Importing Packages

```
# Basic manipulation
import numpy as np
import pandas as pd
np.random.seed(0)

# Visualization features
from textblob import TextBlob

# Plotting
import matplotlib.pyplot as plt

# Utilities imports
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# NLTK
import nltk
from nltk.corpus import stopwords
#nltk.download('stopwords')
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import WordPunctTokenizer
#nltk.download('punkt')

# TensorFlow / Keras
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Misc
from collections import Counter
```

Importing Data

```
# Training data
path = "data/twitter_training.csv"
train_df = pd.read_csv(path, names=["Tweet_ID", "Entity", "Sentiment", "Tweet_Content"])

# Test data (Note to be used until the full model has been trained)
test_path = "data/twitter_validation.csv"
test_df = pd.read_csv(test_path, names=["Tweet_ID", "Entity", "Sentiment", "Tweet_Content"])
```

Dropping NA Values

```
train_df = train_df.dropna()
test_df = test_df.dropna()
```

```
train_df.shape
```

```
(74682, 4)
```

Subsetting Data

```
df = train_df.sample(frac=0.1)
df.reset_index(drop=True, inplace=True)
```

Data Exploration

```
# Some overall information about the data frame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7468 entries, 0 to 7467
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Tweet_ID      7468 non-null   int64
 1   Entity        7468 non-null   object
 2   Sentiment     7468 non-null   object
 3   Tweet_Content 7378 non-null   object
dtypes: int64(1), object(3)
memory usage: 233.4 KB
```

```
# Glimpse a few entries
df.head()
```

	Tweet_ID	Entity	Sentiment	Tweet_Content
0	2535	Borderlands	Negative	@ Borderlands Can we get a big fly please? I...
1	11973	TomClancysGhostRecon	Negative	@GhostRecon You call 1.5 an update. You're b...
2	3024	Dota2	Neutral	I'm excited to watch the EU closed qualifyin...
3	6719	Fortnite	Negative	anyway but my fortnite here, im annoyed.
4	1510	Battlefield	Negative	Collateral with the receiver is so satisfying...

Checking Balance of the Data in Output Categories

```
# Checking balance of target classes
sentiments = list(df["Sentiment"].unique())

sentiment_nums = [len(df.df["Sentiment"] == sentiment)] // len(df) for sentiment in sentiments]

plt.bar(sentiments, sentiment_nums)

<BarContainer object of 4 artists>
```

Labels to Indexes (and Back Again)

```
# Define the indexing for each possible label in a dictionary
class_to_index = {"Neutral": 0, "Irrelevant": 1, "Negative": 2, "Positive": 3}

# Creates a reverse dictionary
index_to_class = dict(zip(class_to_index.values()))

# Creates lambda functions, applying the appropriate dictionary
names_to_ix = lambda x: np.array([class_to_index.get(y) for y in x])
ix_to_names = lambda x: np.array([index_to_class.get(y) for y in x])

# Test each function
print(names_to_ix(["Positive", "Negative", "Irrelevant", "Neutral"]))
print(ix_to_names([1,2,3,0]))

# [ 3 2 1 0]
# ['Neutral' 'Irrelevant' 'Negative' 'Positive']

# Convert the "Sentiment" column into indexes
df["Sentiment"] = names_to_ix(df["Sentiment"])

df.head()
```

	Tweet_ID	Entity	Sentiment	Tweet_Content
0	2535	Borderlands	2	@ Borderlands Can we get a big fly please? I...
1	11973	TomClancysGhostRecon	2	@GhostRecon You call 1.5 an update. You're b...
2	3024	Dota2	0	I'm excited to watch the EU closed qualifyin...
3	6719	Fortnite	2	anyway but my fortnite here, im annoyed.
4	1510	Battlefield	2	Collateral with the receiver is so satisfying...

NLP Data Enrichment

```
def remove_stopwords(s):
    #Lowercasing, then removing stop words
    lc = lc.lower().split(sep=" ")
    lc = [w for w in lc if w not in (stop_english) and (word.isalpha())]

    #Joins the words back into a single string
    lc = " ".join(lc)
    return lc

#Apply each string into a list of words
df["Tweet_Content_Split"] = df["Tweet_Content"].apply(word_tokenize)

#Applies the above function to each entry in the DataFrame
lemmatizer = WordNetLemmatizer()
stop_english = Counter(stopwords.words())

df["Tweet_Content_Split"] = df["Tweet_Content_Split"].apply(remove_stopwords)

df.head()
```

	Tweet_ID	Entity	Sentiment	Tweet_Content	Tweet_Content_Split
0	2535	Borderlands	2	@ Borderlands Can we get a big fly please? I...	[borderlands, can, get, big, fly, please, ?, ...
1	11973	TomClancysGhostRecon	2	@GhostRecon You call 1.5 an update. You're b...	[ghostrecon, you, call, 1.5, an, update, ., y...
2	3024	Dota2	0	I'm excited to watch the EU closed qualifyin...	[, excited, watch, eu, closed, qualifying, match, ...
3	6719	Fortnite	2	anyway but my fortnite here, im annoyed.	[, fortnite, annoyed,]
4	1510	Battlefield	2	Collateral with the receiver is so satisfying...	[, collateral, receiver, ,,]

Tokenisation

```
#Define the Tokenizer
tokenizer = Tokenizer(num_words=10000, lower=True)

#Create the corpus by fitting the word index
tokenizer.fit_on_texts(df["Tweet_Content_Split"])

#Tokenize our column of edited tweet content
tweet_tokens = tokenizer.texts_to_sequences(list(df["Tweet_Content_Split"]))

#Pad these sequences to make them the same length
tweet_tokens = pad_sequences(tweet_tokens, truncating = 'post', padding='post', maxlen=50)

tweet_tokens.shape
```

```
(7468, 50)
```

Final Data Preparation

```
#Remove dependent variable
y = df["Sentiment"]

#Drop all non-useful columns
df = pd.DataFrame(tweet_tokens)

#Displaying final shape
df.shape
```

```
(7468, 50)
```

Test-Train Split

```
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.3, random_state=1)
```

```
model = tf.keras.models.Sequential([
tf.keras.layers.Embedding(10000,16,input_length=50),
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20, return_sequences=True)),
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20)),
tf.keras.layers.Dense(4, activation='softmax')
])
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

Training

```
h = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=15,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=5)]
)
```

```
Epoch 1/15
163/163 [==============================] - 25s 52ms/step - loss: 1.1367 - accuracy: 0.3566 - val_loss: 1.2005 - val_accuracy: 0.4912
Epoch 2/15
105/105 [==============================] - 11s 92ms/step - loss: 1.0009 - accuracy: 0.5882 - val_loss: 1.1272 - val_accuracy: 0.5207
Epoch 3/15
105/105 [==============================] - 11s 92ms/step - loss: 0.8693 - accuracy: 0.7444 - val_loss: 1.0159 - val_accuracy: 0.5455
Epoch 4/15
105/105 [==============================] - 11s 92ms/step - loss: 0.6450 - accuracy: 0.8420 - val_loss: 1.2503 - val_accuracy: 0.5795
Epoch 5/15
105/105 [==============================] - 11s 94ms/step - loss: 0.3357 - accuracy: 0.8982 - val_loss: 1.6492 - val_accuracy: 0.5676
Epoch 6/15
105/105 [==============================] - 11s 97ms/step - loss: 0.2766 - accuracy: 0.9040 - val_loss: 1.5005 - val_accuracy: 0.5615
Epoch 7/15
105/105 [==============================] - 11s 97ms/step - loss: 0.2173 - accuracy: 0.9216 - val_loss: 1.6208 - val_accuracy: 0.5764
Epoch 8/15
105/105 [==============================] - 11s 97ms/step - loss: 0.1850 - accuracy: 0.9334 - val_loss: 1.6959 - val_accuracy: 0.5602
Epoch 9/15
105/105 [==============================] - 11s 97ms/step - loss: 0.1659 - accuracy: 0.9394 - val_loss: 1.7002 - val_accuracy: 0.5707
Epoch 10/15
105/105 [==============================] - 11s 98ms/step - loss: 0.1555 - accuracy: 0.9433 - val_loss: 1.8086 - val_accuracy: 0.5764
Epoch 11/15
105/105 [==============================] - 11s 98ms/step - loss: 0.1389 - accuracy: 0.9502 - val_loss: 1.8994 - val_accuracy: 0.5855
Epoch 12/15
105/105 [==============================] - 11s 98ms/step - loss: 0.1264 - accuracy: 0.9537 - val_loss: 1.9981 - val_accuracy: 0.5797
Epoch 13/15
105/105 [==============================] - 11s 98ms/step - loss: 0.1193 - accuracy: 0.9569 - val_loss: 1.9299 - val_accuracy: 0.5723
Epoch 14/15
105/105 [==============================] - 11s 98ms/step - loss: 0.1121 - accuracy: 0.9592 - val_loss: 2.0170 - val_accuracy: 0.5791
Epoch 15/15
105/105 [==============================] - 11s 98ms/step - loss: 0.1147 - accuracy: 0.9573 - val_loss: 2.0209 - val_accuracy: 0.5730
```

Model Evaluation

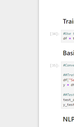
```
#Generate predictions
y_pred = np.argmax(model.predict(X_test), axis=1)

#Assign labels to predictions and test data
y_pred_labels = ix_to_names(y_pred)
y_test_labels = ix_to_names(y_test)

#List
y_unique = list(set(y_test_labels))
cm = confusion_matrix(y_test_labels, y_pred_labels, labels = y_unique, normalize='true')

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=y_unique)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f3cc8f0d0>
```



Training with Full Dataset

```
#Use the full dataset!
df = train_df
```

Basic Data Enrichment

```
#Convert the "Sentiment" column into indexes
df.head()
#df["Sentiment"] = names_to_ix(df["Sentiment"])
y = df["Sentiment"]

#Convert
df["Sentiment"] = names_to_ix(df["Sentiment"])
y_test = test_df["Sentiment"]
```

NLP Data Enrichment

```
#Removing stopwords and lemmatizing

#Whole
#Apply each string into a list of words
df["Tweet_Content_Split"] = df["Tweet_Content"].apply(word_tokenize)

#Applies the above function to each entry in the DataFrame
lemmatizer = WordNetLemmatizer()
stop_english = Counter(stopwords.words())

df["Tweet_Content_Split"] = df["Tweet_Content_Split"].apply(remove_stopwords)

#Whole
test_df["Tweet_Content_Split"] = test_df["Tweet_Content"].apply(word_tokenize)

test_df["Tweet_Content_Split"] = test_df["Tweet_Content_Split"].apply(remove_stopwords)

#Tokenisation

#Define the Tokenizer
tokenizer = Tokenizer(num_words=10000, lower=True)

#Create the corpus by fitting the word index
tokenizer.fit_on_texts(df["Tweet_Content_Split"])

#Whole
#Tokenize our column of edited tweet content
tweet_tokens = tokenizer.texts_to_sequences(list(df["Tweet_Content_Split"]))
test_tokens = tokenizer.texts_to_sequences(test_tokens), truncating = 'post', padding='post', maxlen=50)

#Whole
#Tokenize our column of edited tweet content
tweet_tokens_test = tokenizer.texts_to_sequences(list(test_df["Tweet_Content_Split"]))
tweet_tokens_test = pad_sequences(tweet_tokens_test, truncating = 'post', padding='post', maxlen=50)

#Final prep

#Whole
#Drop all non-useful columns
full_df = pd.DataFrame(tweet_tokens)

#Whole
full_test_df = pd.DataFrame(tweet_tokens_test)
```

Model Definition and Training

This time, we train with all of the available training data

```
#Original
model = tf.keras.models.Sequential([
tf.keras.layers.Embedding(10000,16,input_length=50),
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20, return_sequences=True)),
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20)),
tf.keras.layers.Dense(4, activation='softmax')
])

model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

```
h = model.fit(
    full_df, y,
    validation_data=(full_test_df, y_test),
    epochs=15,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=5)]
)
```

```
Epoch 1/15
233/233 [==============================] - 21s 56ms/step - loss: 0.8008 - accuracy: 0.6387 - val_loss: 1.3487 - val_accuracy: 0.2090
Epoch 2/15
233/233 [==============================] - 14s 61ms/step - loss: 0.5436 - accuracy: 0.7964 - val_loss: 1.5752 - val_accuracy: 0.2610
Epoch 3/15
233/233 [==============================] - 14s 61ms/step - loss: 0.4243 - accuracy: 0.8404 - val_loss: 1.3762 - val_accuracy: 0.5150
Epoch 4/15
233/233 [==============================] - 14s 61ms/step - loss: 0.3602 - accuracy: 0.8636 - val_loss: 1.3955 - val_accuracy: 0.2750
Epoch 5/15
233/233 [==============================] - 14s 61ms/step - loss: 0.3192 - accuracy: 0.8808 - val_loss: 1.5795 - val_accuracy: 0.5150
Epoch 6/15
233/233 [==============================] - 14s 61ms/step - loss: 0.2895 - accuracy: 0.8902 - val_loss: 1.5790 - val_accuracy: 0.2850
Epoch 7/15
233/233 [==============================] - 14s 61ms/step - loss: 0.2659 - accuracy: 0.8969 - val_loss: 1.4829 - val_accuracy: 0.2750
Epoch 8/15
233/233 [==============================] - 14s 61ms/step - loss: 0.2456 - accuracy: 0.9040 - val_loss: 1.5863 - val_accuracy: 0.5290
Epoch 9/15
233/233 [==============================] - 14s 61ms/step - loss: 0.2276 - accuracy: 0.9110 - val_loss: 1.7159 - val_accuracy: 0.5070
Epoch 10/15
233/233 [==============================] - 14s 61ms/step - loss: 0.2123 - accuracy: 0.9165 - val_loss: 1.6980 - val_accuracy: 0.2570
Epoch 11/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1991 - accuracy: 0.9215 - val_loss: 1.8152 - val_accuracy: 0.2920
Epoch 12/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1885 - accuracy: 0.9252 - val_loss: 1.8208 - val_accuracy: 0.2980
Epoch 13/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1772 - accuracy: 0.9308 - val_loss: 1.8288 - val_accuracy: 0.5340
Epoch 14/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1687 - accuracy: 0.9325 - val_loss: 1.8868 - val_accuracy: 0.5310
Epoch 15/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1612 - accuracy: 0.9352 - val_loss: 1.9370 - val_accuracy: 0.2980
Epoch 16/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1524 - accuracy: 0.9379 - val_loss: 1.8235 - val_accuracy: 0.2910
Epoch 17/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1443 - accuracy: 0.9402 - val_loss: 1.9127 - val_accuracy: 0.5120
Epoch 18/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1369 - accuracy: 0.9431 - val_loss: 1.8280 - val_accuracy: 0.2880
Epoch 19/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1310 - accuracy: 0.9457 - val_loss: 1.6807 - val_accuracy: 0.2860
Epoch 20/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1235 - accuracy: 0.9468 - val_loss: 1.5779 - val_accuracy: 0.2850
Epoch 21/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1187 - accuracy: 0.9462 - val_loss: 1.4847 - val_accuracy: 0.2810
Epoch 22/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1138 - accuracy: 0.9472 - val_loss: 1.4864 - val_accuracy: 0.5280
Epoch 23/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1122 - accuracy: 0.9486 - val_loss: 1.3868 - val_accuracy: 0.5210
Epoch 24/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1109 - accuracy: 0.9501 - val_loss: 1.3565 - val_accuracy: 0.3860
Epoch 25/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1155 - accuracy: 0.9511 - val_loss: 1.4981 - val_accuracy: 0.5140
Epoch 26/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1155 - accuracy: 0.9510 - val_loss: 1.3994 - val_accuracy: 0.5120
Epoch 27/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1132 - accuracy: 0.9516 - val_loss: 1.3979 - val_accuracy: 0.5100
Epoch 28/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1091 - accuracy: 0.9519 - val_loss: 1.4852 - val_accuracy: 0.2840
Epoch 29/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1080 - accuracy: 0.9521 - val_loss: 1.4875 - val_accuracy: 0.3860
Epoch 30/15
233/233 [==============================] - 14s 61ms/step - loss: 0.1067 - accuracy: 0.9541 - val_loss: 1.4452 - val_accuracy: 0.2880
```

Final Model Evaluation

```
#Generate predictions
y_pred = np.argmax(model.predict(full_test_df), axis=1)

#Assign labels to predictions and test data
y_pred_labels = ix_to_names(y_pred)
y_test_labels = ix_to_names(y_test)

1122/1122 [==============================] - 2s 1ms/step

y_unique = list(set(y_test_labels))
cm = confusion_matrix(y_test_labels, y_pred_labels, labels = y_unique, normalize='true')

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=y_unique)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f3cc8f0a0>
```

```
#To see the final accuracy
accuracy_score(y_test, y_pred)
```

```
0.2202
```