Configuration Manual

MSc Research Project

MSc Data Analytics

Pawan Kumar

Student ID: x22186115

School of Computing

National College of Ireland

Supervisor: Dr. Muslim Jameel Syed

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Pawan Kumar		
Student ID:	X22186115		
Programme:	MSc. Data Analytics	**Year:**	2024
Module:	Research Project		
Lecturer:	Dr. Muslim Jameel Syed		
Submission Due Date:	12 August 2024		
Project Title:	Sentiment Analysis Techniques for Restaurant Reviews Across Multiple Attributes		
Word Count:	**707**	**Page Count: 18**	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Pawan Kumar
Date:	12 August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Pawan Kumar

Student ID: x22186115

System Configuration

The project is done on Google Colab, a cloud-based platform provided by Google that allows Python Code to be written in a web-based Jupyter Notebook Environment. It is mainly used for machine learning and deep learning. The study uses the T4 GPU by Google, which has 15 GB GPU RAM and 32GB System RAM.

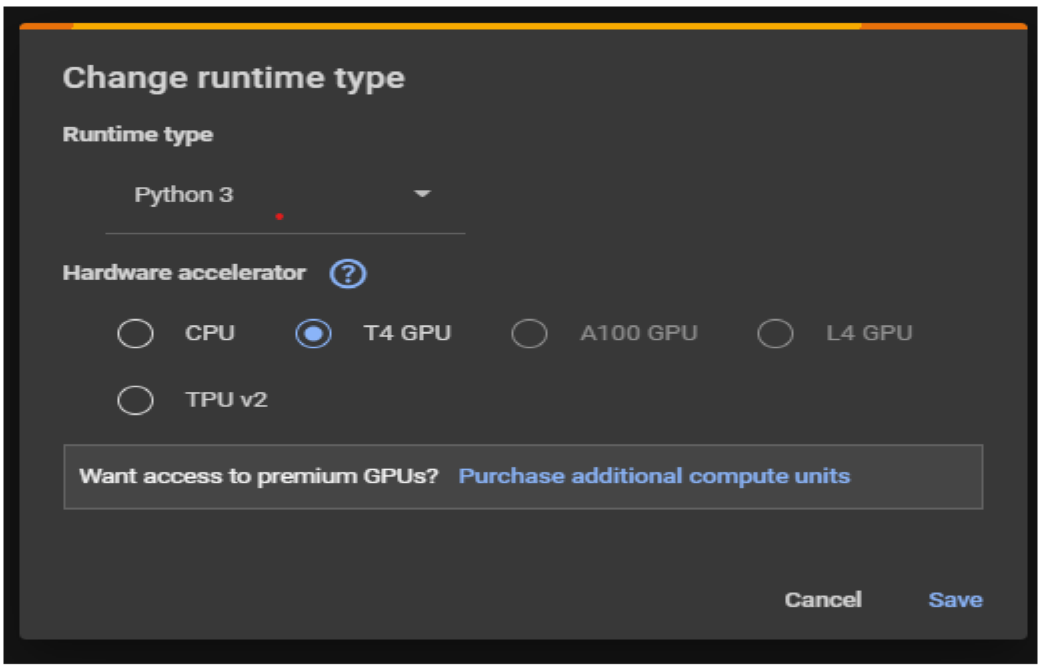


Figure 1 Environment Setup

Software Requirements

For building the project major software used are:

- Google Collab
- Python 3.10

Python Libraries Used

The major Python Libraries used are:

- NumPy
- Pandas
- NLTK
- Seaborn
- Matplotlib
- Plotly
- Keras
- Sci-kit Learn

Dataset

The dataset used in the research is taken from the Zenodo website, which has data available from various restaurants in Dublin across 65 locations.

5 Data Preprocessing

- The dataset is loaded into Google Colab to be used in a notebook.
- Data is analysed 360 degrees to get the insights from the data.

```
data = pd.read_excel('/content/Restaurant Reviews Detailed.xlsx')
```

```
data.head()
```

	Restaurant ID	Location ID	Review	Review Sentiment	Cuisine	Price Range	Food Rating	Service Rating	Ambience Rating	Overall Rating	Restaurant Rating
0	R0001	L01	A great place for Irish music great bar and fu...	Positive	Irish	30 and under	5	5	5	5	4.4
1	R0001	L01	Absolutely exceptional food and staff were ver...	Positive	Irish	30 and under	5	5	5	5	4.4
2	R0001	L01	Cant remember pizza better than this anywhere ...	Positive	Irish	30 and under	5	5	5	5	4.4
3	R0001	L01	Cocktails were very impressive and a perfect e...	Positive	Irish	30 and under	5	5	5	5	4.4
4	R0001	L01	Decent clean and fast Food wasnt bad but ma...	Positive	Irish	30 and under	5	5	5	5	4.4

Figure 2 Data Loading

- A basic data analysis was performed on the dataset

```
plt.figure(figsize=(13,6))
sns.kdeplot(data[data['Review Sentiment']=='Positive']['Food Rating'], label="Positive")
sns.kdeplot(data[data['Review Sentiment']=='Negative']['Food Rating'], label="Negative")
plt.legend(title='Sentiment V/s Food Rating', fontsize=15, title_fontsize=15)
plt.show()
```

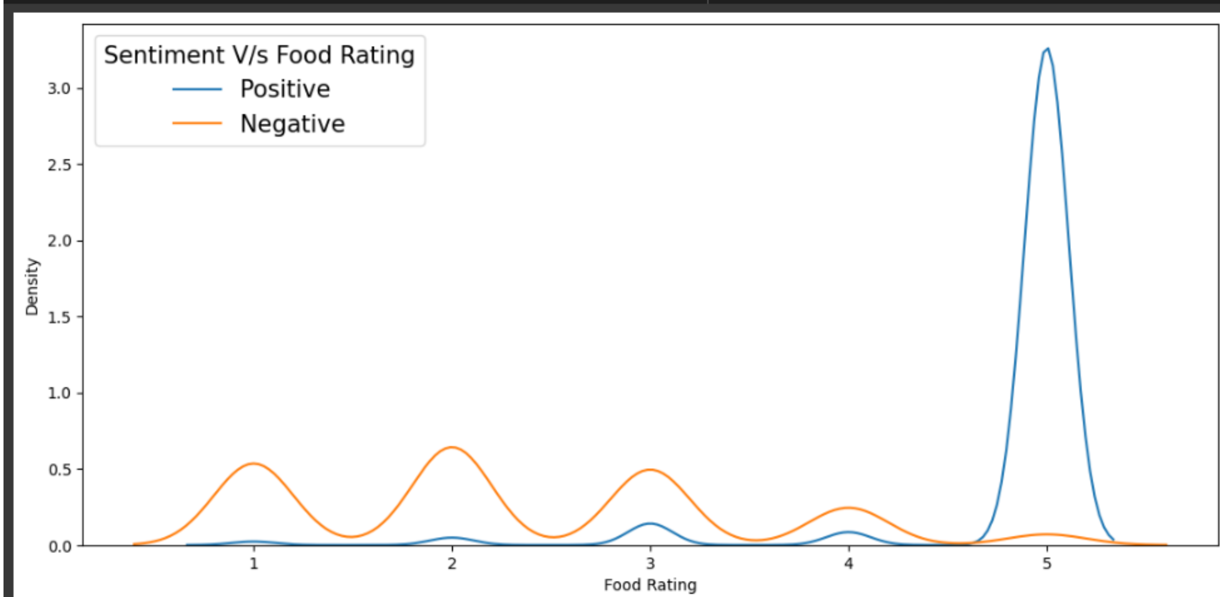


Figure 3 Data Analysis of Sentiment Score and Food Rating

```
plt.figure(figsize=(13,6))
sns.kdeplot(data[data['Review Sentiment']=='Positive']['Service Rating'], label="Positive")
sns.kdeplot(data[data['Review Sentiment']=='Negative']['Service Rating'], label="Negative")
plt.legend(title='Sentiment V/s Service Rating', fontsize=15, title_fontsize=15)
plt.show()
```

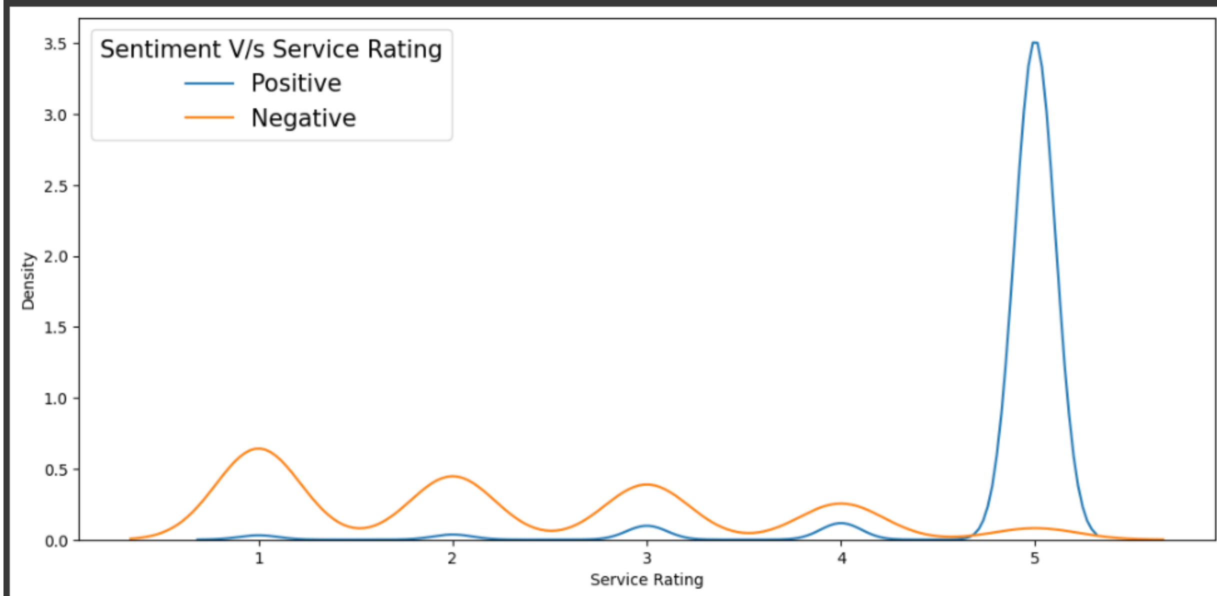


Figure 4 Data Analysis of Sentiment Score and Service Rating

- Text processing on the Independent Variable.
- Figure 4 highlights the case standardization and text cleaning using Regex on the review dataset. The output is highlighting the first review as cleaned output.

```
review = pd.DataFrame(data.Review)
```

```
review
```

	Review
0	A great place for Irish music great bar and fu...
1	Absolutely exceptional food and staff were ver...
2	Cant remember pizza better than this anywhere ...
3	Cocktails were very impressive and a perfect e...
4	Decent clean and fast Food wasnt bad but ma...
...	...
9995	Fabulous food great service Will definitely ...
9996	Great food and excellent service The early bi...
9997	Lovely staff but take away quality food Im afr...
9998	This is our favourite Chinese Restaurant Serv...
9999	We had a fantastic time The food and the serv...

10000 rows × 1 columns

```
review['Review'] = review['Review'].astype(str)
```

```
review['Review'] = review['Review'].apply(lambda x:x.lower())
```

```
def clean_text(x):
    for i in x.split(" "):
        return "".join(re.sub(r"[^a-zA-Z \t]","",x))
```

```
data.Review[0]
```

'A great place for Irish music great bar and full of great people After the Irish band more modern music on a must go'

Figure 5 Data Processing Step 1

- Next step is to remove the stopwords from the dataset.

```
clean_text(review.Review[0])
```

```
'a great place for irish music great bar and full of great people  after the irish band more modern music on a must go'
```

```
review['Review'] = review['Review'].apply(clean_text)
```

```
nltk.download('stopwords') # Download the stopwords corpus

from nltk.corpus import stopwords
stop = stopwords.words('english') # Load the English stopwords
```

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

```
review['Review'] = review['Review'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
```

```
review
```

	Review
0	great place irish music great bar full great p...
1	absolutely exceptional food staff friendly hel...
2	cant remember pizza better anywhere dublin gor...
3	cocktails impressive perfect end seriously tas...
4	decent clean fast food wasnt bad maybe bit sea...
...	...
9995	fabulous food great service definitely return
9996	great food excellent service early bird menu f...
9997	lovely staff take away quality food im afraid ...
9998	favourite chinese restaurant service excellent...
9999	fantastic time food service five star definite...

10000 rows × 1 columns

Figure 6 Data Processing Step 2

- Step 3 of data processing showcase the stemming and lemmatization and the final dataset combined with Y-variable.

```
nltk.download('wordnet')
def lemmatize_stemming(text):
    stemmer=SnowballStemmer('english')
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
review['Review']  = review['Review'].apply(lambda x:lemmatize_stemming(x))
```

```
data = pd.concat([review['Review'] ,data['Review Sentiment']],axis = 1)
```

```
data.rename(columns={'Review Sentiment':'Review_Sentiment'},inplace=True)
```

EDA

```
data.head(2)
```

	Review	Review_Sentiment
0	great place irish music great bar full great p...	Positive
1	absolutely exceptional food staff friendly hel...	Positive

Figure 7 Data Processing Step 3

- Few Word clouds are created for both the Positive Sentiment Reviews and Negative Sentiment Reviews.

Figure 8 Word Cloud of Positive Words

Figure 9 Word Cloud of Negative Words

- The data is divided into training and testing in a split of 70:30.

```python
data['Review_Sentiment'] = data.Review_Sentiment.map({'Negative':0,'Positive':1})
```

```python
data.Review_Sentiment.value_counts()
```

```
Review_Sentiment
1    5000
0    5000
Name: count, dtype: int64
```

```python
X = data.Review
y = data.Review_Sentiment
```

```python
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(X,y,random_state=107,test_size=0.3)
```

```python
print("Shape of train_x:",train_x.shape)
print("Shape of test_x:",test_x.shape)
print("Shape of train_y:",train_y.shape)
print("Shape of test_y:",test_y.shape)
```

```
Shape of train_x: (7000,)
Shape of test_x: (3000,)
Shape of train_y: (7000,)
Shape of test_y: (3000,)
```

Figure 10 Split of Training and Testing Data

- Different Text metrics like TF-IDF, Count Vectorizer and Word2Vec are built on the dataset

```python
count_vect = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}', ngram_range=(1, 2))
xtrain_count = count_vect.fit_transform(train_x)

tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(xtrain_count)
xtest_count = count_vect.transform(test_x)


X_test_tfidf = tfidf_transformer.transform(xtest_count)
```

```python
import gensim
data_model = gensim.models.Word2Vec(sentences=data['Review'], vector_size=100, window=5, min_count=5, workers=4)
word_vectors = data_model.wv

def data_word2vec(text):
    vecs = []
    for word in text.split():
        if word in word_vectors:
            vecs.append(word_vectors[word])
    if len(vecs) > 0:
        return np.mean(vecs, axis=0)
    else:
        return np.zeros(100)

data['Review'] = data['Review'].apply(data_word2vec)
```

```
WARNING:gensim.models.word2vec:Each 'sentences' item should be a list of words (usually unicode strings). First item here is instead plain <class 'str'>.
```

```python
data_features = np.array(list(map(np.array, data['Review'])))
```

```python
word2vec_train_x, word2vec_test_x, word2vec_train_y, word2vec_test_y = train_test_split(data_features, data['Review_Sentiment'], test_size=0.3, random_state=42)
```

Figure 11 Text Metrics

6 Model Training and Testing

5 different approaches are used in the research with 3 text metrics techniques making a total of 15 experiments to get the best model

1. Logistic Regression

1.1 Logistic Regression With CountVectorizer

```
logreg = LogisticRegression()
logreg.fit(xtrain_count, train_y)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
pred_lg_cnt = logreg.predict(xtest_count)
logit_cnt = accuracy_score(test_y, pred_lg_cnt).round(3)

print("Accuracy:",accuracy_score(test_y, pred_lg_cnt).round(3))
logit_cnt_auc = roc_auc_score(test_y, pred_lg_cnt).round(3)
print("ROC-AUC Score: ",logit_cnt_auc)
print(classification_report(test_y, pred_lg_cnt,digits=3))
```

```
Accuracy: 0.947
ROC-AUC Score:  0.947
              precision    recall  f1-score   support

           0      0.950     0.944     0.947      1505
           1      0.944     0.951     0.947      1495

    accuracy                          0.947      3000
   macro avg      0.947     0.947     0.947      3000
weighted avg      0.947     0.947     0.947      3000
```

Figure 12 Logistic Regression With Count Vectorizer

1.2 Logistic Regression With TF-IDF

```
logreg = LogisticRegression()
logreg.fit(X_train_tfidf, train_y)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
pred_lg_tfidf = logreg.predict(X_test_tfidf)
logit_tfidf = accuracy_score(test_y, pred_lg_tfidf).round(3)
print("Accuracy:",accuracy_score(test_y, pred_lg_tfidf).round(3))
logit_tfidf_auc = roc_auc_score(test_y, pred_lg_tfidf).round(3)
print("ROC-AUC Score: ",logit_tfidf_auc)
print(classification_report(test_y, pred_lg_tfidf,digits=3))
```

```
Accuracy: 0.944
ROC-AUC Score:  0.944
              precision    recall  f1-score   support

           0      0.926     0.966     0.945      1505
           1      0.964     0.922     0.943      1495

    accuracy                          0.944      3000
   macro avg      0.945     0.944     0.944      3000
weighted avg      0.945     0.944     0.944      3000
```

Figure 13 Logistic Regression With TF-IDF

1.3 Logistic Regression with Word2Vec

```
data_lr_model = LogisticRegression(max_iter=1000)
data_lr_model.fit(word2vec_train_x, word2vec_train_y)
data_lr_pred_y = data_lr_model.predict(word2vec_test_x)

acc_word2vec_logit = accuracy_score(word2vec_test_y, data_lr_pred_y).round(3)
print("Accuracy:", accuracy_score(word2vec_test_y, data_lr_pred_y).round(3))
logit_word2vec_auc = roc_auc_score(word2vec_test_y, data_lr_pred_y).round(3)
print("ROC-AUC Score: ",logit_word2vec_auc)
print(classification_report(word2vec_test_y, data_lr_pred_y,digits=3))
```

```
Accuracy: 0.522
ROC-AUC Score:  0.519
              precision    recall  f1-score   support

           0      0.777     0.054     0.100      1492
           1      0.513     0.985     0.674      1508

    accuracy                          0.522      3000
   macro avg      0.645     0.519     0.387      3000
weighted avg      0.644     0.522     0.389      3000
```

Figure 14 Logistic Regression With Word2Vec

2. Naïve Bayes

2.1 Naive Bayes With CountVectorizer

```
nb = MultinomialNB()
# Fit the data
nb.fit(xtrain_count, train_y)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
pred_nb_cnt = nb.predict(xtest_count)
nb_cnt = accuracy_score(test_y, pred_nb_cnt).round(3)
print("Accuracy:",accuracy_score(test_y, pred_nb_cnt).round(3))
nb_cnt_auc = roc_auc_score(test_y, pred_nb_cnt).round(3)
print("ROC-AUC Score: ",nb_cnt_auc)
print(classification_report(test_y, pred_nb_cnt,digits=3))
```

```
Accuracy: 0.937
ROC-AUC Score:  0.937
              precision    recall  f1-score   support

           0      0.907     0.975     0.940      1505
           1      0.973     0.900     0.935      1495

    accuracy                          0.937      3000
   macro avg      0.940     0.937     0.937      3000
weighted avg      0.940     0.937     0.937      3000
```

Figure 15 Naive Bayes With Count Vectorizer

2.2 Naive Bayes With TF-IDF

```
nb = MultinomialNB()
# Fit the data
nb.fit(X_train_tfidf, train_y)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
pred_nb_tfidf = nb.predict(X_test_tfidf)
nb_tfidf = accuracy_score(test_y, pred_nb_tfidf).round(3)
print("Accuracy:",accuracy_score(test_y, pred_nb_tfidf).round(3))
nb_tfidf_auc = roc_auc_score(test_y, pred_nb_tfidf).round(3)
print("ROC-AUC Score: ",nb_tfidf_auc)
print(classification_report(test_y, pred_nb_tfidf,digits=3))
```

```
Accuracy: 0.939
ROC-AUC Score:  0.939
              precision    recall  f1-score   support

           0      0.911     0.974     0.942      1505
           1      0.972     0.904     0.937      1495

    accuracy                          0.939      3000
   macro avg      0.942     0.939     0.939      3000
weighted avg      0.941     0.939     0.939      3000
```

Figure 16 Naive Bayes with TF-IDF

2.3 Naive Bayes with Word2Vec

```python
# Train and Evaluate Naive Bayes with Word2Vec features for Crisis Dataset
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score

# Naive Bayes for the Crisis dataset
word2vec_nb_model = GaussianNB()
word2vec_nb_model.fit(word2vec_train_x, word2vec_train_y)
word2vec_nb_pred_y = word2vec_nb_model.predict(word2vec_test_x)
acc_nb_word2vec =  accuracy_score(word2vec_test_y, word2vec_nb_pred_y).round(3)
print("Accuracy:", accuracy_score(word2vec_test_y, word2vec_nb_pred_y))
nb_word2vec_auc = roc_auc_score(word2vec_test_y, word2vec_nb_pred_y).round(3)
print("ROC-AUC Score: ",nb_word2vec_auc)
print(classification_report(word2vec_test_y, word2vec_nb_pred_y,digits=3))
```

```
Accuracy: 0.523
ROC-AUC Score:  0.521
              precision    recall  f1-score   support

           0      0.748     0.062     0.114      1492
           1      0.513     0.979     0.674      1508

    accuracy                          0.523      3000
   macro avg      0.631     0.521     0.394      3000
weighted avg      0.630     0.523     0.395      3000
```

Figure 17 Naive Bayes with Word2Vec

3. Xg-Boost

3.1 XgBoost with CountVectorizer

```python
from xgboost import XGBClassifier
xgb_estimator = XGBClassifier(n_estimators=200,
                              random_state = 42,
                             n_jobs=-1)
xgb_estimator.fit(xtrain_count, train_y)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=-1,
              num_parallel_tree=None, random_state=42, ...)
```

```python
pred_xgb_cnt = xgb_estimator.predict(xtest_count)
xgb_cnt = accuracy_score(test_y, pred_xgb_cnt).round(3)

print("Accuracy:",accuracy_score(test_y, pred_xgb_cnt).round(3))
xgb_cnt_auc = roc_auc_score(test_y, pred_xgb_cnt).round(3)
print("ROC-AUC Score: ",xgb_cnt_auc)
print(classification_report(test_y, pred_xgb_cnt,digits=3))
```

```
Accuracy: 0.938
ROC-AUC Score:  0.938
              precision    recall  f1-score   support

           0      0.941     0.936     0.938      1505
           1      0.936     0.940     0.938      1495

    accuracy                          0.938      3000
   macro avg      0.938     0.938     0.938      3000
weighted avg      0.938     0.938     0.938      3000
```

Figure 18 XgBoost with Count Vectorizer

3.2 XgBoost With TF-IDF

```
from xgboost import XGBClassifier
xgb_estimator = XGBClassifier(n_estimators=200,
                        random_state = 42,
                       n_jobs=-1)
xgb_estimator.fit(X_train_tfidf, train_y)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=-1,
              num_parallel_tree=None, random_state=42, ...)
```

```
pred_xgb_tfidf = xgb_estimator.predict(X_test_tfidf)
xgb_tfidf = accuracy_score(test_y, pred_xgb_tfidf).round(3)
print("Accuracy:",accuracy_score(test_y, pred_xgb_tfidf).round(3))
xgb_tfidf_auc = roc_auc_score(test_y, pred_xgb_tfidf).round(3)
print("ROC-AUC Score: ",xgb_tfidf_auc)
print(classification_report(test_y, pred_xgb_tfidf,digits=3))
```

```
Accuracy: 0.936
ROC-AUC Score:  0.936
              precision    recall  f1-score   support

           0      0.945     0.928     0.936      1505
           1      0.928     0.945     0.937      1495

    accuracy                          0.936      3000
   macro avg      0.936     0.936     0.936      3000
weighted avg      0.936     0.936     0.936      3000
```

Figure 19 XgBoost with TF-IDF

3.3 XgBoost with Word2Vec

```
from xgboost import XGBClassifier
xgb_estimator = XGBClassifier(n_estimators=200,
                        random_state = 42,
                       n_jobs=-1)
xgb_estimator.fit(word2vec_train_x, word2vec_train_y)
data_xgb_pred_y = xgb_estimator.predict(word2vec_test_x)

acc_word2vec_xgb = accuracy_score(word2vec_test_y, data_xgb_pred_y).round(3)
print("Accuracy:", accuracy_score(word2vec_test_y, data_xgb_pred_y).round(3))
xgb_word2vec_auc = roc_auc_score(word2vec_test_y, data_xgb_pred_y).round(3)
print("ROC-AUC Score: ",xgb_word2vec_auc)
print(classification_report(word2vec_test_y, data_xgb_pred_y,digits=3))
```

```
Accuracy: 0.522
ROC-AUC Score:  0.519
              precision    recall  f1-score   support

           0      0.766     0.055     0.103      1492
           1      0.513     0.983     0.674      1508

    accuracy                          0.522      3000
   macro avg      0.639     0.519     0.388      3000
weighted avg      0.639     0.522     0.390      3000
```

Figure 20 XgBoost with Word2Vec

4. Random Forest

4.1 Random Forest With CountVectorizer

```
from sklearn.ensemble import RandomForestClassifier
radm_clf = RandomForestClassifier(oob_score=True,n_estimators=100, n_jobs=-1)
radm_clf.fit( xtrain_count, train_y)
```

```
RandomForestClassifier
RandomForestClassifier(n_jobs=-1, oob_score=True)
```

```
pred_rf_cnt = radm_clf.predict(xtest_count)
rf_cnt = accuracy_score(test_y, pred_rf_cnt).round(3)
print("Accuracy:",accuracy_score(test_y, pred_rf_cnt).round(3))
rf_cnt_auc = roc_auc_score(test_y, pred_rf_cnt).round(3)
print("ROC-AUC Score: ",rf_cnt_auc)
print(classification_report(test_y, pred_rf_cnt,digits=3))
```

```
Accuracy: 0.937
ROC-AUC Score:  0.937
              precision    recall  f1-score   support

           0      0.949     0.925     0.937      1505
           1      0.926     0.950     0.938      1495

    accuracy                          0.937      3000
   macro avg      0.938     0.937     0.937      3000
weighted avg      0.938     0.937     0.937      3000
```

Figure 21 Random Forest With Count Vectorizer

4.2 Random Forest With TF-IDF

```
radm_clf = RandomForestClassifier(oob_score=True,n_estimators=100, n_jobs=-1)
radm_clf.fit( X_train_tfidf, train_y)
```

```
RandomForestClassifier
RandomForestClassifier(n_jobs=-1, oob_score=True)
```

```
pred_rf_tfidf = radm_clf.predict(X_test_tfidf)
rf_tfidf = accuracy_score(test_y, pred_rf_tfidf).round(3)
print("Accuracy:",accuracy_score(test_y, pred_rf_tfidf).round(3))
rf_tfidf_auc = roc_auc_score(test_y, pred_rf_tfidf).round(3)
print("ROC-AUC Score: ",rf_tfidf_auc)
print(classification_report(test_y, pred_rf_tfidf,digits=3))
```

```
Accuracy: 0.921
ROC-AUC Score:  0.921
              precision    recall  f1-score   support

           0      0.954     0.885     0.918      1505
           1      0.892     0.957     0.924      1495

    accuracy                          0.921      3000
   macro avg      0.923     0.921     0.921      3000
weighted avg      0.923     0.921     0.921      3000
```

Figure 22 Random Forest With TF-IDF

4.3 Random Forest With Word2Vec

```
radm_clf = RandomForestClassifier(oob_score=True,n_estimators=100, n_jobs=-1)
radm_clf.fit( word2vec_train_x, word2vec_train_y)
```

```
▾            RandomForestClassifier
RandomForestClassifier(n_jobs=-1, oob_score=True)
```

```
data_word2vec_pred_y_rf = xgb_estimator.predict(word2vec_test_x)

acc_word2vec_rf = accuracy_score(word2vec_test_y, data_word2vec_pred_y_rf).round(3)
print("Accuracy:", accuracy_score(word2vec_test_y, data_word2vec_pred_y_rf).round(3))
rf_word2vec_auc = roc_auc_score(word2vec_test_y, data_word2vec_pred_y_rf).round(3)
print("ROC-AUC Score: ",xgb_word2vec_auc)
print(classification_report(word2vec_test_y, data_word2vec_pred_y_rf,digits=3))
```

```
Accuracy: 0.522
ROC-AUC Score:  0.519
              precision    recall  f1-score   support

           0      0.766     0.055     0.103      1492
           1      0.513     0.983     0.674      1508

    accuracy                          0.522      3000
   macro avg      0.639     0.519     0.388      3000
weighted avg      0.639     0.522     0.390      3000
```

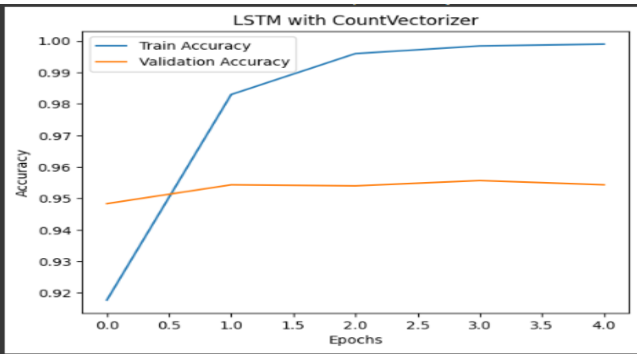
Figure 23 Random Forest With Word2Vec

5. LSTM

```
from tensorflow.keras.utils import Sequence
# Define a custom data generator
class SparseDataGenerator(Sequence):
    def __init__(self, x_data, y_data, batch_size):
        self.x, self.y = x_data, y_data
        self.batch_size = batch_size

    def __len__(self):
        return np.ceil(self.x.shape[0] / self.batch_size).astype(np.int64)

    def __getitem__(self, idx):
        batch_x = self.x[idx * self.batch_size:(idx + 1) * self.batch_size]
        batch_y = self.y[idx * self.batch_size:(idx + 1) * self.batch_size]
        # Convert sparse matrix batch to dense and reshape for LSTM
        batch_x_dense = batch_x.toarray()
        batch_x_reshaped = np.reshape(batch_x_dense, (batch_x_dense.shape[0], 1, batch_x_dense.shape[1]))
        return np.array(batch_x_reshaped), np.array(batch_y)

# Batch size
batch_size = 32

# Create generators for training and testing data
train_generator = SparseDataGenerator(xtrain_count, train_y, batch_size)
test_generator = SparseDataGenerator(xtest_count, test_y, batch_size)

# Define LSTM model
crisis_count_lstm = Sequential()
crisis_count_lstm.add(LSTM(64, input_shape=(1, xtrain_count.shape[1])))
crisis_count_lstm.add(Dropout(0.2))
crisis_count_lstm.add(Dense(1, activation='sigmoid'))

# Compile the model
crisis_count_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model using generators
crisis_count_lstm_history = crisis_count_lstm.fit(train_generator, epochs=5, validation_data=test_generator)

# Plot training history
plt.plot(crisis_count_lstm_history.history['accuracy'], label='Train Accuracy')
plt.plot(crisis_count_lstm_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('LSTM with CountVectorizer')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/5
219/219 ━━━━━━━━━━━━━━━━━━━━ 163s 735ms/step - accuracy: 0.8904 - loss: 0.4456 - val_accuracy: 0.9443 - val_loss: 0.1670
Epoch 2/5
219/219 ━━━━━━━━━━━━━━━━━━━━ 189s 676ms/step - accuracy: 0.9795 - loss: 0.0830 - val_accuracy: 0.9543 - val_loss: 0.1398
Epoch 3/5
219/219 ━━━━━━━━━━━━━━━━━━━━ 149s 679ms/step - accuracy: 0.9956 - loss: 0.0316 - val_accuracy: 0.9580 - val_loss: 0.1351
Epoch 4/5
219/219 ━━━━━━━━━━━━━━━━━━━━ 152s 695ms/step - accuracy: 0.9979 - loss: 0.0143 - val_accuracy: 0.9573 - val_loss: 0.1393
Epoch 5/5
219/219 ━━━━━━━━━━━━━━━━━━━━ 151s 687ms/step - accuracy: 0.9988 - loss: 0.0088 - val_accuracy: 0.9560 - val_loss: 0.1428
```

Figure 24 LSTM with Count Vectorizer



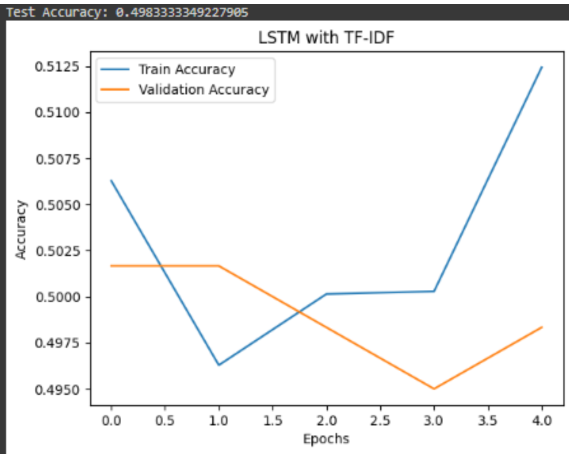

```
[ ] pred_lstm_cnt = crisis_count_lstm.predict(test_generator)

94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step

[ ] print("Accuracy:", accuracy_score(test_y, pred_lstm_cnt.round().astype(int), normalize=True).round(3))

Accuracy: 0.954

[ ]
    pred_lstm_cnt_binary = pred_lstm_cnt.round().astype(int)


    print("ROC-AUC Score: ",lstm_cnt_auc)
    print(classification_report(test_y, pred_lstm_cnt_binary, digits=3))

ROC-AUC Score:  0.989
              precision    recall  f1-score   support

           0      0.952     0.957     0.955      1505
           1      0.957     0.951     0.954      1495

    accuracy                          0.954      3000
   macro avg      0.954     0.954     0.954      3000
weighted avg      0.954     0.954     0.954      3000
```

Figure 25 LSTM with Count Vectorizer Evaluation

```
from sklearn.decomposition import TruncatedSVD

# Dimensionality reduction
svd = TruncatedSVD(n_components=100)
X_train_reduced = svd.fit_transform(X_train_tfidf)
X_test_reduced = svd.transform(X_test_tfidf)

# Create a generator function for batches
class DataGenerator(Sequence):
    def __init__(self, X_data, y_data, batch_size):
        self.X, self.y = X_data, y_data
        self.batch_size = batch_size

    def __len__(self):
        return np.ceil(len(self.X) / self.batch_size).astype(np.int64)

    def __getitem__(self, idx):
        batch_x = self.X[idx * self.batch_size:(idx + 1) * self.batch_size]
        batch_y = self.y[idx * self.batch_size:(idx + 1) * self.batch_size]
        return np.array(batch_x), np.array(batch_y)

# Batch size
batch_size = 32

# Create generators
train_generator = DataGenerator(X_train_reduced, train_y, batch_size)
test_generator = DataGenerator(X_test_reduced, test_y, batch_size)

# LSTM model
crisis_tfidf_lstm = Sequential()
crisis_tfidf_lstm.add(LSTM(128, return_sequences=True, input_shape=(X_train_reduced.shape[1], 1))) # Increased LSTM units and added return_sequences
crisis_tfidf_lstm.add(Dropout(0.3)) # Increased dropout for regularization
crisis_tfidf_lstm.add(LSTM(64)) # Additional LSTM layer
crisis_tfidf_lstm.add(Dropout(0.2)) # Additional dropout layer
crisis_tfidf_lstm.add(Dense(32, activation='relu')) # Additional Dense layer
crisis_tfidf_lstm.add(Dense(1, activation='sigmoid'))

# Compile and train
crisis_tfidf_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
crisis_tfidf_lstm_history = crisis_tfidf_lstm.fit(train_generator, epochs=5, validation_data=test_generator)

# Evaluate
print('Test Accuracy:', crisis_tfidf_lstm.evaluate(X_test_reduced, test_y)[1])
plt.plot(crisis_tfidf_lstm_history.history['accuracy'], label='Train Accuracy')
plt.plot(crisis_tfidf_lstm_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('LSTM with TF-IDF')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

Epoch 1/5
219/219 ━━━━━━━━━━━━━━━━━━━━ 6s 14ms/step - accuracy: 0.5051 - loss: 0.6936 - val_accuracy: 0.5017 - val_loss: 0.6957
Epoch 2/5
219/219 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.4954 - loss: 0.6933 - val_accuracy: 0.5017 - val_loss: 0.6932
Epoch 3/5
219/219 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.4992 - loss: 0.6934 - val_accuracy: 0.4983 - val_loss: 0.6933
Epoch 4/5
219/219 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.5020 - loss: 0.6932 - val_accuracy: 0.4950 - val_loss: 0.6929
Epoch 5/5
219/219 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.5046 - loss: 0.6964 - val_accuracy: 0.4983 - val_loss: 0.6992
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5098 - loss: 0.6967
```

Figure 26 LSTM with TF-IDF

```
Test Accuracy: 0.4983333349227905
```




```
[ ] pred_lstm_tfidf = crisis_tfidf_lstm.predict(X_test_reduced)
    print("Accuracy:", accuracy_score(test_y, pred_lstm_tfidf.round(), normalize=True).round(3))
    tfidf_lstm_acc = accuracy_score(test_y, pred_lstm_tfidf.round(), normalize=True).round(3)
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
Accuracy: 0.498
```

```
[ ] lstm_tfidf_auc = roc_auc_score(test_y, pred_lstm_tfidf).round(3)
    print("ROC-AUC Score: ",lstm_tfidf_auc)
    print(classification_report(test_y, np.round(abs(pred_lstm_tfidf))))
```

```
ROC-AUC Score:  0.5
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1505
           1       0.50      1.00      0.67      1495

    accuracy                           0.50      3000
   macro avg       0.25      0.50      0.33      3000
weighted avg       0.25      0.50      0.33      3000
```

Figure 27 LSTM with TF-IDF Evaluation

```
restaurant_word2vec_lstm = Sequential()
restaurant_word2vec_lstm.add(LSTM(64, input_shape=(word2vec_train_x.shape[1], 1)))
restaurant_word2vec_lstm.add(Dropout(0.2))
restaurant_word2vec_lstm.add(Dense(1, activation='sigmoid'))
restaurant_word2vec_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
restaurant_word2vec_lstm_history = restaurant_word2vec_lstm.fit(word2vec_train_x, word2vec_train_y, epochs=5, batch_size=64, validation_data=(word2vec_test_x, word2vec_test_y))
```

```
Epoch 1/5
110/110 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.4970 - loss: 0.6933 - val_accuracy: 0.4967 - val_loss: 0.6930
Epoch 2/5
110/110 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.5037 - loss: 0.6932 - val_accuracy: 0.4967 - val_loss: 0.6925
Epoch 3/5
110/110 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.5063 - loss: 0.6968 - val_accuracy: 0.5027 - val_loss: 0.7008
Epoch 4/5
110/110 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.4952 - loss: 0.6992 - val_accuracy: 0.4973 - val_loss: 0.6947
Epoch 5/5
110/110 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.4981 - loss: 0.6956 - val_accuracy: 0.5050 - val_loss: 0.6930
```

[ ] Start coding or generate with AI.

```
plt.plot(restaurant_word2vec_lstm_history.history['accuracy'], label='Train Accuracy')
plt.plot(restaurant_word2vec_lstm_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('LSTM with Word2Vec')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

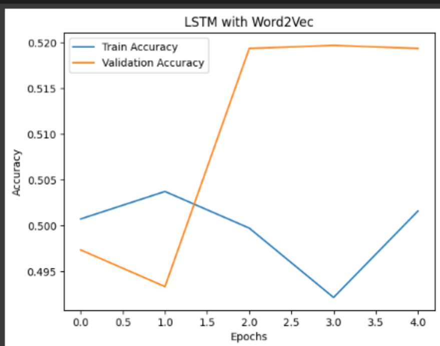


Figure 28 LSTM with Word2Vec

```
pred_lstm_word2vec = restaurant_word2vec_lstm.predict(word2vec_test_x)
print("Accuracy:", accuracy_score(word2vec_test_y, pred_lstm_word2vec.round(),normalize=True).round(3))
word2vec_lstm_acc = accuracy_score(word2vec_test_y, pred_lstm_word2vec.round(),normalize=True).round(3)
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
Accuracy: 0.505
```

```
lstm_word2vec_auc = roc_auc_score(word2vec_test_y, pred_lstm_word2vec).round(3)
print("ROC-AUC Score: ",lstm_word2vec_auc)
print(classification_report(word2vec_test_y, pred_lstm_word2vec.round(),digits=3))
```

```
ROC-AUC Score:  0.515
              precision    recall  f1-score   support

           0      0.769     0.007     0.013      1492
           1      0.504     0.998     0.670      1508

    accuracy                          0.505      3000
   macro avg      0.637     0.502     0.341      3000
weighted avg      0.636     0.505     0.343      3000
```

Figure 29 LSTM with Word2Vec Evaluation