TWITTER SENTIMENT ANALYSIS

Steps to Complete the Task

1. **Data Collection:** Gather Twitter data related to specific topics.
2. **Data Preprocessing:** Clean and prepare the data for analysis.
3. **Sentiment Labeling:** Assign sentiment labels (positive, negative, neutral) to the tweets.
4. **Feature Extraction:** Convert text data into numerical features.
5. **Model Building:** Train a machine learning model for sentiment classification.
6. **Evaluation:** Evaluate the model's performance.
7. **Visualization:** Visualize the results to understand public sentiment.

Step 1: Data Collection

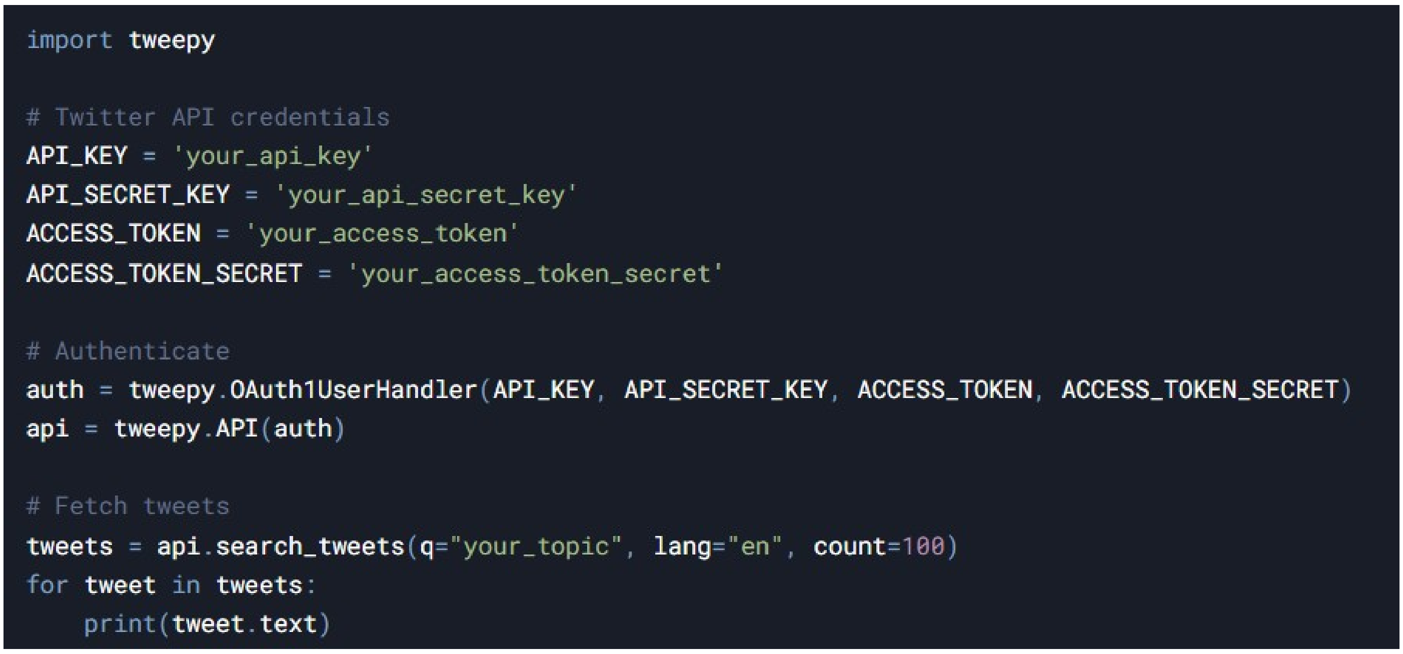
You can use the Twitter API (now called X API) to collect tweets. Alternatively, you can use pre-existing datasets like:

- Sentiment140
- Kaggle Twitter Sentiment Analysis Dataset

If you want to collect live data, you'll need to:

1. Create a Twitter Developer account.
2. Use the tweepy library in Python to fetch tweets.

Example code to fetch tweets:



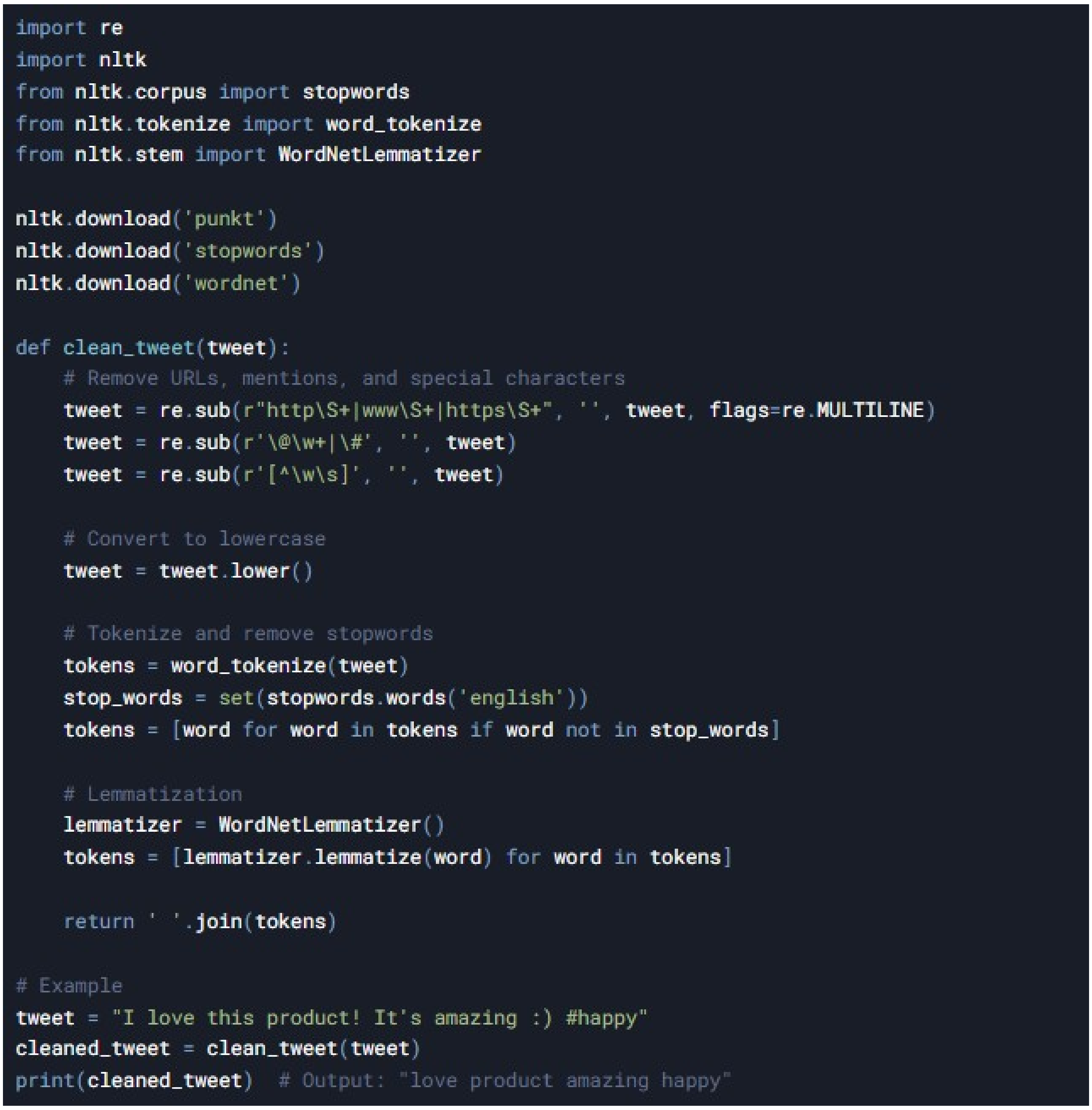
```python
import tweepy

# Twitter API credentials
API_KEY = 'your_api_key'
API_SECRET_KEY = 'your_api_secret_key'
ACCESS_TOKEN = 'your_access_token'
ACCESS_TOKEN_SECRET = 'your_access_token_secret'

# Authenticate
auth = tweepy.OAuth1UserHandler(API_KEY, API_SECRET_KEY, ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
api = tweepy.API(auth)

# Fetch tweets
tweets = api.search_tweets(q="your_topic", lang="en", count=100)
for tweet in tweets:
    print(tweet.text)
```

Step 2: Data Preprocessing

Clean the tweets to remove noise:

- Remove URLs, mentions, and special characters.
- Convert text to lowercase.
- Remove stop words (e.g., "the", "and").
- Perform tokenization and stemming/lemmatization.

Example code:



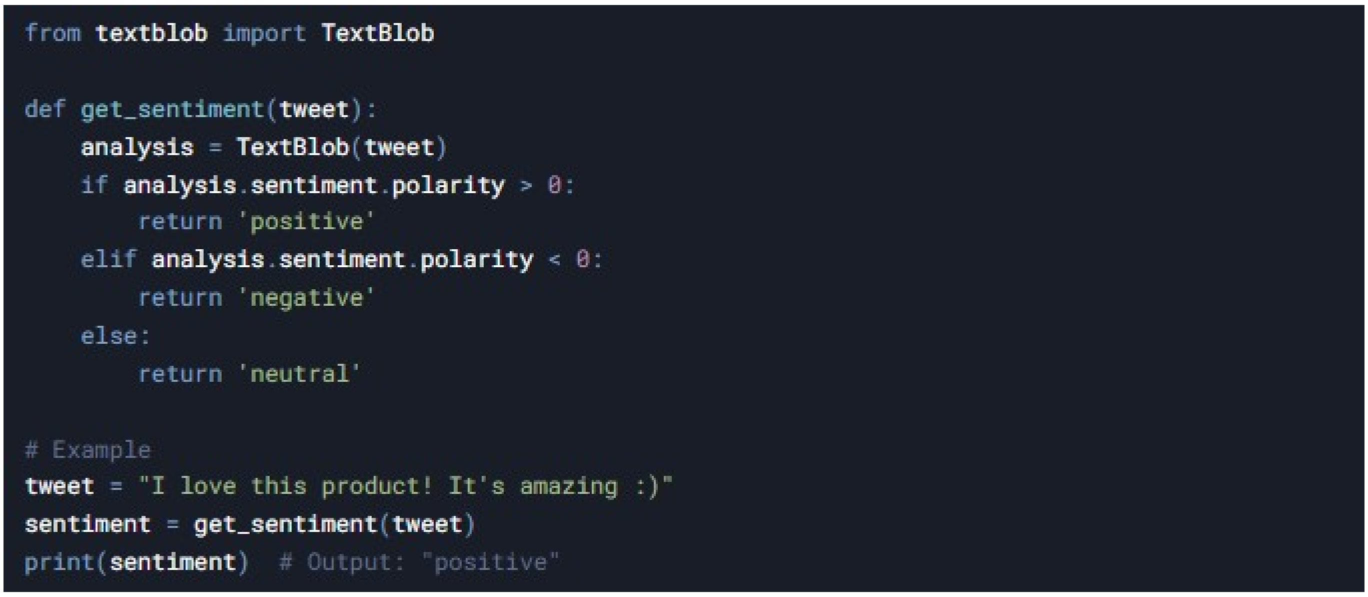
```python
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

def clean_tweet(tweet):
    # Remove URLs, mentions, and special characters
    tweet = re.sub(r"http\S+|www\S+|https\S+", '', tweet, flags=re.MULTILINE)
    tweet = re.sub(r'\@\w+|\#', '', tweet)
    tweet = re.sub(r'[^\w\s]', '', tweet)

    # Convert to lowercase
    tweet = tweet.lower()

    # Tokenize and remove stopwords
    tokens = word_tokenize(tweet)
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    return ' '.join(tokens)

# Example
tweet = "I love this product! It's amazing :) #happy"
cleaned_tweet = clean_tweet(tweet)
print(cleaned_tweet)  # Output: "love product amazing happy"
```

Step 3: Sentiment Labeling

If your dataset doesn’t have labels, you can use a pre-trained sentiment analysis tool like:

- TextBlob
- VADER (Valence Aware Dictionary and sEntiment Reasoner

Example using TextBlob:



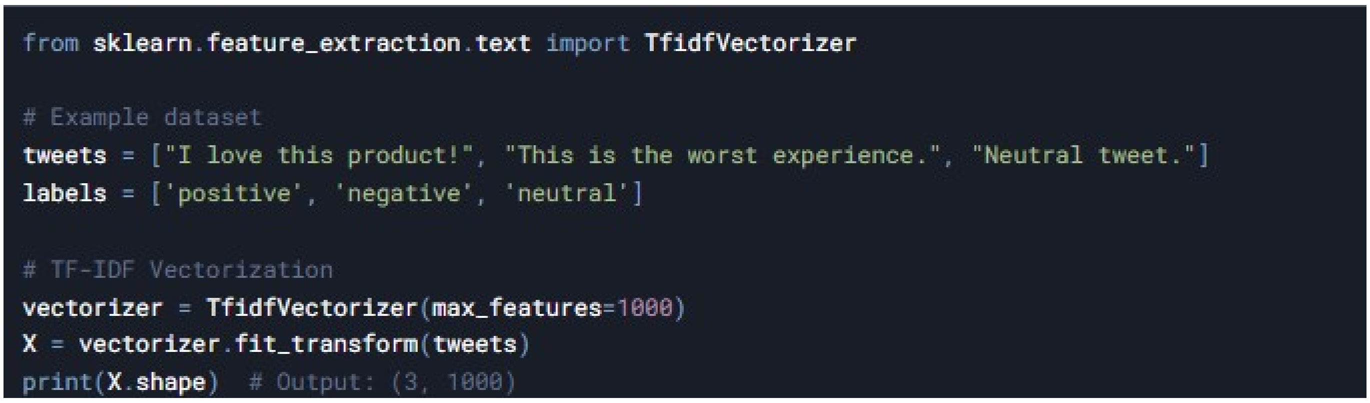
```python
from textblob import TextBlob

def get_sentiment(tweet):
    analysis = TextBlob(tweet)
    if analysis.sentiment.polarity > 0:
        return 'positive'
    elif analysis.sentiment.polarity < 0:
        return 'negative'
    else:
        return 'neutral'

# Example
tweet = "I love this product! It's amazing :)"
sentiment = get_sentiment(tweet)
print(sentiment)  # Output: "positive"
```

Step 4: Feature Extraction

Convert text into numerical features using techniques like:

- Bag of Words (BoW)
- TF-IDF
- Word Embeddings (e.g., Word2Vec, GloVe)

Example using TF-IDF:



```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Example dataset
tweets = ["I love this product!", "This is the worst experience.", "Neutral tweet."]
labels = ['positive', 'negative', 'neutral']

# TF-IDF Vectorization
vectorizer = TfidfVectorizer(max_features=1000)
X = vectorizer.fit_transform(tweets)
print(X.shape)  # Output: (3, 1000)
```

Step 5: Model Building

Train a machine learning model for sentiment classification. Common models include:

- Logistic Regression
- Naive Bayes
- Support Vector Machines (SVM)
- Deep Learning (e.g., LSTM, BERT)

Example using Logistic Regression:

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

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Train model
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Step 6: Evaluation

Evaluate the model using metrics like:

- Accuracy
- Precision, Recall, F1-Score
- Confusion Matrix

Step 7: Visualization

Visualize the results using libraries like matplotlib or seaborn. For example:

- Plot the distribution of sentiments (positive, negative, neutral).
- Show word clouds for each sentiment.

Example code for a word cloud:

```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(tweets))

# Plot
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Further Steps

1. Finalize Data Collection

- If using the **Twitter API,** replace placeholders with your API keys and fetch real tweets.
- If using a **pre-existing dataset** (e.g., Sentiment140), load it into a DataFrame.

```
import pandas as pd
df = pd.read_csv('twitter_dataset.csv', encoding='latin1')  # Example for Sentiment140
```

2. Preprocess the Entire Dataset

- Apply the **clean_tweet()** function to all tweets in your dataset.

```
df['cleaned_text'] = df['tweet_column'].apply(clean_tweet)
```

3. Assign Sentiment Labels (If Not Labeled)

- Use **TextBlob/VADER** to auto-label sentiments (if your dataset is unlabeled).

```
df['sentiment'] = df['cleaned_text'].apply(get_sentiment)
```

4. Train and Evaluate the Model

- Vectorize text (TF-IDF/BOW) and split into train/test sets.
- Train a model (e.g., Logistic Regression) and evaluate metrics (accuracy, F1-score).

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

5. Visualize Results

- Create plots:
  - **Sentiment distribution** (bar chart).
  - **Word clouds** for positive/negative tweets.
  - **Confusion matrix heatmap**.

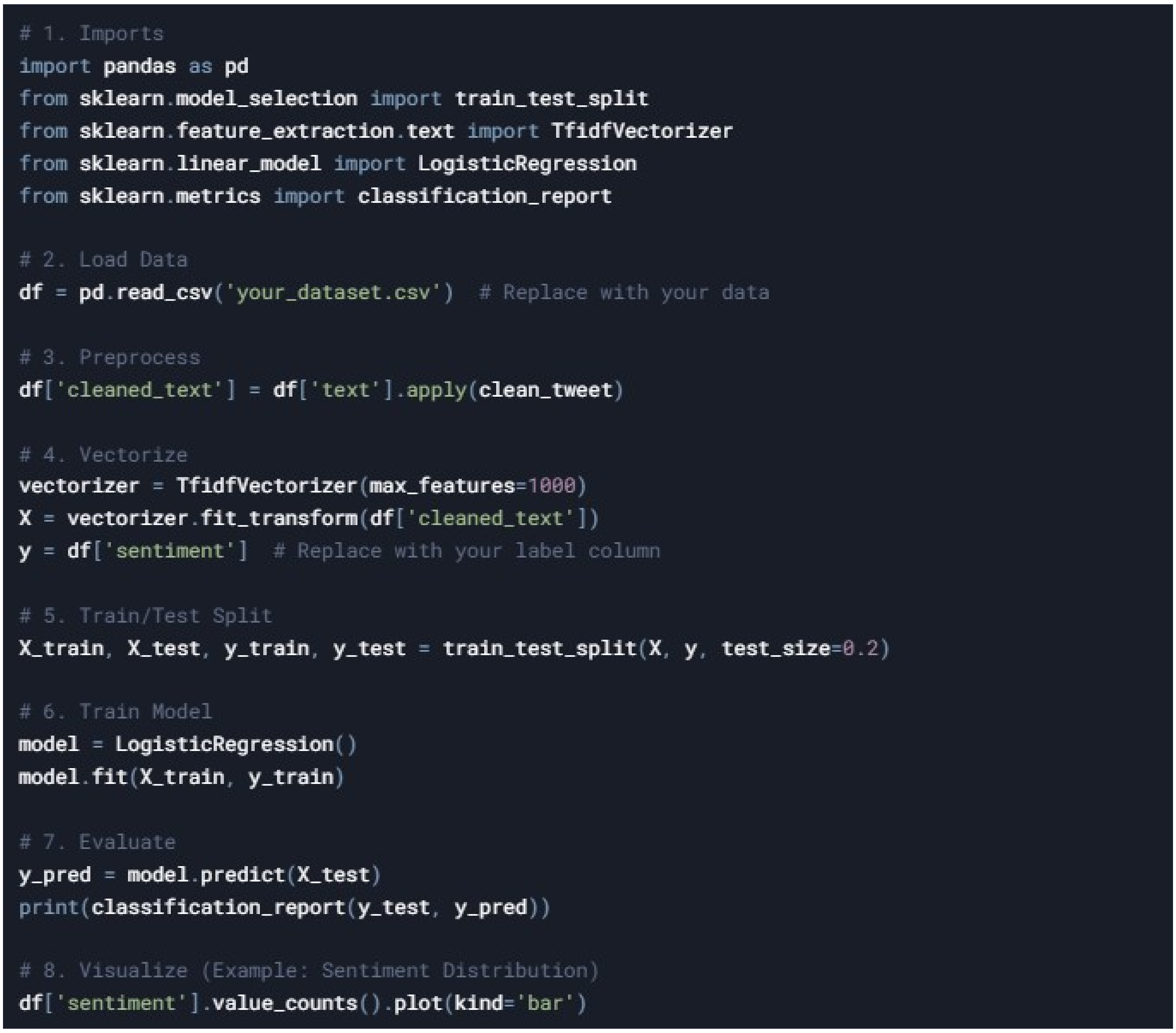
6. Test on New Tweets (Optional but Recommended)

- Deploy the model to analyze live tweets or new data.

```
new_tweet = "This movie was terrible!"
cleaned = clean_tweet(new_tweet)
vectorized = vectorizer.transform([cleaned])
print(model.predict(vectorized))  # Output: 'negative'
```

Final Deliverable Template

Here’s the complete structure of the script:



```python
# 1. Imports
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# 2. Load Data
df = pd.read_csv('your_dataset.csv')  # Replace with your data

# 3. Preprocess
df['cleaned_text'] = df['text'].apply(clean_tweet)

# 4. Vectorize
vectorizer = TfidfVectorizer(max_features=1000)
X = vectorizer.fit_transform(df['cleaned_text'])
y = df['sentiment']  # Replace with your label column

# 5. Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# 6. Train Model
model = LogisticRegression()
model.fit(X_train, y_train)

# 7. Evaluate
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

# 8. Visualize (Example: Sentiment Distribution)
df['sentiment'].value_counts().plot(kind='bar')
```