In [1]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import GridSearchCV
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from tqdm import tqdm
import copy
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import seaborn as sns
from textblob import TextBlob
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
```

In [3]:
```
import spacy
```

In [4]:
```
import tensorflow as tf
```

In [5]:
```
from keras.models import Sequential
```

In [6]:
```
from keras import layers
```

In [7]:
```
from keras import backend as K
```

In [8]:
```
from keras.preprocessing.text import Tokenizer
```

In [9]:
```
from sklearn.cluster import KMeans
```

In [10]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [11]:
```
from sklearn.ensemble import RandomForestClassifier
import re
from sklearn import metrics
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
import operator
```

In [12]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

In [13]:
```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score,f1_score,precision_score, recall
from scipy.sparse import hstack
```

In [14]:
```
import pandas as pd
import json

yelp_review=pd.read_json("yelp_academic_dataset_review.json", lines=True)
```

In [15]:
```
import pandas as pd
import json
yelp_business=pd.read_json("yelp_academic_dataset_business.json" ,lines=Tru
```

In [16]:
```
def reviews_restaurants(business_data,review_data):
    restaurant_data =  business_data[business_data['categories'].str.contai
    restaurant_reviews = review_data[review_data.business_id.isin(restauran
    return(restaurant_reviews)
```

In [17]:
```
review_restaurant_data = reviews_restaurants(yelp_business, yelp_review)
```

In [18]:
```
review_restaurant_data.to_csv("review_rest_data.csv",index = False)

review_restaurant_data = pd.read_csv("review_rest_data.csv")
```

In [19]:
```
review_restaurant_data = review_restaurant_data[["text","stars"]]
review_restaurant_data = review_restaurant_data.reset_index(drop = True)
```

In [20]:
```
def label_data(data):
    target = {"Target_sentiment":[]}
    for i in data["stars"]:
        j = 0
        if i > 3:
            j = 1
            target["Target_sentiment"].append(j)
        else:
            target["Target_sentiment"].append(j)
    data = data.join(pd.DataFrame(target, index = data.index))

    #we drop the stars column because it is not useful to us now
    data = data.drop('stars', axis = 1)
    data_sample = data.sample(n = 10000, random_state = 42)
    data_sample = data_sample.reset_index(drop = True)
    return(data_sample)
```

In [21]:
```
restaurant_reviews_after_labels = label_data(review_restaurant_data)
```

In [22]:

```
## Function for replacing contractions with normal words
def replace_contractions(data):
    data = re.sub(r"ain't", "am not", data)
    data = re.sub(r"aren't", "are not", data)
    data = re.sub(r"can't", "can not", data)
    data = re.sub(r"can't've", "can not have", data)
    data = re.sub(r"'cause", "because", data)
    data = re.sub(r"could've", "could have", data)
    data = re.sub(r"couldn't", "could not", data)
    data = re.sub(r"couldn't've", "could not have", data)
    data = re.sub(r"doesn't", "does not", data)
    data = re.sub(r"hadn't", "had not", data)
    data = re.sub(r"hadn't've", "had not have", data)
    data = re.sub(r"hasn't", "has not", data)
    data = re.sub(r"haven't", "have not", data)
    data = re.sub(r"he'd", "he had", data)
    data = re.sub(r"he'd've", "he would have", data)
    data = re.sub(r"he'll", "he will", data)
    data = re.sub(r"he'll've", "he will have", data)
    data = re.sub(r"he's", "he has", data)
    data = re.sub(r"how'd", "how did", data)
    data = re.sub(r"how'd'y", "how do you", data)
    data = re.sub(r"how'll", "how will", data)
    data = re.sub(r"how's", "how has", data)
    data = re.sub(r"i'd", "i had", data)
    data = re.sub(r"i'd've", "i would have", data)
    data = re.sub(r"i'll", "i shall", data)
    data = re.sub(r"i'll've", "i shall have", data)
    data = re.sub(r"i'm", "i am", data)
    data = re.sub(r"i've", "i have", data)
    data = re.sub(r"isn't", "is not", data)
    data = re.sub(r"it'd", "it had", data)
    data = re.sub(r"it'd've", "it would have", data)
    data = re.sub(r"it'll", "it shall", data)
    data = re.sub(r"it'll've", "it shall have", data)
    data = re.sub(r"it's", "it has", data)
    data = re.sub(r"let's", "let us", data)
    data = re.sub(r"ma'am", "madam", data)
    data = re.sub(r"mayn't", "may not", data)
    data = re.sub(r"might've", "might have", data)
    data = re.sub(r"mightn't", "might not", data)
    data = re.sub(r"mightn't've", "might not have", data)
    data = re.sub(r"must've", "must have", data)
    data = re.sub(r"mustn't", "must not", data)
    data = re.sub(r"mustn't've", "must not have", data)
    data = re.sub(r"needn't", "need not", data)
    data = re.sub(r"needn't've", "need not have", data)
    data = re.sub(r"o'clock", "of the clock", data)
    data = re.sub(r"oughtn't", "ought not", data)
    data = re.sub(r"oughtn't've", "ought not have", data)
    data = re.sub(r"shan't", "shall not", data)
    data = re.sub(r"sha'n't", "shall not", data)
    data = re.sub(r"shan't've", "shall not have", data)
    data = re.sub(r"she'd", "she had", data)
    data = re.sub(r"she'd've", "she would have", data)
    data = re.sub(r"she'll", "she shall", data)
```

```
data = re.sub(r"she'll've", "she shall have", data)
data = re.sub(r"she's", "she has", data)
data = re.sub(r"should've", "should have", data)
data = re.sub(r"shouldn't", "should not", data)
data = re.sub(r"shouldn't've", "should not have", data)
data = re.sub(r"so've", "so have", data)
data = re.sub(r"so's", "so as", data)
data = re.sub(r"that'd", "that would", data)
data = re.sub(r"that'd've", "that would have", data)
data = re.sub(r"that's", "that has", data)
data = re.sub(r"there'd", "there had", data)
data = re.sub(r"there'd've", "there would have", data)
data = re.sub(r"there's", "there has", data)
data = re.sub(r"they'd", "they had", data)
data = re.sub(r"they'd've", "they would have", data)
data = re.sub(r"they'll", "they shall", data)
data = re.sub(r"they'll've", "they shall have", data)
data = re.sub(r"they're", "they are", data)
data = re.sub(r"they've", "they have", data)
data = re.sub(r"to've", "to have", data)
data = re.sub(r"wasn't", "was not", data)
data = re.sub(r"we'd", "we had", data)
data = re.sub(r"we'd've", "we would have", data)
data = re.sub(r"we'll", "we will", data)
data = re.sub(r"we'll've", "we will have", data)
data = re.sub(r"we're", "we are", data)
data = re.sub(r"we've", "we have", data)
data = re.sub(r"weren't", "were not", data)
data = re.sub(r"what'll", "what shall", data)
data = re.sub(r"what'll've", "what shall have", data)
data = re.sub(r"what're", "what are", data)
data = re.sub(r"what's", "what has", data)
data = re.sub(r"what've", "what have", data)
data = re.sub(r"when's", "when has", data)
data = re.sub(r"when've", "when have", data)
data = re.sub(r"where'd", "where did", data)
data = re.sub(r"where's", "where has", data)
data = re.sub(r"where've", "where have", data)
data = re.sub(r"who'll", "who shall", data)
data = re.sub(r"who'll've", "who shall have", data)
data = re.sub(r"who's", "who has", data)
data = re.sub(r"who've", "who have", data)
data = re.sub(r"why's", "why has", data)
data = re.sub(r"why've", "why have", data)
data = re.sub(r"will've", "will have", data)
data = re.sub(r"won't", "will not", data)
data = re.sub(r"won't've", "will not have", data)
data = re.sub(r"would've", "would have", data)
data = re.sub(r"wouldn't", "would not", data)
data = re.sub(r"wouldn't've", "would not have", data)
data = re.sub(r"y'all", "you all", data)
data = re.sub(r"y'all'd", "you all would", data)
data = re.sub(r"y'all'd've", "you all would have", data)
data = re.sub(r"y'all're", "you all are", data)
data = re.sub(r"y'all've", "you all have", data)
data = re.sub(r"you'd", "you had", data)
data = re.sub(r"you'd've", "you would have", data)
```

```
    data = re.sub(r"you'll", "you shall", data)
    data = re.sub(r"you'll've", "you shall have", data)
    data = re.sub(r"how's", "how has", data)
    data = re.sub(r"you're", "you are", data)
    data = re.sub(r"you've", "you have", data)
    data = re.sub(r"didn't", "did not", data)
    data = re.sub(r"don't", "do not", data)
    data = re.sub(r"'","",data)
    data = re.sub(r". . .","",data)
    return(data)
```

In [23]:
```
def remove_unnecessary(data):
    for index, row in tqdm(data.iterrows()):
        cleaned_text = ""
        preprocess_word = re.sub(r'([\d]+[a-zA-Z]+)|([a-zA-Z]+[\d]+)', "",
        preprocess_word = re.sub(r"(^|\s)(\-?\d+(?:\.\d)*|\d+|[\d]+[A-Za-z]
        preprocess_word = re.sub('[^A-Za-z\']+', " ", preprocess_word)
        cleaned_text = cleaned_text + preprocess_word
        cleaned_text = replace_contractions(cleaned_text)
        data["text"][index] = cleaned_text
    return(data)
```

In [24]:
```
def remove_stopwords_and_lemmatize(data):
    copy_data = copy.deepcopy(data)
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english')) - set(['no', 'not'])
    for index, row in tqdm(copy_data.iterrows()):
        sent = ''
        for e in row["text"].split():
            if e not in stop_words:
                e = lemmatizer.lemmatize(e, pos ="a")
                sent = ' '.join([sent,e])
        copy_data["text"][index] = sent
    return(copy_data)
```

In [25]:
```
restaurants_reviews_preprocessed = remove_unnecessary(restaurant_reviews_af
```

```
10000it [00:03, 2774.69it/s]
```

In [26]:
```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')

restaurants_reviews_preprocessed_lemmatize = remove_stopwords_and_lemmatize
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/chiragarora/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/chiragarora/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
10000it [00:02, 3573.57it/s]
```

In [27]:
```
def restructure_data(pre_processed_data):
    cleaned_data = copy.deepcopy(pre_processed_data)
    cleaned_data["sentiment_polarity"] = cleaned_data["text"].map(lambda te
    cleaned_data["text_length"] = cleaned_data["text"].astype(str).apply(le
    cleaned_data["Word_count"] = cleaned_data["text"].apply(lambda x: len(s
    return(cleaned_data)
cleaned_data = restructure_data(restaurants_reviews_preprocessed_lemmatize)
```

In [28]:
```
## This code is for spliting the data for train, dev and test set
X_train, X_test, Y_train, Y_test = train_test_split(cleaned_data.drop(colum
                                                    test_size = 0.2, random_

X_train, X_dev, Y_train, Y_dev = train_test_split(X_train, Y_train,
                                                    test_size = 0.2, random_
```

In [29]:
```
## Function for creating tf-idf vectors from text

def tfidf(train,dev,test):
    tfidf_vectorizer = TfidfVectorizer(ngram_range=(2,2),max_features = 300
    tfidf_vectorizer.fit(train['text'].values)

    X_train_reviews_tfidf = tfidf_vectorizer.transform(train['text'].values
    X_dev_reviews_tfidf = tfidf_vectorizer.transform(dev['text'].values)
    X_test_reviews_tfidf = tfidf_vectorizer.transform(test['text'].values)

    print("After vectorizations")
    print(X_train_reviews_tfidf.shape)
    print(X_dev_reviews_tfidf.shape)
    print(X_test_reviews_tfidf.shape)
    print("="*100)
    return(X_train_reviews_tfidf,X_dev_reviews_tfidf,X_test_reviews_tfidf )
```

In [30]:
```
def numerical_feature_standardization(train,dev,test):
    normaliser = Normalizer()
    normaliser.fit(train['text_length'].values.reshape(-1,1))


    X_train_text_len_stand = normaliser.transform(train['text_length'].valu
    X_dev_text_len_stand = normaliser.transform(dev['text_length'].values.r
    X_test_text_len_stand = normaliser.transform(test['text_length'].values
    print("After Normalization")
    print(X_train_text_len_stand.shape)
    print(X_dev_text_len_stand.shape)
    print(X_test_text_len_stand.shape)
    print("="*100)

    return(X_train_text_len_stand,X_dev_text_len_stand, X_test_text_len_sta
```

In [31]:
```
## Calling the above tfidf_vec function to create 3000 dimensional bigram f
X_train_reviews_tfidf,X_dev_text_len_stand_tfidf,X_test_reviews_tfidf = tfi
```

```
After vectorizations
(6400, 3000)
(1600, 3000)
(2000, 3000)
============================================================================
============================
```

In [32]:
```
## calling the above function to normalise the numerical feature text lengt
X_train_text_len_stand,X_dev_text_len_stand, X_test_text_len_stand = numeri
```

```
After Normalization
(6400, 1)
(1600, 1)
(2000, 1)
================================================================================
============================
```

In [33]:
```
def merge_text_vectors_and_numerical_features(train1,train2,dev1,dev2,test1
    train_datam = hstack((train1,train2)).tocsr()
    dev_datam = hstack((dev1,dev2)).tocsr()
    test_datam = hstack((test1,test2 )).tocsr()


    print(tx +"final data matrix developed")
    print(train_datam.shape)
    print(dev_datam.shape)
    print(test_datam.shape)
    print("="*100)

    return(train_datam,dev_datam,test_datam)
```

In [34]:
```
train_data_tfidf, dev_data_tfidf, test_data_tfidf = merge_text_vectors_and_
```

```
TFIDF final data matrix developed
(6400, 3001)
(1600, 3001)
(2000, 3001)
================================================================================
============================
```

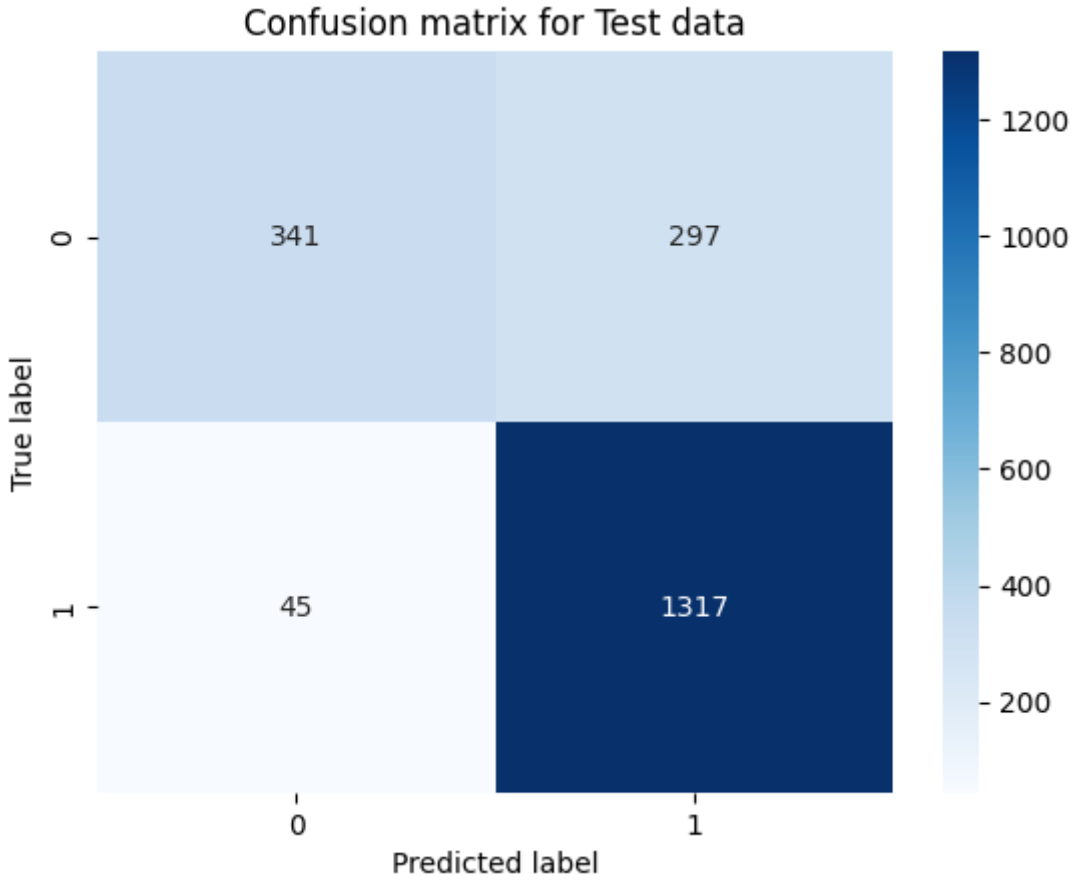
In [35]:
```
def evaluate_models(y_test,y_pred):
    confusion_matrix_given = confusion_matrix(y_test,y_test_pred)
    sns.heatmap(confusion_matrix_given, annot = True, fmt = 'd',cmap="Blues
    plt.title('Confusion matrix for Test data')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


    print("Precision Score of the model:", precision_score(y_test,y_pred)*1
    print("Recall Score of the model:", recall_score(y_test,y_pred)*100)
    print("Accuracy score of the model:",accuracy_score(y_test,y_pred)*100)
    print("F1 score of the model:",f1_score(y_test,y_pred)*100)
```

In [36]:
```
nb_model =  MultinomialNB()
nb_model = nb_model.fit(train_data_tfidf, Y_train)
```

In [37]:
```
y_test_pred = nb_model.predict(test_data_tfidf)
evaluate_models(Y_test,y_test_pred )
```

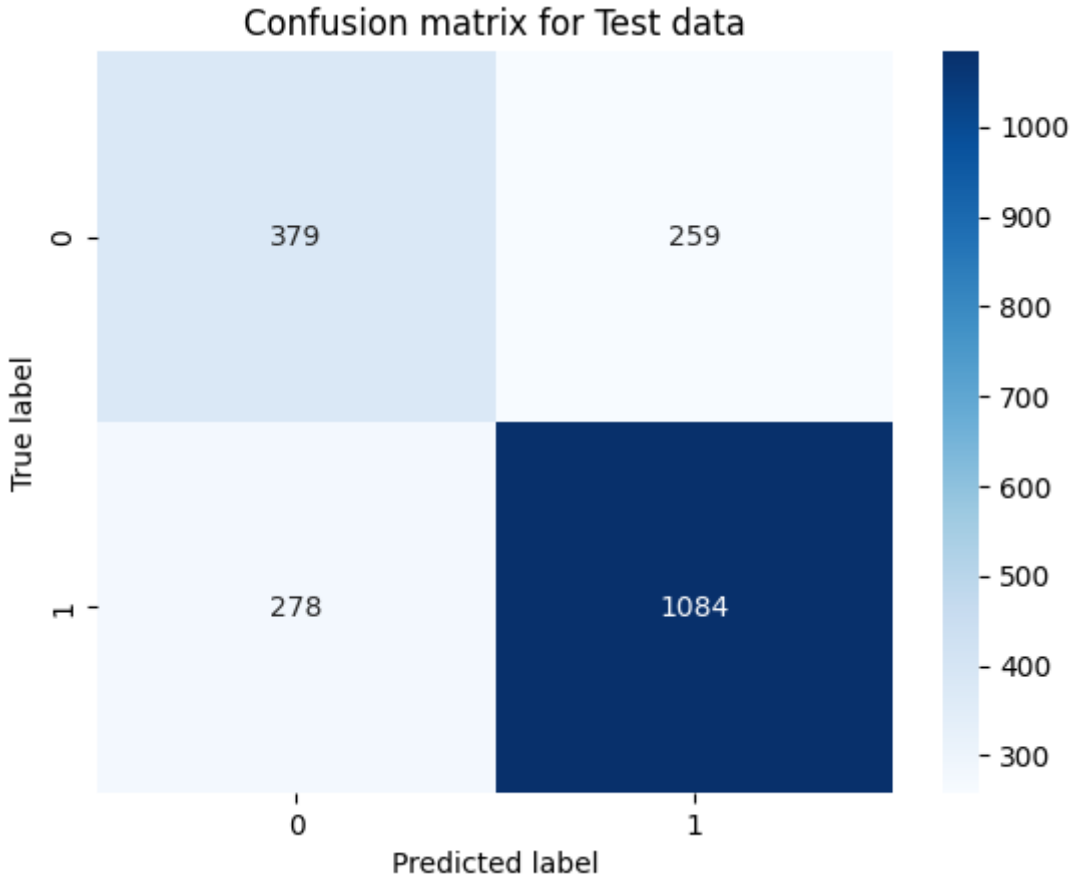
```
Precision Score of the model: 81.59851301115242
Recall Score of the model: 96.69603524229075
Accuracy score of the model: 82.89999999999999
F1 score of the model: 88.50806451612902
```



In [38]:
```
dc_model = DecisionTreeClassifier()
dc_model = dc_model.fit(train_data_tfidf,Y_train)
```

In [39]:
```
y_test_pred = dc_model.predict(test_data_tfidf)
evaluate_models(Y_test,y_test_pred )
```

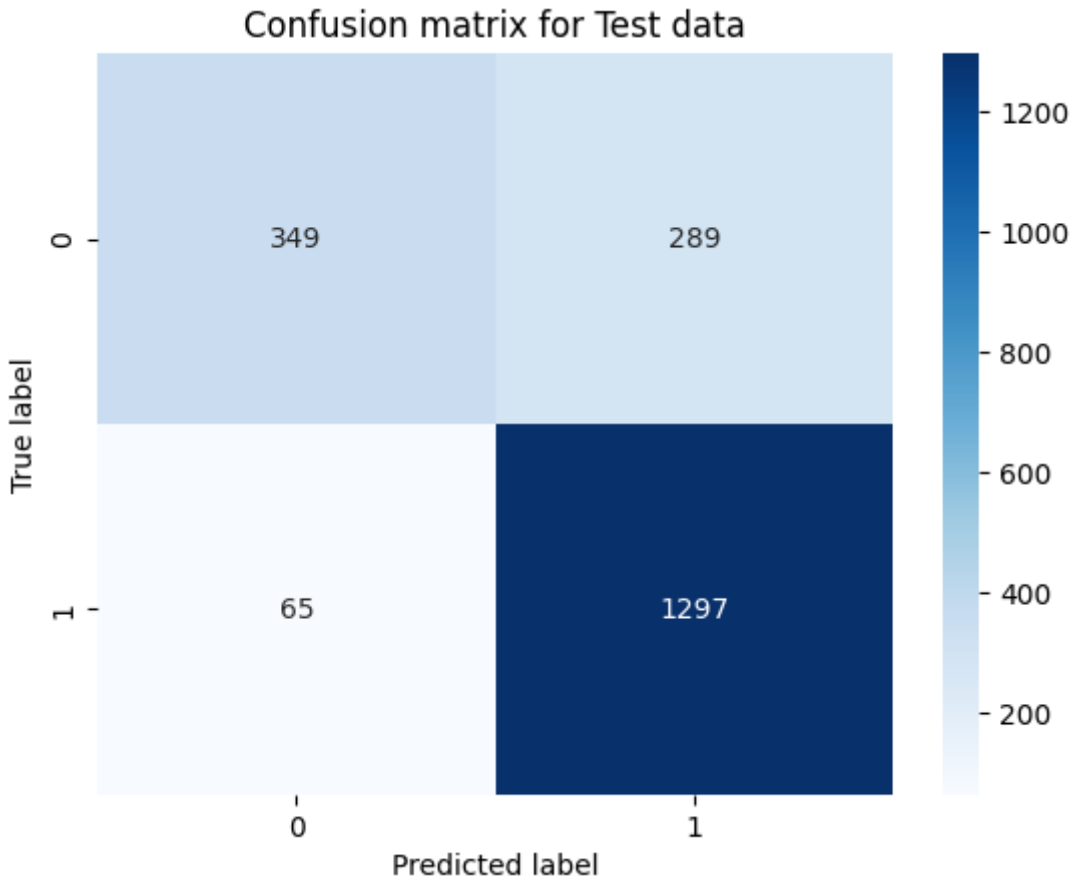
```
Precision Score of the model: 80.71481757259866
Recall Score of the model: 79.58883994126285
Accuracy score of the model: 73.15
F1 score of the model: 80.1478743068392
```



In [40]:
```
svc_model = svm.SVC()
svc_model = svc_model.fit(train_data_tfidf,Y_train)
```

In [41]:
```
y_test_pred = svc_model.predict(test_data_tfidf)
evaluate_models(Y_test,y_test_pred )
```

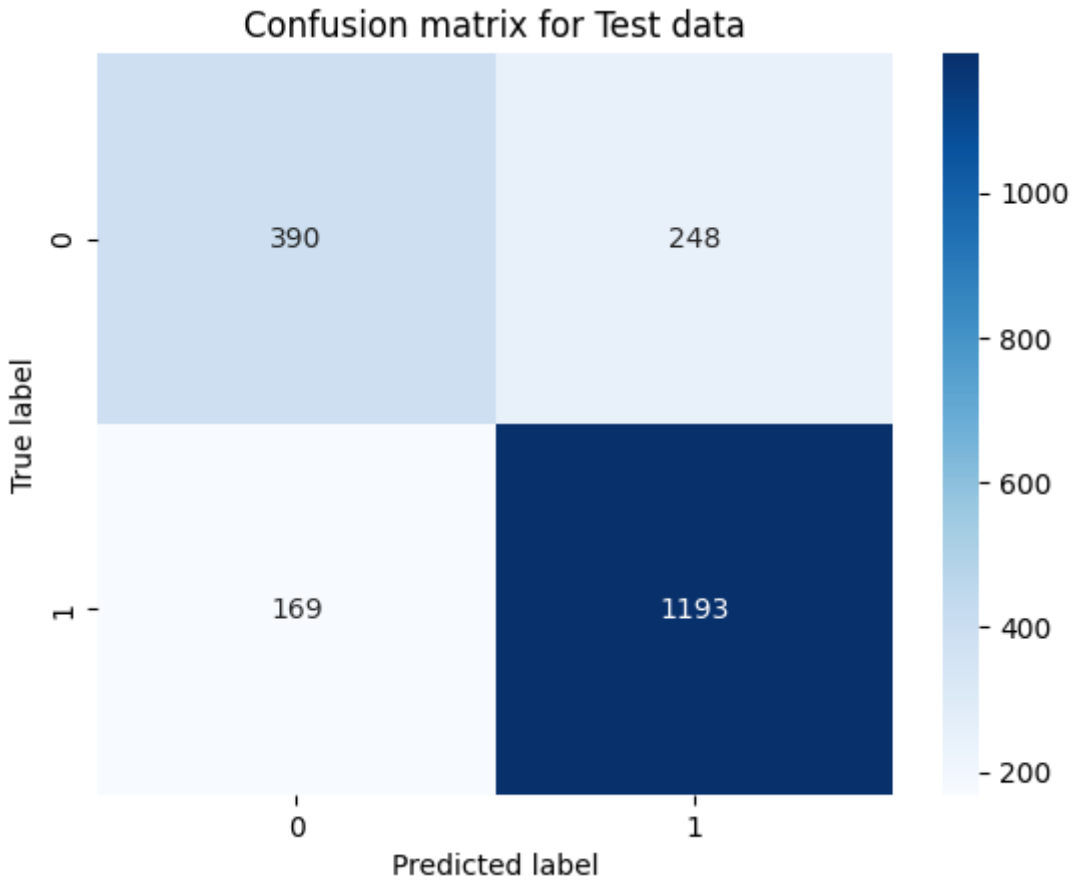
```
Precision Score of the model: 81.7780580075662
Recall Score of the model: 95.22760646108664
Accuracy score of the model: 82.3
F1 score of the model: 87.99185888738127
```



In [42]:
```
rf_model =  RandomForestClassifier()
rf_model = rf_model.fit(train_data_tfidf, Y_train)
```

In [43]:
```
y_test_pred = rf_model.predict(test_data_tfidf)
evaluate_models(Y_test,y_test_pred )
```

```
Precision Score of the model: 82.78972935461485
Recall Score of the model: 87.59177679882526
Accuracy score of the model: 79.14999999999999
F1 score of the model: 85.12308241170174
```



In [44]:
```
def oversampling_data(train_s,y_trains):
    random_oversampler = RandomOverSampler(random_state=0)
    train_data1, y_train1 = random_oversampler.fit_resample(train_s, y_trai
    return(train_data1, y_train1)
```

In [45]:
```
train_data1_tfidf,y_train1_tfidf = oversampling_data(train_data_tfidf,Y_tra
```

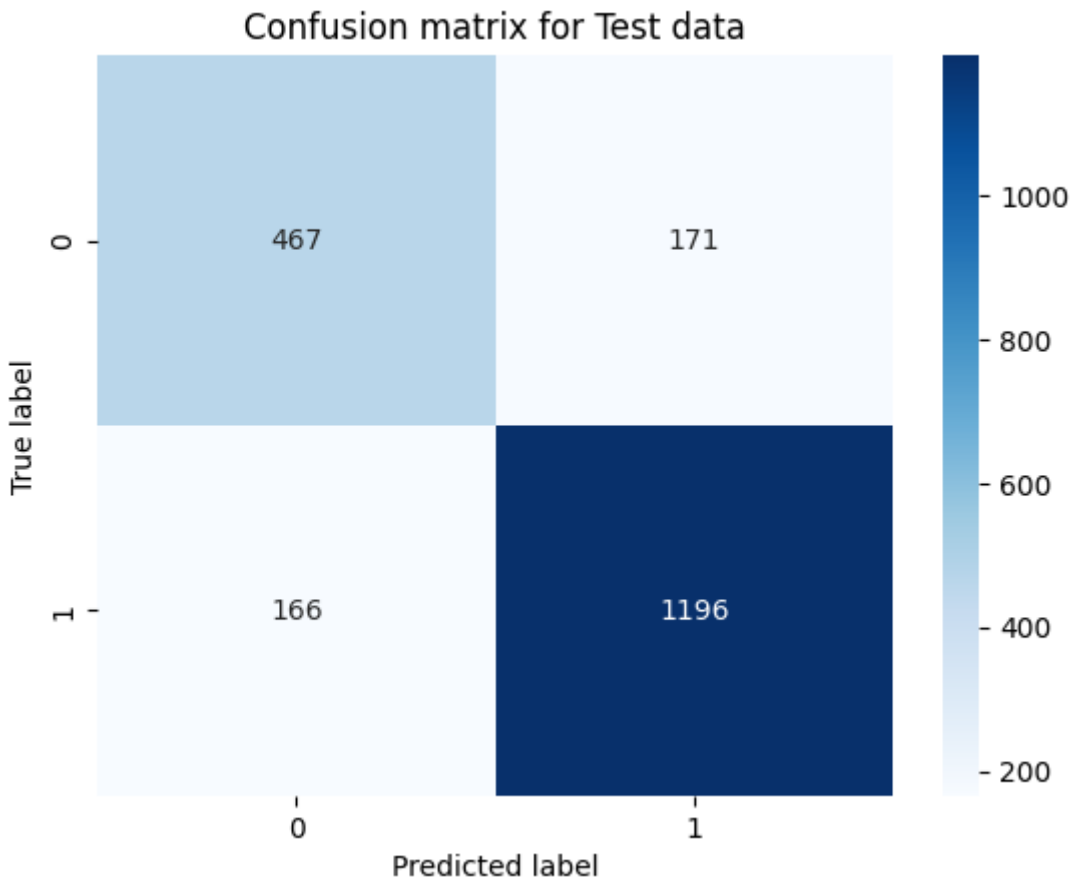
In [46]:
```
def hyper_tuning_grid_search_cross_validation(t_d,y_t,alpha,parameters):
    clf = GridSearchCV(alpha, param_grid= parameters, cv=5, scoring='f1',re
    hyper = clf.fit(t_d,y_t)
    print("Best parameters for the algorithm", hyper.best_estimator_)
    print("Best cross validation score :", hyper.best_score_)
    return(hyper.best_estimator_)
```

In [47]:
```
nb_model =  hyper_tuning_grid_search_cross_validation(train_data1_tfidf,y_t
nb_model = nb_model.fit(train_data1_tfidf, y_train1_tfidf)
```

```
Best parameters for the algorithm MultinomialNB(alpha=0.001)
Best cross validation score : 0.8561095600130338
```

In [48]:
```
y_test_pred = nb_model.predict(test_data_tfidf)
evaluate_models(Y_test,y_test_pred)
```

```
Precision Score of the model: 87.4908558888076
Recall Score of the model: 87.81204111600587
Accuracy score of the model: 83.15
F1 score of the model: 87.651154268963
```

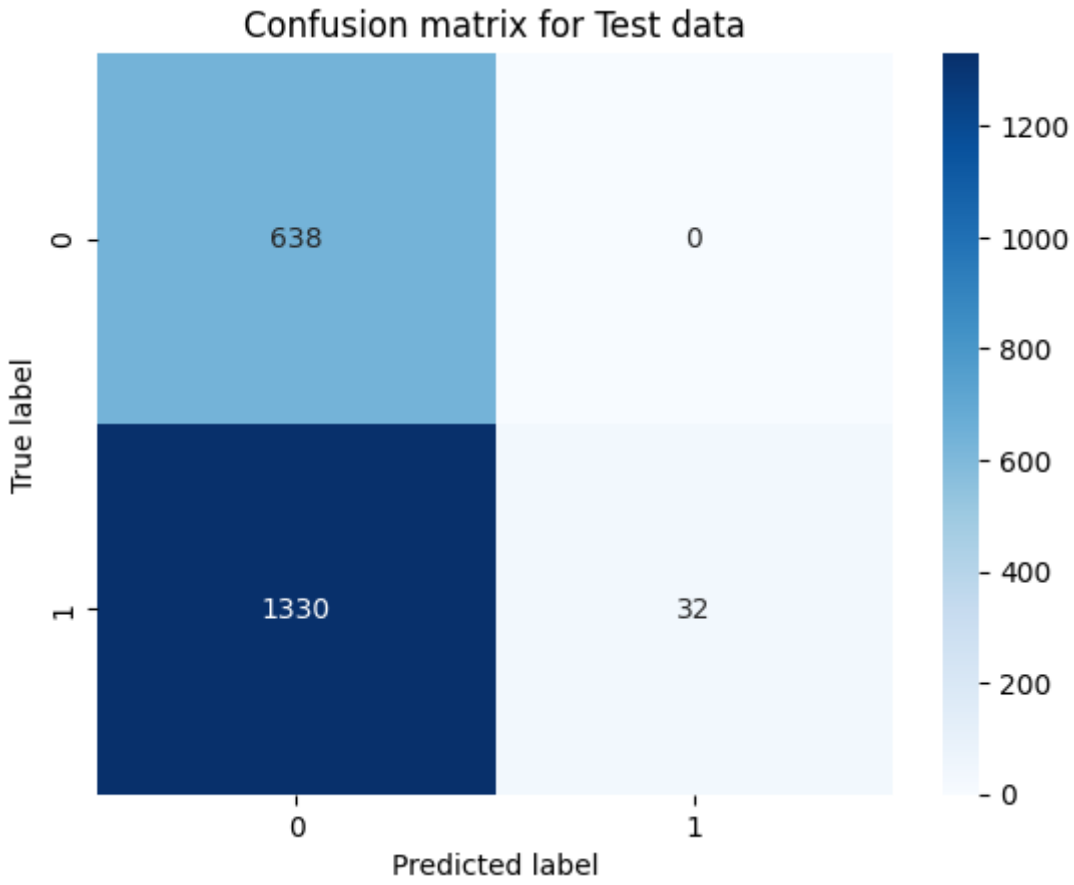


In [49]:
```
del =  hyper_tuning_grid_search_cross_validation(train_data1_tfidf,y_train1_
del = dt_model.fit(train_data1_tfidf, y_train1_tfidf)
```

```
Best parameters for the algorithm DecisionTreeClassifier(criterion='entro
py', max_depth=8, max_features=5,
                       min_samples_leaf=5, min_samples_split=3)
Best cross validation score : 0.6732066611828127
```

In [50]:
```
y_test_pred = dt_model.predict(test_data_tfidf)
evaluate_models(Y_test,y_test_pred)
```

```
Precision Score of the model: 100.0
Recall Score of the model: 2.3494860499265786
Accuracy score of the model: 33.5
F1 score of the model: 4.591104734576758
```

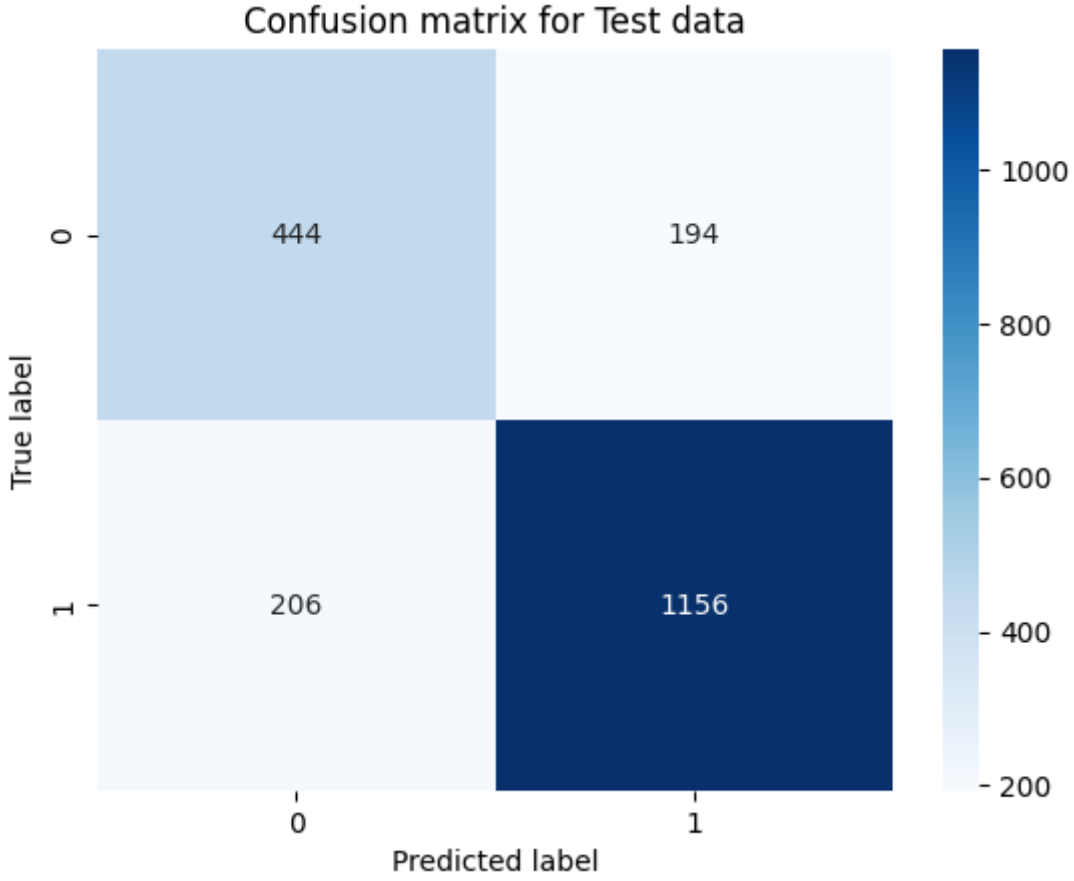


In [51]:
```
tion(train_data1_tfidf,y_train1_tfidf,alpha =  SGDClassifier(loss = 'hinge'
ain1_tfidf)
```

```
Best parameters for the algorithm SGDClassifier(max_iter=20)
Best cross validation score : 0.8650257799374182
```

In [52]:
```
## prediction on the test dataset and then evalauting the model performance
y_test_pred = svc_model.predict(test_data_tfidf)
evaluate_models(Y_test,y_test_pred)
```

```
Precision Score of the model: 85.62962962962963
Recall Score of the model: 84.87518355359765
Accuracy score of the model: 80.0
F1 score of the model: 85.25073746312685
```



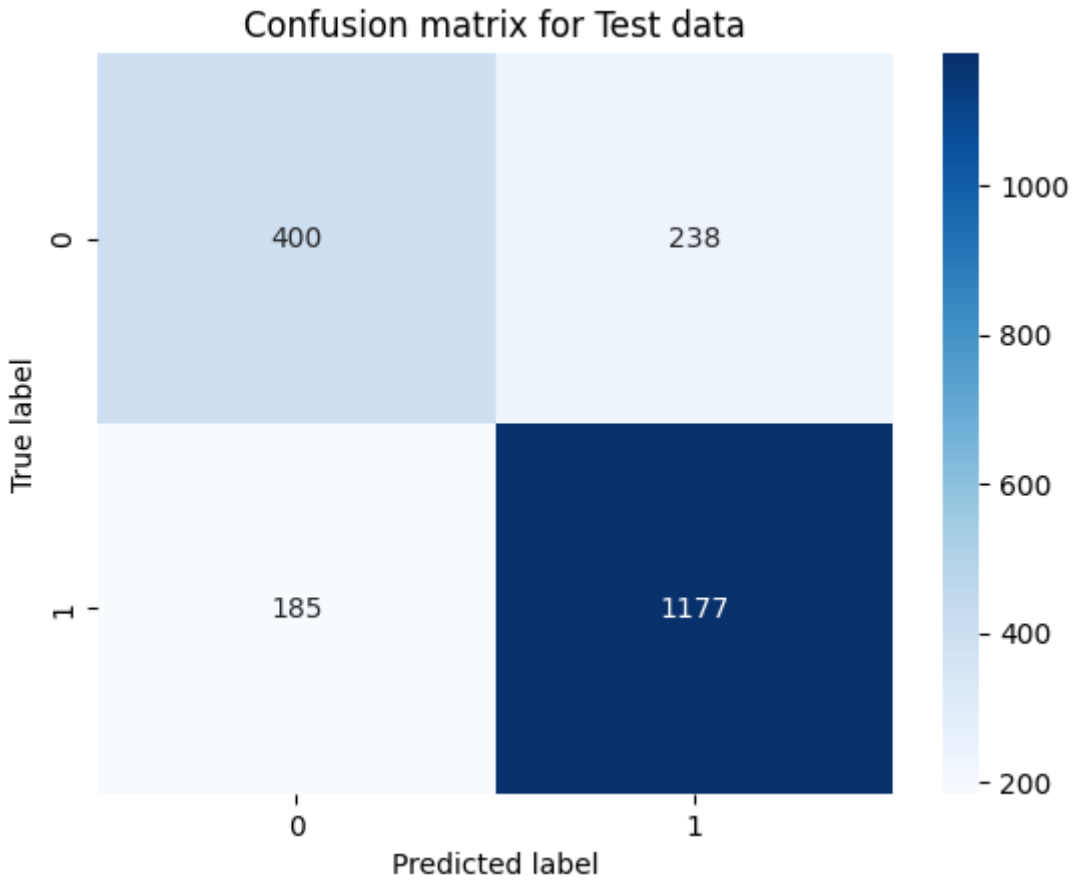
In [53]:
```
## Calling the above function for tuning Random Forest classifier algorithm
rf_model = hyper_tuning_grid_search_cross_validation(train_data1_tfidf,y_tr
rf_model = rf_model.fit(train_data1_tfidf, y_train1_tfidf)
```

```
Best parameters for the algorithm RandomForestClassifier(max_depth=110, n
_estimators=600, n_jobs=-1)
Best cross validation score : 0.8252979764896864
```

In [57]:
```
## prediction on the test dataset and then evalauting the model performance

y_test_pred = rf_model.predict(test_data_tfidf)
evaluate_models(Y_test,y_test_pred)
```

```
Precision Score of the model: 83.18021201413428
Recall Score of the model: 86.41703377386197
Accuracy score of the model: 78.85
F1 score of the model: 84.76773496579042
```



In [ ]:
```

```