1.1 Reading Data

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from tqdm import tqdm
import os
# from plotly import plotly
# import plotly.offline as offline
# import plotly.graph_objs as go
# offline.init_notebook_mode()
from collections import Counter
```

importing file into googel colab

```
#  resources data
import gdown

url = 'https://drive.google.com/uc?id=1OcMV5zjAJI7OvNxxN4Ant52BDF3jrZOZ'
output = 'resources.csv'
# https://drive.google.com/file/d/1OcMV5zjAJI7OvNxxN4Ant52BDF3jrZOZ/view?usp=sharing
gdown.download(url, output, quiet=False)

```

```
# test data
import gdown

url = 'https://drive.google.com/uc?id=1JGtsNLea4Q2HZQIgBp3pRrOfRN80qIg0'
# https://drive.google.com/file/d/1JGtsNLea4Q2HZQIgBp3pRrOfRN80qIg0/view?usp=sharing
output = 'train_data.csv'
gdown.download(url, output, quiet=False)
```

```
ls
```

```
project_data =pd.read_csv("train_data.csv")
resource_data = pd.read_csv("resources.csv")
```

```
print("The shape of the Train data ",project_data.shape)
print("-"*50)
print("The number of attributes in Train data","-"*5,project_data.columns.values)
```

```
cols = ["Date" if x=="project_submitted_datetime" else x for x in list(project_data.columns)]
project_data["Date"] = pd.to_datetime(project_data["project_submitted_datetime"])
project_data.drop("project_submitted_datetime",axis=1,inplace = True )
project_data.sort_values(by=["Date"],inplace=True)

project_data = project_data[cols]
project_data.head(2)
```

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

```
# we cannot remove rows where teacher prefix is not available therefore we are replacing 'nan' value with
# 'null'(string)
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-split
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')
```

1.2 preprocessing of project_subject_categories

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
```

```
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&",\
#             "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip().lower())

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
print("The Worlds in sorted_cat_dict",sorted_cat_dict)
```

1.3 preprocessing of project_subject_subcategories

```
catogories1 = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list1 = []
for i in catogories1:
    temp1 = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&",\
#             "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp1+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp1 = temp1.replace('&','_') # we are replacing the & value into
    cat_list1.append(temp1.strip().lower())

project_data['clean_sub_categories'] = cat_list1
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

from collections import Counter
my_counter1 = Counter()
for word in project_data['clean_sub_categories'].values:
    my_counter1.update(word.split())
```

```python

cat_dict1 = dict(my_counter1)
sorted_cat_dict1 = dict(sorted(cat_dict1.items(), key=lambda kv: kv[1]))
print("The Worlds in sorted_cat_dict1",sorted_cat_dict1)
```

1.4 Text preprocessing of project essay

```python
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

```python
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

```python
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
preprocessed_essays[20000]
```

1.5 Preprocessing of project_title

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_title'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
preprocessed_titles[1000]
```

```
# here we are removing the unwanted coloumns from the data that has been processed .
project_data["clean_titles"] = preprocessed_titles
project_data.drop(["project_essay_1"],axis=1,inplace=True)
project_data.drop(["project_essay_2"],axis=1,inplace=True)
project_data.drop(["project_essay_3"],axis=1,inplace=True)
project_data.drop(["project_essay_4"],axis=1,inplace=True)
project_data.drop(["project_title"],axis=1,inplace=True)
```

1.6 Merging the tow DataFrame (Resources.csv and Train.csv)

```
project_data.columns
```

```
resource_data.head(2)
```

```
price_data= resource_data.groupby("id").agg({"price" : "sum" , "quantity" : "sum"}).reset_index()
price_data.head(2)
```

```
project_data = pd.merge(project_data,price_data,on="id",how = "left")
project_data.head(2)
# project_data.drop(["Unnamed: 0"],axis=1,inplace=True)
# project_data.drop(["id"],axis=1,inplace=True)
# project_data.drop(["teacher_id"],axis=1,inplace=True)
```

1.7 Preprocessing of school_state

```
school_state = list(project_data['school_state'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

```

```
school_state_list = []
for i in school_state:
    temp2 = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=>"Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp2 +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp2 = temp2.replace('&','_')
    school_state_list.append(temp2.strip().lower())

# droping the school_state column
project_data['School_state'] = school_state_list
project_data.drop(['school_state'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter3 = Counter()
for word in project_data['School_state'].values:
    my_counter3.update(word.split())

school_state_dict = dict(my_counter3)
sorted_school_state_dict = dict(sorted(school_state_dict.items(), key=lambda kv: kv[1]))
print("The Values in sorted_school_state_dict : ", sorted_school_state_dict)


```

2.1 Splitting data

```
project_data = project_data.head(50000)
Y = project_data["project_is_approved"].values
X = project_data.drop(["project_is_approved"],axis = 1)
print(X.columns)
```

```
#Splitting the data into train and test data_set
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, stratify=Y)
X_train, X_cv, Y_train, Y_cv = train_test_split(X_train, Y_train, test_size=0.33, stratify=Y_train)
```

2.2 Resampling data using the Randomover sampler for Imblearn.over sampling

```
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
import warnings
warnings.filterwarnings("ignore")

ros = RandomOverSampler(sampling_strategy='minority',random_state=42)
x_train, y_train = ros.fit_resample(X_train, Y_train)
print('Resampled dataset shape %s' % Counter(y_train))
print("Capitial" ,"X","represents the original train_data and lower case" ,"x", "represnts the ramdonly over-sampled data")
```

```
# here we have to convert x into a dataframe
x_train = pd.DataFrame(x_train,columns = X.columns)
x_train.head(1)
```

```
print(x_train.shape, y_train.shape)
print(X_cv.shape, Y_cv.shape)
print(X_test.shape, Y_test.shape)

print("="*100)

```

2.3 Make Data Model Ready: Vectorizing data

Bag of Words is used for Vectorizing of Text data

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=2000)
vectorizer.fit(x_train['essay'].values) # fit has to happen only on train data

# we use the transform Text_data to vector , BOW CountVectorizer
x_train_essay_bow = vectorizer.transform(x_train['essay'].values) # this the vectorization of the oversampled data we do it as we
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)  # here we use the X_CV for vectorization( only x_train is oversampled
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

print("="*100)
print("After vectorizations of the text data the shape of the data is ")
print(x_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, Y_cv.shape)
print(X_test_essay_bow.shape, Y_test.shape)



```

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=2000)
vectorizer.fit(x_train['clean_titles'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_titles_bow = vectorizer.transform(x_train['clean_titles'].values)
X_cv_titles_bow = vectorizer.transform(X_cv['clean_titles'].values)
X_test_titles_bow = vectorizer.transform(X_test['clean_titles'].values)

print("After vectorizations of the clean_titles data the shape of the data is")
print(x_train_titles_bow.shape, y_train.shape)
print(X_cv_titles_bow.shape, Y_cv.shape)
print(X_test_titles_bow.shape, Y_test.shape)
print("*"*100)
```

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=2000)
vectorizer.fit(x_train['project_resource_summary']) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_summary_bow = vectorizer.transform(x_train['project_resource_summary'])
X_cv_summary_bow = vectorizer.transform(X_cv['project_resource_summary'])
X_test_summary_bow = vectorizer.transform(X_test['project_resource_summary'])

print("After vectorizations of the project_resource_summary data the shape of the data is")
print(x_train_summary_bow.shape, y_train.shape)
print(X_cv_summary_bow.shape, Y_cv.shape)
print(X_test_summary_bow.shape, Y_test.shape)
print("*"*100)
```

2.4 One-hot-encoding of the Catogorical Features

```
vectorizer = CountVectorizer()
vectorizer.fit(x_train['clean_sub_categories']) # fit has to happen only on train data


# we use the fitted CountVectorizer to convert the text to vector
x_train_clean_subcat_ohe = vectorizer.transform(x_train['clean_sub_categories'])
X_cv_clean_subcat_ohe = vectorizer.transform(X_cv['clean_sub_categories'])
X_test_clean_subcat_ohe = vectorizer.transform(X_test['clean_sub_categories'])

print("After vectorizations of the clean_sub_categories , One-hot-encoding shape of the data is")
print(x_train_clean_subcat_ohe.shape, y_train.shape)
print(X_cv_clean_subcat_ohe.shape, Y_cv.shape)
print(X_test_clean_subcat_ohe.shape, Y_test.shape)
# print(vectorizer.get_feature_names())
print("*"*100)
```

```
vectorizer = CountVectorizer()
vectorizer.fit(x_train['clean_categories']) # fit has to happen only on train data


# we use the fitted CountVectorizer to convert the text to vector
x_train_clean_categories_ohe = vectorizer.transform(x_train['clean_categories'])
X_cv_clean_categories_ohe = vectorizer.transform(X_cv['clean_categories'])
X_test_clean_categories_ohe = vectorizer.transform(X_test['clean_categories'])

print("After vectorizations of the clean_categories, One-hot-encoding shape of the data is")
print(x_train_clean_categories_ohe.shape, y_train.shape)
print(X_cv_clean_categories_ohe.shape, Y_cv.shape)
print(X_test_clean_categories_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("*"*100)
```

```
vectorizer = CountVectorizer()
vectorizer.fit(x_train['teacher_prefix'].values.astype('U')) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_teacher_ohe = vectorizer.transform(x_train['teacher_prefix'].values.astype('U'))
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values.astype('U'))

print("After vectorizations of the teacher_prefix , One-hot-encoding shape of the data is")
print(x_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, Y_cv.shape)
print(X_test_teacher_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("*"*100)

```

```python
vectorizer = CountVectorizer()
vectorizer.fit(x_train['School_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_state_ohe = vectorizer.transform(x_train['School_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['School_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['School_state'].values)

print("After vectorizations of the School_state , One-hot-encoding shape of the data is")
print(x_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, Y_cv.shape)
print(X_test_state_ohe.shape, Y_test.shape)
# print(vectorizer.get_feature_names())
print("="*100)
```

```python
# #This step is to intialize a vectorizer with the vocabulary created form the project_grade_category values
from collections import Counter
my_counter5 = Counter()
for word in X_train['project_grade_category'].values:
  if "Grades" in word:
    word = word.replace("Grades","")
  my_counter5.update(word.split())
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
project_grade_category_dict = dict(my_counter5)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda kv: kv[1]))
print(sorted_project_grade_category_dict)
```

```python
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(x_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_grade_ohe = vectorizer.transform(x_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

```

```
print("After vectorizations of the project_grade_category , One-hot-encoding shape of the data is")
print(x_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, Y_cv.shape)
print(X_test_grade_ohe.shape,Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

2.5 Normalizing the numerical features: Price

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
standard_vec.fit(x_train['price'].values.reshape(-1,1))

x_train_price_std = standard_vec.transform(x_train['price'].values.reshape(-1,1))
X_cv_price_std = standard_vec.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_std = standard_vec.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations of the price data ,  shape of the data after standazing")
print(x_train_price_std.shape, y_train.shape)
print(X_cv_price_std.shape, Y_cv.shape)
print(X_test_price_std.shape, Y_test.shape)
print("="*100)
```

```
from sklearn.preprocessing import StandardScaler
```

```python
standard_vector = StandardScaler(with_mean = False)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
standard_vector.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

x_train_projects_std = standard_vector.transform(x_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_cv_projects_std = standard_vector.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_projects_std = standard_vector.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations of the teacher_number_of_previously_posted_projects , shape of the data after standazing")
print(x_train_projects_std.shape, y_train.shape)
print(X_cv_projects_std.shape,Y_cv.shape)
print(X_test_projects_std.shape, Y_test.shape)
print("="*100)
```

```python
from sklearn.preprocessing import StandardScaler
standard_vector1 = StandardScaler(with_mean = False)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
standard_vector1.fit(x_train['quantity'].values.reshape(-1,1))

x_train_qty_std = standard_vector1.transform(x_train['quantity'].values.reshape(-1,1))
X_cv_qty_std = standard_vector1.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_qty_std = standard_vector1.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(x_train_qty_std.shape, y_train.shape)
print(X_cv_qty_std.shape, Y_cv.shape)
print(X_test_qty_std.shape, Y_test.shape)
print("="*100)
```

3 Appling KNN on different kind of featurization ("BOW" and "TFIDF")

3.1 **Set 1**: Categorical Vectorised data , Numerical Vectorised data, Project_title(BOW) + Preprocessed_essay (BOW)

```
from scipy.sparse import hstack
X1_tr = hstack((x_train_clean_categories_ohe,x_train_clean_subcat_ohe,x_train_teacher_ohe,x_train_state_ohe,\
                x_train_grade_ohe,x_train_titles_bow,x_train_essay_bow,x_train_price_std,x_train_projects_std,x_train_qty_std)).t
X1_cv = hstack((X_cv_clean_categories_ohe,X_cv_clean_subcat_ohe,X_cv_teacher_ohe,X_cv_state_ohe,X_cv_grade_ohe,\
                X_cv_price_std,X_cv_projects_std,X_cv_qty_std,X_cv_essay_bow,X_cv_titles_bow)).tocsr()
X1_te =hstack((X_test_clean_categories_ohe,X_test_clean_subcat_ohe,X_test_teacher_ohe,X_test_state_ohe,\
               X_test_grade_ohe,X_test_essay_bow,X_test_titles_bow,X_test_price_std,X_test_projects_std,X_test_qty_std)).tocsr()


print("The final Data Matrix for Set:1" , " All the shapes of the data represent the merged features as mentioned in the tittle")
print("shape of X_train is : ",            X1_tr.shape)
print("shape of X_Cross validation is :" , X1_cv.shape)
print("shape of X_test is ",               X1_te.shape)
```

```
def batch_predict(clf, data):
  y_data_pred = []
  tr_loop = data.shape[0] - data.shape[0]%1000
  # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
  # in this for loop we will iterate unti the last 1000 multiplier
  for i in range(0, tr_loop, 1000):
      y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
  # we will be predicting for the last data points
  if data.shape[0]%1000 !=0:
      y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

  return y_data_pred
```

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

```

```
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X1_tr, y_train)

    y_train_pred = batch_predict(neigh, X1_tr)
    y_cv_pred = batch_predict(neigh, X1_cv)

    # roc_auc_score(y_tr, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(Y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=1)
plt.show()
```

1. The error plot above representes the Train AUC and CV AUC Curve, wherein we choose K(hyperparameter value) such that,it will be the maximum AU data and distance between the train-data line("Blue") and CV-line("orange") results be to minimum.
2. The best value of k is found to be 102
3. Here I have used the "AUCROC" curve for choosing the best hypermeter as the data is imbalanced(though i have balanced it) , the "AUCROC" curve co both the class labels equally .
4. The other hyperparameter tuning techiniques can also be choosen as "CV" and "K-fold CV"

```
best_k = 101
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X1_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X1_tr)
y_test_pred = batch_predict(neigh, X1_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Fasle Poitive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=1)
plt.show()
```

Conclusion for SET : 1

1. The above Represents the TRP and FPR rates on the either axis and this curve is know as the AUCROC curve , it is a metrix to evaluate the performar model.
2. As the grap represents here the Train Auc = 0.67 , so from this we can conclude that the model is predecting the values with 67 % probabilty.

```python
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

```

```python
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
Confusion_metrix_Train_data = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
Confusion_metrix_Test_data = pd.DataFrame(confusion_matrix(Y_test, predict_with_best_t(y_test_pred, best_t)))
import seaborn as sns
sns.set(font_scale=1.4)#for label size
print("The Confusion metrix of train data")
sns.heatmap(Confusion_metrix_Train_data,annot=True, annot_kws={"size": 20},fmt ="g")
```

```
print("The Confusion metrix of test data ")
sns.heatmap(Confusion_metrix_Test_data,annot=True, annot_kws={"size": 20},fmt ="g")
```

1. The Ouput Stated above represents the maximun value of TRP (i.e tpr*(1-fpr) "0.3944289258349809" corrosponding to whcih the maximum thersho
   .
2. The Second output represnts the Confusion metrix Based on the thershold of 0.48 , which states that the values below the thershold of 0.48 are clas 0 and the values above 0.48 are classified as 1.

3.2 **Set 2**: Categorical, Numerical features + Project_title(TFIDF)+ Preprocessed_essay (TFIDF)

```python
# Applying TF-IDF on Prohect title :
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(x_train["clean_titles"].values)

x_tain_project_titles_tfidf = vectorizer.transform(x_train["clean_titles"].values)
X_cv_project_titles_tfidf = vectorizer.transform(X_cv["clean_titles"].values)
X_test_project_titles_tfidf = vectorizer.transform(X_test["clean_titles"].values)


print("After TFIDF vectorizations of the clean_titles , shape of the data after standazing")
print(x_tain_project_titles_tfidf.shape, y_train.shape)
print(X_cv_project_titles_tfidf.shape,Y_cv.shape)
print(X_test_project_titles_tfidf.shape, Y_test.shape)
print("*"*100)
```

```python
# Applying the Tfidf Vectrization on Preprocessed_essay
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(x_train["essay"])

x_tain_essay_tfidf = vectorizer.transform(x_train["essay"].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv["essay"].values)
X_test_essay_tfidf = vectorizer.transform(X_test["essay"].values)


print("After TFIDF vectorizations of the essay , shape of the data after standazing")
print(x_tain_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape,Y_cv.shape)
print(X_test_essay_tfidf.shape, Y_test.shape)
print("*"*100)
```

Creating new data-set for set -2

Catogorial data , Numerical data , Essay(TFIDF) , Project_tittle(TFIDF)

```
# the data points are merged using the Hstack which we import from sklearn
from scipy.sparse import hstack
X2_tr = hstack((x_train_clean_categories_ohe,x_train_clean_subcat_ohe,x_train_teacher_ohe,x_train_state_ohe,\
                x_train_grade_ohe,x_train_price_std,x_train_projects_std,x_train_qty_std,x_tain_project_titles_tfidf,x_tain_essay_
X2_cv = hstack((X_cv_clean_categories_ohe,X_cv_clean_subcat_ohe,X_cv_teacher_ohe,X_cv_state_ohe,X_cv_grade_ohe,\
                X_cv_price_std,X_cv_projects_std,X_cv_qty_std,X_cv_project_titles_tfidf,X_cv_essay_tfidf)).tocsr()
X2_te =hstack((X_test_clean_categories_ohe,X_test_clean_subcat_ohe,X_test_teacher_ohe,X_test_state_ohe,\
               X_test_grade_ohe,X_test_price_std,X_test_projects_std,X_test_qty_std,X_test_project_titles_tfidf,X_test_essay_tfid


print("The final Data Matrix for Set:2" , " All the shapes of the data represent the merged features as mentioned in the tittle")
print("shape of X_train is : ",            X2_tr.shape)
print("shape of X_Cross validation is :" , X2_cv.shape)
print("shape of X_test is ",               X2_te.shape)
```

```
def batch_predict(clf, data):
  y1_data_pred = []
  tr_loop = data.shape[0] - data.shape[0]%1000
  # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
  # in this for loop we will iterate unti the last 1000 multiplier
  for i in range(0, tr_loop, 1000):
      y1_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
  # we will be predicting for the last data points
  if data.shape[0]%1000 !=0:
      y1_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

  return y1_data_pred
```

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc1 = []
cv_auc1 = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X2_tr, y_train)

    y1_train_pred = batch_predict(neigh, X2_tr)
    y1_cv_pred = batch_predict(neigh, X2_cv)

    # roc_auc_score(y_tr, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc1.append(roc_auc_score(y_train,y1_train_pred))
    cv_auc1.append(roc_auc_score(Y_cv, y1_cv_pred))

plt.plot(K, train_auc1, label='Train AUC')
plt.plot(K, cv_auc1, label='CV AUC')

plt.scatter(K, train_auc1, label='Train AUC points')
plt.scatter(K, cv_auc1, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=1)
plt.show()
```

1. The error plot above representes the Train AUC and CV AUC Curve, wherein we choose K(hyperparameter value) such that,it will be the maximum AU data and distance between the train-data line("Blue") and CV-line("orange") results be to minimum.
2. The best value of k is found to be 101
3. Here I have used the "AUCROC" curve for choosing the best hypermeter as the data is imbalanced(though i have balanced it) , the "AUCROC" curve co both the class labels equally .
4. The other hyperparameter tuning techiniques can also be choosen as "CV" and "K-fold CV"

```
best_k1 = 101
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k1, n_jobs=-1)
neigh.fit(X2_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y2_train_pred = batch_predict(neigh, X2_tr)
y2_test_pred = batch_predict(neigh, X2_te)

train1_fpr, train1_tpr, tr1_thresholds = roc_curve(y_train, y2_train_pred)
test1_fpr, test1_tpr, te1_thresholds = roc_curve(Y_test, y2_test_pred)

plt.plot(train1_fpr, train1_tpr, label="Tain AUC ="+str(auc(train1_fpr, train1_tpr)))
plt.plot(test1_fpr, test1_tpr, label="Test AUC ="+str(auc(test1_fpr, test1_tpr)))
plt.legend()
plt.xlabel("Fasle Postive Rate")
plt.ylabel("True Pstive Rate")
plt.title("ROC_AUC Curve")
plt.grid(color='black', linestyle='-', linewidth=1)
plt.show()
```

Conclusion for SET : 2

1. The above Represents the TRP and FPR rates on the either axis and this curve is know as the AUCROC curve , it is a metrix to evaluate the performar model.
2. As teh grap represents here the Train Auc = 0.67 , so from this we can conclude that the model is predecting the values with 67 % probabilty.

```python
#this is the custom function for predecting the best thershold and sorting the values according the threshould
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t


```

```python
def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

```python
from sklearn.metrics import confusion_matrix
```

```
best_t = find_best_threshold(tr1_thresholds, train1_fpr, train1_tpr)
Confusion_metrix_Train_data = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y2_train_pred, best_t)))
Confusion_metrix_Test_data = pd.DataFrame(confusion_matrix(Y_test, predict_with_best_t(y2_test_pred, best_t)))
import seaborn as sns
sns.set(font_scale=1.4)#for label size
print("The Confusion metrix of train data")
sns.heatmap(Confusion_metrix_Train_data,annot=True, annot_kws={"size": 20},fmt ="g")

```

```
print("The Confusion metrix of test data ")
sns.heatmap(Confusion_metrix_Test_data,annot=True, annot_kws={"size": 20},fmt ="g")
```

1. The Ouput Stated above represents the maximun value of TRP (i.e tpr*(1-fpr) "0.3944" corrosponding to whcih the maximum thershold is 0.48 .
2. The Second output represnts the Confusion metrix Based on the thershold of 0.48 , which states that the values below the thershold of 0.48 are clas 0 and the values above 0.48 are classified as 1.

3.3 Set 3: Categorical, Numerical features + Project_title(AVG W2V)+ Preprocessed_essay (AVG W2V)

```
# I am using the predefined word to vector which is pre-trained , hence we use the pickel file to access the file
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
# please review the above link for more information
# code to import file in google colab form drive


import gdown

url = 'https://drive.google.com/uc?id=1MqUasf7jYoPbG35MJ28VQcOjjNp-ZDDp'
output = 'glove_vectors'
gdown.download(url, output, quiet=False)
```

```
#checking for files are present in the directory we are working or not
ls
```

```
import pickle
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
# The code below represents the Avg-word-to-vector of Project tittle .
# here we are calculating the Avg-word to vec for "x_train" .
x_train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_train_avg_w2v_vectors.append(vector)

print(len(x_train_avg_w2v_vectors))
print(len(x_train_avg_w2v_vectors[0]))
# print(x_train_avg_w2v_vectors[0])
```

```
# here we are calculating the Avg-word to vec for "X_CV"
X_cv_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_cv_avg_w2v_vectors.append(vector)

print(len(X_cv_avg_w2v_vectors))
print(len(X_cv_avg_w2v_vectors[0]))
# print(X_cv_avg_w2v_vectors[0])
# here we are calculating the Avg-word to vec for "X_test"
X_test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_avg_w2v_vectors.append(vector)

print(len(X_test_avg_w2v_vectors))
print(len(X_test_avg_w2v_vectors[0]))
# print(X_cv_avg_w2v_vectors[0])
```

```
# The code below represents the Avg-word-to-vector of Essay.
# here we are calculating the Avg-word to vec for "x_train" .
x_train_essay_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_train_essay_avg_w2v_vectors.append(vector)

print(len(x_train_essay_avg_w2v_vectors))
print(len(x_train_essay_avg_w2v_vectors[0]))
# print(x_train_essay_avg_w2v_vectors[0])
```

```
# here we are calculating the Avg-word to vec for "X_CV"
X_cv_essay_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_cv_essay_avg_w2v_vectors.append(vector)

print(len(X_cv_essay_avg_w2v_vectors))
print(len(X_cv_essay_avg_w2v_vectors[0]))
# print(X_cv_essay_avg_w2v_vectors[0])
```

```
# here we are calculating the Avg-word to vec for "X_test"
X_test_essay_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_essay_avg_w2v_vectors.append(vector)

print(len(X_test_essay_avg_w2v_vectors))
print(len(X_test_essay_avg_w2v_vectors[0]))
# print(X_test_essay_avg_w2v_vectors[0])
```

```
from scipy.sparse import hstack
X3_tr = hstack((x_train_clean_categories_ohe,x_train_clean_subcat_ohe,x_train_teacher_ohe,x_train_state_ohe,\
                x_train_grade_ohe,x_train_price_std,x_train_projects_std,x_train_qty_std,x_train_avg_w2v_vectors,x_train_essay_avg
X3_cv = hstack((X_cv_clean_categories_ohe,X_cv_clean_subcat_ohe,X_cv_teacher_ohe,X_cv_state_ohe,X_cv_grade_ohe,\
                X_cv_price_std,X_cv_projects_std,X_cv_qty_std,X_cv_avg_w2v_vectors,X_cv_essay_avg_w2v_vectors)).tocsr()
X3_te =hstack((X_test_clean_categories_ohe,X_test_clean_subcat_ohe,X_test_teacher_ohe,X_test_state_ohe,\
               X_test_grade_ohe,X_test_price_std,X_test_projects_std,X_test_qty_std,X_test_avg_w2v_vectors,X_test_essay_avg_w2v_ve


```

```
print("The final Data Matrix for Set:3" , " All the shapes of the data represent the merged features as mentioned in the tittle")
print("shape of X_train is : ",            X3_tr.shape)
print("shape of X_Cross validation is :" , X3_cv.shape)
print("shape of X_test is ",               X3_te.shape)
```

```
def batch_predict(clf, data):
  y2_data_pred = []
  tr_loop = data.shape[0] - data.shape[0]%1000
  # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
  # in this for loop we will iterate unti the last 1000 multiplier
  for i in range(0, tr_loop, 1000):
      y2_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
  # we will be predicting for the last data points
  if data.shape[0]%1000 !=0:
      y2_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

  return y2_data_pred
```

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc2 = []
cv_auc2 = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X3_tr, y_train)

    y2_train_pred = batch_predict(neigh, X3_tr)
    y2_cv_pred = batch_predict(neigh, X3_cv)

    # roc_auc_score(y_tr, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc2.append(roc_auc_score(y_train,y2_train_pred))
    cv_auc2.append(roc_auc_score(Y_cv, y2_cv_pred))

plt.plot(K, train_auc2, label='Train AUC')
plt.plot(K, cv_auc2, label='CV AUC')

plt.scatter(K, train_auc2, label='Train AUC points')
plt.scatter(K, cv_auc2, label='CV AUC points')
```

```python

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=1)
plt.show()
```

1. The error plot above representes the Train AUC and CV AUC Curve, wherein we choose K(hyperparameter value) such that,it will be the maximum AU data and distance between the train-data line("Blue") and CV-line("orange") results be to minimum.
2. The best value of k is found to be 101
3. Here I have used the "AUCROC" curve for choosing the best hypermeter as the data is imbalanced(though i have balanced it) , the "AUCROC" curve co both the class labels equally .
4. The other hyperparameter tuning techiniques can also be choosen as "CV" and "K-fold CV"

```python
best_k1 = 101
```

```python
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

```

```

neigh = KNeighborsClassifier(n_neighbors=best_k1, n_jobs=-1)
neigh.fit(X3_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y3_train_pred = batch_predict(neigh, X3_tr)
y3_test_pred = batch_predict(neigh, X3_te)

train2_fpr, train2_tpr, tr2_thresholds = roc_curve(y_train, y3_train_pred)
test2_fpr, test2_tpr, te2_thresholds = roc_curve(Y_test, y3_test_pred)

plt.plot(train2_fpr, train2_tpr, label="Tain AUC ="+str(auc(train2_fpr, train2_tpr)))
plt.plot(test2_fpr, test2_tpr, label="Test AUC ="+str(auc(test2_fpr, test2_tpr)))
plt.legend()
plt.xlabel("Fasle Postive Rate")
plt.ylabel("True Pstive RAte")
plt.title("ROC_AUC Curve")
plt.grid(color='black', linestyle='-', linewidth=1)
plt.show()
```

Conclusion for SET : 3

1. The above Represents the TRP and FPR rates on the either axis and this curve is know as the AUCROC curve , it is a metrix to evaluate the performar model.
2. As teh grap represents here the Train Auc = 0.67 , so from this we can conclude that the model is predecting the values with 67 % probabilty.

```
#this is the custom function for predecting the best thershold and sorting the values according the threshould
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t
```

```
def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr2_thresholds, train2_fpr, train2_tpr)
Confusion_metrix_Train_data = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y3_train_pred, best_t)))
Confusion_metrix_Test_data = pd.DataFrame(confusion_matrix(Y_test, predict_with_best_t(y3_test_pred, best_t)))
import seaborn as sns
sns.set(font_scale=1.4)#for label size
print("The Confusion metrix of train data")
sns.heatmap(Confusion_metrix_Train_data,annot=True, annot_kws={"size": 20},fmt ="g")

```

```
print("The Confusion metrix of test data ")
sns.heatmap(Confusion_metrix_Test_data,annot=True, annot_kws={"size": 20},fmt ="g")
```

1. The Ouput Stated above represents the maximun value of TRP (i.e tpr*(1-fpr) "0.401" corrosponding to whcih the maximum thershold is 0.48 .
2. The Second output represnts the Confusion metrix Based on the thershold of 0.48 , which states that the values below the thershold of 0.48 are clas 0 and the values above 0.48 are classified as 1.

3.4 Set 4: Categorical, Numerical features + Project_title(TFIDF W2V)+ Preprocessed_essay (TFIDF W2V)

```
# TFIDF W2V of x_train "essays"
x_train_tfidf_model = TfidfVectorizer()
x_train_tfidf_model.fit(x_train["essay"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(x_train_tfidf_model.get_feature_names(), list(x_train_tfidf_model.idf_)))
tfidf_words = set(x_train_tfidf_model.get_feature_names())
```

```python
x_train1_tfidf_model = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_train1_tfidf_model.append(vector)

print(len(x_train1_tfidf_model))
print(len(x_train1_tfidf_model[0]))
```

```python
# TFIDF W2V of X_CV "essays"
# here alos we will fit in the train data only as we dont want our data to be leacked
X_cv_tfidf_model = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_cv_tfidf_model.append(vector)

print(len(X_cv_tfidf_model))
print(len(X_cv_tfidf_model[0]))
```

```
X_test_tfidf_model = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_tfidf_model.append(vector)

print(len(X_test_tfidf_model))
print(len(X_test_tfidf_model[0]))
```

```
# TFIDF W2V of x_train "essays"
x_train_tfidf_model1 = TfidfVectorizer()
x_train_tfidf_model1.fit(x_train["clean_titles"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(x_train_tfidf_model1.get_feature_names(), list(x_train_tfidf_model.idf_)))
tfidf_words = set(x_train_tfidf_model1.get_feature_names())
```

```
x_train_tfidf_clean_titles_model = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_train_tfidf_clean_titles_model.append(vector)

print(len(x_train_tfidf_clean_titles_model))
print(len(x_train_tfidf_clean_titles_model[0]))
```

```
X_cv_tfidf_clean_titles_model = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_cv_tfidf_clean_titles_model.append(vector)

print(len(X_cv_tfidf_clean_titles_model))
print(len(X_cv_tfidf_clean_titles_model[0]))
```

```
X_test_tfidf_clean_titles_model = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_tfidf_clean_titles_model.append(vector)

print(len(X_test_tfidf_clean_titles_model))
print(len(X_test_tfidf_clean_titles_model[0]))
```

```
from scipy.sparse import hstack
X4_tr = hstack((x_train_clean_categories_ohe,x_train_clean_subcat_ohe,x_train_teacher_ohe,x_train_state_ohe,\
                x_train_grade_ohe,x_train_price_std,x_train_projects_std,x_train_qty_std,x_train1_tfidf_model, x_train_tfidf_clea
X4_cv = hstack((X_cv_clean_categories_ohe,X_cv_clean_subcat_ohe,X_cv_teacher_ohe,X_cv_state_ohe,X_cv_grade_ohe,\
                X_cv_price_std,X_cv_projects_std,X_cv_qty_std,X_cv_tfidf_model,X_cv_tfidf_clean_titles_model)).tocsr()
X4_te =hstack((X_test_clean_categories_ohe,X_test_clean_subcat_ohe,X_test_teacher_ohe,X_test_state_ohe,\
               X_test_grade_ohe,X_test_price_std,X_test_projects_std,X_test_qty_std,X_test_tfidf_model,X_test_tfidf_clean_titles_


print("The final Data Matrix for Set:4" , " All the shapes of the data represent the merged features as mentioned in the tittle")
print("shape of X_train is : ",            X4_tr.shape)
print("shape of X_Cross validation is :" , X4_cv.shape)
print("shape of X_test is ",               X4_te.shape)
```

```
def batch_predict(clf, data):
  y3_data_pred = []
  tr_loop = data.shape[0] - data.shape[0]%1000
  # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
  # in this for loop we will iterate unti the last 1000 multiplier
  for i in range(0, tr_loop, 1000):
      y3_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
  # we will be predicting for the last data points
  if data.shape[0]%1000 !=0:
      y3_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

  return y3_data_pred
```

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc3 = []
cv_auc3 = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X4_tr, y_train)

    y3_train_pred = batch_predict(neigh, X4_tr)
    y3_cv_pred = batch_predict(neigh, X4_cv)

    # roc_auc_score(y_tr, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc3.append(roc_auc_score(y_train,y3_train_pred))
    cv_auc3.append(roc_auc_score(Y_cv, y3_cv_pred))

plt.plot(K, train_auc3, label='Train AUC')
plt.plot(K, cv_auc3, label='CV AUC')

plt.scatter(K, train_auc3, label='Train AUC points')
plt.scatter(K, cv_auc3, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=1)
plt.show()
```

1. The error plot above representes the Train AUC and CV AUC Curve, wherein we choose K(hyperparameter value) such that,it will be the maximum AU data and distance between the train-data line("Blue") and CV-line("orange") results be to minimum.
2. The best value of k is found to be 102

3. Here I have used the "AUCROC" curve for choosing the best hypermeter as the data is imbalanced(though i have balanced it) , the "AUCROC" curve c
   both the class labels equally .
4. The other hyperparameter tuning techiniques can also be choosen as "CV" and "K-fold CV"

```
best_k2= 101
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k2, n_jobs=-1)
neigh.fit(X4_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y4_train_pred = batch_predict(neigh, X4_tr)
y4_test_pred = batch_predict(neigh, X4_te)

train3_fpr, train3_tpr, tr3_thresholds = roc_curve(y_train, y4_train_pred)
test3_fpr, test3_tpr, te3_thresholds = roc_curve(Y_test, y4_test_pred)

plt.plot(train3_fpr, train3_tpr, label="Tain AUC ="+str(auc(train3_fpr, train3_tpr)))
plt.plot(test3_fpr, test3_tpr, label="Test AUC ="+str(auc(test3_fpr, test3_tpr)))
plt.legend()
plt.xlabel("Fasle Postive Rate")
plt.ylabel("True Pstive RAte")
plt.title("ROC_AUC Curve")
plt.grid(color='black', linestyle='-', linewidth=1)
plt.show()
```

Conclusion for SET : 4

1. The above Represents the TRP and FPR rates on the either axis and this curve is know as the AUCROC curve , it is a metrix to evaluate the performar model.
2. As teh grap represents here the Train Auc = 0.67 , so from this we can conclude that the model is predecting the values with 67 % probabilty.

```python
#this is the custom function for predecting the best thershold and sorting the values according the threshould
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t
```

```python

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

```python
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr3_thresholds, train3_fpr, train3_tpr)
Confusion_metrix_Train_data = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y4_train_pred, best_t)))
Confusion_metrix_Test_data = pd.DataFrame(confusion_matrix(Y_test, predict_with_best_t(y4_test_pred, best_t)))
import seaborn as sns
sns.set(font_scale=1.4)#for label size
print("The Confusion metrix of train data")
sns.heatmap(Confusion_metrix_Train_data,annot=True, annot_kws={"size": 20},fmt ="g")
```

```python
print("The Confusion metrix of test data ")
sns.heatmap(Confusion_metrix_Test_data,annot=True, annot_kws={"size": 20},fmt ="g")
```

1. The Ouput Stated above represents the maximun value of TRP (i.e tpr*(1-fpr) "0.3939" corrosponding to whcih the maximum thershold is 0.45 .

2. The Second output represnts the Confusion metrix Based on the thershold of 0.45 , which states that the values below the thershold of 0.45 are clas 0 and the values above 0.48 are classified as 1.

3.5 Applying the SelectKBest for selecting the Top best 2000 features on the set2 of the Data matrix t text

This data metrix consits of Set 2: Categorical, Numerical features + Project_title(TFIDF)+ Preprocessed_essay (TFIDF)

```
# Preprocessed_essay (TFIDF) and selcting the best 2000 Features using the  SelectKBest, chi2
%%time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(x_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_essay_tfidf = vectorizer.transform(x_train['essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)


#Selecting top 2000 best features from the generated tfidf features
selector = SelectKBest(chi2, k = 2000 )
selector.fit(x_train_essay_tfidf,y_train)
x_train_essay_2000 = selector.transform(x_train_essay_tfidf)
X_cv_essay_2000 = selector.transform(X_cv_essay_tfidf)
X_test_essay_2000 = selector.transform(X_test_essay_tfidf)
print(x_train_essay_2000.shape)
print(X_cv_essay_2000.shape)
print(X_test_essay_2000.shape)
```

```
# Project_title(TFIDF) and selcting the best 2000 Features using the  SelectKBest, chi2
%%time
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(x_train['clean_titles'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_clean_titles_tfidf = vectorizer.transform(x_train['clean_titles'].values)
X_cv_clean_titles_tfidf = vectorizer.transform(X_cv['clean_titles'].values)
X_test_clean_titles_tfidf = vectorizer.transform(X_test['clean_titles'].values)


#Selecting top 2000 best features from the generated tfidf features
selector = SelectKBest(chi2, k = 1811 )
selector.fit(x_train_clean_titles_tfidf,y_train)
x_train_clean_titles_2000 = selector.transform(x_train_clean_titles_tfidf)
X_cv_clean_titles_2000 = selector.transform(X_cv_clean_titles_tfidf)
X_test_clean_titles_2000 = selector.transform(X_test_clean_titles_tfidf)
print(x_train_clean_titles_2000.shape)
print(X_cv_clean_titles_2000.shape)
print(X_test_clean_titles_2000.shape)
```

```
#now merging all the data matrix
from scipy.sparse import hstack
X5_tr = hstack((x_train_clean_categories_ohe,x_train_clean_subcat_ohe,x_train_teacher_ohe,x_train_state_ohe,\
                x_train_grade_ohe,x_train_price_std,x_train_projects_std,x_train_qty_std,x_train_essay_2000,x_train_clean_titles_
X5_cv = hstack((X_cv_clean_categories_ohe,X_cv_clean_subcat_ohe,X_cv_teacher_ohe,X_cv_state_ohe,\
                X_cv_grade_ohe,X_cv_price_std,X_cv_projects_std,X_cv_qty_std,X_cv_essay_2000,X_cv_clean_titles_2000)).tocsr()
X5_te =hstack((X_test_clean_categories_ohe,X_test_clean_subcat_ohe,X_test_teacher_ohe,X_test_state_ohe,\
               X_test_grade_ohe,X_test_price_std,X_test_projects_std,X_test_qty_std,X_test_essay_2000,X_test_clean_titles_2000)).


print("The final Data Matrix for Set:3" , " All the shapes of the data represent the merged features as mentioned in the tittle")
print("shape of X_train is : ",            X5_tr.shape)
print("shape of X_Cross validation is :" , X5_cv.shape)
print("shape of X_test is ",               X5_te.shape)
```

```
def batch_predict(clf, data):
  y4_data_pred = []
  tr_loop = data.shape[0] - data.shape[0]%1000
  # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
  # in this for loop we will iterate unti the last 1000 multiplier
  for i in range(0, tr_loop, 1000):
      y4_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
  # we will be predicting for the last data points
  if data.shape[0]%1000 !=0:
      y4_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

  return y4_data_pred
```

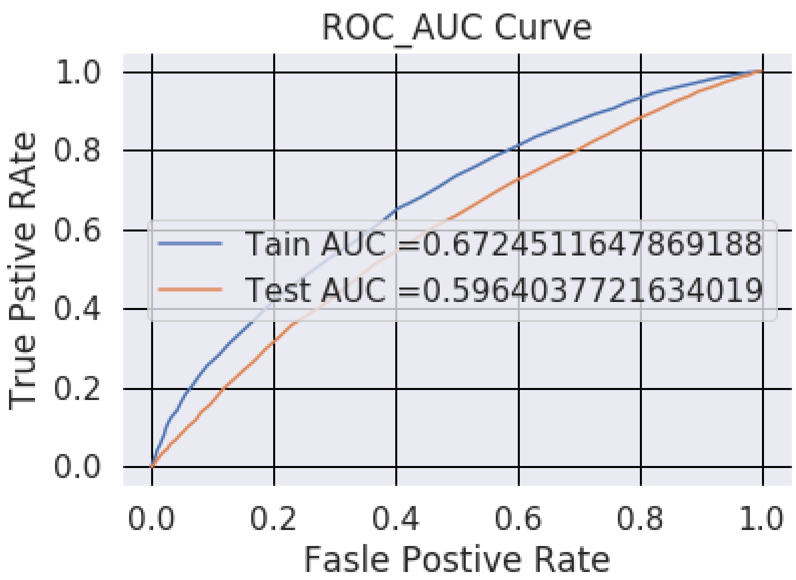
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc4 = []
cv_auc4 = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X5_tr, y_train)

    y4_train_pred = batch_predict(neigh, X5_tr)
    y4_cv_pred = batch_predict(neigh, X5_cv)

    # roc_auc_score(y_tr, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc4.append(roc_auc_score(y_train,y4_train_pred))
    cv_auc4.append(roc_auc_score(Y_cv, y4_cv_pred))

plt.plot(K, train_auc4, label='Train AUC')
plt.plot(K, cv_auc4, label='CV AUC')

plt.scatter(K, train_auc4, label='Train AUC points')
plt.scatter(K, cv_auc4, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=1)
plt.show()
```

1. The error plot above representes the Train AUC and CV AUC Curve, wherein we choose K(hyperparameter value) such that,it will be the maximum AU data and distance between the train-data line("Blue") and CV-line("orange") results be to minimum.
2. The best value of k is found to be 100
3. Here I have used the "AUCROC" curve for choosing the best hypermeter as the data is imbalanced(though i have balanced it) , the "AUCROC" curve co both the class labels equally .
4. The other hyperparameter tuning techiniques can also be choosen as "CV" and "K-fold CV"

```
best_k2= 101
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k2, n_jobs=-1)
neigh.fit(X5_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y5_train_pred = batch_predict(neigh, X5_tr)
y5_test_pred = batch_predict(neigh, X5_te)

train5_fpr, train5_tpr, tr5_thresholds = roc_curve(y_train, y5_train_pred)
test5_fpr, test5_tpr, te5_thresholds = roc_curve(Y_test, y5_test_pred)

plt.plot(train5_fpr, train5_tpr, label="Tain AUC ="+str(auc(train5_fpr, train5_tpr)))
```

```python
plt.plot(test5_fpr, test5_tpr, label="Test AUC ="+str(auc(test5_fpr, test5_tpr)))
plt.legend()
plt.xlabel("Fasle Postive Rate")
plt.ylabel("True Pstive RAte")
plt.title("ROC_AUC Curve")
plt.grid(color='black', linestyle='-', linewidth=1)
plt.show()
```



Conclusion for SET : 5

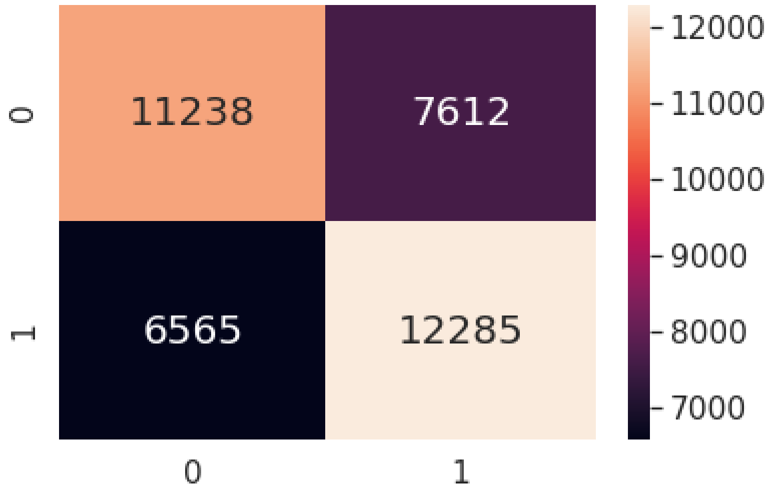
1. The above Represents the TRP and FPR rates on the either axis and this curve is know as the AUCROC curve , it is a metrix to evaluate the performar model.
2. As teh grap represents here the Train Auc = 0.67 , so from this we can conclude that the model is predecting the values with 67 % probabilty.

```python
#this is the custom function for predecting the best thershold and sorting the values according the threshould
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t
```

```python
def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

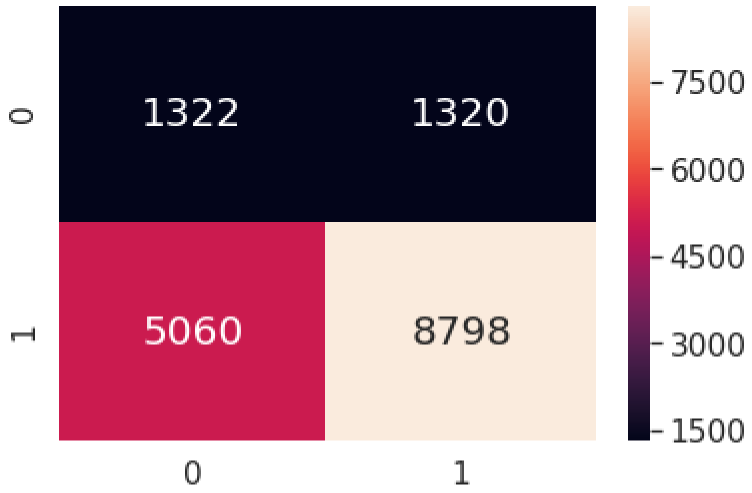
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr5_thresholds, train5_fpr, train5_tpr)
Confusion_metrix_Train_data = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y5_train_pred, best_t)))
Confusion_metrix_Test_data = pd.DataFrame(confusion_matrix(Y_test, predict_with_best_t(y5_test_pred, best_t)))
import seaborn as sns
sns.set(font_scale=1.4)#for label size
print("The Confusion metrix of train data")
sns.heatmap(Confusion_metrix_Train_data,annot=True, annot_kws={"size": 20},fmt ="g")
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.38854513857129785 for threshold 0.475
The Confusion metrix of train data
<matplotlib.axes._subplots.AxesSubplot at 0x7fd41dba3cc0>
```



```
print("The Confusion metrix of test data ")
sns.heatmap(Confusion_metrix_Test_data,annot=True, annot_kws={"size": 20},fmt ="g")
```

```
The Confusion metrix of test data
<matplotlib.axes._subplots.AxesSubplot at 0x7fd41c69ff28>
```



1. The Ouput Stated above represents the maximun value of TRP (i.e tpr*(1-fpr) "0.3973" corrosponding to whcih the maximum thershold is 0.49 .
2. The Second output represnts the Confusion metrix Based on the thershold of 0.49 , which states that the values below the thershold of 0.49 are clas 0 and the values above 0.49 are classified as 1.

```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Hyperparameter", "AUC"]
x.add_row(["SET 1 : project_title(BOW) + preprocessed_essay (BOW)", 101 ,  0.55])
x.add_row(["SET 2 : project_title(TFIDF)+ preprocessed_essay (TFIDF)", 101 , 0.59 ])
x.add_row(["SET 3 :  project_title(AVG W2V)+ preprocessed_essay (AVG W2V)", 101, 0.59])
x.add_row(["SET 4 : project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)", 101, 0.60])
x.add_row(["SET 5 : project_title(TFIDF)+ preprocessed_essay (TFIDF) of 2000 Words", 100, 0.59])

print(x)
```

```
+------------------------------------------------------------------------+----------------+------+
|                               Vectorizer                               | Hyperparameter | AUC  |
+------------------------------------------------------------------------+----------------+------+
|         SET 1 : project_title(BOW) + preprocessed_essay (BOW)          |      101       | 0.55 |
|        SET 2 : project_title(TFIDF)+ preprocessed_essay (TFIDF)        |      101       | 0.59 |
|     SET 3 :  project_title(AVG W2V)+ preprocessed_essay (AVG W2V)      |      101       | 0.59 |
|    SET 4 : project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)    |      101       | 0.6  |
| SET 5 : project_title(TFIDF)+ preprocessed_essay (TFIDF) of 2000 Words |      100       | 0.59 |
+------------------------------------------------------------------------+----------------+------+
```