```
import pandas as pd
import numpy as np
from typing import Union
from sklearn.model_selection import train_test_split
from nltk.stem import WordNetLemmatizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,recall_score,f1_score,precision_score
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
import math
import string
import spacy
import seaborn as sns
import matplotlib.pyplot as plt


df = pd.read_csv("/content/FinalBalancedDataset.csv")
len(df)
```

```
56745
```

```
def drop(DataFrame : object, columns : Union[str,list]):
    try:
        DataFrame.drop(columns,axis=1,inplace=True)
        print(f'Succ                    s}" columns')
    except Exception
        print(e)
```



```
#Removing punctuations and digits from the string
def remove_punc_dig(text : str):
    '''
    text : str
    This function will remove all the punctuations and digits from the "text"
    '''
    to_remove = string.punctuation + string.digits
    cur_text = ""
    for i in range(len(text)):
        if text[i] in to_remove:
            cur_text += " "
        else:
            cur_text += text[i].lower()
    cur_text = " ".join(cur_text.split())
    return cur_text


#Removing punctuations and digits from the string
def remove_punc_dig(text : str):
    '''
    text : str
    This function will remove all the punctuations and digits from the "text"
    '''
    to_remove = string.punctuation + string.digits
    cur_text = ""
    for i in range(len(text)):
        if text[i] in to_remove:
            cur_text += " "
        else:
            cur_text += text[i].lower()
    cur_text = " ".join(cur_text.split())
    return cur_text
```

```
df['cur_tweet'] = df['tweet'].apply(lambda x:remove_punc_dig(x))


# we don't need tweet column now so dropping the column
drop(df,'tweet')
```

```
Succefully Dropped "tweet" columns
```

```
# removing stop words like I,my,myself,etc
from spacy.lang.en.stop_words import STOP_WORDS


# we will use spacy lemmatizer API to perform lemmatization on cur_tweet and removing stop words
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])


def remove_stop_words(text: str):
    '''
    text : str
    This function will remove stop words like I,my,myself etc
    '''
    filtered_sentence = []
    for word in text.split(' '):
        lexeme = nlp.vocab[word]
        if lexeme.is_stop == False:
            filtered_sentence.append(word)
    return " ".join(filtered_sentence)

#applying remove_stop_words function on cur_tweets of dataframe df
df['filtered_cur_tweet'] = df['cur_tweet'].apply(lambda x : remove_stop_words(x))


# we don't need the cur_tweet now so dropping the cur_tweet column
drop(df,'cur_tweet')
```

```
Succefully Dropped "cur_tweet" columns
```

```
def lemmatizer(text : str):
    '''
    text : str
    Applying lemmatization for all words of "text"
    '''
    return " ".join([token.lemma_ for token in nlp(text)])


#applying lemmatizer function on cur_tweets of dataframe df
df['lemma_cur_tweet'] = df['filtered_cur_tweet'].apply(lambda x : lemmatizer(x))


# droping filtered_cur_tweet column since we don't need it any more
drop(df,'filtered_cur_tweet')
```

```
Succefully Dropped "filtered_cur_tweet" columns
```

```
#TfidfVectorizer, CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import gensim
tfidf_vectorizer = TfidfVectorizer(max_df=0.90, min_df=2, stop_words='english')
tfidf = tfidf_vectorizer.fit_transform(df['lemma_cur_tweet'])
tfidf.shape
```

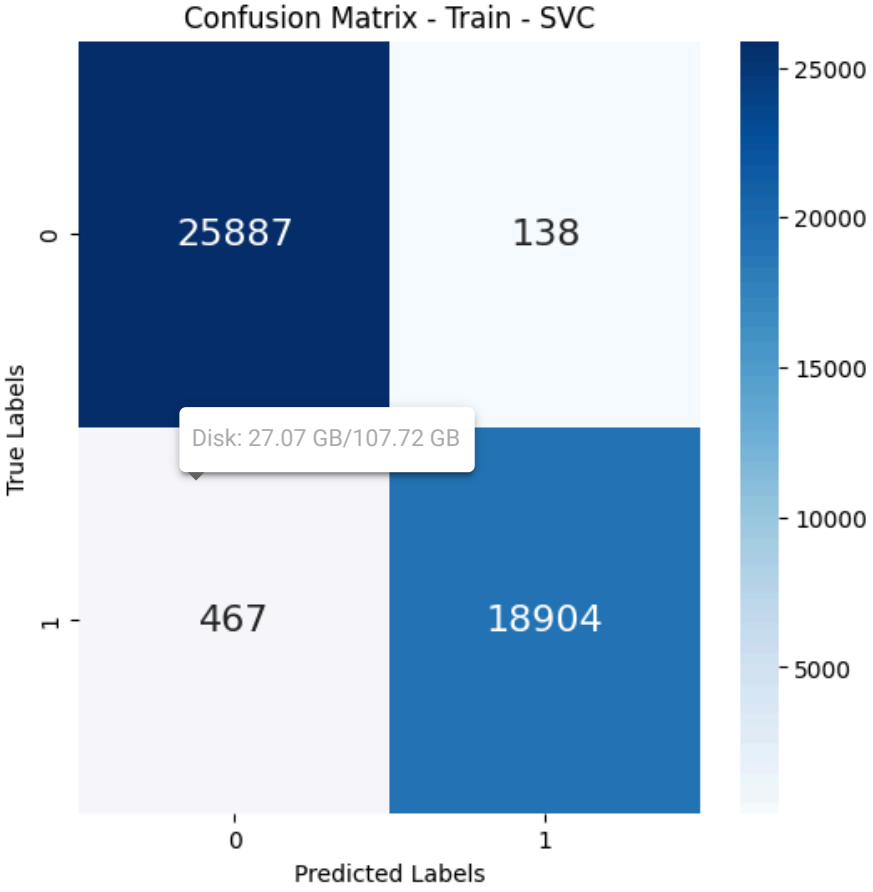
```
(56745, 20056)
```

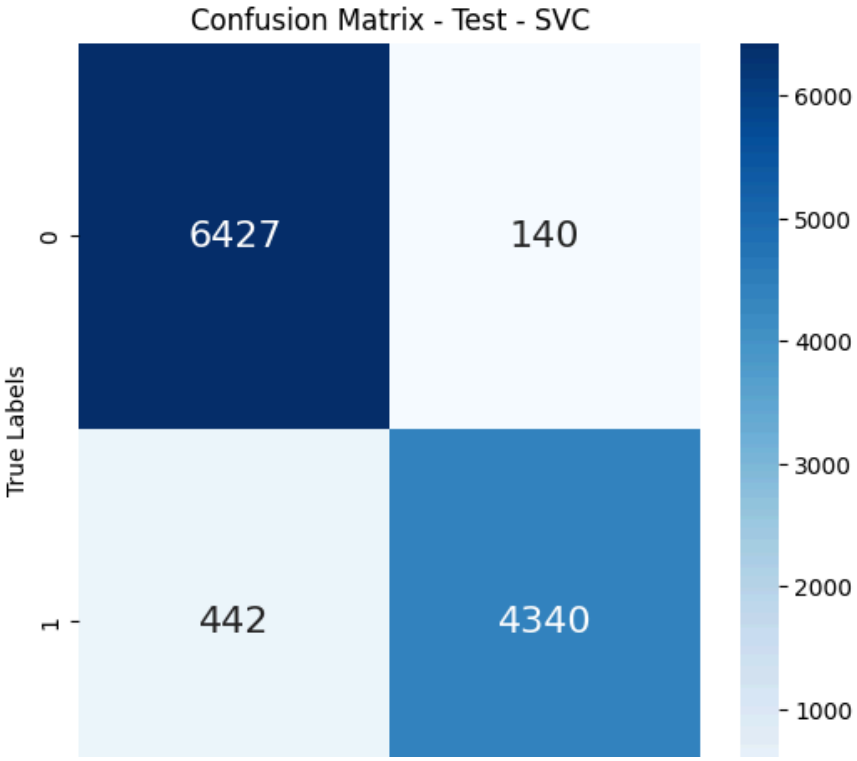
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score


x_train, x_test,y_train, y_test = train_test_split(tfidf,df['Toxicity'] ,
                                   test_size=0.20)


def plot_confusion_matrices(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 16})
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(title)
    plt.show()


def plot_roc_auc_curve(y_true, y_scores, title):
    fpr, tpr, _ = roc_curve(y_true, y_scores)
    auc = roc_auc_score(y_true, y_scores)
    plt.figure(figsi
    plt.plot(fpr, tp                                    UC = {auc:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate (FPR)')
    plt.ylabel('True Positive Rate (TPR)')
    plt.title(title)
    plt.legend(loc='lower right')
    plt.show()


models = [SVC(),KNeighborsClassifier(),LogisticRegression(),DecisionTreeClassifier(),RandomForestClassifier(
train_accuracies = []
train_precisions = []
train_recalls = []
train_f1s = []
test_accuracies = []
test_precisions = []
test_recalls = []
test_f1s = []
model_names = []
for model in models:
    model.fit(x_train,y_train)
    train_pred = model.predict(x_train)
    #train_probs = model.predict_proba(x_train)[:, 1]
    test_pred = model.predict(x_test)
    #test_probs = model.predict_proba(x_test)[:, 1]
    print(type(model).__name__)
    model_names.append(type(model).__name__)
    print("*******Train************")
    print("Accuracy: ",accuracy_score(y_train,train_pred))
    print("Precision: ",precision_score(y_train,train_pred))
    print("Recall: ",recall_score(y_train,train_pred))
    print("F1 Score: ",f1_score(y_train,train_pred))
    train_accuracies.append(accuracy_score(y_train,train_pred))
    train_precisions.append(precision_score(y_train,train_pred))
    train_recalls.append(recall_score(y_train,train_pred))
    train_f1s.append(f1_score(y_train,train_pred))
```

```
        print("*******Test************")
        print("Accuracy: ",accuracy_score(y_test,test_pred))
        print("Precision: ",precision_score(y_test,test_pred))
        print("Recall: ",recall_score(y_test,test_pred))
        print("F1 Score: ",f1_score(y_test,test_pred))
        test_accuracies.append(accuracy_score(y_test,test_pred))
        test_precisions.append(precision_score(y_test,test_pred))
        test_recalls.append(recall_score(y_test,test_pred))
        test_f1s.append(f1_score(y_test,test_pred))
        print("\n \n")
        # Calculate and display the confusion matrix for training data
        plot_confusion_matrices(y_train, train_pred, f"Confusion Matrix - Train - {type(model).__name__}")

        # Calculate and display the confusion matrix for testing data
        plot_confusion_matrices(y_test, test_pred, f"Confusion Matrix - Test - {type(model).__name__}")

        # Calculate and display the ROC-AUC curve for training data
        '''plot_roc_auc_curve(y_train, train_probs, f"ROC-AUC Curve - Train - {type(model).__name__}")

        # Calculate and display the ROC-AUC curve for testing data
        plot_roc_auc_curve(y_test, test_probs, f"ROC-AUC Curve - Test - {type(model).__name__}")'''

    train_df = pd.DataFrame()
    train_df['Accuracy'] =  train_accuracies
    train_df['Precision'] =  train_precisions
    train_df['Recall'] =  train_recalls
    train_df['F1 Score'] =  train_f1s
    train_df['Mechanism'] = "Train"
    train_df['Model'] = model_names

    test_df = pd.DataFrame()
    test_df['Accuracy'] =  test_accuracies
    test_df['Precision'] =  test_precisions
    test_df['Recall'] =  test_recalls
    test_df['F1 Score'] =  test_f1s
    test_df['Mechanism'] = "Test"
    test_df['Model'] = model_names

    result_df = pd.concat([train_df, test_df])
    for metric in ['Accuracy','Precision','Recall','F1 Score']:
        sns.barplot(data =result_df,x ='Model',y = metric,hue="Mechanism")
        plt.xticks(rotation=60)
        plt.show()
```

SVC

```
*******Train************
Accuracy:  0.986672834610979
Precision:  0.9927528620943178
Recall:  0.9758917970161581
F1 Score:  0.984250123656054
*******Test************
Accuracy:  0.9487179487179487
Precision:  0.96875
Recall:  0.9075700543705563
F1 Score:  0.9371625998704384
```






```
KNeighborsClassifier
*******Train************
Accuracy:  0.6878359326812935
Precision:  0.9498269896193772
Recall:  0.28341334985287286
F1 Score:  0.436563158522524
*******Test************
Accuracy:  0.6539783240814169
Precision:  0.9203539823008849
Recall:  0.19573400250941028
F1 Score:  0.32281427832384896
```