```
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

Helper Function for Text Cleaning:

Implement a Helper Function as per Text Preprocessing Notebook and Complete the following pipeline.

Build a Text Cleaning Pipeline

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer, PorterStemmer

# Download necessary NLTK resources
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

# Define stop words
stop_words = set(stopwords.words('english'))

def text_cleaning_pipeline(dataset, rule="lemmatize"):
  """
  This function performs a complete text cleaning process including:
  - Lowercasing
  - Removing URLs, emojis, and unwanted characters
  - Tokenization
  - Stopword removal
  - Lemmatization or stemming

  Args:
  dataset (str): Input text to be cleaned
  rule (str): "lemmatize" (default) or "stem" to apply desired
transformation

  Returns:
  str: Cleaned text string
  """

  def lower_order(text):
    return text.lower()

  def remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', text)

  def remove_emoji(string):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols &
pictographs
                               u"\U0001F680-\U0001F6FF"  # transport &
map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r' ', string)

  def removeunwanted_characters(document):
    document = re.sub("@[A-Za-z0-9_]+", " ", document)
    document = re.sub("#[A-Za-z0-9_]+", "", document)
    document = re.sub("[^0-9A-Za-z ]", "", document)
    document = remove_emoji(document)
    document = document.replace('  ', " ")
    return document.strip()

  def remove_punct(text):
    tokenizer = RegexpTokenizer(r"\w+")
    lst = tokenizer.tokenize(' '.join(text)) if isinstance(text, list)
else tokenizer.tokenize(text)
    return lst

  def remove_stopwords(text_tokens):
    return [token for token in text_tokens if token not in stop_words]

  def lemmatization(token_text):
    wordnet = WordNetLemmatizer()
    lemmatized_tokens = [wordnet.lemmatize(token, pos='v') for token
in token_text]
    return lemmatized_tokens

  def stemming(text):
    porter = PorterStemmer()
    stemm_tokens = [porter.stem(word) for word in text]
    return stemm_tokens

  # --- Actual pipeline flow ---
  data = lower_order(dataset)
  data = remove_urls(data)
  data = remove_emoji(data)
  data = removeunwanted_characters(data)
  tokens = remove_punct(data)
  tokens = remove_stopwords(tokens)

  if rule == "lemmatize":
    tokens = lemmatization(tokens)
  elif rule == "stem":
    tokens = stemming(tokens)
  else:
    print("Pick between lemmatize or stem")
    return ""

  return " ".join(tokens)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

Text Classification using Machine Learning Models

Instructions: Trump Tweet Sentiment Classification

1. **Load the Dataset**
   Load the dataset named `"trump_tweet_sentiment_analysis.csv"` using `pandas`. Ensure the dataset contains at least two columns: `"text"` and `"label"`.

2. **Text Cleaning and Tokenization**
   Apply a text preprocessing pipeline to the `"text"` column. This should include:

   - Lowercasing the text
   - Removing URLs, mentions, punctuation, and special characters
   - Removing stopwords
   - Tokenization (optional: stemming or lemmatization)
   - "Complete the above function"

3. **Train-Test Split**
   Split the cleaned and tokenized dataset into **training** and **testing** sets using `train_test_split` from `sklearn.model_selection`.

4. **TF-IDF Vectorization**
   Import and use the `TfidfVectorizer` from

`sklearn.feature_extraction.text` to transform the training and testing texts into numerical feature vectors.

5. **Model Training and Evaluation**
   Import **Logistic Regression** (or any machine learning model of your choice) from `sklearn.linear_model`. Train it on the TF-IDF-embedded training data, then evaluate it using the test set.

   – Print the **classification report** using `classification_report` from `sklearn.metrics`.

Step 1: Load the Dataset

```
import pandas as pd

# Load the dataset
df =
pd.read_csv("/content/drive/MyDrive/AI(LastSem)/week8/trum_tweet_senti
ment_analysis.csv")

# Quick check
print(df.head())
print(df.columns)

                                                text  Sentiment
0  RT @JohnLeguizamo: #trump not draining swamp b...          0
1  ICYMI: Hackers Rig FM Radio Stations To Play A...          0
2  Trump protests: LGBTQ rally in New York https:...          1
3  "Hi I'm Piers Morgan. David Beckham is awful b...          0
4  RT @GlennFranco68: Tech Firm Suing BuzzFeed fo...          0
Index(['text', 'Sentiment'], dtype='object')
```

Apply Your Text Cleaning Pipeline

```
# Apply the text cleaning pipeline to the 'text' column
df['clean_text'] = df['text'].apply(lambda x:
text_cleaning_pipeline(str(x), rule="lemmatize"))

# Optional: Show before and after
print(df[['text', 'clean_text']].head())

                                                text  \
0  RT @JohnLeguizamo: #trump not draining swamp b...
1  ICYMI: Hackers Rig FM Radio Stations To Play A...
2  Trump protests: LGBTQ rally in New York https:...
3  "Hi I'm Piers Morgan. David Beckham is awful b...
4  RT @GlennFranco68: Tech Firm Suing BuzzFeed fo...

                                          clean_text
0  rt drain swamp taxpayer dollars trip advertise...
```

```
1  icymi hackers rig fm radio station play antitr...
2               trump protest lgbtq rally new york via
3  hi im piers morgan david beckham awful donald ...
4  rt tech firm sue buzzfeed publish unverified t...
```

Step 3: Train-Test Split

```python
from sklearn.model_selection import train_test_split

# Split into features and labels
X = df['clean_text']
y = df['Sentiment']

# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print("Training samples:", len(X_train))
print("Testing samples:", len(X_test))
```

```
Training samples: 1480098
Testing samples: 370025
```

TF-IDF Vectorization

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

# Initialize TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Fit and transform training data, transform test data
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

print("Shape of TF-IDF matrix (train):", X_train_tfidf.shape)
```

```
Shape of TF-IDF matrix (train): (1480098, 147171)
```

Step 5: Train a Machine Learning Model

```python
from sklearn.linear_model import LogisticRegression

# Increase the number of iterations
model = LogisticRegression(max_iter=1000)  # or even 2000 if needed
model.fit(X_train_tfidf, y_train)
```

```
LogisticRegression(max_iter=1000)
```

Step 6: Evaluate the Model

```python
from sklearn.metrics import classification_report

# Predict on test data
y_pred = model.predict(X_test_tfidf)

# Print classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96    248563
           1       0.93      0.90      0.91    121462

    accuracy                           0.94    370025
   macro avg       0.94      0.93      0.93    370025
weighted avg       0.94      0.94      0.94    370025
```