Assignment 3

Name - Harman Bhutani | 300144160

Question 2

a) Represent the words from the Covid-19 tweets sample data in vector space using the Word2Vec approach, and compare the accuracy of your assignment 2A model (which used the tf-idf approach) with the new model

```
import nltk
import pandas as pd
import numpy as np
import csv
import nltk
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

```
import pandas as pd
import numpy as np
import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import re
import matplotlib
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score,
from sklearn.svm import SVC
from sklearn.naive_bayes import BaseNB
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoos
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBRFClassifier
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.model_selection import RandomizedSearchCV# Number of trees in random for
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


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

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

```
tweets_df = pd.read_csv("Covid_train_data.csv", encoding='latin-1')


tweets_df.head()
```

	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	3799	48751	London	16-03-2020	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral
1	3800	48752	UK	16-03-2020	advice Talk to your neighbours family to excha...	Positive
2	3801	48753	Vagabonds	16-03-2020	Coronavirus Australia: Woolworths to give elde...	Positive
				16-03-	Mv food stock is not the onlv one	

```
len(tweets_df)
```

```
41157
```

```
Location_count=tweets_df["Location"].value_counts()
Location_count
```

```
London                 540
United States          528
London, England        520
New York, NY           395
Washington, DC         373
                      ...
El Dorado Hills, CA      1
Sonoma County, CA        1
Evanston, Ill.           1
Always Hungry, USA       1
????, ????? ????         1
Name: Location, Length: 12220, dtype: int64
```

```
# Print the value counts of Sentiments column
Sentiments_count=tweets_df["Sentiment"].value_counts()
Sentiments_count
```

```
Positive              11422
Negative               9917
Neutral                7713
Extremely Positive     6624
Extremely Negative     5481
Name: Sentiment, dtype: int64
```

```
display(tweets_df.head())
print(tweets_df.describe())
print(tweets_df.info())
```

	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	3799	48751	London	16-03-2020	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral
1	3800	48752	UK	16-03-2020	advice Talk to your neighbours family to excha...	Positive
2	3801	48753	Vagabonds	16-03-2020	Coronavirus Australia: Woolworths to give elde...	Positive
3	3802	48754	NaN	16-03-2020	My food stock is not the only one which is emp...	Positive
4	3803	48755	NaN	16-03-2020	Me, ready to go at supermarket during the #COV...	Extremely Negative

```
           UserName    ScreenName
count  41157.000000  41157.000000
mean   24377.000000  69329.000000
std    11881.146851  11881.146851
min     3799.000000  48751.000000
25%    14088.000000  59040.000000
50%    24377.000000  69329.000000
75%    34666.000000  79618.000000
max    44955.000000  89907.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41157 entries, 0 to 41156
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   UserName       41157 non-null  int64
 1   ScreenName     41157 non-null  int64
 2   Location       32567 non-null  object
 3   TweetAt        41157 non-null  object
 4   OriginalTweet  41157 non-null  object
```

```
def process_tweets(tweet):

    # Remove links
    tweet = re.sub(r"http\S+|www\S+|https\S+", '', tweet, flags=re.MULTILINE)
```

```
    # Remove mentions and hashtag
    tweet = re.sub(r'\@\w+|\#','', tweet)

    # Tokenize the words
    tokenized = word_tokenize(tweet)

    # Remove the stop words
    tokenized = [token for token in tokenized if token not in stopwords.words("englis

    # Lemmatize the words
    lemmatizer = WordNetLemmatizer()
    tokenized = [lemmatizer.lemmatize(token, pos='a') for token in tokenized]

    # Remove non-alphabetic characters and keep the words contains three or more lett
    tokenized = [token for token in tokenized if token.isalpha() and len(token)>2]

    return tokenized

# Call the function and store the result into a new column
tweets_df["Processed"] = tweets_df["OriginalTweet"].str.lower().apply(process_tweets)

# Print the first fifteen rows of Processed
display(tweets_df[["Processed"]].head(15))
```

	Processed
0	[]
1	[advice, talk, neighbours, family, exchange, p...
2	[coronavirus, australia, woolworths, give, eld...
3	[food, stock, one, empty, please, panic, enoug...
4	[ready, supermarket, outbreak, paranoid, food,...
5	[news, first, confirmed, case, came, sullivan,...
6	[cashier, grocery, store, sharing, insights, p...
7	[supermarket, today, buy, toilet, paper, rebel...
8	[due, retail, store, classroom, atlanta, open,...
9	[corona, prevention, stop, buy, things, cash, ...
10	[month, crowding, supermarkets, restaurants, h...
11	[due, situation, increased, demand, food, prod...
12	[horningsea, caring, community, look, less, ca...
13	[need, stock, food, amazon, deliver, whatever,...
14	[adara, releases, resource, center, travel, br...

```
# Get the tweet lengths
tweets_df["Length"] = tweets_df["OriginalTweet"].str.len()
# Get the number of words in tweets
tweets_df["Words"] = tweets_df["OriginalTweet"].str.split().str.len()
# Display the new columns
display(tweets_df[["Length", "Words"]])
```

	Length	Words
0	111	8
1	237	38
2	131	14
3	306	42
4	310	40
...	...	...
41152	102	12
41153	138	23
41154	136	18
41155	111	18
41156	255	46

41157 rows × 2 columns

```
tweets_df["Location"].fillna("unknown", inplace=True)
```

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
# Initialize a Tf-idf Vectorizer
vectorizer = TfidfVectorizer(max_features=100)
# Fit and transform the vectorizer
tfidf_matrix = vectorizer.fit_transform([' '.join(l) for l in tweets_df["Processed"]]
# Let's see what we have
display(tfidf_matrix)
# Create a DataFrame for tf-idf vectors and display the first rows
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns= vectorizer.get_feature_names
display(tfidf_df)
```

```
<41157x100 sparse matrix of type '<class 'numpy.float64'>'
	with 186001 stored elements in Compressed Sparse Row format>
```

	also	amid	amp	back	business	buy	buying	consumer	coronavirus	cou
0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	
1	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	
2	0.0	0.617436	0.0	0.0	0.0	0.0	0.0	0.000000	0.243451	
3	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.115334	
4	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.197031	
...	...	...	...	...	...	...	...	...	...	
41152	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	
41153	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.558534	0.000000	
41154	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.190497	
41155	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.554381	

```
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.preprocessing import LabelEncoder
# Encode the labels
le = LabelEncoder()
tweets_df["Label_enc"] = le.fit_transform(tweets_df["Sentiment"])
# Display the encoded labels
display(tweets_df[["Label_enc"]].head())
```

	Label_enc
0	3
1	4
2	4
3	4
4	0

```
X = tweets_df['Processed']
y = tweets_df["Label_enc"]
```

```
tweets_test = pd.read_csv("Covid_test_data.csv")
```

```
# Encode the labels
le = LabelEncoder()
tweets_test["Label_enc"] = le.fit_transform(tweets_test["Sentiment"])
# Display the encoded labels
display(tweets_test[["Label_enc"]].head())
```

	Label_enc
0	0
1	4
2	1
3	2
4	3

```
def process_tweets(tweet):

    # Remove links
    tweet = re.sub(r"http\S+|www\S+|https\S+", '', tweet, flags=re.MULTILINE)

    # Remove mentions and hashtag
    tweet = re.sub(r'\@\w+|\#','', tweet)

    # Tokenize the words
    tokenized = word_tokenize(tweet)

    # Remove the stop words
    tokenized = [token for token in tokenized if token not in stopwords.words("englis

    # Lemmatize the words
    lemmatizer = WordNetLemmatizer()
    tokenized = [lemmatizer.lemmatize(token, pos='a') for token in tokenized]

    # Remove non-alphabetic characters and keep the words contains three or more lett
    tokenized = [token for token in tokenized if token.isalpha() and len(token)>2]

    return tokenized

# Call the function and store the result into a new column
tweets_test["Processed"] = tweets_test["OriginalTweet"].str.lower().apply(process_twe

# Print the first fifteen rows of Processed
display(tweets_test[["Processed"]].head(15))
```

	Processed
0	[trending, new, yorkers, encounter, empty, sup...
1	[could, find, hand, sanitizer, fred, meyer, tu...
2	[find, protect, loved, ones, coronavirus]
3	[panic, buying, hits, newyork, city, anxious, ...
4	[toiletpaper, dunnypaper, coronavirus, coronav...
5	[remember, last, time, paid, gallon, regular, ...
6	[voting, age, coronavirus, hand, sanitizer, su...
7	[stop, without, protecting, healthworkers, pri...
8	[twitter, pharmacist, sell, hand, sanitizer, l...

```
X_test = tweets_test['Processed']
y_test = tweets_test["Label_enc"]
```

```
pip install sklearn
```

```
Requirement already satisfied: sklearn in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.7/dist-pa
```

```
model_vectorizer = TfidfVectorizer()
# First fit the vectorizer with our training set
tfidf_train = vectorizer.fit_transform([' '.join(l) for l in X])
# Now we can fit our test data with the same vectorizer
tfidf_test = vectorizer.transform([' '.join(l) for l in X_test])
```

```
from functools import reduce
```

```
from sklearn.ensemble import RandomForestClassifier
# from sklearn.cross_validation import train_test_split
from sklearn.datasets import load_iris

def generate_rf(X_train, y_train, X_test, y_test):
    rf = RandomForestClassifier(n_estimators=5, min_samples_leaf=3)
    rf.fit(X_train, y_train)
    print ("rf score ", rf.score(X_test, y_test))
    return rf

def combine_rfs(rf_a, rf_b):
    rf_a.estimators_ += rf_b.estimators_
    rf_a.n_estimators = len(rf_a.estimators_)
    return rf_a
```

```
    return rf_a


# in the line below, we create 10 random forest classifier models
rfs = [generate_rf(tfidf_train, y, tfidf_test, y_test)]
# in this step below, we combine the list of random forest models into one giant mode
rf_combined = reduce(combine_rfs, rfs)
# the combined model scores better than *most* of the component models
print ("rf combined score", rf_combined.score(tfidf_test, y_test))
```

```
rf score  0.3504476040021064
rf combined score 0.3504476040021064
```

```
# Print the accuracy score
best_accuracy = cross_val_score(rf_combined, tfidf_test, y_test, cv=10, scoring='accu
print("Accuracy:",best_accuracy)
```

```
Accuracy: 0.3447368421052632
```

```
from gensim.models import word2vec

wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in  X.str.join(' ')]

feature_size = 100
window_context = 10  # Context window size
min_word_count = 1   # Minimum word count, i.e., words > this are going to be include
sample = 1e-3        # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                              window=window_context, min_count = min_word_count,
                              sample=sample)


def average_word_vectors(words, model, vocabulary, num_features):

    feature_vector = np.zeros((num_features,),dtype="float64")
    nwords = 0.

    for word in words:
        if word in vocabulary:
            nwords = nwords + 1.
            feature_vector = np.add(feature_vector, model[word])

    if nwords:
        feature_vector = np.divide(feature_vector, nwords)

    return feature_vector

def averaged_word_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index2word)
    features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_featu
```

```
    features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_featu
                    for tokenized_sentence in corpus]
    return np.array(features)


w2v_feature_array = averaged_word_vectorizer(corpus=tokenized_corpus, model=w2v_model
                                             num_features=feature_size)
X_train_wordvec = pd.DataFrame(w2v_feature_array)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: DeprecationWarni
  if __name__ == '__main__':
```

```
from gensim.models import word2vec

wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in  X_test.str.join(' ')]

feature_size = 100
window_context = 10  # Context window size
min_word_count = 1   # Minimum word count, i.e., words > this are going to be include
sample = 1e-3        # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                              window=window_context, min_count = min_word_count,
                              sample=sample)


def average_word_vectors(words, model, vocabulary, num_features):

    feature_vector = np.zeros((num_features,),dtype="float64")
    nwords = 0.

    for word in words:
        if word in vocabulary:
            nwords = nwords + 1.
            feature_vector = np.add(feature_vector, model[word])

    if nwords:
        feature_vector = np.divide(feature_vector, nwords)

    return feature_vector

def averaged_word_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index2word)
    features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_featu
                    for tokenized_sentence in corpus]
    return np.array(features)


w2v_feature_array = averaged_word_vectorizer(corpus=tokenized_corpus, model=w2v_model
                                             num_features=feature_size)
X_test_wordvec = pd.DataFrame(w2v_feature_array)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: DeprecationWarni
  if __name__ == '__main__':
```

```
from sklearn.ensemble import RandomForestClassifier
# from sklearn.cross_validation import train_test_split
from sklearn.datasets import load_iris

def generate_rf(X_train, y_train, X_test, y_test):
    rf = RandomForestClassifier(n_estimators=5, min_samples_leaf=3)
    rf.fit(X_train, y_train)
    print ("rf score ", rf.score(X_test, y_test))
    return rf

def combine_rfs(rf_a, rf_b):
    rf_a.estimators_ += rf_b.estimators_
    rf_a.n_estimators = len(rf_a.estimators_)
    return rf_a


# in the line below, we create 10 random forest classifier models
rf_wvec = [generate_rf(X_train_wordvec, y, X_test_wordvec, y_test)]
# in this step below, we combine the list of random forest models into one giant mode
rf_wvec_combined = reduce(combine_rfs, rf_wvec)
# the combined model scores better than *most* of the component models
print ("rf combined score", rf_wvec_combined.score(X_test_wordvec, y_test))
```

```
rf score  0.16903633491311215
rf combined score 0.16903633491311215
```

```
# Print the accuracy score
best_accuracy = cross_val_score(rf_wvec_combined, X_test_wordvec, y_test, cv=10, scor
print("Accuracy:",best_accuracy)
```

```
Accuracy: 0.2868421052631579
```

Here we observed Tf-idf has the better accuracy

b)Using the best model from (a), perform a k-fold (k=10) cross-validation in combination with Gridsearch or Randomsearch to fine-tune the performance of your model by varying its hyperparameter values

```
def performance(ypred, yactl):
    print(f'F1-score  : {f1_score(yactl, ypred , average = "macro")}')
    print(f'Accuracy  : {accuracy_score(yactl, ypred)}')
    print(f'Precision : {precision_score(yactl, ypred, average = "macro")}')
    print(f'Recall    : {recall_score(yactl, ypred, average = "macro")}')
```

```
rfc = RandomForestClassifier(n_jobs=-1)
n_estimators = [150,200,300]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [10,20,40]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [150, 200, 300], 'max_features': ['auto', 'sqrt'], 'max_depth':
```

```
rf_random = RandomizedSearchCV(estimator=rfc, param_distributions=random_grid, cv=10,
# Fit the random search model
rf_random.fit(tfidf_train, y)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 11.8min
/usr/local/lib/python3.7/dist-packages/joblib/externals/loky/process_executor.py
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 52.4min finished
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0
                                                    n_estimators=100, n_job...
                                                    verbose=0,
                                                    warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 40, None],
```

```
                                                'max_features': ['auto', 'sqrt'],
                                                'min_samples_leaf': [1, 2, 4],
                                                'min_samples_split': [2, 5, 10],
                                                'n_estimators': [150, 200, 300]},
                                pre_dispatch='2*n_jobs', random_state=42, refit=True,
                                return_train_score=False, scoring=None, verbose=2)
```

```
rf_random.best_params_
```

```
{'bootstrap': True,
 'max_depth': 40,
 'max_features': 'sqrt',
 'min_samples_leaf': 4,
 'min_samples_split': 2,
 'n_estimators': 200}
```

```
rfc = RandomForestClassifier(n_estimators=200,
                             min_samples_split= 2,
                             min_samples_leaf= 4,
                             max_features= 'sqrt',
                             max_depth= 40,
                             bootstrap= True)
```

```
rfc.fit(tfidf_train, y)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=40, max_features='sqrt',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=4, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
pred = rfc.predict(tfidf_test)
```

```
pred_prob = rfc.predict_proba(tfidf_test)
```

```
performance(pred, y_test)
```

```
F1-score  : 0.37443212086706157
Accuracy  : 0.37651395471300686
Precision : 0.4096311070020642
Recall    : 0.3721776514954227
```

```
# Print the Confusion Matrix
cm = confusion_matrix(y_test, pred)
print("Confusion Matrix\n")
print(cm)
# Print the Classification Report
```

```
# Print the classification report
cr = classification_report(y_test, pred)
print("\n\nClassification Report\n")
print(cr)
```

```
Confusion Matrix

[[157  18 188  85 144]
 [ 25 178  75  72 249]
 [110  37 345 218 331]
 [ 10  10 137 313 149]
 [ 35  98 190 187 437]]


Classification Report

              precision    recall  f1-score   support

           0       0.47      0.27      0.34       592
           1       0.52      0.30      0.38       599
           2       0.37      0.33      0.35      1041
           3       0.36      0.51      0.42       619
           4       0.33      0.46      0.39       947

    accuracy                           0.38      3798
   macro avg       0.41      0.37      0.37      3798
weighted avg       0.40      0.38      0.37      3798
```

```
def confusion_metrics (conf_matrix):
# save confusion matrix and slice into four pieces
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
    print('True Positives:', TP)
    print('True Negatives:', TN)
    print('False Positives:', FP)
    print('False Negatives:', FN)

    # calculate the sensitivity
    conf_sensitivity = (TP / float(TP + FN))
    # calculate the specificity
    conf_specificity = (TN / float(TN + FP))

    print(f'Sensitivity: {round(conf_sensitivity,2)}')
    print(f'Specificity: {round(conf_specificity,2)}')


confusion_metrics(cm)
```

```
True Positives: 178
True Negatives: 157
False Positives: 18
```

```
False Negatives: 25
Sensitivity: 0.88
Specificity: 0.9
```

c)Select another classifier and compare the results of your model based on the following criteria: Accuracy, Sensitivity and Specificity

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (
```

```
from xgboost import XGBClassifier
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.model_selection import RandomizedSearchCV# Number of trees in random for
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
```

```
silent = False,
max_depth= 6, 10, 15, 20,
learning_rate = 0.001, 0.01, 0.1, 0.2, 0,3,
subsample= 0.5, 0.6, 0.7, 0.8, 0.9, 1.0,
colsample_bytree= 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0,
colsample_bylevel= 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0,
min_child_weight= 0.5, 1.0, 3.0, 5.0, 7.0, 10.0,
gamma = 0, 0.25, 0.5, 1.0,
reg_lambda= 0.1, 1.0, 5.0, 10.0, 50.0, 100.0,
n_estimators = [100]
```

```
xg = XGBClassifier()
param_grid = dict(
        max_depth = max_depth,
        learning_rate = learning_rate,
        subsample = subsample,
        colsample_bytree = colsample_bytree,
        colsample_bylevel = colsample_bylevel,
        min_child_weight = min_child_weight,
        gamma = gamma,
        reg_lambda = reg_lambda,
         n_estimators = n_estimators)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
```

```
xg_random = RandomizedSearchCV(estimator=xg, param_distributions=param_grid, cv=kfold
```

```
xg_random = RandomizedSearchCV(estimator=xg, param_distributions=param_grid, cv=kfold
```

```
xg_random.fit(tfidf_train, y)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 11.0min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 26.1min finished
RandomizedSearchCV(cv=StratifiedKFold(n_splits=10, random_state=7, shuffle=True)
                   error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='bina...
                                                        0.8, 0.9, 1.0),
                                        'gamma': (0, 0.25, 0.5, 1.0),
                                        'learning_rate': (0.001, 0.01, 0.1, 0.2,
                                                          0, 3),
                                        'max_depth': (6, 10, 15, 20),
                                        'min_child_weight': (0.5, 1.0, 3.0, 5.0,
                                                             7.0, 10.0),
                                        'n_estimators': [100],
                                        'reg_lambda': (0.1, 1.0, 5.0, 10.0,
                                                       50.0, 100.0),
                                        'subsample': (0.5, 0.6, 0.7, 0.8, 0.9,
                                                      1.0)},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring=None, verbose=2)
```

```
xg_random.best_params_
```

```
{'colsample_bylevel': 0.4,
 'colsample_bytree': 1.0,
 'gamma': 0.5,
 'learning_rate': 0.1,
 'max_depth': 20,
 'min_child_weight': 7.0,
 'n_estimators': 100,
 'reg_lambda': 100.0,
 'subsample': 0.8}
```

```
xg = XGBClassifier()
param_grid = dict(
        max_depth = 20,
        learning_rate = 0.1,
        subsample = 0.8,
        colsample_bytree = 1.0,
        colsample_bylevel =0.4,
        min_child_weight = 7.0,
```

```
        gamma = 0.5,
        reg_lambda = 100.0,
         n_estimators = 100)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)


xg.fit(tfidf_train, y)
```

```
    XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                  colsample_bynode=1, colsample_bytree=1, gamma=0,
                  learning_rate=0.1, max_delta_step=0, max_depth=3,
                  min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                  nthread=None, objective='multi:softprob', random_state=0,
                  reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                  silent=None, subsample=1, verbosity=1)
```

```
print(xg)
```

```
    XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                  colsample_bynode=1, colsample_bytree=1, gamma=0,
                  learning_rate=0.1, max_delta_step=0, max_depth=3,
                  min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                  nthread=None, objective='multi:softprob', random_state=0,
                  reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                  silent=None, subsample=1, verbosity=1)
```

```
ypred = xg.predict(tfidf_test)


predictions = [round(value) for value in ypred]


from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
    Accuracy: 34.99%
```

```
cm = confusion_matrix(y_test, ypred)
print("Confusion Matrix\n")
print(cm)
# Print the Classification Report
cr = classification_report(y_test, pred)
print("\n\nClassification Report\n")
print(cr)
```

```
    Confusion Matrix

    [[156  12 131  59 234]
     [ 21 138  58  47 335]
     [122  29 230 149 511]
     [  8   7  71 196 337]
     [ 35  73 116 114 609]]
```

```
     Classification Report

                   precision    recall  f1-score   support

                0       0.47      0.27      0.34       592
                1       0.52      0.30      0.38       599
                2       0.37      0.33      0.35      1041
                3       0.36      0.51      0.42       619
                4       0.33      0.46      0.39       947

         accuracy                           0.38      3798
        macro avg       0.41      0.37      0.37      3798
     weighted avg       0.40      0.38      0.37      3798
```

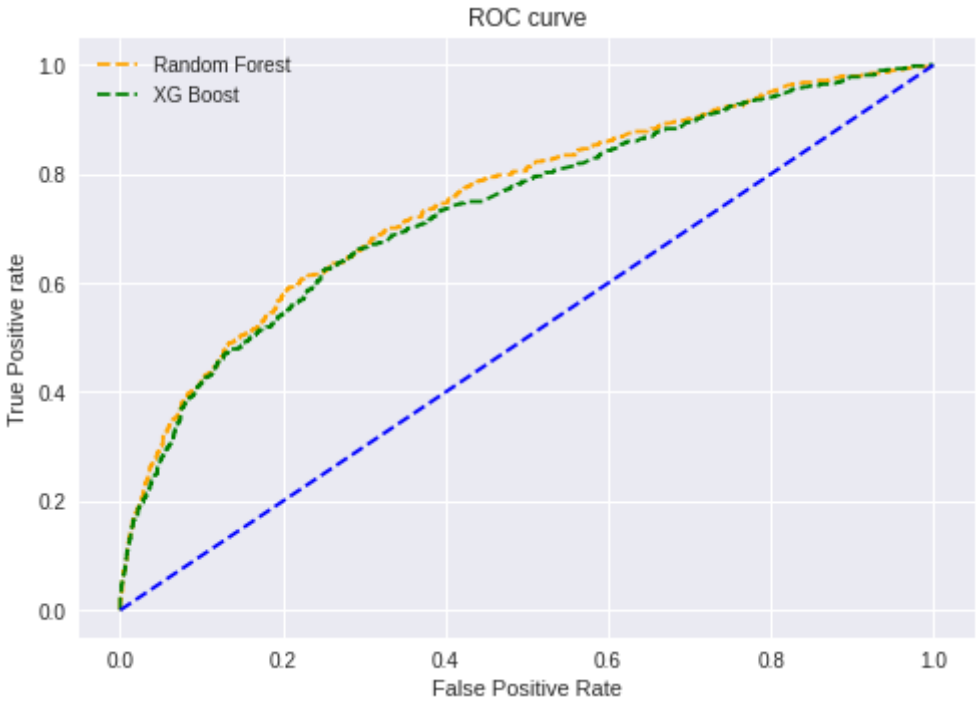
```
def confusion_metrics (conf_matrix):
# save confusion matrix and slice into four pieces
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
    print('True Positives:', TP)
    print('True Negatives:', TN)
    print('False Positives:', FP)
    print('False Negatives:', FN)

    # calculate the sensitivity
    conf_sensitivity = (TP / float(TP + FN))
    # calculate the specificity
    conf_specificity = (TN / float(TN + FP))

    print(f'Sensitivity: {round(conf_sensitivity,2)}')
    print(f'Specificity: {round(conf_specificity,2)}')


confusion_metrics(cm)
```

```
     True Positives: 138
     True Negatives: 156
     False Positives: 12
     False Negatives: 21
     Sensitivity: 0.87
     Specificity: 0.93
```

d)Carry out a ROC analysis to compare the performance of your model with the selected classifier. Plot the ROC graph of the models

```
pred_prob = rfc.predict_proba(tfidf_test)
pred_prob2 = xg.predict_proba(tfidf_test)
```

```
from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, pred_prob2[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)


import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Random Forest')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='XG Boost')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```



e)After tuning your final model, persist using Pickle or Joblib

```
import pickle


filename = 'Random Forest.sav'
pickle.dump(rfc, open(filename, 'wb'))


filename = 'XGBoost.sav'
pickle.dump(xg, open(filename, 'wb'))
```

Thank you