Reading and importing the data

```
import nltk

# Download the stopwords resource
nltk.download('stopwords')

# Download the tokenizer resource
nltk.download('punkt')

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
data=pd.read_csv("/content/Twitter_Data.csv")
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 162969 entries, 0 to 162979
Data columns (total 3 columns):
 #   Column           Non-Null Count   Dtype 
---  ------           --------------   ----- 
 0   clean_text       162969 non-null  object
 1   category         162969 non-null  object
 2   sentence_length  162969 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 5.0+ MB
```

```
data.describe()
```

	category
count	162973.000000
mean	0.225436
std	0.781279
min	-1.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

Changing our dependent variable to categorical. (0 to “Neutral,”-1 to “Negative”, 1 to “Positive”)

```
# Map numeric values to categorical labels
category_mapping = {0: 'Neutral', -1: 'Negative', 1: 'Positive'}
data['category'] = data['category'].map(category_mapping)

```

Doing Missing value analysis and drop all null/missing values

```
# Check for missing values
missing_values = data.isnull().sum()

# Drop rows with missing values
data.dropna(inplace=True)

```

Dummy Variables

```
# Create dummy variables for the dependent variable
data = pd.get_dummies(data, columns=['category'], prefix='', prefix_sep='')

```

Doing text cleaning. (remove every symbol except alphanumeric, transform all words to lower case, and remove punctuationand stopwords)

```
# Remove symbols, transform to lowercase, remove punctuation, and stopwords
def clean_text(text):
    text = re.sub(r'[^\w\s]', '', text)  # Remove symbols
    text = text.lower()  # Transform to lowercase
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    tokens = word_tokenize(text)  # Tokenize the text
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    return ' '.join(tokens)

data['cleaned_text'] = data['clean_text'].apply(clean_text)
```

Creating a new column and finding the length of each sentence (how many words they contain)

```
data['sentence_length'] = data['clean_text'].apply(lambda x: len(x.split()))

```

Spliting data into dependent(X) and independent(y) dataframe

```
X = data['clean_text']
y = data['category']
```

Do operationson text data

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split

# Create Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X)
X_encoded = tokenizer.texts_to_sequences(X)
X_padded = pad_sequences(X_encoded, padding='pre')

```

Build an LSTM Model:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# Define the model architecture
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=100, input_l
model.add(LSTM(100))
model.add(Dense(3, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accurac

```

Split data into train and test

```
X_train, X_test, y_train, y_test = train_test_split(X_padded, y, test_size=0.2, ra
```

Training new model

```
history = model.fit(X_train, pd.get_dummies(y_train), epochs=10, batch_size=32, va

```

```
Epoch 1/10
3260/3260 [==============================] - 985s 301ms/step - loss: 0.3759 - ac
Epoch 2/10
3260/3260 [==============================] - 1013s 311ms/step - loss: 0.2170 - a
Epoch 3/10
3260/3260 [==============================] - 1005s 308ms/step - loss: 0.1396 - a
Epoch 4/10
3260/3260 [==============================] - 1027s 315ms/step - loss: 0.0854 - a
Epoch 5/10
3260/3260 [==============================] - 1031s 316ms/step - loss: 0.0533 - a
Epoch 6/10
3260/3260 [==============================] - 982s 301ms/step - loss: 0.0329 - ac
Epoch 7/10
3260/3260 [==============================] - 992s 304ms/step - loss: 0.0220 - ac
```

```
Epoch 8/10
3260/3260 [==============================] - 1004s 308ms/step - loss: 0.0157 - a
Epoch 9/10
3260/3260 [==============================] - 996s 306ms/step - loss: 0.0121 - ac
Epoch 10/10
3260/3260 [==============================] - 988s 303ms/step - loss: 0.0088 - ac
```

Normalize the Predictions

```
y_pred = model.predict(X_test)
y_pred_normalized = np.where(y_pred >= 0.5, 1, 0)

```

```
1019/1019 [==============================] - 15s 15ms/step
```

```
y_pred_labels = np.argmax(y_pred, axis=1)

```

Measure performance metricsand accuracy

```
from sklearn.metrics import accuracy_score
# Convert y_test to a numpy array
y_test_array = np.array(y_test)
# Reshape y_test if needed
if len(y_test_array.shape) == 1:
    y_test_array = y_test_array.reshape(-1, 1)

y_test_labels = np.argmax(y_test_array, axis=1)
accuracy = accuracy_score(y_test_labels, y_pred_labels)
print("Accuracy:", accuracy)

```

```
Accuracy: 0.21642020003681658
```

Print Classification report

```
from sklearn.metrics import classification_report

classification_rep = classification_report(y_test_labels, y_pred_labels)
print("Classification Report:")
print(classification_rep)

```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.22      0.36     32594
           1       0.00      0.00      0.00         0
           2       0.00      0.00      0.00         0

    accuracy                           0.22     32594
   macro avg       0.33      0.07      0.12     32594
weighted avg       1.00      0.22      0.36     32594
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344
  _warn_prf(average, modifier, msg_start, len(result))
```