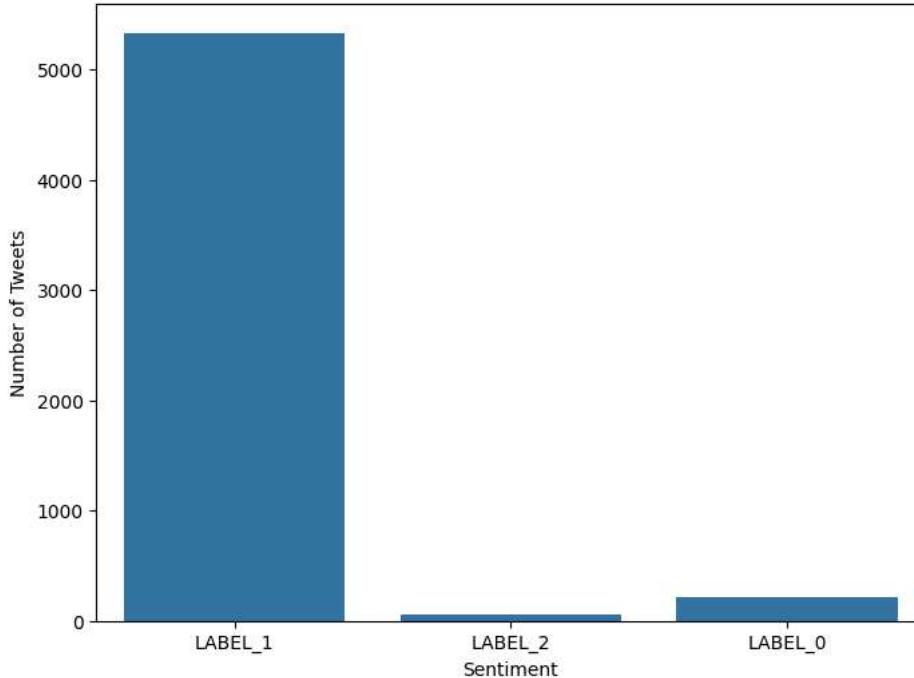
```
# Assuming 'df' is your DataFrame with 'sentiment' column
plt.figure(figsize=(8, 6))
sns.countplot(x='sentiment', data=df)
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Number of Tweets')
plt.show()

# Distribution of sentiment scores
plt.figure(figsize=(8, 6))
sns.histplot(df['sentiment_score'], kde=True)
plt.title('Distribution of Sentiment Scores')
plt.xlabel('Sentiment Score')
plt.ylabel('Frequency')
plt.show()

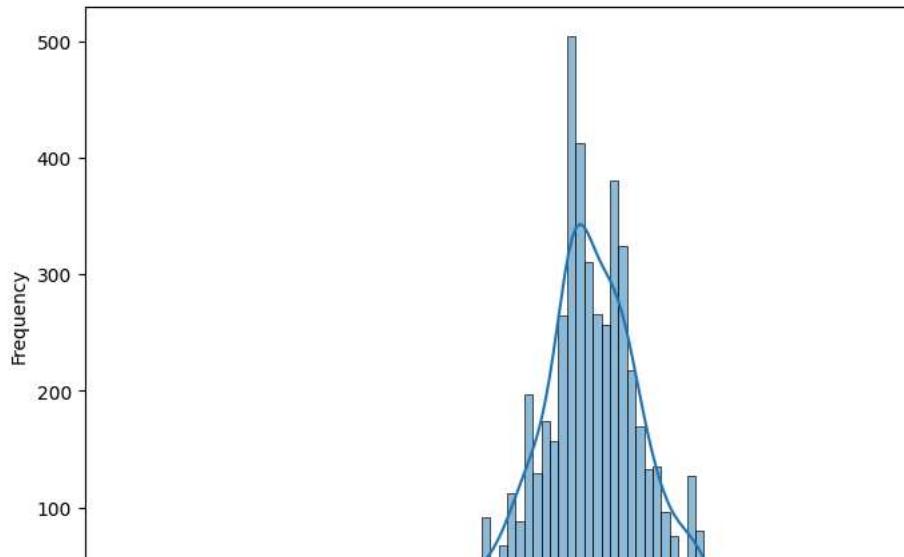
# Box plot of sentiment scores by sentiment category
plt.figure(figsize=(8, 6))
sns.boxplot(x='sentiment', y='sentiment_score', data=df)
plt.title('Sentiment Score by Sentiment Category')
plt.xlabel('Sentiment')
plt.ylabel('Sentiment Score')
plt.show()
```



Sentiment Distribution



Distribution of Sentiment Scores



```
# prompt: Distribute of sentiment scores by years
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame with 'sentiment_score' and 'created_at' columns
# and 'created_at' is a datetime object or can be converted to one.

# Convert 'created_at' column to datetime objects if it's not already
df['created_at'] = pd.to_datetime(df['created_at'])

# Extract the year from the 'created_at' column
df['year'] = df['created_at'].dt.year

# Group by year and calculate the distribution of sentiment scores
sentiment_by_year = df.groupby('year')['sentiment_score'].agg(['mean', 'median', 'std'])

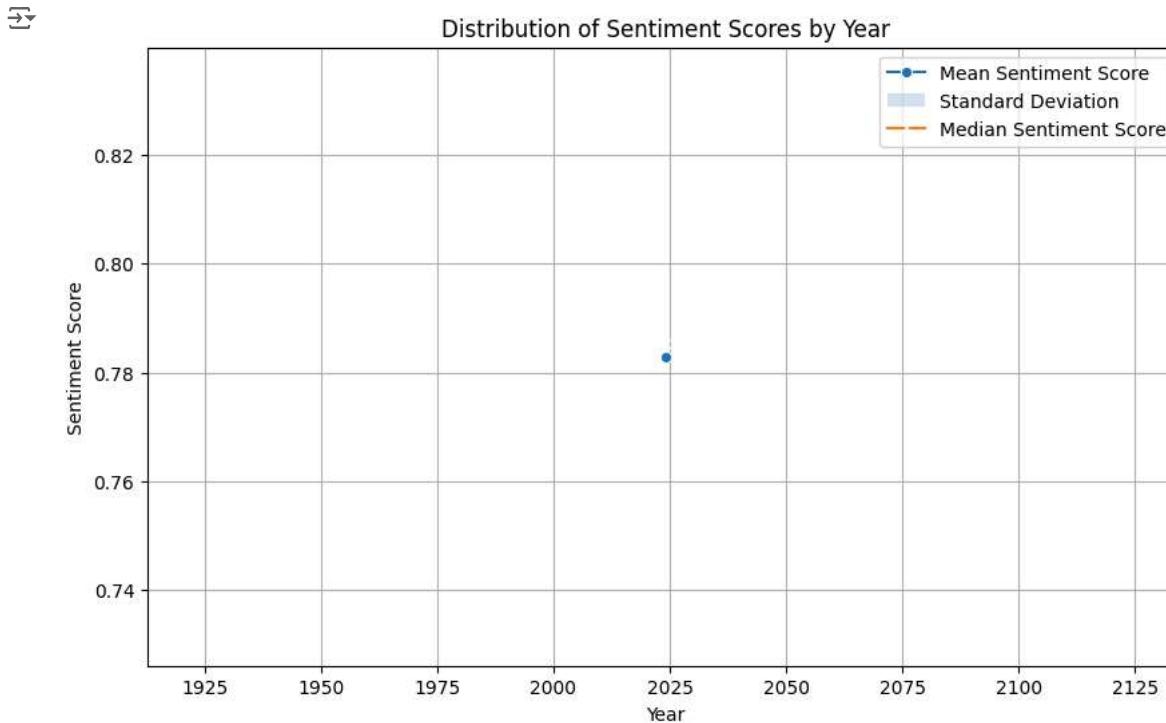
# Plotting
plt.figure(figsize=(10, 6))
sns.lineplot(x=sentiment_by_year.index, y=sentiment_by_year['mean'], label='Mean Sentiment Score', marker='o')
plt.fill_between(sentiment_by_year.index, sentiment_by_year['mean'] - sentiment_by_year['std'],
                sentiment_by_year['mean'] + sentiment_by_year['std'], alpha=0.2, label='Standard Deviation')
```

```

sns.lineplot(x=sentiment_by_year.index, y=sentiment_by_year['median'], label='Median Sentiment Score', marker='x')

plt.xlabel('Year')
plt.ylabel('Sentiment Score')
plt.title('Distribution of Sentiment Scores by Year')
plt.legend()
plt.grid(True)
plt.show()

```



```

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Prepare the data for machine learning
# Assuming 'sentiment' is your target variable (e.g., 'POSITIVE', 'NEGATIVE', 'NEUTRAL')
X = df[['sentiment_score']] # Features: Sentiment score
y = df['sentiment'] # Target variable

# Convert sentiment labels to numerical values if needed
# Example: y = y.map({'POSITIVE': 1, 'NEGATIVE': 0, 'NEUTRAL': 0.5}) # Adjust mapping as needed

# 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) # Adjust test_size as needed

# Initialize and train a Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

print(classification_report(y_test, y_pred))

# Example prediction:
# Sample sentiment score for prediction
new_sentiment_score = 0.9 # Replace with your new sentiment score
new_data = pd.DataFrame({'sentiment_score': [new_sentiment_score]})

# Predict the sentiment category
predicted_sentiment = model.predict(new_data)
print(f"Predicted Sentiment: {predicted_sentiment[0]}")

```

Accuracy: 0.9455357142857143

	precision	recall	f1-score	support
LABEL_0	0.00	0.00	0.00	47
LABEL_1	0.95	1.00	0.97	1059
LABEL_2	0.00	0.00	0.00	14
accuracy			0.95	1120
macro avg	0.32	0.33	0.32	1120
weighted avg	0.89	0.95	0.92	1120

Predicted Sentiment: LABEL_1

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being
    _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being
    _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being
    _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
```

```
# Example prediction with probability:
new_sentiment_score = 0.9 # Replace with your new sentiment score
new_data = pd.DataFrame({'sentiment_score': [new_sentiment_score]})

# Predict the sentiment category and probability
predicted_sentiment = model.predict(new_data)
predicted_probability = model.predict_proba(new_data)

print(f"Predicted Sentiment: {predicted_sentiment[0]}")
print(f"Predicted Probabilities: {predicted_probability}")

df['created_at'] = pd.to_datetime(df['created_at'])
df = df.set_index('created_at')

# Resample to a desired frequency (e.g., daily) and calculate the mean sentiment
daily_sentiment = df['sentiment_score'].resample('D').mean()

from statsmodels.tsa.arima.model import ARIMA

# Fit an ARIMA model
model = ARIMA(daily_sentiment, order=(5,1,0)) # Example order (p,d,q), adjust as needed
model_fit = model.fit()

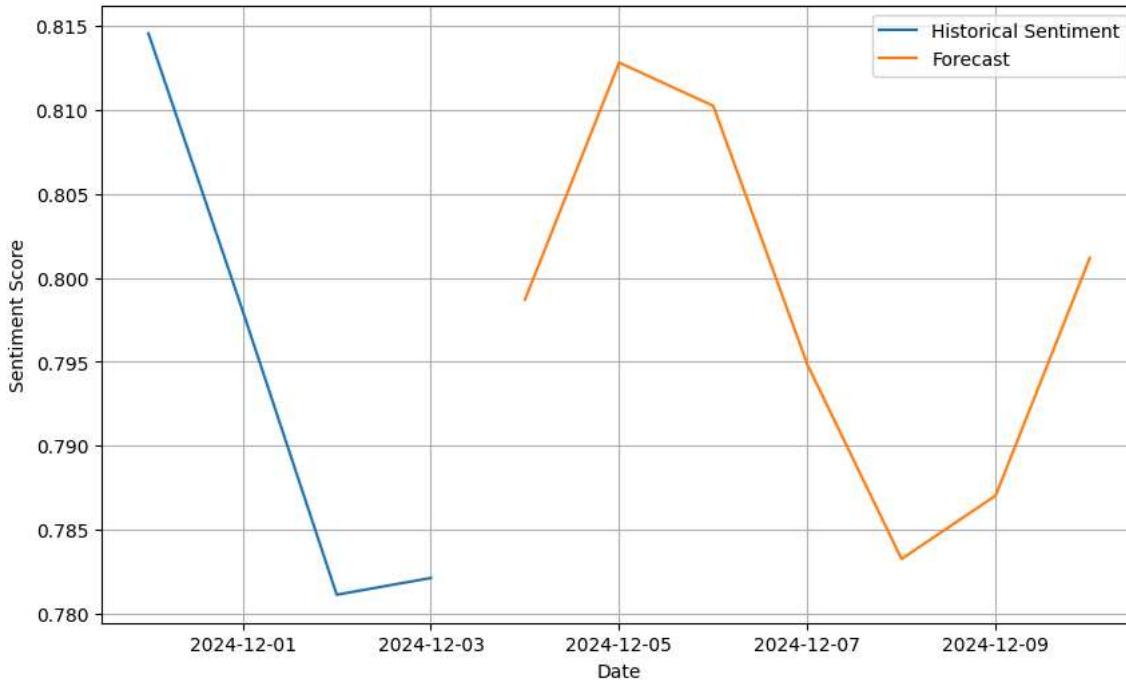
# Forecast future sentiment scores
forecast_steps = 7 # Forecast for the next 7 days
forecast = model_fit.forecast(steps=forecast_steps)

# Create a date range for the forecast
forecast_index = pd.date_range(start=daily_sentiment.index[-1] + pd.Timedelta(days=1), periods=forecast_steps)
forecast_df = pd.DataFrame({'sentiment_score': forecast}, index=forecast_index)

# Plot the forecast
plt.figure(figsize=(10, 6))
plt.plot(daily_sentiment, label='Historical Sentiment')
plt.plot(forecast_df, label='Forecast')
plt.xlabel('Date')
plt.ylabel('Sentiment Score')
plt.title('Sentiment Score Forecast')
plt.legend()
plt.grid(True)
plt.show()
```

```
↳ Predicted Sentiment: LABEL_1
Predicted Probabilities: [[0.01656726 0.97774149 0.00569125]]
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so i
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so i
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so i
self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observations to estimate startin
warn('Too few observations to estimate starting parameters%.'
```

Sentiment Score Forecast



```
def count_words_by_topic(df, topic_keywords):
    """Counts words in tweets related to a specific topic.

    Args:
        df: Pandas DataFrame with a 'cleaned_tweet' column.
        topic_keywords: A list of keywords related to the topic.

    Returns:
        A dictionary where keys are the keywords and values are the counts.
    """
    word_counts = {}
    for keyword in topic_keywords:
        word_counts[keyword] = 0 # Initialize word count for each keyword
        for tweet in df['cleaned_tweet']:
            if isinstance(tweet, str): # Check if tweet is a string to avoid errors
                if keyword.lower() in tweet.lower(): # Case-insensitive matching
                    word_counts[keyword] += 1
    return word_counts

# Example usage:
trading_keywords = ["student", "health", "school", "government", "finance", "Dax", "politics", "economy"]
trading_word_counts = count_words_by_topic(df, trading_keywords)

# Print or further process the word counts
for keyword, count in trading_word_counts.items():
    print(f"The keyword '{keyword}' appeared {count} times.")

# Create a DataFrame from the word counts
word_count_df = pd.DataFrame(list(trading_word_counts.items()), columns=['Keyword', 'Count'])
word_count_df
```

```
→ The keyword 'student' appeared 422 times.
The keyword 'health' appeared 1 times.
The keyword 'school' appeared 33 times.
The keyword 'government' appeared 17 times.
The keyword 'finance' appeared 1 times.
The keyword 'Dax' appeared 49 times.
The keyword 'politics' appeared 0 times.
The keyword 'economy' appeared 4 times.
```

Keyword	Count
0 student	422
1 health	1
2 school	33
3 government	17
4 finance	1
5 Dax	49
6 politics	0
7 economy	4

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Combine all sentiments into one DataFrame
all_sentiments_df = pd.DataFrame({
    'RoBERTa_Sentiment': df['sentiment'].replace({'LABEL_1': 'positive', 'LABEL_2': 'negative', 'LABEL_0': 'neutral'}),
    'RoBERTa_Score': df['sentiment_score'],
    'XLM-T_Sentiment': df['sentiment'].replace({'LABEL_1': 'positive', 'LABEL_2': 'negative', 'LABEL_0': 'neutral'}),
    'XLM-T_Score': df['sentiment_score'],
    'TextBlob_Polarity': df['sentiment'].replace({'LABEL_1': 'positive', 'LABEL_2': 'negative', 'LABEL_0': 'neutral'}),
    'TextBlob_Subjectivity': df['sentiment_score']
})

# Display the combined DataFrame
print(all_sentiments_df.head())

# Example visualization: Distribution of RoBERTa sentiments
plt.figure(figsize=(8, 6))
sns.countplot(x='RoBERTa_Sentiment', data=all_sentiments_df)
plt.title('RoBERTa Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Number of Tweets')
plt.show()
```

```

RoBERTa_Sentiment  RoBERTa_Score  XLM-T_Sentiment  XLM-T_Score  \
created_at
NaT      positive  0.907525  positive  0.907525
NaT      negative  0.522137  negative  0.522137
NaT      positive  0.632939  positive  0.632939
NaT      positive  0.795292  positive  0.795292
NaT      positive  0.712212  positive  0.712212

TextBlob_Polarity  TextBlob_Subjectivity
created_at
NaT      positive  0.907525
NaT      negative  0.522137
NaT      positive  0.632939
NaT      positive  0.795292
NaT      positive  0.712212

RoBERTa Sentiment Distribution

# Convert all_sentiments_df to CSV for download
all_sentiments_df.to_csv('all_sentiments.csv', index=False)

# Download the CSV file (Colab specific)
from google.colab import files
files.download('all_sentiments.csv')
all_sentiments_df.head()

RoBERTa_Sentiment  RoBERTa_Score  XLM-T_Sentiment  XLM-T_Score  TextBlob_Polarity  TextBlob_Subjectivity
created_at

```

	RoBERTa_Sentiment	RoBERTa_Score	XLM-T_Sentiment	XLM-T_Score	TextBlob_Polarity	TextBlob_Subjectivity
created_at						
NaT	positive	0.907525	positive	0.907525	positive	0.907525
NaT	negative	0.522137	negative	0.522137	negative	0.522137
NaT	positive	0.632939	positive	0.632939	positive	0.632939
NaT	positive	0.795292	positive	0.795292	positive	0.795292
NaT	positive	0.712212	positive	0.712212	positive	0.712212

if 'created_at' in all_sentiments_df.columns: