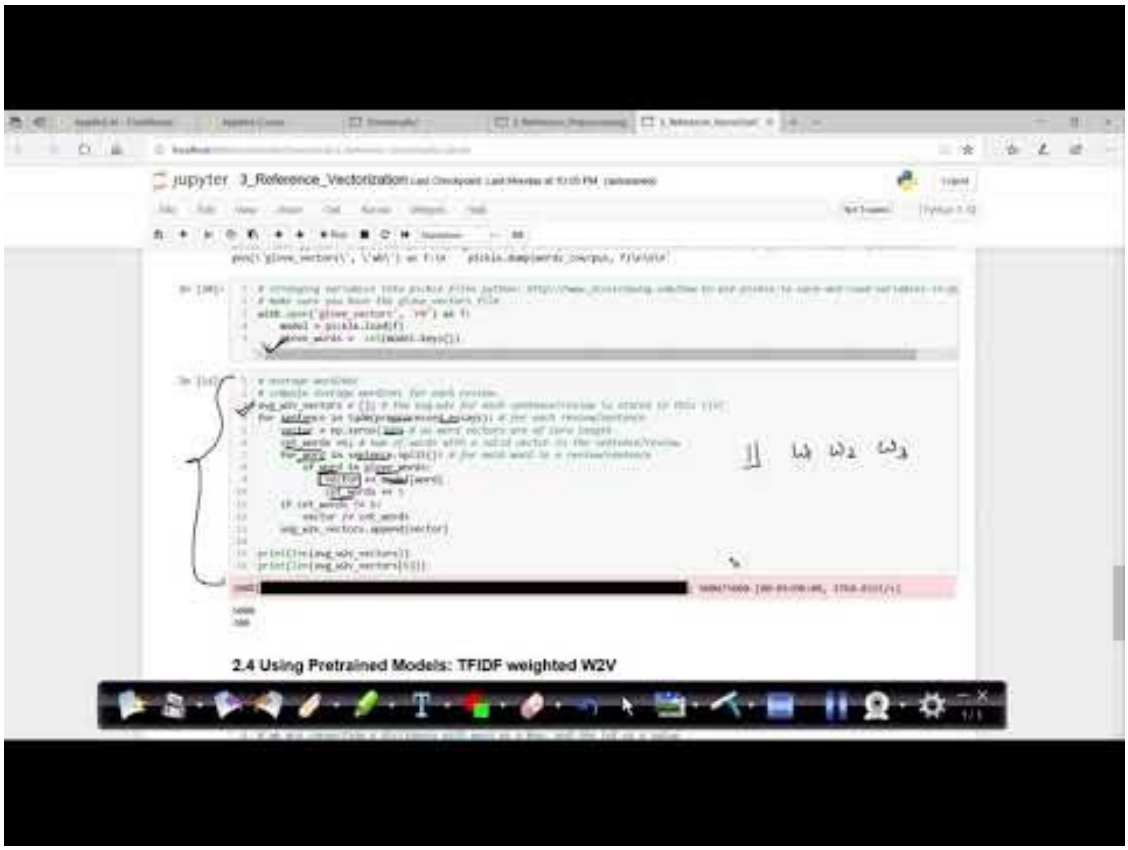
Assignment : DT

Please check below video before attempting this assignment

```
from IPython.display import YouTubeVideo
YouTubeVideo('ZhLXULFjIjQ', width="1000",height="500")
```



TF-IDFW2V

Tfidf w2v (w1,w2..) = (tfidf(w1) * w2v(w1) + tfidf(w2) * w2v(w2) + ...) / (tfidf(w1) + tfidf(w2) + ...)

(Optional) Please check course video on AVgw2V and TF-IDFW2Vfor more details.

Glove vectors

In this assignment you will be working with glove vectors , please check this and this for more details.

Download glove vectors from this link

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

```
import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
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 sklearn.model_selection import GridSearchCV

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

import pickle
from tqdm import tqdm
import os

import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
import warnings
warnings.simplefilter("ignore", UserWarning)

#please use below code to load glove vectors
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

or else , you can use below code

```
'''
# Reading glove vectors in python:
https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')
```

```
# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and
our coupus", \

len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python:
http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-
variables-in-python/

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)


'''

'\n# Reading glove vectors in python:
https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f =
open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    for line
```

```
in tqdm(f):\n        splitLine = line.split()\n        word =
splitLine[0]\n        embedding = np.array([float(val) for val in
splitLine[1:]])\n        model[word] = embedding\n    print
("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n#
============================\nOutput:\n    \nLoading Glove Model\
n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\
n    words.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n
words.extend(i.split(\' \'))\nprint("all the words in the coupus",
len(words))\nwords = set(words)\nprint("the unique words in the
coupus", len(words))\n\ninter_words =
set(model.keys()).intersection(words)\nprint("The number of words that
are present in both glove vectors and our coupus",
len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")
\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in
words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\
nprint("word 2 vec length", len(words_courpus))\n\n\n# stronging
variables into pickle files python: http://www.jessicayung.com/how-to-
use-pickle-to-save-and-load-variables-in-python/\n\nimport pickle\
nwith open(\'glove_vectors\', \'wb\') as f:\n
pickle.dump(words_courpus, f)\n\n\n'
```

Task - 1

```
<li><strong>Apply Decision Tree Classifier(DecisionTreeClassifier) on
these feature sets</strong>
    <ul>
        <li><font color='red'>Set 1</font>: categorical, numerical
features +  preprocessed_essay (TFIDF) + Sentiment
scores(preprocessed_essay)</li>
        <li><font color='red'>Set 2</font>: categorical, numerical
features +  preprocessed_essay (TFIDF W2V) + Sentiment
scores(preprocessed_essay)</li>        </ul>
</li>
<li><strong>The hyper paramter tuning (best `depth` in range [1, 3,
10, 30], and the best `min_samples_split` in range [5, 10, 100,
500])</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a
href='https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/receiver-operating-characteristic-curve-roc-curve-and-
auc-1/'>AUC</a> value</li>
<li>find the best hyper paramter using k-fold cross validation(use
gridsearch cv or randomsearch cv)/simple cross validation data(you can
write your own for loops refer sample solution)</li>
    </ul>
</li>
<li>
<strong>Representation of results</strong>
```

```
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
<img src='https://i.imgur.com/Gp2DQmh.jpg' width=500px> with X-axis as <strong>min_sample_split</strong>, Y-axis as <strong>max_depth</strong>, and Z-axis as <strong>AUC Score</strong> , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive <i>3d_scatter_plot.ipynb</i></li>
        <p style="text-align:center;font-size:30px;color:red;"><strong>or</strong></p> <br>
<li>You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
<img src='https://i.imgur.com/fgN9aUP.jpg' width=300px> <a href='https://seaborn.pydata.org/generated/seaborn.heatmap.html'>seaborn heat maps</a> with rows as <strong>min_sample_split</strong>, columns as <strong>max_depth</strong>, and values inside the cell representing <strong>AUC Score</strong> </li>
<li>You choose either of the plotting techniques out of 3d plot or heat map</li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    Make sure that you are using predict_proba method to calculate AUC curves, because AUC is calcualted on class probabilities and not on class labels.
<img src='https://i.imgur.com/wMQDTFe.jpg' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/'>confusion matrix</a> with predicted and original labels of test data points
<img src='https://i.imgur.com/IdN5Ctv.png' width=300px></li>
<li>Once after you plot the confusion matrix with the test data, get all the `false positive data points`
    <ul>
        <li> Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these `false positive data points`</li>
        <li> Plot the box plot with the `price` of these `false positive data points`</li>
        <li> Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`</li>
    </ul>
    </ul>
</li>
```

Task - 2

For this task consider **set-1** features.

- Select all the features which are having non-zero feature importance.You can get the feature importance using 'feature_importances_` (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html), discard the all other remaining features and then apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM).
- You need to do hyperparameter tuning corresponding to the model you selected and procedure in step 2 and step 3 **Note**: when you want to find the feature importance make sure you don't use max_depth parameter keep it None. You need to summarize the results at the end of the notebook, summarize it in the table format

Hint for calculating Sentiment scores

```
 import nltk

nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\akash.ragothu\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!

True
```

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

sample_sentence_1='I am happy.'
ss_1 = sid.polarity_scores(sample_sentence_1)
print('sentiment score for sentence 1',ss_1)

sample_sentence_2='I am sad.'
ss_2 = sid.polarity_scores(sample_sentence_2)
print('sentiment score for sentence 2',ss_2)

sample_sentence_3='I am going to New Delhi tommorow.'
ss_3 = sid.polarity_scores(sample_sentence_3)
print('sentiment score for sentence 3',ss_3)
```

```
sentiment score for sentence 1 {'neg': 0.0, 'neu': 0.213, 'pos':
0.787, 'compound': 0.5719}
sentiment score for sentence 2 {'neg': 0.756, 'neu': 0.244, 'pos':
0.0, 'compound': -0.4767}
```

```
sentiment score for sentence 3 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0,
'compound': 0.0}
```

Task - 1

1.1 Loading Data

```
#make sure you are loading atleast 50k datapoints
#you can work with features of preprocessed_data.csv for the
assignment.
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows=10000)

print(data.shape)
data.head(2)
```

```
(10000, 9)

  school_state teacher_prefix project_grade_category  \
0           ca            mrs          grades_prek_2
1           ut             ms             grades_3_5

   teacher_number_of_previously_posted_projects
project_is_approved  \
0                                            53                    1

1                                             4                    1


  clean_categories                 clean_subcategories  \
0     math_science  appliedsciences health_lifescience
1     specialneeds                          specialneeds

                                               essay   price
0  i fortunate enough use fairy tale stem kits cl...  725.05
1  imagine 8 9 years old you third grade classroo...  213.03
```

```
# write your code in following steps for task 1
# 1. calculate sentiment scores for the essay feature
sentiment_list =[]
for sentence in data["essay"]:
    sentiment_list.append(sid.polarity_scores(sentence)["compound"])

#print(sentiment_list)
data["essay_sentimental_score"] = sentiment_list
data.head(2)
```

```
  school_state teacher_prefix project_grade_category  \
0           ca            mrs          grades_prek_2
```

```
1           ut             ms             grades_3_5

   teacher_number_of_previously_posted_projects
project_is_approved  \
0                                            53                     1

1                                             4                     1


  clean_categories                 clean_subcategories  \
0     math_science  appliedsciences health_lifescience
1     specialneeds                          specialneeds

                                                 essay   price  \
0  i fortunate enough use fairy tale stem kits cl...  725.05
1  imagine 8 9 years old you third grade classroo...  213.03

   essay_sentimental_score
0                   0.9867
1                   0.9897
```

```
# 2. Split your data.
from sklearn.model_selection import train_test_split

data_set1 = data
data_set2 = data.copy()

#Data Set1:
Y = data.project_is_approved
# Split data set into train and test
X_train1, X_test1, y_train1, y_test1 = train_test_split( data_set1, Y,
test_size=0.2, random_state=12)
print("X_train1 (60%):", X_train1.shape)
print("X_test1 (20%):",X_train1.shape)

#Data Set2:
Y = data.project_is_approved
# Split data set into train and test
X_train2, X_test2, y_train2, y_test2 = train_test_split( data_set2, Y,
test_size=0.2, random_state=12)
print("X_train2 (60%):", X_train2.shape)
print("X_test2 (20%):",X_test2.shape)

X_train1.head()
X_train1['essay_sentimental_score'].shape
```

```
X_train1 (60%): (8000, 10)
X_test1 (20%): (8000, 10)
X_train2 (60%): (8000, 10)
X_test2 (20%): (2000, 10)
```

```
(8000,)
```

Preparing Data set1: Vectorization,standardization

TFIDF vectorization: Data set1

```
# 3. perform tfidf vectorization of text data.

preprocessed_essays_Xtrain1 = X_train1['essay'].values
preprocessed_essays_Xtest1 = X_test1['essay'].values
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf_Xtrain1 =
vectorizer.fit_transform(preprocessed_essays_Xtrain1)
text_tfidf_Xtest1 = vectorizer.transform(preprocessed_essays_Xtest1)

print("Shape of matrix after one hot encodig (Xtrain1):
",text_tfidf_Xtrain1.shape)
print("Shape of matrix after one hot encodig (Xtest1):
",text_tfidf_Xtest1.shape)

#print(type(text_tfidf_Xtrain1))
```

```
Shape of matrix after one hot encodig (Xtrain1):  (8000, 5619)
Shape of matrix after one hot encodig (Xtest1):  (2000, 5619)
```

Encoding of categorical features: Data set1

```
# 5. perform encoding of categorical features.

# School_state
vectorizer = CountVectorizer(binary=True)
school_state_ohe_Xtrain1 =
vectorizer.fit_transform(X_train1['school_state'].values)
school_state_ohe_Xtest1 =
vectorizer.transform(X_test1['school_state'].values)

print("Shape of matrix after one hot encoding (Xtrain1: school_state):
",school_state_ohe_Xtrain1.shape)
print("Shape of matrix after one hot encoding (Xtest1: school_state):
",school_state_ohe_Xtest1.shape)

# teacher_prefix
vectorizer = CountVectorizer(binary=True)
teacher_prefix_ohe_Xtrain1 =
vectorizer.fit_transform(X_train1['teacher_prefix'].values)
teacher_prefix_ohe_Xtest1 =
vectorizer.transform(X_test1['teacher_prefix'].values)
```

```
print("Shape of matrix after one hot encoding
(Xtrain1:teacher_prefix): ",teacher_prefix_ohe_Xtrain1.shape)
print("Shape of matrix after one hot encoding (Xtest1:teacher_prefix)
",teacher_prefix_ohe_Xtest1.shape)

# project_grade_category
vectorizer = CountVectorizer(binary=True)
project_grade_category_ohe_Xtrain1 =
vectorizer.fit_transform(X_train1['project_grade_category'].values)
project_grade_category_ohe_Xtest1 =
vectorizer.transform(X_test1['project_grade_category'].values)

print("Shape of matrix after one hot encoding
(Xtrain1:project_grade_category):
",project_grade_category_ohe_Xtrain1.shape)
print("Shape of matrix after one hot encoding
(Xtest1:project_grade_category)
",project_grade_category_ohe_Xtest1.shape)

#clean_categories
vectorizer = CountVectorizer(binary=True)
clean_categories_ohe_Xtrain1 =
vectorizer.fit_transform(X_train1['clean_categories'].values)
clean_categories_ohe_Xtest1 =
vectorizer.transform(X_test1['clean_categories'].values)

print("Shape of matrix after one hot encoding (Xtrain1:
clean_categories): ",clean_categories_ohe_Xtrain1.shape)
print("Shape of matrix after one hot encoding (Xtest1:
clean_categories) ",clean_categories_ohe_Xtest1.shape)

#clean_subcategories
vectorizer = CountVectorizer(binary=True)
clean_subcategories_ohe_Xtrain1 =
vectorizer.fit_transform(X_train1['clean_subcategories'].values)
clean_subcategories_ohe_Xtest1 =
vectorizer.transform(X_test1['clean_subcategories'].values)

print("Shape of matrix after one hot encoding
(Xtrain1:clean_subcategories):
",clean_subcategories_ohe_Xtrain1.shape)
print("Shape of matrix after one hot encoding
(Xtest1:clean_subcategories) ",clean_subcategories_ohe_Xtest1.shape)
```

```
Shape of matrix after one hot encoding (Xtrain1: school_state):
(8000, 51)
Shape of matrix after one hot encoding (Xtest1: school_state):  (2000,
51)
Shape of matrix after one hot encoding (Xtrain1:teacher_prefix):
```

```
(8000, 5)
Shape of matrix after one hot encoding (Xtest1:teacher_prefix)  (2000,
5)
Shape of matrix after one hot encoding
(Xtrain1:project_grade_category):  (8000, 4)
Shape of matrix after one hot encoding (Xtest1:project_grade_category)
(2000, 4)
Shape of matrix after one hot encoding (Xtrain1: clean_categories):
(8000, 7)
Shape of matrix after one hot encoding (Xtest1: clean_categories)
(2000, 7)
Shape of matrix after one hot encoding (Xtrain1:clean_subcategories):
(8000, 28)
Shape of matrix after one hot encoding (Xtest1:clean_subcategories)
(2000, 28)
```

Encoding of numerical features: Data set1

```
# 6. perform standardizing numerical features
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
# Price
scaler = StandardScaler()
standardized_price_Xtrain1 =
scaler.fit_transform(X_train1['price'].values.reshape(-1, 1))
standardized_price_Xtest1 =
scaler.transform(X_test1['price'].values.reshape(-1, 1))
scaler = MinMaxScaler()
nrm_price_Xtrain1=scaler.fit_transform(standardized_price_Xtrain1)
nrm_price_Xtest1=scaler.transform(standardized_price_Xtest1)

#print(nrm_price_Xtrain1.shape)
#print(type(nrm_price_Xtrain1))
#print(nrm_price_Xtrain1[:5])
#project_data['nrm_price_Xtrain1'].head()

# teacher_number_of_previously_posted_projects

scaler = StandardScaler()
standardized_teacher_number_of_previously_posted_projects_Xtrain1 =
scaler.fit_transform(X_train1['teacher_number_of_previously_posted_pro
jects'].values.reshape(-1, 1))
standardized_teacher_number_of_previously_posted_projects_Xtest1 =
scaler.transform(X_test1['teacher_number_of_previously_posted_projects
'].values.reshape(-1, 1))
scaler = MinMaxScaler()
nrm_teacher_number_of_previously_posted_projects_Xtrain1=scaler.fit_tr
ansform(standardized_teacher_number_of_previously_posted_projects_Xtra
in1)
```

```
nrm_teacher_number_of_previously_posted_projects_Xtest1=scaler.transfo
rm(standardized_teacher_number_of_previously_posted_projects_Xtest1)

# essay_sentimental_score

scaler = StandardScaler()
standardized_essay_sentimental_score_Xtrain1 =
scaler.fit_transform(X_train1['essay_sentimental_score'].values.reshap
e(-1, 1))
standardized_essay_sentimental_score_Xtest1 =
scaler.transform(X_test1['essay_sentimental_score'].values.reshape(-1,
1))
scaler = MinMaxScaler()
nrm_essay_sentimental_score_Xtrain1=scaler.fit_transform(standardized_
essay_sentimental_score_Xtrain1)
nrm_essay_sentimental_score_Xtest1=scaler.transform(standardized_essay
_sentimental_score_Xtest1)
```

Stack up all the features : Data set1

```
# 7. For task 1 set 1 stack up all the features

#Xtrain of data set1

#convert school_state_ohe sparse matrix to dense
#print(school_state_ohe_Xtrain1.shape)
column_names = [ "school_ohe_"+str(i) for i in
range(school_state_ohe_Xtrain1.shape[1])]
#print(column_names)
school_state_ohe_Xtrain1_df =
pd.DataFrame(school_state_ohe_Xtrain1.todense(),columns =column_names)
#school_state_ohe_df.column = column_names
school_state_ohe_Xtrain1_df.head(2)


#convert teacher_prefix_ohe_Xtrain1 sparse matrix to dense
#print(teacher_prefix_ohe_Xtrain1.shape)
column_names = [ "teacher_prefix_ohe_"+str(i) for i in
range(teacher_prefix_ohe_Xtrain1.shape[1])]
#print(column_names)
teacher_prefix_ohe_Xtrain1_df =
pd.DataFrame(teacher_prefix_ohe_Xtrain1.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
teacher_prefix_ohe_Xtrain1_df.head(2)
```

```
#convert project_grade_category_ohe_Xtrain1 sparse matrix to dense
#print(project_grade_category_ohe_Xtrain1.shape)
column_names = [ "project_grade_category_ohe_"+str(i) for i in
range(project_grade_category_ohe_Xtrain1.shape[1])]
#print(column_names)
project_grade_category_ohe_Xtrain1_df =
pd.DataFrame(project_grade_category_ohe_Xtrain1.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
project_grade_category_ohe_Xtrain1_df.head(2)

#convert clean_categories_ohe_Xtrain1 sparse matrix to dense
#print(clean_categories_ohe_Xtrain1.shape)
column_names = [ "clean_categories_ohe_"+str(i) for i in
range(clean_categories_ohe_Xtrain1.shape[1])]
#print(column_names)
clean_categories_ohe_Xtrain1_df =
pd.DataFrame(clean_categories_ohe_Xtrain1.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
clean_categories_ohe_Xtrain1_df.head(2)

#convert clean_subcategories_ohe_Xtrain1 sparse matrix to dense
#print(clean_subcategories_ohe_Xtrain1.shape)
column_names = [ "clean_subcategories_ohe_"+str(i) for i in
range(clean_subcategories_ohe_Xtrain1.shape[1])]
#print(column_names)
clean_subcategories_ohe_Xtrain1_df =
pd.DataFrame(clean_subcategories_ohe_Xtrain1.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
clean_subcategories_ohe_Xtrain1_df.head(2)

#convert text_tfidf sparse matrix to dense
#print(text_tfidf.shape)
column_names = [ "text_tfidf"+str(i) for i in
range(text_tfidf_Xtrain1.shape[1])]
#print(column_names)
text_tfidf_Xtrain1_df =
pd.DataFrame(text_tfidf_Xtrain1.todense(),columns =column_names)
#school_state_ohe_df.column = column_names
text_tfidf_Xtrain1_df.head(2)

#sent_score =X_train1.essay_sentimental_score

X_train1_df =
pd.concat([school_state_ohe_Xtrain1_df,teacher_prefix_ohe_Xtrain1_df,p
roject_grade_category_ohe_Xtrain1_df,

pd.DataFrame(nrm_teacher_number_of_previously_posted_projects_Xtrain1,
```

```
columns =["nrm_teacher_number_of_previously_posted_projects"]),


clean_categories_ohe_Xtrain1_df,clean_subcategories_ohe_Xtrain1_df,
                      text_tfidf_Xtrain1_df,
                      pd.DataFrame(nrm_price_Xtrain1,columns
=["nrm_price"]),

pd.DataFrame(nrm_essay_sentimental_score_Xtrain1,columns
=["nrm_essay_sentimental_score"])],axis=1)
#project_grade_category_ohe_df.head(2)
print("Xtrain of Data Set 1 ")
print("-"*50)
print("size: ",X_train1_df.shape)
X_train1_df.head(2)

#print(X_train1_df.shape)
```

```
Xtrain of Data Set 1
--------------------------------------------------
size:  (8000, 5717)
```

```
   school_ohe_0  school_ohe_1  school_ohe_2  school_ohe_3
school_ohe_4  \
0             0             0             0             0
0
1             0             0             0             0
0

   school_ohe_5  school_ohe_6  school_ohe_7  school_ohe_8
school_ohe_9  \
0             0             0             0             0
0
1             0             0             0             0
0

                ...               text_tfidf5611  text_tfidf5612  \
0               ...                          0.0             0.0
1               ...                          0.0             0.0

   text_tfidf5613  text_tfidf5614  text_tfidf5615  text_tfidf5616  \
0             0.0             0.0             0.0             0.0
1             0.0             0.0             0.0             0.0

   text_tfidf5617  text_tfidf5618  nrm_price
nrm_essay_sentimental_score
0             0.0             0.0   0.057476
0.997536
1             0.0             0.0   0.013664
0.995122
```

```
[2 rows x 5717 columns]
```

```
# 7. For task 1 set 1 stack up all the features

#Xtest of data set1

#convert school_state_ohe sparse matrix to dense
#print(school_state_ohe_Xtest1.shape)
column_names = [ "school_ohe_"+str(i) for i in
range(school_state_ohe_Xtest1.shape[1])]
#print(column_names)
school_state_ohe_Xtest1_df =
pd.DataFrame(school_state_ohe_Xtest1.todense(),columns =column_names)
#school_state_ohe_df.column = column_names
school_state_ohe_Xtest1_df.head(2)


#convert teacher_prefix_ohe_Xtest1 sparse matrix to dense
#print(teacher_prefix_ohe_Xtest1.shape)
column_names = [ "teacher_prefix_ohe_"+str(i) for i in
range(teacher_prefix_ohe_Xtest1.shape[1])]
#print(column_names)
teacher_prefix_ohe_Xtest1_df =
pd.DataFrame(teacher_prefix_ohe_Xtest1.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
teacher_prefix_ohe_Xtest1_df.head(2)


#convert project_grade_category_ohe_Xtest1 sparse matrix to dense
#print(project_grade_category_ohe_Xtest1.shape)
column_names = [ "project_grade_category_ohe_"+str(i) for i in
range(project_grade_category_ohe_Xtest1.shape[1])]
#print(column_names)
project_grade_category_ohe_Xtest1_df =
pd.DataFrame(project_grade_category_ohe_Xtest1.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
project_grade_category_ohe_Xtest1_df.head(2)

#convert clean_categories_ohe_Xtest1 sparse matrix to dense
#print(clean_categories_ohe_Xtest1.shape)
column_names = [ "clean_categories_ohe_"+str(i) for i in
range(clean_categories_ohe_Xtest1.shape[1])]
#print(column_names)
clean_categories_ohe_Xtest1_df =
pd.DataFrame(clean_categories_ohe_Xtest1.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
```

```
clean_categories_ohe_Xtest1_df.head(2)

#convert clean_subcategories_ohe_Xtest1 sparse matrix to dense
#print(clean_subcategories_ohe_Xtest1.shape)
column_names = [ "clean_subcategories_ohe_"+str(i) for i in
range(clean_subcategories_ohe_Xtest1.shape[1])]
#print(column_names)
clean_subcategories_ohe_Xtest1_df =
pd.DataFrame(clean_subcategories_ohe_Xtest1.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
clean_subcategories_ohe_Xtest1_df.head(2)

#convert text_tfidf sparse matrix to dense
#print(text_tfidf.shape)
column_names = [ "text_tfidf"+str(i) for i in
range(text_tfidf_Xtest1.shape[1])]
#print(column_names)
text_tfidf_Xtest1_df =
pd.DataFrame(text_tfidf_Xtest1.todense(),columns =column_names)
#school_state_ohe_df.column = column_names
text_tfidf_Xtest1_df.head(2)

#sent_score =X_train1.essay_sentimental_score

X_test1_df =
pd.concat([school_state_ohe_Xtest1_df,teacher_prefix_ohe_Xtest1_df,pro
ject_grade_category_ohe_Xtest1_df,

pd.DataFrame(nrm_teacher_number_of_previously_posted_projects_Xtest1,c
olumns =["nrm_teacher_number_of_previously_posted_projects"]),


clean_categories_ohe_Xtest1_df,clean_subcategories_ohe_Xtest1_df,
                        text_tfidf_Xtest1_df,
                        pd.DataFrame(nrm_price_Xtest1,columns
=["nrm_price"]),

pd.DataFrame(nrm_essay_sentimental_score_Xtest1,columns
=["nrm_essay_sentimental_score"])],axis=1)
#project_grade_category_ohe_df.head(2)
print("Xtest of Data Set 1 ")
print("-"*50)
print("size: ",X_test1_df.shape)
X_train1_df.head(2)

#print(X_train1_df.shape)
```

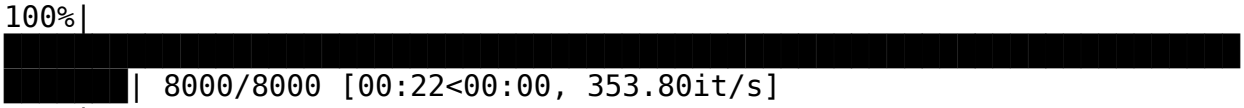
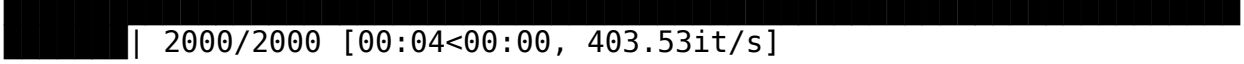
```
Xtest of Data Set 1
--------------------------------------------------
size:  (2000, 5717)

   school_ohe_0  school_ohe_1  school_ohe_2  school_ohe_3
school_ohe_4  \
0             0             0             0             0
0
1             0             0             0             0
0

   school_ohe_5  school_ohe_6  school_ohe_7  school_ohe_8
school_ohe_9  \
0             0             0             0             0
0
1             0             0             0             0
0

               ...               text_tfidf5611  text_tfidf5612  \
0              ...                          0.0             0.0
1              ...                          0.0             0.0

   text_tfidf5613  text_tfidf5614  text_tfidf5615  text_tfidf5616  \
0             0.0             0.0             0.0             0.0
1             0.0             0.0             0.0             0.0

   text_tfidf5617  text_tfidf5618  nrm_price
nrm_essay_sentimental_score
0             0.0             0.0   0.057476
0.997536
1             0.0             0.0   0.013664
0.995122

[2 rows x 5717 columns]
```

Preparing Data set2: Vectorization,standardization

TFIDF W2V Vectorization: Data set2

```
# 4. perform tfidf w2v vectorization of text data.

preprocessed_essays_Xtrain2 = X_train2['essay'].values
preprocessed_essays_Xtest2 = X_test2['essay'].values
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_Xtrain2)

# we are converting a dictionary with word as a key, and the tf-idf as
a value
```

```
dictionary = dict(zip(tfidf_model.get_feature_names(),
list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

# average Word2Vec
# compute average word2vec for each review.

def get_tfidf_w2v(essays):

    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is
stored in this list
    for sentence in tqdm(essays): # 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
        tfidf_w2v_vectors.append(vector)
    return tfidf_w2v_vectors

text_tfidf_w2v_Xtrain2 =
np.array(get_tfidf_w2v(preprocessed_essays_Xtrain2))
text_tfidf_w2v_Xtest2 =
np.array(get_tfidf_w2v(preprocessed_essays_Xtest2))


#print(len(text_tfidf_w2v_Xtrain2))
#print(len(text_tfidf_w2v_Xtrain2[0]))
```

```
100%|
```

```
| 8000/8000 [00:22<00:00, 353.80it/s]
100%|
```

```
| 2000/2000 [00:04<00:00, 403.53it/s]
```

Encoding of categorical features: Data set2

```
# 5. perform encoding of categorical features.

# School_state
vectorizer = CountVectorizer(binary=True)
school_state_ohe_Xtrain2 =
vectorizer.fit_transform(X_train2['school_state'].values)
school_state_ohe_Xtest2 =
vectorizer.transform(X_test2['school_state'].values)

print("Shape of matrix after one hot encoding (Xtrain2: school_state):
",school_state_ohe_Xtrain2.shape)
print("Shape of matrix after one hot encoding (Xtest2: school_state):
",school_state_ohe_Xtest2.shape)

# teacher_prefix
vectorizer = CountVectorizer(binary=True)
teacher_prefix_ohe_Xtrain2 =
vectorizer.fit_transform(X_train2['teacher_prefix'].values)
teacher_prefix_ohe_Xtest2 =
vectorizer.transform(X_test2['teacher_prefix'].values)

print("Shape of matrix after one hot encoding
(Xtrain2:teacher_prefix): ",teacher_prefix_ohe_Xtrain2.shape)
print("Shape of matrix after one hot encoding (Xtest2:teacher_prefix)
",teacher_prefix_ohe_Xtest2.shape)

# project_grade_category
vectorizer = CountVectorizer(binary=True)
project_grade_category_ohe_Xtrain2 =
vectorizer.fit_transform(X_train2['project_grade_category'].values)
project_grade_category_ohe_Xtest2 =
vectorizer.transform(X_test2['project_grade_category'].values)

print("Shape of matrix after one hot encoding
(Xtrain2:project_grade_category):
",project_grade_category_ohe_Xtrain2.shape)
print("Shape of matrix after one hot encoding
(Xtest2:project_grade_category)
",project_grade_category_ohe_Xtest2.shape)

#clean_categories
vectorizer = CountVectorizer(binary=True)
clean_categories_ohe_Xtrain2 =
vectorizer.fit_transform(X_train2['clean_categories'].values)
clean_categories_ohe_Xtest2 =
vectorizer.transform(X_test2['clean_categories'].values)

print("Shape of matrix after one hot encoding (Xtrain2:
```

```
clean_categories): ",clean_categories_ohe_Xtrain2.shape)
print("Shape of matrix after one hot encoding (Xtest2:
clean_categories) ",clean_categories_ohe_Xtest2.shape)

#clean_subcategories
vectorizer = CountVectorizer(binary=True)
clean_subcategories_ohe_Xtrain2 =
vectorizer.fit_transform(X_train2['clean_subcategories'].values)
clean_subcategories_ohe_Xtest2 =
vectorizer.transform(X_test2['clean_subcategories'].values)

print("Shape of matrix after one hot encoding
(Xtrain2:clean_subcategories):
",clean_subcategories_ohe_Xtrain2.shape)
print("Shape of matrix after one hot encoding
(Xtest2:clean_subcategories) ",clean_subcategories_ohe_Xtest2.shape)
```

```
Shape of matrix after one hot encoding (Xtrain2: school_state):
(8000, 51)
Shape of matrix after one hot encoding (Xtest2: school_state):  (2000,
51)
Shape of matrix after one hot encoding (Xtrain2:teacher_prefix):
(8000, 5)
Shape of matrix after one hot encoding (Xtest2:teacher_prefix)  (2000,
5)
Shape of matrix after one hot encoding
(Xtrain2:project_grade_category):  (8000, 4)
Shape of matrix after one hot encoding (Xtest2:project_grade_category)
(2000, 4)
Shape of matrix after one hot encoding (Xtrain2: clean_categories):
(8000, 7)
Shape of matrix after one hot encoding (Xtest2: clean_categories)
(2000, 7)
Shape of matrix after one hot encoding (Xtrain2:clean_subcategories):
(8000, 28)
Shape of matrix after one hot encoding (Xtest2:clean_subcategories)
(2000, 28)
```

Encoding of numerical features: Data set2

```
# 6. perform standardizing numerical features
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
# Price
scaler = StandardScaler()
standardized_price_Xtrain2 =
scaler.fit_transform(X_train2['price'].values.reshape(-1, 1))
standardized_price_Xtest2 =
scaler.transform(X_test2['price'].values.reshape(-1, 1))
scaler = MinMaxScaler()
```

```
nrm_price_Xtrain2=scaler.fit_transform(standardized_price_Xtrain2)
nrm_price_Xtest2=scaler.transform(standardized_price_Xtest2)

print(nrm_price_Xtrain2.shape)
#print(type(nrm_price_Xtrain2))
#print(nrm_price_Xtrain2[:5])
#project_data['nrm_price_Xtrain2'].head()

# teacher_number_of_previously_posted_projects

scaler = StandardScaler()
standardized_teacher_number_of_previously_posted_projects_Xtrain2 =
scaler.fit_transform(X_train2['teacher_number_of_previously_posted_pro
jects'].values.reshape(-1, 1))
standardized_teacher_number_of_previously_posted_projects_Xtest2 =
scaler.transform(X_test2['teacher_number_of_previously_posted_projects
'].values.reshape(-1, 1))
scaler = MinMaxScaler()
nrm_teacher_number_of_previously_posted_projects_Xtrain2=scaler.fit_tr
ansform(standardized_teacher_number_of_previously_posted_projects_Xtra
in2)
nrm_teacher_number_of_previously_posted_projects_Xtest2=scaler.transfo
rm(standardized_teacher_number_of_previously_posted_projects_Xtest2)

# essay_sentimental_score

scaler = StandardScaler()
standardized_essay_sentimental_score_Xtrain2 =
scaler.fit_transform(X_train2['essay_sentimental_score'].values.reshap
e(-1, 1))
standardized_essay_sentimental_score_Xtest2 =
scaler.transform(X_test2['essay_sentimental_score'].values.reshape(-1,
1))
scaler = MinMaxScaler()
nrm_essay_sentimental_score_Xtrain2=scaler.fit_transform(standardized_
essay_sentimental_score_Xtrain2)
nrm_essay_sentimental_score_Xtest2=scaler.transform(standardized_essay
_sentimental_score_Xtest2)



(8000, 1)
```

Stack up all the features : Data set2

```
# 7. For task 2 set 2 stack up all the features

#Xtrain of data set2
```

```
#convert school_state_ohe sparse matrix to dense
#print(school_state_ohe_Xtrain2.shape)
column_names = [ "school_ohe_"+str(i) for i in
range(school_state_ohe_Xtrain2.shape[1])]
#print(column_names)
school_state_ohe_Xtrain2_df =
pd.DataFrame(school_state_ohe_Xtrain2.todense(),columns =column_names)
#school_state_ohe_df.column = column_names
school_state_ohe_Xtrain2_df.head(2)


#convert teacher_prefix_ohe_Xtrain2 sparse matrix to dense
#print(teacher_prefix_ohe_Xtrain2.shape)
column_names = [ "teacher_prefix_ohe_"+str(i) for i in
range(teacher_prefix_ohe_Xtrain2.shape[1])]
#print(column_names)
teacher_prefix_ohe_Xtrain2_df =
pd.DataFrame(teacher_prefix_ohe_Xtrain2.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
teacher_prefix_ohe_Xtrain2_df.head(2)


#convert project_grade_category_ohe_Xtrain2 sparse matrix to dense
#print(project_grade_category_ohe_Xtrain2.shape)
column_names = [ "project_grade_category_ohe_"+str(i) for i in
range(project_grade_category_ohe_Xtrain2.shape[1])]
#print(column_names)
project_grade_category_ohe_Xtrain2_df =
pd.DataFrame(project_grade_category_ohe_Xtrain2.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
project_grade_category_ohe_Xtrain2_df.head(2)

#convert clean_categories_ohe_Xtrain2 sparse matrix to dense
#print(clean_categories_ohe_Xtrain2.shape)
column_names = [ "clean_categories_ohe_"+str(i) for i in
range(clean_categories_ohe_Xtrain2.shape[1])]
#print(column_names)
clean_categories_ohe_Xtrain2_df =
pd.DataFrame(clean_categories_ohe_Xtrain2.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
clean_categories_ohe_Xtrain2_df.head(2)

#convert clean_subcategories_ohe_Xtrain2 sparse matrix to dense
#print(clean_subcategories_ohe_Xtrain2.shape)
column_names = [ "clean_subcategories_ohe_"+str(i) for i in
range(clean_subcategories_ohe_Xtrain2.shape[1])]
#print(column_names)
```

```
clean_subcategories_ohe_Xtrain2_df =
pd.DataFrame(clean_subcategories_ohe_Xtrain2.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
clean_subcategories_ohe_Xtrain2_df.head(2)

#convert text_tfidf sparse matrix to dense
#print(text_tfidf.shape)
column_names = [ "text_tfidf"+str(i) for i in
range(text_tfidf_w2v_Xtrain2.shape[1])]
#print(column_names)
text_tfidf_w2v_Xtrain2_df =
pd.DataFrame(text_tfidf_w2v_Xtrain2,columns =column_names)
#school_state_ohe_df.column = column_names
text_tfidf_w2v_Xtrain2_df.head(2)

#sent_score =X_train2.essay_sentimental_score

X_train2_df =
pd.concat([school_state_ohe_Xtrain2_df,teacher_prefix_ohe_Xtrain2_df,p
roject_grade_category_ohe_Xtrain2_df,

pd.DataFrame(nrm_teacher_number_of_previously_posted_projects_Xtrain2,
columns =["nrm_teacher_number_of_previously_posted_projects"]),


clean_categories_ohe_Xtrain2_df,clean_subcategories_ohe_Xtrain2_df,
                      text_tfidf_w2v_Xtrain2_df,
                      pd.DataFrame(nrm_price_Xtrain2,columns
=["nrm_price"]),

pd.DataFrame(nrm_essay_sentimental_score_Xtrain2,columns
=["nrm_essay_sentimental_score"])],axis=1)
#project_grade_category_ohe_df.head(2)
print("Xtrain of Data Set 2 ")
print("-"*50)
print("size: ",X_train2_df.shape)
X_train2_df.head(2)

#print(X_train2_df.shape)
```

```
Xtrain of Data Set 2
--------------------------------------------------
size:  (8000, 398)

   school_ohe_0  school_ohe_1  school_ohe_2  school_ohe_3
school_ohe_4  \
0             0             0             0             0
0
1             0             0             0             0
```

```
0

   school_ohe_5  school_ohe_6  school_ohe_7  school_ohe_8
school_ohe_9  \
0             0             0             0             0
0
1             0             0             0             0
0

              ...               text_tfidf292  text_tfidf293
text_tfidf294  \
0             ...                    0.004161      -0.014986
0.062333
1             ...                   -0.207442      -0.067349
0.070186

   text_tfidf295  text_tfidf296  text_tfidf297  text_tfidf298
text_tfidf299  \
0       0.082496       0.019540       0.267045       0.130154
0.014083
1      -0.019640      -0.185976       0.177698       0.132500
0.027462

   nrm_price  nrm_essay_sentimental_score
0   0.057476                     0.997536
1   0.013664                     0.995122

[2 rows x 398 columns]
```

```
# 7. For task 2 set 2 stack up all the features

#Xtrain of data set2

#convert school_state_ohe sparse matrix to dense
#print(school_state_ohe_Xtest2.shape)
column_names = [ "school_ohe_"+str(i) for i in
range(school_state_ohe_Xtest2.shape[1])]
#print(column_names)
school_state_ohe_Xtest2_df =
pd.DataFrame(school_state_ohe_Xtest2.todense(),columns =column_names)
#school_state_ohe_df.column = column_names
school_state_ohe_Xtest2_df.head(2)


#convert teacher_prefix_ohe_Xtest2 sparse matrix to dense
#print(teacher_prefix_ohe_Xtest2.shape)
column_names = [ "teacher_prefix_ohe_"+str(i) for i in
range(teacher_prefix_ohe_Xtest2.shape[1])]
#print(column_names)
```

```
teacher_prefix_ohe_Xtest2_df =
pd.DataFrame(teacher_prefix_ohe_Xtest2.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
teacher_prefix_ohe_Xtest2_df.head(2)


#convert project_grade_category_ohe_Xtest2 sparse matrix to dense
#print(project_grade_category_ohe_Xtest2.shape)
column_names = [ "project_grade_category_ohe_"+str(i) for i in
range(project_grade_category_ohe_Xtest2.shape[1])]
#print(column_names)
project_grade_category_ohe_Xtest2_df =
pd.DataFrame(project_grade_category_ohe_Xtest2.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
project_grade_category_ohe_Xtest2_df.head(2)

#convert clean_categories_ohe_Xtest2 sparse matrix to dense
#print(clean_categories_ohe_Xtest2.shape)
column_names = [ "clean_categories_ohe_"+str(i) for i in
range(clean_categories_ohe_Xtest2.shape[1])]
#print(column_names)
clean_categories_ohe_Xtest2_df =
pd.DataFrame(clean_categories_ohe_Xtest2.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
clean_categories_ohe_Xtest2_df.head(2)

#convert clean_subcategories_ohe_Xtest2 sparse matrix to dense
#print(clean_subcategories_ohe_Xtest2.shape)
column_names = [ "clean_subcategories_ohe_"+str(i) for i in
range(clean_subcategories_ohe_Xtest2.shape[1])]
#print(column_names)
clean_subcategories_ohe_Xtest2_df =
pd.DataFrame(clean_subcategories_ohe_Xtest2.todense(),columns
=column_names)
#school_state_ohe_df.column = column_names
clean_subcategories_ohe_Xtest2_df.head(2)

#convert text_tfidf sparse matrix to dense
#print(text_tfidf.shape)
column_names = [ "text_tfidf"+str(i) for i in
range(text_tfidf_w2v_Xtest2.shape[1])]
#print(column_names)
text_tfidf_w2v_Xtest2_df = pd.DataFrame(text_tfidf_w2v_Xtest2,columns
=column_names)
#school_state_ohe_df.column = column_names
text_tfidf_w2v_Xtest2_df.head(2)
```

```
#sent_score =X_test2.essay_sentimental_score

X_test2_df =
pd.concat([school_state_ohe_Xtest2_df,teacher_prefix_ohe_Xtest2_df,pro
ject_grade_category_ohe_Xtest2_df,

pd.DataFrame(nrm_teacher_number_of_previously_posted_projects_Xtest2,c
olumns =["nrm_teacher_number_of_previously_posted_projects"]),


clean_categories_ohe_Xtest2_df,clean_subcategories_ohe_Xtest2_df,
                      text_tfidf_w2v_Xtest2_df,
                      pd.DataFrame(nrm_price_Xtest2,columns
=["nrm_price"]),

pd.DataFrame(nrm_essay_sentimental_score_Xtest2,columns
=["nrm_essay_sentimental_score"])],axis=1)
#project_grade_category_ohe_df.head(2)
print("Xtest of Data Set 2 ")
print("-"*50)
print("size: ",X_test2_df.shape)
X_test2_df.head(2)

#print(X_test2_df.shape)
```

```
Xtest of Data Set 2
--------------------------------------------------
size:  (2000, 398)

   school_ohe_0  school_ohe_1  school_ohe_2  school_ohe_3
school_ohe_4  \
0             0             0             0             0
0
1             0             0             0             0
0

   school_ohe_5  school_ohe_6  school_ohe_7  school_ohe_8
school_ohe_9  \
0             0             0             0             0
0
1             0             0             0             0
0

              ...               text_tfidf292  text_tfidf293
text_tfidf294  \
0             ...                   -0.013635      -0.105754
0.072701
1             ...                   -0.102784      -0.079473      -
0.015621
```

```
   text_tfidf295  text_tfidf296  text_tfidf297  text_tfidf298
text_tfidf299  \
0      -0.006954      -0.097349       0.251716       0.031096      -
0.083033
1       0.023749       0.054284       0.125298       0.095684      -
0.006980

   nrm_price  nrm_essay_sentimental_score
0   0.038441                     0.993965
1   0.036825                     0.993512

[2 rows x 398 columns]
```

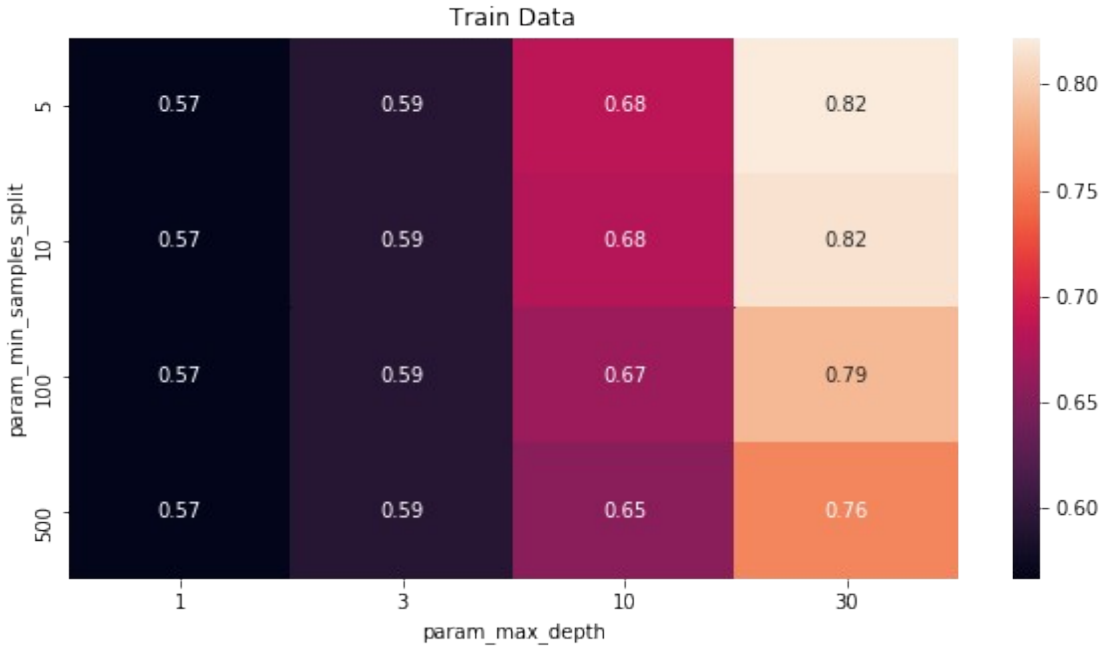
Applying DecisionTree Classifier on data set 1 (i.e., TFIDF)

```
#The hyper paramter tuning (best depth in range [1, 3, 10, 30], and
the best min_samples_split in range [5, 10, 100, 500])
#Find the best hyper parameter which will give the maximum AUC value
#find the best hyper paramter using k-fold cross validation(use
gridsearch cv or randomsearch cv)/simple cross validation data(you can
write your own for loops refer sample solution)

from sklearn.tree import DecisionTreeClassifier

# fit the model with best parameters found
clf = DecisionTreeClassifier(random_state =20)
clf.fit(X_train1_df,y_train1)

parameters = {'max_depth':[1, 3, 10, 30],'min_samples_split': [5, 10,
100, 500]}
GridSearch_clf = GridSearchCV(clf, parameters,scoring='roc_auc',cv
=3,return_train_score=True)
GridSearch_clf.fit(X_train1_df, y_train1)

print(GridSearch_clf.score(X_train1_df,y_train1))
print("------------Best Hyperparameters----------------")
# print best parameter after tuning
print(GridSearch_clf.best_params_)

# print how our model looks after hyper-parameter tuning
print(GridSearch_clf.best_estimator_)

#print(GridSearch_clf.cv_results_)

best_max_depth = GridSearch_clf.best_params_['max_depth']

best_min_samples_split =
GridSearch_clf.best_params_['min_samples_split']
```

```
0.734194937590675
------------Best Hyperparameters----------------
{'max_depth': 30, 'min_samples_split': 500}
DecisionTreeClassifier(max_depth=30, min_samples_split=500,
random_state=20)
```

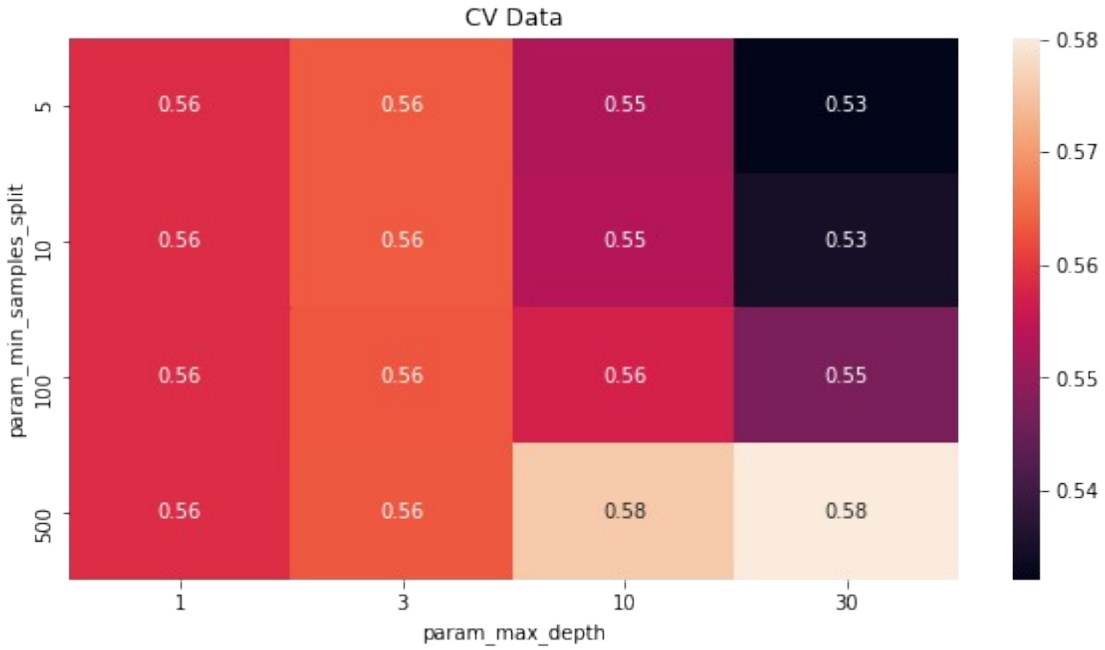
```
#Perform hyperparameter tuning and plot either heatmap or 3d plot.
#On Train data
temp =
np.stack((GridSearch_clf.cv_results_['param_min_samples_split'],GridSe
arch_clf.cv_results_['param_max_depth'],GridSearch_clf.cv_results_['me
an_train_score']),axis =1)
#print(temp)
scores = (pd.DataFrame(temp,columns =
['param_min_samples_split','param_max_depth','mean_test_score']).group
by(['param_min_samples_split','param_max_depth'])).max().unstack()
print(scores)
fig = plt.figure(figsize=(10,5))
sns.heatmap(scores.mean_test_score,annot = True)
plt.title("Train Data")
plt.show()
```

	mean_test_score			
param_max_depth	1	3	10	30
param_min_samples_split				
5	0.566385	0.593176	0.684766	0.821891
10	0.566385	0.593176	0.681024	0.815571
100	0.566385	0.592801	0.665660	0.788787
500	0.566385	0.592633	0.654973	0.757116



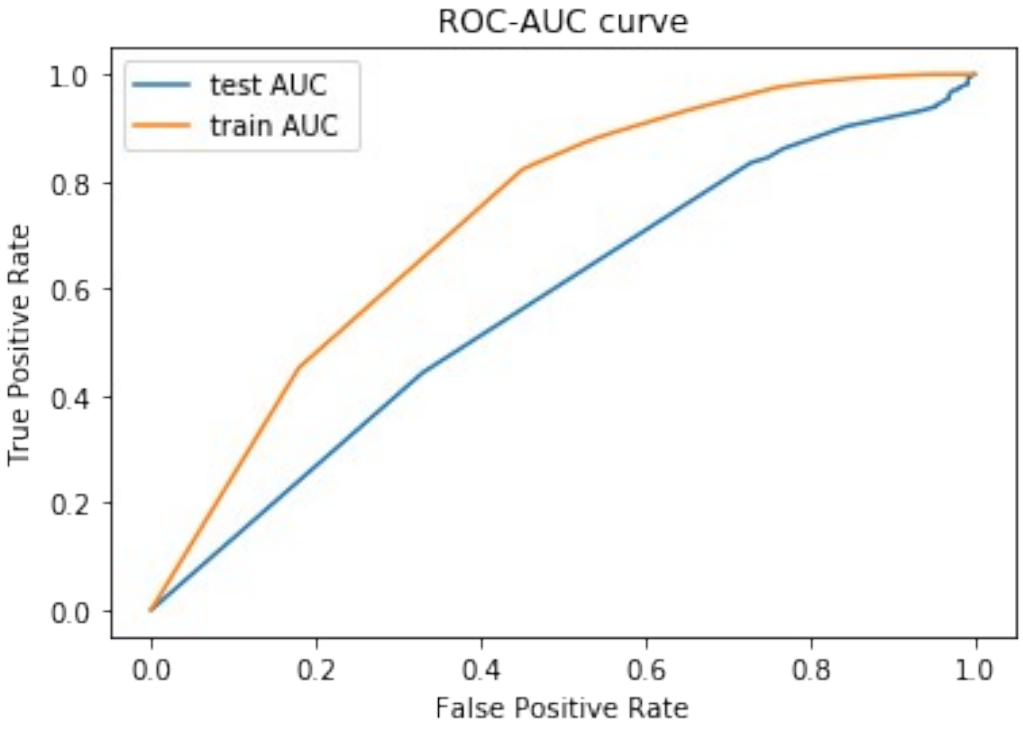
```
#Perform hyperparameter tuning and plot either heatmap or 3d plot.
#On CV data
temp =
np.stack((GridSearch_clf.cv_results_['param_min_samples_split'],GridSe
arch_clf.cv_results_['param_max_depth'],GridSearch_clf.cv_results_['me
an_test_score']),axis =1)
scores = (pd.DataFrame(temp,columns =
['param_min_samples_split','param_max_depth','mean_test_score']).group
by(['param_min_samples_split','param_max_depth'])).max().unstack()
print(scores)
fig = plt.figure(figsize=(10,5))
sns.heatmap(scores.mean_test_score,annot = True)
plt.title("CV Data")
plt.show()
```

	mean_test_score			
param_max_depth	1	3	10	30
param_min_samples_split				
5	0.558818	0.563501	0.552823	0.532085
10	0.558818	0.563501	0.553580	0.534968
100	0.558818	0.563205	0.557313	0.546924
500	0.558818	0.563223	0.575886	0.580103




```
# 10. Find the best parameters and fit the model. Plot ROC-AUC
curve(using predict proba method)
# Re-fit the model with best parameters found
clf = DecisionTreeClassifier(max_depth =
best_max_depth ,min_samples_split =best_min_samples_split )
clf.fit(X_train1_df,y_train1)
```

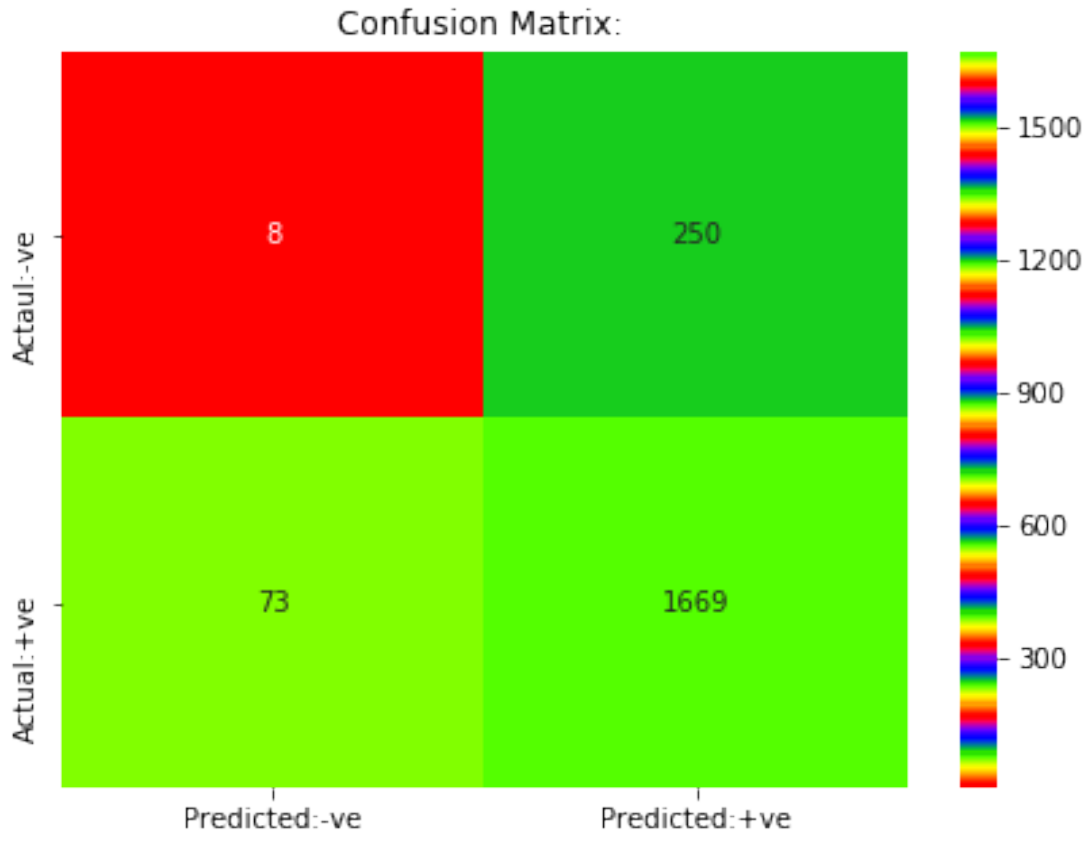
```
DecisionTreeClassifier(max_depth=30, min_samples_split=500)

#create ROC curve

y_test_pred_proba = clf.predict_proba(X_test1_df)[::,1]
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test1,
y_test_pred_proba)

y_train_pred_proba = clf.predict_proba(X_train1_df)[::,1]
fpr_train, tpr_train, threshold_train = metrics.roc_curve(y_train1,
y_train_pred_proba)

#roc_auc = auc(false_positive_rate, true_positive_rate)
plt.plot(fpr_test,tpr_test,label="test AUC ")
plt.plot(fpr_train,tpr_train,label="train AUC ")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title("ROC-AUC curve")
plt.legend()
plt.show()
#print(threshold_test)
#print(y_pred_proba)
```




```
# 11. Plot confusion matrix based on best threshold value
y_pred = clf.predict(X_test1_df)
cm = confusion_matrix(y_test1,  y_pred)
```

```
cm_df =pd.DataFrame(cm,columns=["Predicted:-ve","Predicted:+ve"],index
= ["Actaul:-ve","Actual:+ve"])
print(cm_df)
print("***************confusion matrix***********************")

fig = plt.figure(figsize=(7,5))
ax =sns.heatmap(cm_df,annot =True, xticklabels=True,
yticklabels=True,cmap="prism",fmt="")
plt.title("Confusion Matrix: ")
plt.show()
```

```
            Predicted:-ve  Predicted:+ve
Actaul:-ve              8            250
Actual:+ve             73           1669
***************confusion matrix***********************
```



 # 12. Find all the false positive data points and plot wordcloud of essay text and pdf of teacher_number_of_previously_posted_projects.

```
indx_list = list()
for i in range(y_test1.shape[0]):
```

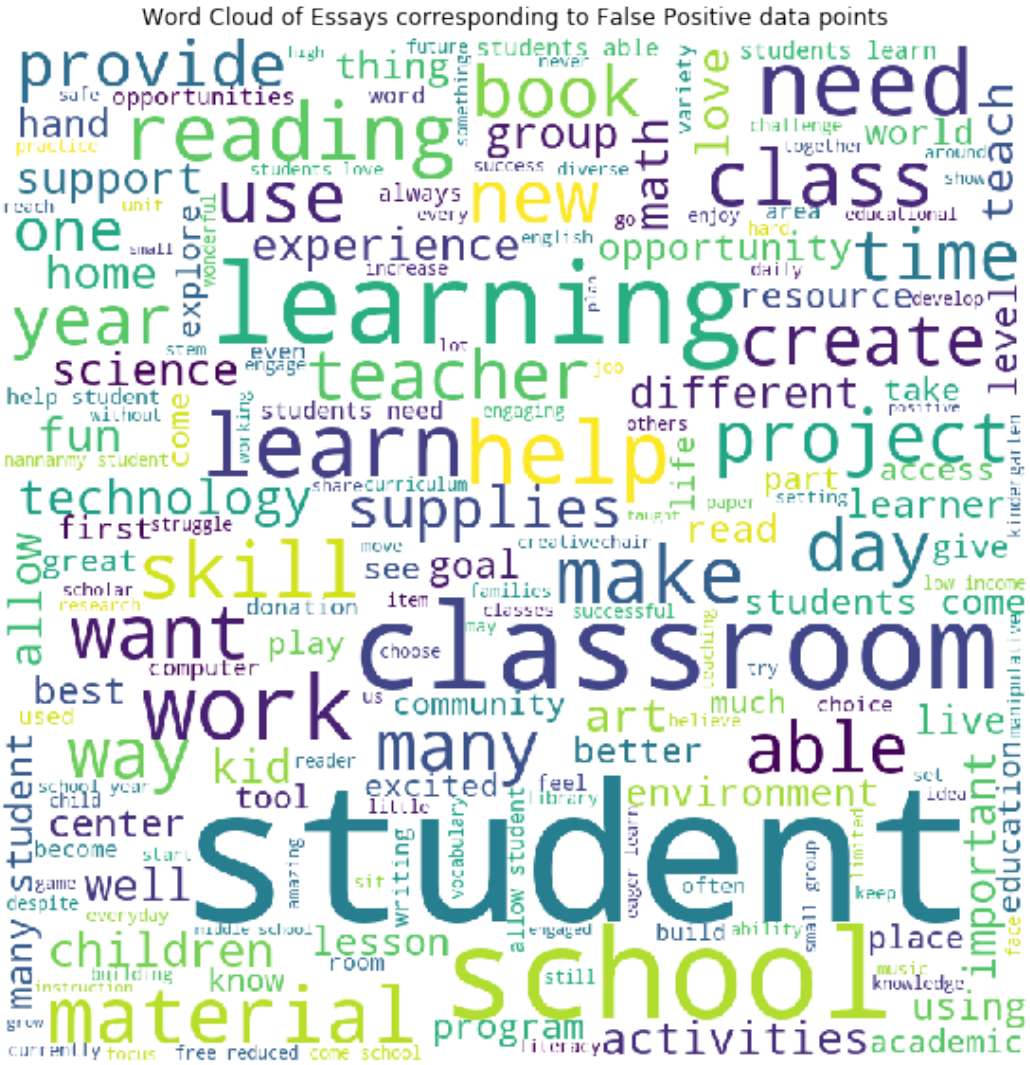
```
    if (y_test1.values[i] == 0) & (y_pred[i] == 1):
        indx_list.append(y_test1.index[i])

#print("actual Y",y_test.values)
#print("pred Y",y_pred)
#print(y_test.index[3])

#print(len(y_test.index))
#print(y_test.index)

#print(len(X_test.index))
#print(X_test.index)
fp_essay = list()
fp_teacher_number_of_previously_posted_projects =list()
print("List of indices for false positive data points: ")
print(indx_list)

for i,ind in enumerate(indx_list):
    essay=data["essay"][data.index==ind].values[0]
    number_of_previously_posted_projects =
data["teacher_number_of_previously_posted_projects"]
[data.index==ind].values[0]
    fp_essay.append(essay)

fp_teacher_number_of_previously_posted_projects.append(number_of_previ
ously_posted_projects)
print(" "*70)
print("List (sample (2)) of essays corresponding to false positive
data poits: ")
print("-"*70)
print(fp_essay[0:2])

#word_count_list = [ len(fp_essay[i]) for i in range(len(fp_essay)) ]
#print()
#print("Word count of essays corresponding to false positive: ")
#print(word_count_list)

#essay_labels = ["essay "+str(i) for i in range(len(fp_essay))]
#fig = plt.figure(figsize = (20, 5))
#plt.bar(essay_labels,word_count_list,width =0.5)
#plt.xlabel("Essays corresponding to false positives")
#plt.ylabel("Word count")
#plt.title("False Positve Data points: Analysis")
#plt.show()



List of indices for false positive data points:
[5795, 9617, 190, 302, 4124, 8242, 2492, 1693, 4540, 2847, 5876, 1726,
7825, 9806, 286, 8117, 1124, 1783, 3448, 3226, 7620, 9867, 6793, 7529,
```

6825, 2020, 7582, 4890, 9222, 3184, 2900, 7153, 1379, 8547, 1704,
5098, 964, 9382, 7618, 4413, 1694, 4512, 8181, 6441, 213, 1878, 7009,
6975, 2424, 9906, 8983, 4464, 5639, 5759, 9989, 4109, 5349, 474, 6853,
1034, 1371, 6940, 5466, 2155, 6472, 6732, 7694, 7748, 847, 4057, 16,
9795, 9634, 8698, 2149, 6422, 9975, 668, 6327, 8225, 9968, 4881, 6405,
8282, 268, 3807, 1282, 4175, 3346, 9350, 9547, 3608, 1329, 5338, 3639,
7709, 2227, 5305, 2792, 4879, 2703, 8655, 2318, 2055, 4004, 2333,
7587, 1293, 8054, 8598, 7889, 3043, 4340, 2889, 2691, 2154, 2081,
4286, 7635, 7897, 8069, 9586, 8298, 1365, 5152, 8468, 6626, 8853,
6703, 640, 5388, 4290, 8209, 9010, 7458, 3265, 2470, 8294, 233, 2361,
1361, 7212, 6258, 1894, 8200, 7824, 5568, 866, 1420, 7771, 7624, 1103,
7453, 4782, 4575, 4079, 3416, 2175, 7266, 7539, 9664, 5016, 5187,
7563, 5242, 5522, 4919, 5228, 5627, 6088, 1551, 3472, 870, 7126, 2295,
226, 3056, 3888, 1259, 9168, 3110, 6423, 5741, 9738, 1465, 6796, 7303,
652, 7994, 1797, 7967, 7796, 2917, 8683, 9627, 9912, 7295, 3590, 2072,
2953, 2973, 1312, 426, 6981, 4343, 5138, 9144, 1626, 4674, 6779, 3119,
4967, 8890, 8965, 6590, 5108, 7875, 8868, 4937, 5205, 4283, 2575,
9230, 1150, 326, 9265, 3023, 4045, 7931, 7729, 3095, 737, 3228, 6676,
6251, 188, 9893, 3148, 6977, 9402, 5122, 3070, 4099, 5756, 2412, 8699,
1346, 5797, 5916, 794]

List (sample (2)) of essays corresponding to false positive data poits:

['other students parents work many jobs struggle give things necessary there students struggle reading grade level distaste reading process then students grade level struggle think critically students intensive reading well language arts class true dislike reading kind never learned love see escape new worlds rather mandatory thing pass state test students come walks life types backgroundsthese donations help students see words seeing traditional settings typically inside textbooks i want able sit together work collaboratively use vocabulary learning well tapping background knowledge this allow challenge discussion priceless it understanding experience students ability learn discussion learning growth deeper knowledge concrete long lasting they able use knowledge speaking writing listening cross curricular setting nannan', 'my classroom ppcd prek class means i could 3 4 5 year olds in order qualify class disability homeless poverty spanish speaking i try give students experiences i part school exposure majority get i already working new centers next year order make sure students every chance learn vocabulary time investigate explore new materials new prek guidelines to insure ready next school year experience donations the explorers project would give students every opportunity learn explore investigate problem solve create it important able expose students real life situations never even experience eating going movies take granted i would like create positive fun learning experience encourage learning well establishing self confidence positive outcomes i incorporated variety new learning centers new learning experiences storage new centers well organized easy access nannan']

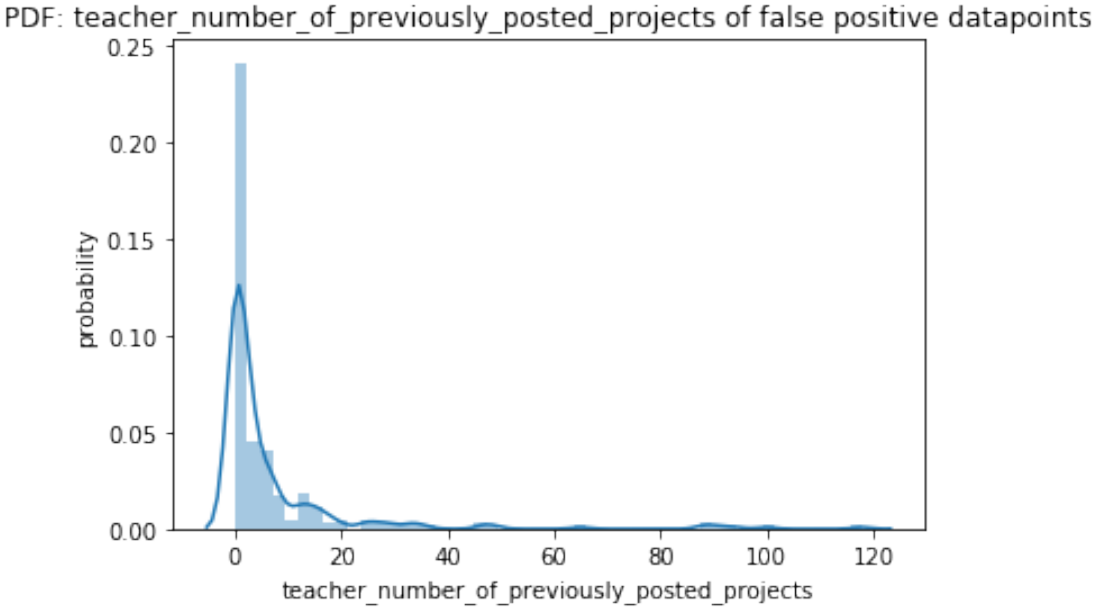
```
#plot wordcloud of essay text
#Ref: https://www.geeksforgeeks.org/generating-word-cloud-python/?
msclkid=7d386f85ba3811ec95b39b1d46408e15
from wordcloud import WordCloud
#fp_essays_text is to store all words of essays corresponding to false
positive data points in a str so we can find the word cloud of it.

fp_essays_text = str()
for i in fp_essay :
    fp_essays_text = fp_essays_text + i

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                min_font_size = 10).generate(fp_essays_text)
# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.title("Word Cloud of Essays corresponding to False Positive data
points")
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```




```
print()
print("List of metric:teacher_number_of_previously_posted_projects
corresponding to false positive data poits: ")
print("-"*70)
print(fp_teacher_number_of_previously_posted_projects)
sns.distplot(fp_teacher_number_of_previously_posted_projects)
plt.title('PDF: teacher_number_of_previously_posted_projects of false
positive datapoints')
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.ylabel('probability')
plt.show()
```

```
List of metric:teacher_number_of_previously_posted_projects
corresponding to false positive data poits:
----------------------------------------------------------------------
[14, 0, 0, 0, 89, 0, 0, 2, 1, 33, 1, 47, 1, 6, 2, 20, 0, 8, 12, 5, 8,
```

```
11, 2, 1, 0, 16, 2, 0, 0, 6, 4, 0, 2, 1, 6, 0, 3, 0, 8, 0, 0, 2, 0, 0,
100, 0, 0, 0, 18, 0, 6, 0, 36, 25, 0, 0, 0, 3, 29, 1, 2, 2, 13, 1, 0,
3, 6, 7, 2, 1, 2, 0, 6, 13, 118, 9, 1, 2, 0, 0, 2, 1, 47, 0, 0, 2, 4,
3, 6, 7, 0, 0, 1, 19, 2, 7, 15, 4, 2, 6, 0, 1, 14, 0, 0, 1, 4, 0, 3,
4, 0, 6, 3, 2, 0, 0, 0, 3, 3, 3, 0, 1, 0, 6, 0, 91, 2, 0, 0, 3, 0, 5,
0, 26, 3, 0, 17, 5, 1, 3, 33, 0, 3, 13, 8, 1, 2, 1, 0, 3, 0, 0, 9, 0,
1, 7, 0, 4, 16, 0, 3, 4, 12, 65, 24, 6, 8, 3, 0, 0, 88, 1, 2, 0, 0,
11, 2, 0, 7, 2, 0, 1, 0, 0, 0, 0, 0, 3, 12, 19, 0, 0, 1, 28, 14, 1, 1,
0, 1, 5, 2, 6, 5, 8, 13, 0, 5, 29, 0, 0, 49, 1, 12, 0, 0, 0, 1, 0, 34,
16, 3, 8, 3, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 16, 2, 15, 8, 11, 1, 0, 0,
3, 2, 94, 16, 0, 7, 25, 1, 1]
```



13. Write your observations about the wordcloud and pdf.

- {student,classroom,learning,school,...} These words found to be more frequent in the essays corresponded to False Positive data points.
- By analysis of PDF of "teacher_number_of_previously_posted_projects", we found that requests with teacher_number_of_previously_posted_projects = 0 are more likely to be classified as False Positive.

Applying DecisionTree Classifier on data set 2 (i.e., TFIDF W2V)

```
#The hyper paramter tuning (best depth in range [1, 3, 10, 30], and
the best min_samples_split in range [5, 10, 100, 500])
#Find the best hyper parameter which will give the maximum AUC value
#find the best hyper paramter using k-fold cross validation(use
gridsearch cv or randomsearch cv)/simple cross validation data(you can
write your own for loops refer sample solution)
```

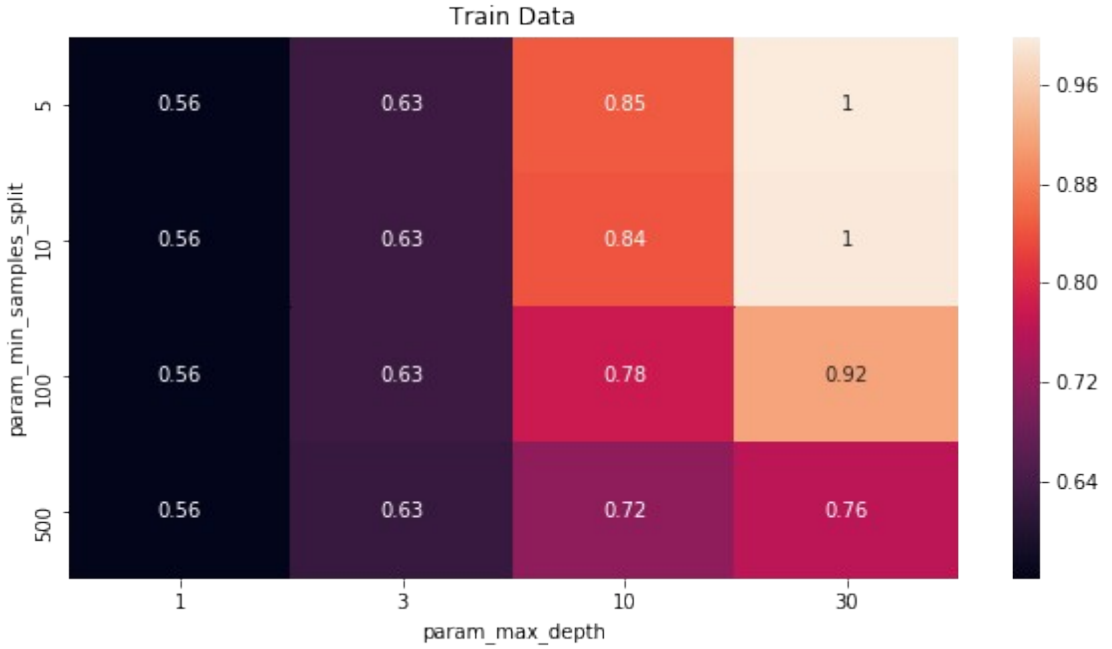
```
clf = DecisionTreeClassifier(random_state =20)
clf.fit(X_train2_df,y_train2)

parameters = {'max_depth':[1, 3, 10, 30],'min_samples_split': [5, 10,
100, 500]}
GridSearch_clf = GridSearchCV(clf, parameters,scoring='roc_auc',cv
=3,return_train_score=True)
GridSearch_clf.fit(X_train2_df, y_train2)

print(GridSearch_clf.score(X_train2_df,y_train2))
print("------------Best Hyperparameters----------------")
# print best parameter after tuning
print(GridSearch_clf.best_params_)

# print how our model looks after hyper-parameter tuning
print(GridSearch_clf.best_estimator_)

#print(GridSearch_clf.cv_results_)

best_max_depth = GridSearch_clf.best_params_['max_depth']

best_min_samples_split =
GridSearch_clf.best_params_['min_samples_split']
```

```
0.7462765883006035
------------Best Hyperparameters----------------
{'max_depth': 10, 'min_samples_split': 500}
DecisionTreeClassifier(max_depth=10, min_samples_split=500,
random_state=20)
```

```
#Perform hyperparameter tuning and plot either heatmap or 3d plot.
#On Train data
temp =
np.stack((GridSearch_clf.cv_results_['param_min_samples_split'],GridSe
arch_clf.cv_results_['param_max_depth'],GridSearch_clf.cv_results_['me
an_train_score']),axis =1)
#print(temp)
scores = (pd.DataFrame(temp,columns =
['param_min_samples_split','param_max_depth','mean_test_score']).group
by(['param_min_samples_split','param_max_depth'])).max().unstack()
print(scores)
fig = plt.figure(figsize=(10,5))
sns.heatmap(scores.mean_test_score,annot = True)
plt.title("Train Data")
plt.show()
```

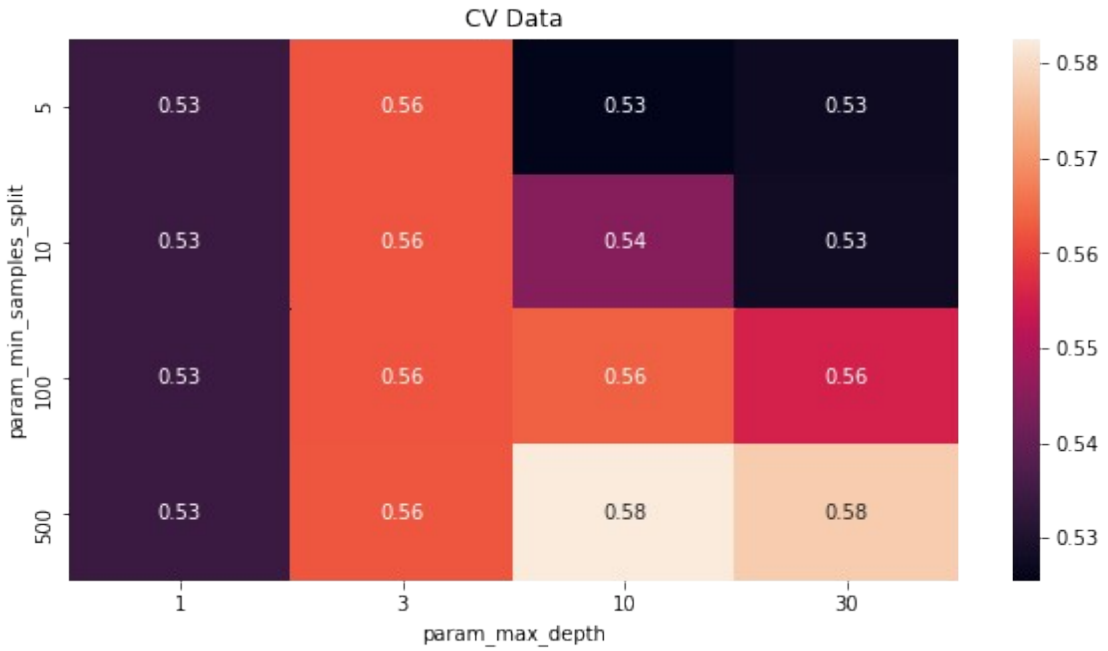
```
                        mean_test_score
param_max_depth                       1         3         10        30
param_min_samples_split
5                              0.560933  0.630545  0.847440  0.998471
```

```
10                              0.560933  0.630545  0.841556  0.995619
100                             0.560933  0.630545  0.778395  0.919273
500                             0.560933  0.625622  0.718436  0.763123
```




```
#Perform hyperparameter tuning and plot either heatmap or 3d plot.
#On CV data
temp =
np.stack((GridSearch_clf.cv_results_['param_min_samples_split'],GridSe
arch_clf.cv_results_['param_max_depth'],GridSearch_clf.cv_results_['me
an_test_score']),axis =1)
scores = (pd.DataFrame(temp,columns =
['param_min_samples_split','param_max_depth','mean_test_score']).group
by(['param_min_samples_split','param_max_depth'])).max().unstack()
print(scores)
fig = plt.figure(figsize=(10,5))
sns.heatmap(scores.mean_test_score,annot = True)
plt.title("CV Data")
plt.show()
```

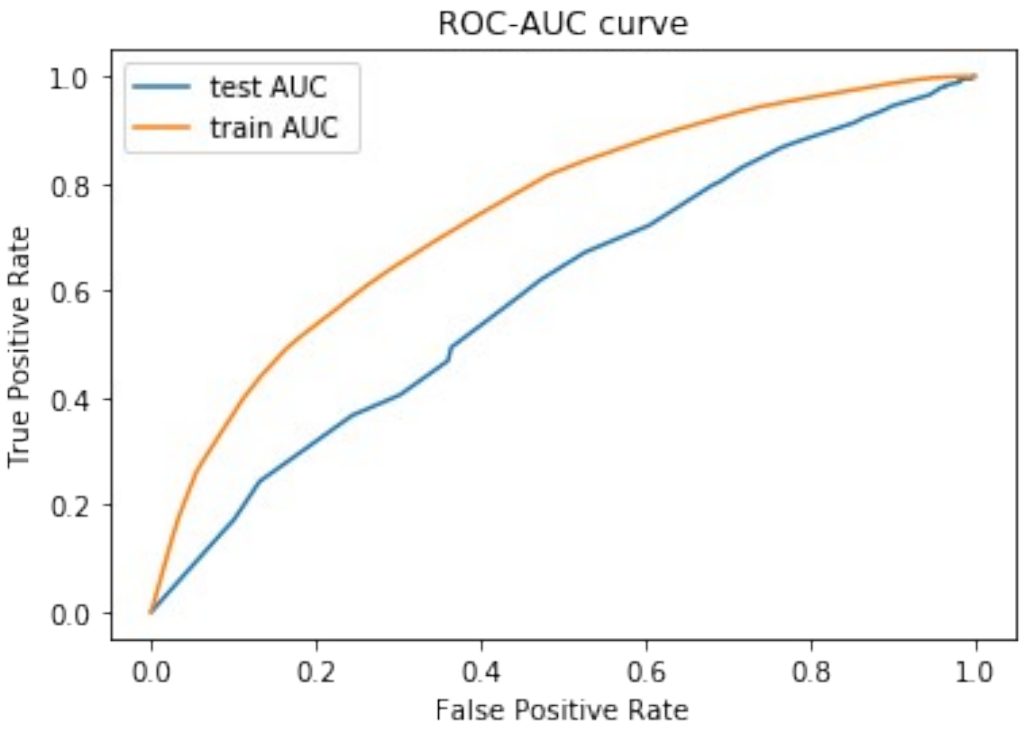
```
                        mean_test_score
param_max_depth                       1         3         10        30
param_min_samples_split
5                               0.534309  0.562042  0.525449  0.527363
10                              0.534309  0.562042  0.544852  0.528118
100                             0.534309  0.562042  0.563491  0.555428
500                             0.534309  0.562379  0.582567  0.577762
```




```
# 10. Find the best parameters and fit the model. Plot ROC-AUC
curve(using predict proba method)
# Re-fit the model with best parameters found
clf = DecisionTreeClassifier(max_depth =
best_max_depth ,min_samples_split =best_min_samples_split )
clf.fit(X_train2_df,y_train2)
```

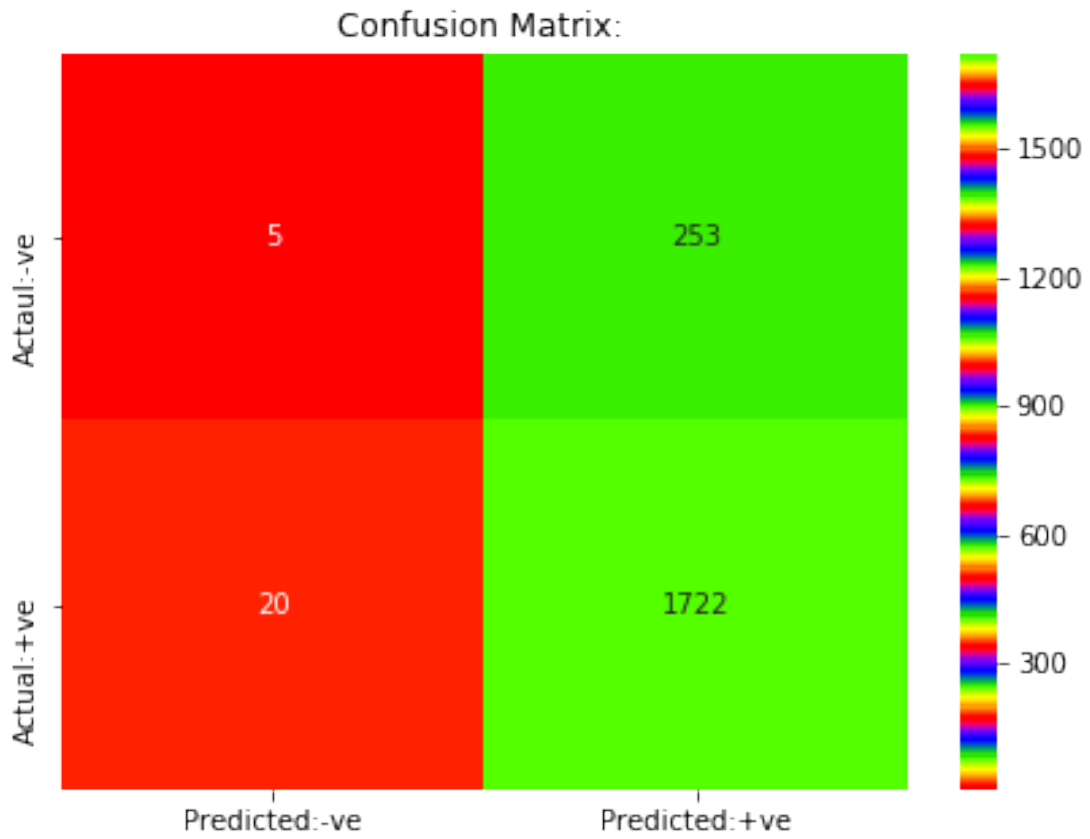
```
DecisionTreeClassifier(max_depth=10, min_samples_split=500)
```

```
#create ROC curve

y_test_pred_proba = clf.predict_proba(X_test2_df)[::,1]
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test2,
y_test_pred_proba)

y_train_pred_proba = clf.predict_proba(X_train2_df)[::,1]
fpr_train, tpr_train, threshold_train = metrics.roc_curve(y_train2,
y_train_pred_proba)

#roc_auc = auc(false_positive_rate, true_positive_rate)
plt.plot(fpr_test,tpr_test,label="test AUC ")
plt.plot(fpr_train,tpr_train,label="train AUC ")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title("ROC-AUC curve")
plt.legend()
plt.show()
#print(threshold_test)
#print(y_pred_proba)
```




```
# 11. Plot confusion matrix based on best threshold value
y_pred = clf.predict(X_test2_df)
cm = confusion_matrix(y_test2,  y_pred)

cm_df =pd.DataFrame(cm,columns=["Predicted:-ve","Predicted:+ve"],index
= ["Actaul:-ve","Actual:+ve"])
print(cm_df)
print("***************confusion matrix***********************")

fig = plt.figure(figsize=(7,5))
ax =sns.heatmap(cm_df,annot =True, xticklabels=True,
yticklabels=True,cmap="prism",fmt="")
plt.title("Confusion Matrix: ")
plt.show()
```

```
            Predicted:-ve  Predicted:+ve
Actaul:-ve              5            253
Actual:+ve             20           1722
***************confusion matrix***********************
```




```
# 12. Find all the false positive data points and plot wordcloud of
essay text and pdf of teacher_number_of_previously_posted_projects.
```

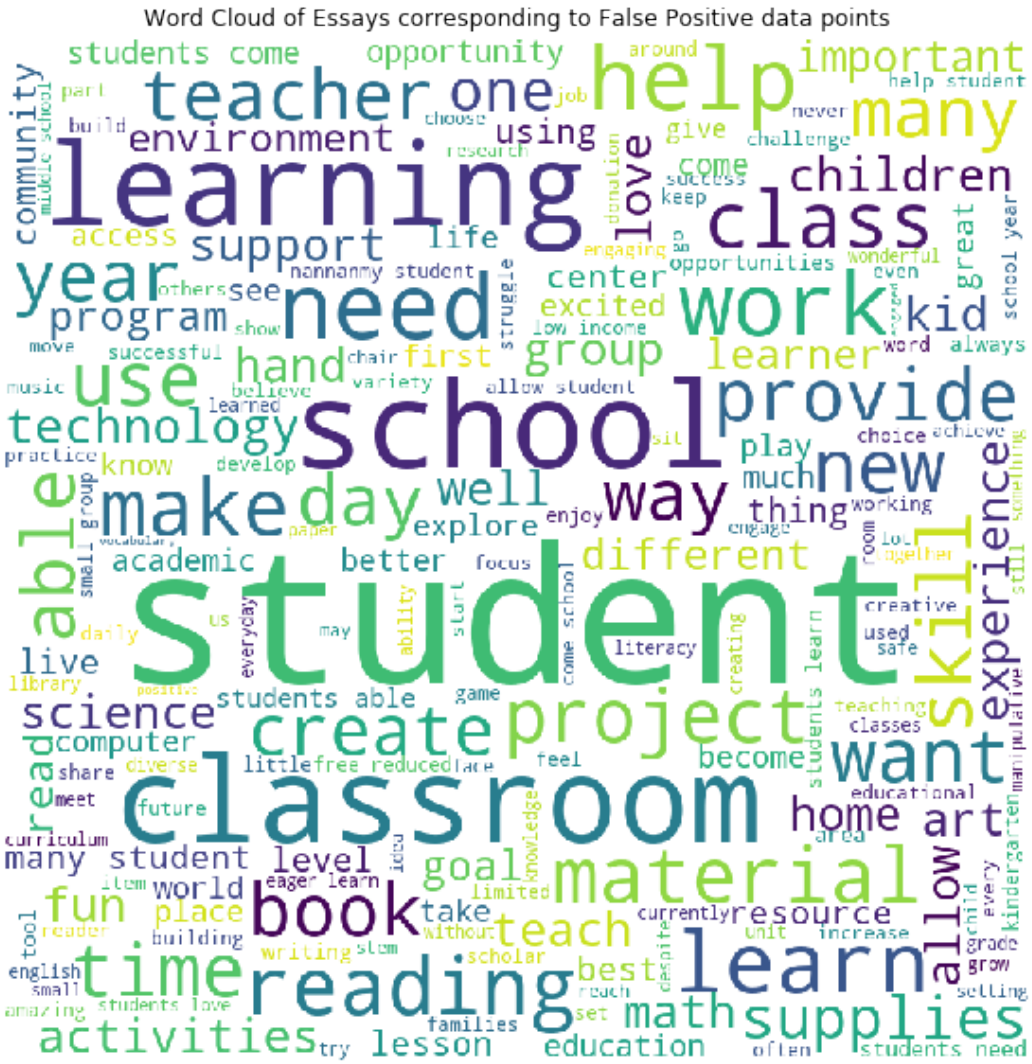
```
indx_list = list()
for i in range(y_test2.shape[0]):
    if (y_test2.values[i] == 0) & (y_pred[i] == 1):
        indx_list.append(y_test2.index[i])

#print("actual Y",y_test.values)
#print("pred Y",y_pred)
#print(y_test.index[3])

#print(len(y_test.index))
#print(y_test.index)

#print(len(X_test.index))
#print(X_test.index)
fp_essay = list()
fp_teacher_number_of_previously_posted_projects =list()
print("List of indices for false positive data points: ")
print(indx_list)

for i,ind in enumerate(indx_list):
    essay=data["essay"][data.index==ind].values[0]
```

```
    number_of_previously_posted_projects =
data["teacher_number_of_previously_posted_projects"]
[data.index==ind].values[0]
    fp_essay.append(essay)

fp_teacher_number_of_previously_posted_projects.append(number_of_previ
ously_posted_projects)
print(" "*70)
print("List (sample (2)) of essays corresponding to false positive
data poits: ")
print("-"*70)
print(fp_essay[0:2])

#word_count_list = [ len(fp_essay[i]) for i in range(len(fp_essay)) ]
#print()
#print("Word count of essays corresponding to false positive: ")
#print(word_count_list)

#essay_labels = ["essay "+str(i) for i in range(len(fp_essay))]
#fig = plt.figure(figsize = (20, 5))
#plt.bar(essay_labels,word_count_list,width =0.5)
#plt.xlabel("Essays corresponding to false positives")
#plt.ylabel("Word count")
#plt.title("False Positve Data points: Analysis")
#plt.show()
```

```
List of indices for false positive data points:
[5795, 9617, 190, 302, 4124, 8242, 3083, 2492, 1693, 4540, 2847, 5876,
1726, 7825, 9806, 286, 8117, 1124, 5256, 1783, 3448, 3226, 7620, 9867,
6793, 7529, 6825, 2020, 7582, 4890, 9222, 3184, 2900, 7153, 1379,
8547, 1704, 5098, 964, 9382, 7618, 4413, 1694, 4512, 8181, 6441, 213,
1878, 7009, 6975, 2424, 9906, 8983, 6313, 4464, 7775, 5639, 5759,
9989, 4109, 5349, 474, 6853, 1034, 1371, 6940, 5466, 2155, 6472, 6732,
7694, 7748, 847, 4057, 16, 9795, 9634, 8698, 2149, 6422, 9975, 6327,
8225, 9968, 4881, 6405, 8282, 268, 3807, 9367, 1282, 4175, 3346, 9350,
9547, 3608, 5338, 3639, 7709, 2227, 5305, 2792, 4879, 2703, 8655,
2318, 2055, 4004, 2333, 7587, 1293, 8054, 8598, 7889, 3043, 4340,
2889, 2691, 2154, 2081, 1467, 4286, 7897, 8069, 9586, 8298, 1365,
5152, 8468, 6626, 8853, 6703, 640, 5388, 4290, 8209, 9010, 7458, 3265,
8294, 233, 2361, 7212, 6258, 1894, 8200, 7824, 5568, 866, 1420, 7771,
7624, 1103, 3849, 7453, 4782, 4575, 4079, 3416, 2175, 7266, 7539,
9664, 5016, 5187, 7563, 5242, 5522, 4919, 5228, 5627, 6088, 1551,
3472, 870, 7126, 2295, 226, 3056, 3888, 1259, 9168, 3110, 6423, 5741,
9738, 1465, 6796, 7303, 652, 7994, 1797, 7967, 7796, 2917, 8683, 9627,
9912, 7295, 3590, 2072, 2953, 2973, 1312, 426, 6981, 4343, 5138, 9144,
1626, 4674, 6779, 3119, 4967, 8890, 8965, 6590, 5108, 7875, 8868,
4937, 5205, 4283, 2575, 9230, 1150, 326, 9265, 3023, 4045, 7931, 7729,
3095, 6823, 737, 3228, 6676, 6251, 188, 9893, 3148, 6977, 9402, 5122,
```

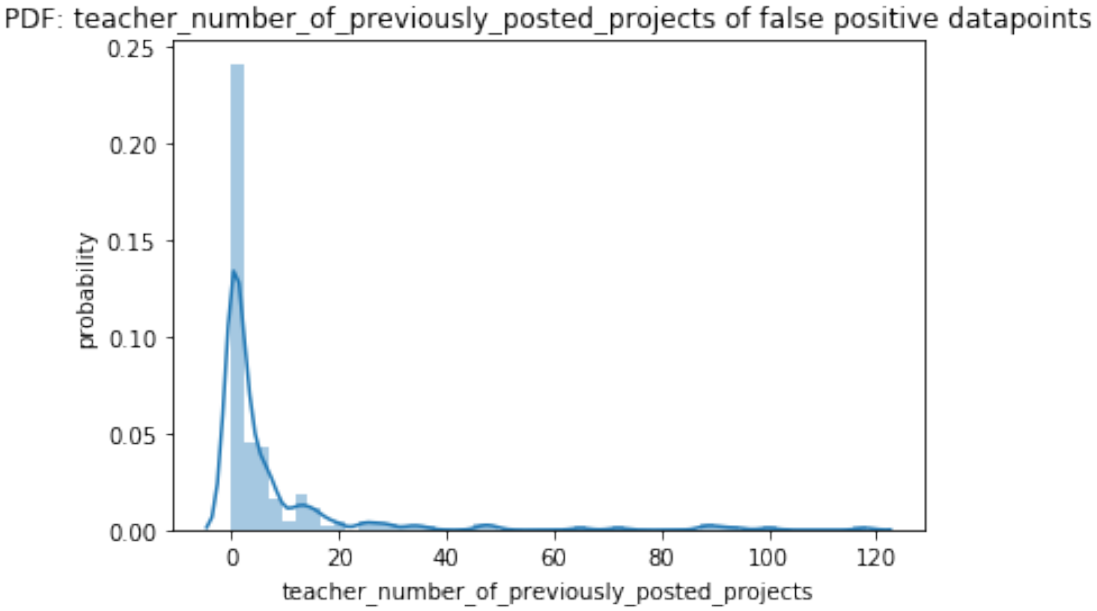
```
3070, 4099, 5756, 2412, 8699, 1346, 5797, 5916, 794]

List (sample (2)) of essays corresponding to false positive data
poits:
--------------------------------------------------------------------------
['other students parents work many jobs struggle give things necessary
there students struggle reading grade level distaste reading process
then students grade level struggle think critically students intensive
reading well language arts class true dislike reading kind never
learned love see escape new worlds rather mandatory thing pass state
test students come walks life types backgroundsthese donations help
students see words seeing traditional settings typically inside
textbooks i want able sit together work collaboratively use vocabulary
learning well tapping background knowledge this allow challenge
discussion priceless it understanding experience students ability
learn discussion learning growth deeper knowledge concrete long
lasting they able use knowledge speaking writing listening cross
curricular setting nannan', 'my classroom ppcd prek class means i
could 3 4 5 year olds in order qualify class disability homeless
poverty spanish speaking i try give students experiences i part school
exposure majority get i already working new centers next year order
make sure students every chance learn vocabulary time investigate
explore new materials new prek guidelines to insure ready next school
year experience donations the explorers project would give students
every opportunity learn explore investigate problem solve create it
important able expose students real life situations never even
experience eating going movies take granted i would like create
positive fun learning experience encourage learning well establishing
self confidence positive outcomes i incorporated variety new learning
centers new learning experiences storage new centers well organized
easy access nannan']

#plot wordcloud of essay text
#Ref: https://www.geeksforgeeks.org/generating-word-cloud-python/?
msclkid=7d386f85ba3811ec95b39b1d46408e15
from wordcloud import WordCloud
#fp_essays_text is to store all words of essays corresponding to false
positive data points in a str so we can find the word cloud of it.

fp_essays_text = str()
for i in fp_essay :
    fp_essays_text = fp_essays_text + i

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                min_font_size = 10).generate(fp_essays_text)
# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.title("Word Cloud of Essays corresponding to False Positive data
```

```
points")
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```




```
print()
print("List of metric:teacher_number_of_previously_posted_projects
corresponding to false positive data poits: ")
print("-"*70)
print(fp_teacher_number_of_previously_posted_projects)
sns.distplot(fp_teacher_number_of_previously_posted_projects)
plt.title('PDF: teacher_number_of_previously_posted_projects of false
positive datapoints')
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.ylabel('probability')
plt.show()
```

```
List of metric:teacher_number_of_previously_posted_projects
corresponding to false positive data poits:
------------------------------------------------------------------------
[14, 0, 0, 0, 89, 0, 0, 0, 2, 1, 33, 1, 47, 1, 6, 2, 20, 0, 72, 8, 12,
5, 8, 11, 2, 1, 0, 16, 2, 0, 0, 6, 4, 0, 2, 1, 6, 0, 3, 0, 8, 0, 0, 2,
0, 0, 100, 0, 0, 0, 18, 0, 6, 0, 0, 5, 36, 25, 0, 0, 0, 3, 29, 1, 2,
2, 13, 1, 0, 3, 6, 7, 2, 1, 2, 0, 6, 13, 118, 9, 1, 0, 0, 2, 1, 47, 0,
0, 2, 3, 4, 3, 6, 7, 0, 0, 19, 2, 7, 15, 4, 2, 6, 0, 1, 14, 0, 0, 1,
4, 0, 3, 4, 0, 6, 3, 2, 0, 0, 0, 6, 3, 3, 0, 1, 0, 6, 0, 91, 2, 0, 0,
3, 0, 5, 0, 26, 3, 0, 5, 1, 3, 0, 3, 13, 8, 1, 2, 1, 0, 3, 0, 0, 0, 9,
0, 1, 7, 0, 4, 16, 0, 3, 4, 12, 65, 24, 6, 8, 3, 0, 0, 88, 1, 2, 0, 0,
11, 2, 0, 7, 2, 0, 1, 0, 0, 0, 0, 0, 3, 12, 19, 0, 0, 1, 28, 14, 1, 1,
0, 1, 5, 2, 6, 5, 8, 13, 0, 5, 29, 0, 0, 49, 1, 12, 0, 0, 0, 1, 0, 34,
16, 3, 8, 3, 0, 1, 1, 0, 0, 0, 0, 0, 2, 1, 0, 16, 2, 15, 8, 11, 1, 0,
0, 3, 2, 94, 16, 0, 7, 25, 1, 1]
```



13. Write your observations about the wordcloud and pdf.

- {student,classroom,learning,school,...} These words found to be more frequent in the essays corresponded to False Positive data points.
- By analysis of PDF of "teacher_number_of_previously_posted_projects", we found that requests with teacher_number_of_previously_posted_projects = 0 are more likely to be classified as False Positive.

Task - 2

#For this task consider set-1 features.

```
#Select all the features which are having non-zero feature importance.
#You can get the feature importance using 'feature_importances_`
(https://scikit-learn.org/stable/modules/generated/sklearn.tree.Decisi
onTreeClassifier.html), discard the all other remaining features and
then apply any of the model of you choice i.e. (Dession tree, Logistic
Regression, Linear SVM).
#You need to do hyperparameter tuning corresponding to the model you
selected and procedure in step 2 and step 3
#Note: when you want to find the feature importance make sure you
don't use max_depth parameter keep it None.

#Dataset1

clf = DecisionTreeClassifier()
clf.fit(X_train1_df,y_train1)

DecisionTreeClassifier()

print("Feature Importance: ")
print(clf.feature_importances_)

Feature Importance:
[0.         0.         0.         ... 0.         0.01270432
0.00973973]

feature_indicies = []
print("Feature index and its corresponding non-zero feature
importance")
for i,feature_weight in enumerate(clf.feature_importances_):
    if(feature_weight != 0):
        feature_indicies.append(i)
        print(i,feature_weight)

Feature index and its corresponding non-zero feature importance
4 0.0023769057174800974
29 0.0007774925244093778
60 0.005822555127243563
75 0.0011662387866140664
107 0.0024549411618181092
111 0.0010181029705534554
198 0.0030072786825789976
200 0.006686484106591947
206 0.0010022194833447416
211 0.000888562885039289
214 0.0017878126731509646
221 0.0025274677618895326
236 0.0018458986809339441
245 0.0015983750266399348
254 0.0010268564167567867
264 0.0018423415772472465
268 0.0018700180917155557
```

279 0.0010252648673530 25
283 0.0010076632149232066
289 0.0013181856393747029
296 0.0009774003216570164
299 0.0010225820809396647
300 0.0028288549330854995
331 0.001245862305974241
338 0.001009006966984331
358 0.0009905328140189787
370 0.003288202302063519
376 0.0019248446978412067
412 0.0010074724266846892
414 0.0005183283496062518
439 0.0007774925244093778
446 0.0010029765006714369
451 0.0010039658903860183
453 0.0011995329346847733
466 0.0010136000529756173
467 0.0038030777651303148
469 0.0010048408061947146
476 0.004966838690732435
482 0.0009885835862051351
485 0.0009984219845347584
496 0.0009961428089879684
518 0.002022265048134764
532 0.0020187072741696056
546 0.0009909781107240222
557 0.0018048596609279313
583 0.0011530127445666088
595 0.0010184384616588344
597 0.0006911044661416691
611 0.0018870742078738456
613 0.0018746141410252702
624 0.0010218389216975496
664 0.0007774925244093778
668 0.0017849950014832403
686 0.0005183283496062518
705 0.0019432099452536117
720 0.0031828937217506
739 0.0018105962006421297
740 0.0010204766969527973
773 0.001000477629177754
805 0.0018654389241234048
808 0.0009918361004302384
840 0.0007774925244093778
846 0.0018147262013973606
877 0.0018368315446103242
878 0.0013096910775181744
886 0.0019169297926265457
888 0.001027315188297393

891 0.002228450544404718
908 0.002755860942466893
910 0.0018478899404528913
919 0.0010109973659554608
929 0.0006911044661416691
938 0.000994448604663955
946 0.0010063363127516469
947 0.0024122203962444796
949 0.0029330937295640215
953 0.000989265467869179
967 0.001701643845412775
985 0.0020128791550654323
992 0.0009952792261452082
997 0.0018257286907277178
999 0.0010243820051334667
1001 0.0008293253593700027
1009 0.0007774925244093778
1014 0.002163457459226094
1021 0.0022183496814365644
1022 0.02083876355254008
1032 0.0010170830287829049
1050 0.0018524085790146698
1080 0.0018738834053752857
1087 0.002207547048573149
1091 0.0019484624241950673
1094 0.0030068608085400714
1098 0.0009926952048914463
1102 0.00420531582691125
1103 0.0027618408201519145
1105 0.0007774925244093778
1122 0.0006911044661416691
1142 0.0010225210571590477
1166 0.002969473556202838
1180 0.0018344832850097602
1197 0.002648868107046308
1204 0.0010207868965059598
1220 0.0013736874081036895
1260 0.0009990464455773907
1261 0.000979873669746642
1265 0.00034325610909270097
1305 0.002400528676346928
1318 0.0010083519312039432
1333 0.0009070746118109407
1335 0.0018024173039039623
1342 0.003512315731233108
1362 0.001784650862262177
1363 0.0007774925244093778
1368 0.0012085369069494715
1369 0.0006911044661416691
1383 0.0009944167658107596

1384 0.0018792271551366225
1397 0.0017771257700785776
1402 0.0005105804095106704
1412 0.001013653224919207
1414 0.00101017760406757495
1415 0.00197880653942205 1
1419 0.0009997058821750624
1448 0.0009996332456691998
1454 0.002415190850385731
1477 0.0006911044661416691
1482 0.0005183283496062518
1501 0.0009790822320654522
1531 0.0027358070984970753
1532 0.0010150549474193173
1548 0.0010103462825756379
1551 0.00187856401271348
1560 0.004927951629003218
1564 0.0019759677134550823
1571 0.0024378541512301496
1580 0.0005138144323144525
1587 0.0018563309042902193
1598 0.0009807684875589712
1606 0.0019869253401573
1607 0.0018677264028776975
1617 0.0018020016092082723
1632 0.0010154296158573692
1642 0.0018429452430444508
1643 0.002622039951993131
1672 0.0011626185061525457
1698 0.0017934680684405025
1699 0.0010266282506923951
1703 0.0007774925244093778
1711 0.0006911044661416691
1716 0.0009626097921258965
1719 0.0011203387265862447
1725 0.0010030921167652556
1729 0.0010140007194195803
1731 0.0005183283496062518
1733 0.001776581838278384
1751 0.002395670945467627
1766 0.005479330632290528
1788 0.0009924878125713327
1789 0.001981719789222724
1812 0.000974070228814401
1830 0.0018221516335465353
1838 0.0009805105717094916
1845 0.0010127667769743974
1853 0.0006911044661416691
1854 0.0017345366993359531
1865 0.001252686150184933

1866 0.0010130329260578782
1871 0.0008638805826770864
1872 0.0012055243513029064
1877 0.002623688441212343
1881 0.001833818042643679
1892 0.001926195381882379
1901 0.0008638805826770864
1908 0.002084586953450214
1911 0.0018222797408855363
1916 0.001853035517103013
1917 0.0017109662636182786
1927 0.0010211574685985395
1930 0.0009782407346590717
1933 0.001009676291672149
1941 0.0011426269528641284
1953 0.0028484868066664935
1960 0.0022282281915801875
1963 0.0009964152132112707
1985 0.0005183283496062518
1988 0.002268240299131632
1996 0.0017856783084262311
2002 0.0018659820585825069
2004 0.000995103493438842
2016 0.0018788795008867567
2027 0.0010013479881419325
2034 0.0009758359019966499
2053 0.0009799248157418203
2107 0.0009918355037887394
2121 0.002279657074494659
2153 0.001841234690116428
2164 0.0018063296148993825
2166 0.0022768662423545117
2177 0.0017821839857329643
2185 0.0009892326890194307
2186 0.001792283366549971
2215 0.002429987768332836
2217 0.0018565118877809782
2219 0.002485490587522485
2227 0.0010118814913301108
2230 0.004232035438952807
2238 0.0003452029552072174
2273 0.0015549850488187555
2276 0.000981613249391664
2282 0.00100923258791252
2283 0.0010333761400377788
2287 0.001252084917376288
2302 0.000888562885039289
2305 0.002479570753521799
2311 0.002107256507293706
2313 0.00499285832718848

2323 0.0006911044661416691
2335 0.001384407384315409
2342 0.0010101143988236313
2379 0.0005183283496062518
2392 0.001819252426492215
2401 0.0008638805826770864
2407 0.0008293253593700027
2409 0.002834180642982321
2413 0.0036026035376051633
2438 0.0015811192513199104
2441 0.0018077245175325098
2454 0.0022994930418895536
2462 0.0026533256884062307
2493 0.0036359364966525947
2519 0.0010233662287097815
2520 0.0007774925244093778
2540 0.0012426217421154537
2546 0.0017917543007768405
2547 0.0039425864306558076
2558 0.00098242500782584
2562 0.0021325509240942933
2563 0.004807347400396371
2579 0.0009867048161887842
2591 0.0006911044661416691
2594 0.0009765609911960336
2607 0.004193793004701752
2647 0.0018769185222680495
2654 0.0018247384270672236
2668 0.0009957590293370089
2671 0.003629596994571225
2692 0.0009792373881429013
2710 0.002410288568551212
2717 0.001023481449524587
2730 0.0009935554260394313
2739 0.0018456453125267008
2746 0.0010216838972937393
2763 0.0012025230458638568
2775 0.003784341539663673
2793 0.0008638805826770864
2812 0.0028442356846365953
2819 0.0006911044661416691
2839 0.0024745967968728155
2854 0.0018761152228849217
2862 0.003093260232599866
2865 0.001014380268158771
2869 0.0010030128792870424
2888 0.00275270473877342
2898 0.003319149097025575
2899 0.0018723140009748276
2904 0.0018411683087596923

2912 0.0018293941750808884
2915 0.0011478022286130773
2921 0.0021586069497099387
2934 0.0010293801429867109
2958 0.0010189964352049974
2966 0.0005183283496062518
3000 0.0009841540937863468
3004 0.0018300468091174398
3008 0.001790637021765691
3016 0.0034203306930537213
3021 0.0005151970472034156
3029 0.0017456482549621192
3058 0.0032580639118107257
3061 0.005337910860861021
3062 0.004385371976062589
3066 0.0018559572524397888
3068 0.0033638980304765724
3084 0.0014136903777250676
3089 0.00451033109421511
3108 0.0034084015716532323
3114 0.0010236984904723463
3119 0.0006911044661416691
3123 0.0010252564408847543
3128 0.0023982886734684734
3133 0.001883857217646323
3139 0.0008293253593700027
3142 0.0012439880390550044
3156 0.0009850032345056713
3160 0.0009857304621209075
3170 0.0005183283496062518
3176 0.003212873028022743
3179 0.000982459103117854
3182 0.0027661117480621217
3187 0.001763838936814635
3197 0.0009502686409447951
3215 0.004455760534768776
3220 0.001007002981392144
3221 0.0034487825408524766
3232 0.0012588634815630753
3235 0.0017657379381600551
3246 0.0018763017180316803
3247 0.0019339317220906892
3248 0.0012115607691020574
3263 0.004579494764759795
3264 0.005206413538108877
3265 0.002478996078325742
3267 0.0010203982002991887
3276 0.008886132172935097
3282 0.001004340274220889
3300 0.0010798507283463577

3306 0.0006911044661416691
3309 0.003101326100220605
3343 0.0010116882665692185
3351 0.0020790817584365236
3361 0.001214595994410787
3374 0.0006911044661416691
3406 0.0009732144576580297
3421 0.0007774925244093778
3432 0.0018163601332865265
3440 0.0015253249031748112
3446 0.0013822089322833384
3448 0.0005131252110772399
3463 0.006266222304319805
3471 0.006349383604565821
3476 0.0012237704211255362
3482 0.0010050049600410832
3490 0.0018278346038479506
3497 0.006543302554555523
3500 0.001843404771934867
3501 0.0017771257700785776
3506 0.0023908279449038504
3511 0.0009875572609544248
3519 0.0024606356816472613
3528 0.002039193745159609
3529 0.0017793796049528749
3537 0.0008293253593700027
3538 0.0015314580953870404
3544 0.0009911838378860547
3546 0.001002350168431733
3553 0.009346428449108775
3558 0.0010197966058519393
3562 0.000993140765079469
3569 0.0018631556451464142
3583 0.0005183283496062518
3606 0.0009502686409447951
3607 0.0009070746118109407
3612 0.0010057168661826244
3613 0.0006911044661416691
3621 0.001005242859842427
3624 0.0018129584936854416
3625 0.0026612300164334216
3627 0.0015549850488187555
3644 0.0018631696421440608
3651 0.0006911044661416691
3665 0.0016125770876638942
3681 0.002946244158634901
3686 0.0009763873419737106
3691 0.0009811480948380884
3699 0.001884052851510137
3728 0.0009879351220619015

3734 0.001023203875895162
3744 0.0018234779757458676
3760 0.0009858534746736188
3803 0.002381185897039073
3804 0.0009773322574386418
3805 0.0010123602614323529
3811 0.0005132950551984813
3816 0.0018250577765027815
3821 0.002622069162993704
3833 0.000997072045916203
3843 0.000514505043110707
3854 0.0018345413328959665
3855 0.001119660274831408
3865 0.0009926606589276174
3879 0.0024936305691966223
3888 0.0018483372803621341
3902 0.001832263032925003
3918 0.000990121233844828
3922 0.0015192284807000436
3930 0.0018608765556723714
3935 0.001008338307628977
3938 0.0023859996151599
3945 0.0029381048491460724
3954 0.0010163195629354057
3979 0.0012207007628785823
4002 0.0009898824313437488
4033 0.0022276225123751482
4034 0.00051238777273233
4042 0.0009833060506206602
4053 0.0009937943621607696
4054 0.0024435019222808817
4061 0.0006911044661416691
4066 0.003851984339927515
4069 0.0006911044661416691
4072 0.0008638805826770864
4090 0.0012386884479832288
4099 0.0027894330411356726
4123 0.0010280028155988659
4128 0.004250858211701779
4172 0.0010259420018618342
4173 0.0016242963978940071
4193 0.0005158904483438599
4205 0.0012176426398818364
4208 0.0013822089322833382
4211 0.002946401129417469
4237 0.0026265347931320716
4248 0.0009779666827924092
4250 0.0008293253593700027
4263 0.0010286911335200778
4274 0.0007774925244093778

4276 0.0012482674455611498
4282 0.0010252837498562458
4287 0.000999608404478251
4301 0.002491791634049662
4308 0.0009817862399402644
4350 0.002907820594161672
4354 0.000888562885039289
4371 0.00125760748027894
4374 0.0023767155184704265
4388 0.0017991503408392095
4404 0.0012363109269699613
4450 0.0005183283496062518
4462 0.0005183283496062518
4472 0.002823118884130551
4485 0.0005172814568311344
4494 0.0007774925244093778
4499 0.0016033296919425938
4533 0.0010164063268344361
4547 0.0010137062613360796
4557 0.0009766984492300104
4567 0.0010245715668428322
4595 0.0010172106804828204
4603 0.0019293175098100584
4628 0.0013422816784621281
4630 0.0007774925244093778
4647 0.002999270383372324
4678 0.0007774925244093778
4688 0.00100567030592649
4709 0.002067617482495269
4712 0.0012818984315304534
4722 0.0017963058343715102
4723 0.0013472668158686178
4730 0.006252733164907097
4737 0.0005165852502944781
4742 0.003506476100412013
4745 0.0018134747319628274
4746 0.0006911044661416691
4749 0.0012410728338311922
4755 0.0009884108108775469
4757 0.0005117509223909575
4760 0.0012765964240867963
4776 0.0010238873788183352
4782 0.0024447728170263446
4783 0.0017277611653541728
4786 0.0012439880390550044
4787 0.001993570575408662
4808 0.001760497692697726
4814 0.0031702562942636414
4818 0.001214345765972977
4823 0.004808169246512674

4843 0.001001688114159368
4876 0.002425040389096207
4881 0.0024435479338580446
4888 0.0036838774197938255
4889 0.001739289048383924
4891 0.001919010614457873
4897 0.00025870323362964134
4910 0.000511482893663594
4920 0.0010036762475932413
4924 0.0010157303000124158
4939 0.0012514710936913497
4940 0.0018389359131142677
4954 0.0013083270005201627
4973 0.0009461109269105213
4980 0.0023025740145970037
4981 0.0018845765471813623
4992 0.00498050184860281
4996 0.0014809381417321482
5004 0.0018279708775026978
5017 0.0009786017260929863
5036 0.0013132424432271148
5037 0.001283449161551432
5041 0.0009977295283070372
5045 0.005383707215730998
5046 0.0023599265691926275
5078 0.000998740312073188
5085 0.0018202149762975167
5091 0.0014809381417321482
5097 0.0008293253593700027
5118 0.0034706588756782024
5120 0.002190500677466421
5123 0.0014809381417321482
5124 0.0015426438976376545
5141 0.0010010267156025775
5143 0.0023268422521487107
5166 0.001760089336847286
5170 0.001933360290521295
5171 0.008528492321401855
5175 0.0005151168644359219
5185 0.0025916417480312593
5187 0.0018930252768228322
5189 0.0028933123933803017
5194 0.0009757227414181513
5198 0.0015549850488187555
5213 0.0017709199484422231
5227 0.0009970075162873547
5229 0.00103198707444127
5237 0.0032329069497877127
5256 0.001009924322561152
5266 0.0007774925244093778

```
5271 0.0030090905151497595
5275 0.0019247197915858876
5290 0.0025278120412816976
5338 0.0018312674960088368
5344 0.000888562885039289
5357 0.0005183283496062518
5384 0.0008638805826770864
5386 0.0009978733499967493
5422 0.001858601645457476
5429 0.0010145408372002976
5436 0.0015549850488187555
5441 0.0039565730686610566
5450 0.0024201081035592677
5459 0.0008293253593700027
5472 0.001011016940579419
5473 0.001019891076500304
5521 0.001969979990266321
5525 0.0009214726215222254
5551 0.00179687161196834
5552 0.0003441718657199969
5553 0.00621683117664234
5575 0.0010276422931323925
5593 0.0010227315849844053
5594 0.0010191171943889747
5596 0.005350645701582943
5607 0.0009983876612414502
5616 0.0010065940723459625
5631 0.0014809381417321482
5633 0.0006911044661416691
5648 0.0007774925244093778
5649 0.0019209976114793119
5650 0.0017878900606432858
5662 0.0010003659718959721
5677 0.0017277611653541726
5683 0.0005183283496062518
5692 0.0005142047495808218
5713 0.0008293253593700027
5715 0.012704318267927398
5716 0.00973972968090309

# Limiting features for xtrain dataset with non zero features
print("Found "+str(len(feature_indicies))+" features with zero feature
importance")
print("Xtrain data before removing zero weighted features")
print(X_train1_df.shape)

#Rename colums with feature indices for mathematical ease
#Ref: https://saugatach.github.io/2022/03/31/3-ways-to-rename-columns-
of-a-pandas-dataframe.html#:~:text=%203%20ways%20to%20rename%20columns
%20of%20a,process...%203%20The%20regular%20expression%20approach
```

```
%20More%20?msclkid=4d3569fbba8611ec95642f37ee146257
dataset_with_limited_features = X_train1_df.copy()
column_list =X_train1_df.columns
column_list_new =[i for i in range(X_train1_df.shape[1])]
mapping = {key1: key2 for key1, key2 in zip(column_list,
column_list_new)}

dataset_with_limited_features=dataset_with_limited_features.rename(col
umns = mapping )
X_train3=dataset_with_limited_features.drop(feature_indicies,axis=1)

#Ytrain doest change
y_train3 = y_train1

print("X train data after removing zero weighted features")
print(X_train3.shape)
```

```
Found 556 features with zero feature importance
Xtrain data before removing zero weighted features
(8000, 5717)
X train data after removing zero weighted features
(8000, 5161)
```

```
# Limiting features for xtrain dataset with non zero features
print("Found "+str(len(feature_indicies))+" features with zero feature
importance")
print("Xtest data before removing zero weighted features")
print(X_test1_df.shape)

#Rename colums with feature indices for mathematical ease
#Ref: https://saugatach.github.io/2022/03/31/3-ways-to-rename-columns-
of-a-pandas-dataframe.html#:~:text=%203%20ways%20to%20rename%20columns
%20of%20a,process...%203%20The%20regular%20expression%20approach
%20More%20?msclkid=4d3569fbba8611ec95642f37ee146257
dataset_with_limited_features = X_test1_df.copy()
column_list =X_test1_df.columns
column_list_new =[i for i in range(X_test1_df.shape[1])]
mapping = {key1: key2 for key1, key2 in zip(column_list,
column_list_new)}

dataset_with_limited_features=dataset_with_limited_features.rename(col
umns = mapping )
X_test3=dataset_with_limited_features.drop(feature_indicies,axis=1)

#Ytest doest change
y_test3 = y_test1

print("X test data after removing zero weighted features")
print(X_test3.shape)
```

```
Found 556 features with zero feature importance
Xtest data before removing zero weighted features
(2000, 5717)
X test data after removing zero weighted features
(2000, 5161)
```

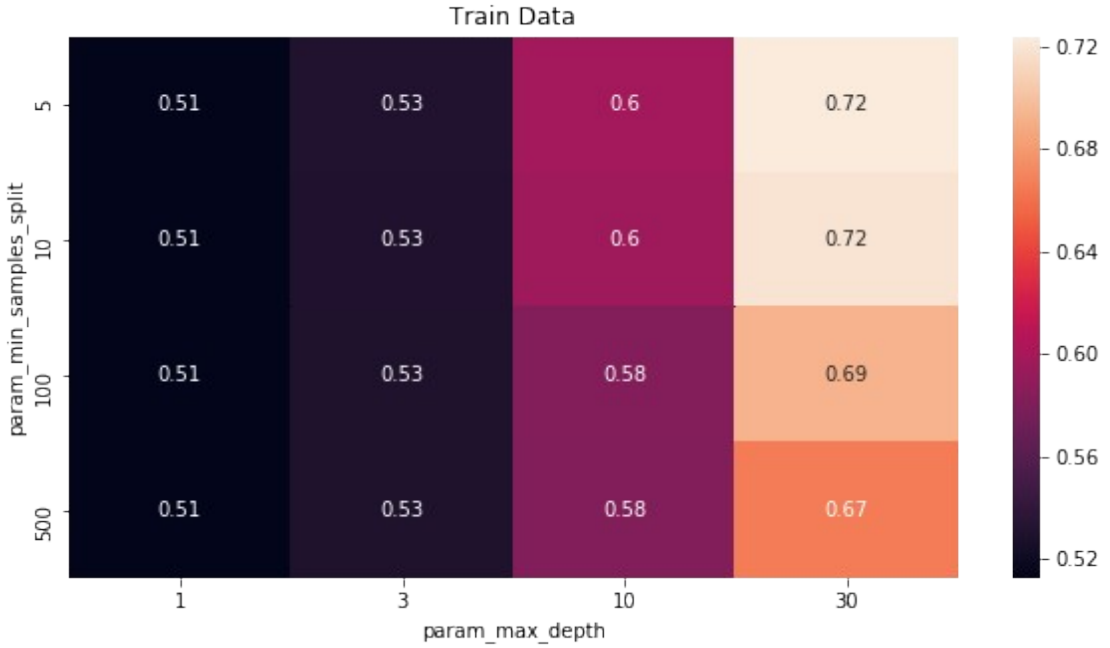
```
#The hyper paramter tuning (best depth in range [1, 3, 10, 30], and
the best min_samples_split in range [5, 10, 100, 500])
#Find the best hyper parameter which will give the maximum AUC value
#find the best hyper paramter using k-fold cross validation(use
gridsearch cv or randomsearch cv)/simple cross validation data(you can
write your own for loops refer sample solution)

clf = DecisionTreeClassifier(random_state =20)
clf.fit(X_train3,y_train3)

parameters = {'max_depth':[1, 3, 10, 30],'min_samples_split': [5, 10,
100, 500]}
GridSearch_clf = GridSearchCV(clf, parameters,scoring='roc_auc',cv
=3,return_train_score=True)
GridSearch_clf.fit(X_train3, y_train3)

print(GridSearch_clf.score(X_train3,y_train3))
print("------------Best Hyperparameters----------------")
# print best parameter after tuning
print(GridSearch_clf.best_params_)

# print how our model looks after hyper-parameter tuning
print(GridSearch_clf.best_estimator_)

#print(GridSearch_clf.cv_results_)

best_max_depth = GridSearch_clf.best_params_['max_depth']

best_min_samples_split =
GridSearch_clf.best_params_['min_samples_split']
```

```
0.6543340802444851
------------Best Hyperparameters----------------
{'max_depth': 30, 'min_samples_split': 500}
DecisionTreeClassifier(max_depth=30, min_samples_split=500,
random_state=20)
```

```
#Perform hyperparameter tuning and plot either heatmap or 3d plot.
#On Train data
temp =
np.stack((GridSearch_clf.cv_results_['param_min_samples_split'],GridSe
arch_clf.cv_results_['param_max_depth'],GridSearch_clf.cv_results_['me
an_train_score']),axis =1)
#print(temp)
```

```
scores = (pd.DataFrame(temp,columns =
['param_min_samples_split','param_max_depth','mean_test_score']).group
by(['param_min_samples_split','param_max_depth'])).max().unstack()
print(scores)
fig = plt.figure(figsize=(10,5))
sns.heatmap(scores.mean_test_score,annot = True)
plt.title("Train Data")
plt.show()
```

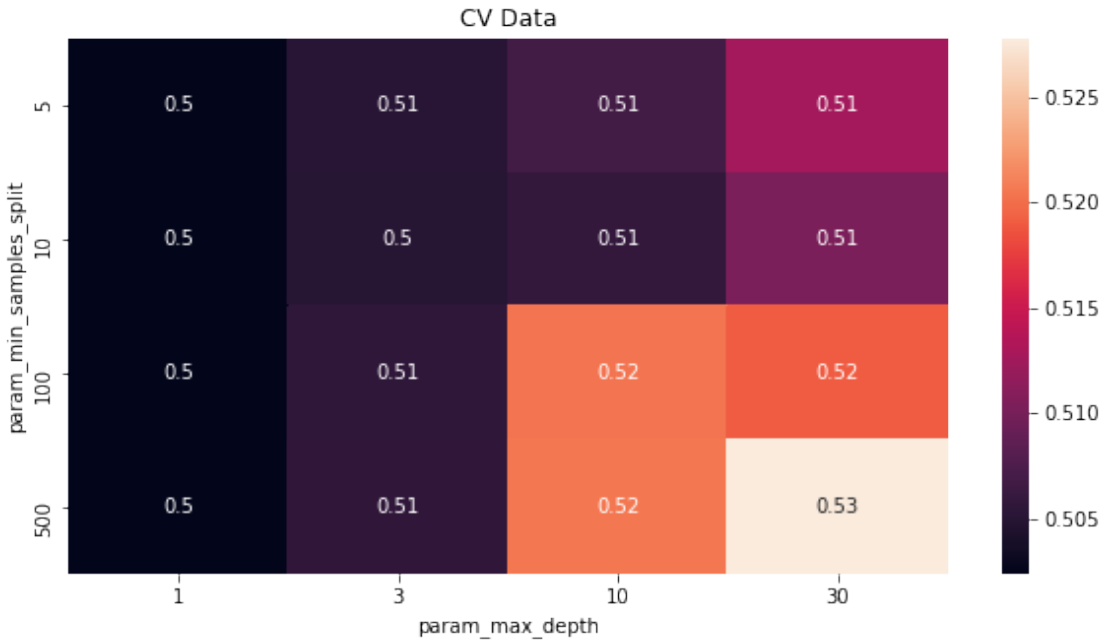
	mean_test_score			
param_max_depth	1	3	10	30
param_min_samples_split				
5	0.512354	0.530537	0.598050	0.723746
10	0.512354	0.530323	0.596387	0.718930
100	0.512354	0.529389	0.583022	0.692939
500	0.512354	0.529312	0.581768	0.666660




```
#Perform hyperparameter tuning and plot either heatmap or 3d plot.
#On CV data
temp =
np.stack((GridSearch_clf.cv_results_['param_min_samples_split'],GridSe
arch_clf.cv_results_['param_max_depth'],GridSearch_clf.cv_results_['me
an_test_score']),axis =1)
scores = (pd.DataFrame(temp,columns =
['param_min_samples_split','param_max_depth','mean_test_score']).group
by(['param_min_samples_split','param_max_depth'])).max().unstack()
print(scores)
fig = plt.figure(figsize=(10,5))
sns.heatmap(scores.mean_test_score,annot = True)
```

```
plt.title("CV Data")
plt.show()
```

	mean_test_score			
param_max_depth	1	3	10	30
param_min_samples_split				
5	0.502351	0.505106	0.506903	0.512696
10	0.502351	0.504962	0.505744	0.510273
100	0.502351	0.505507	0.520413	0.519122
500	0.502351	0.505536	0.520590	0.527791



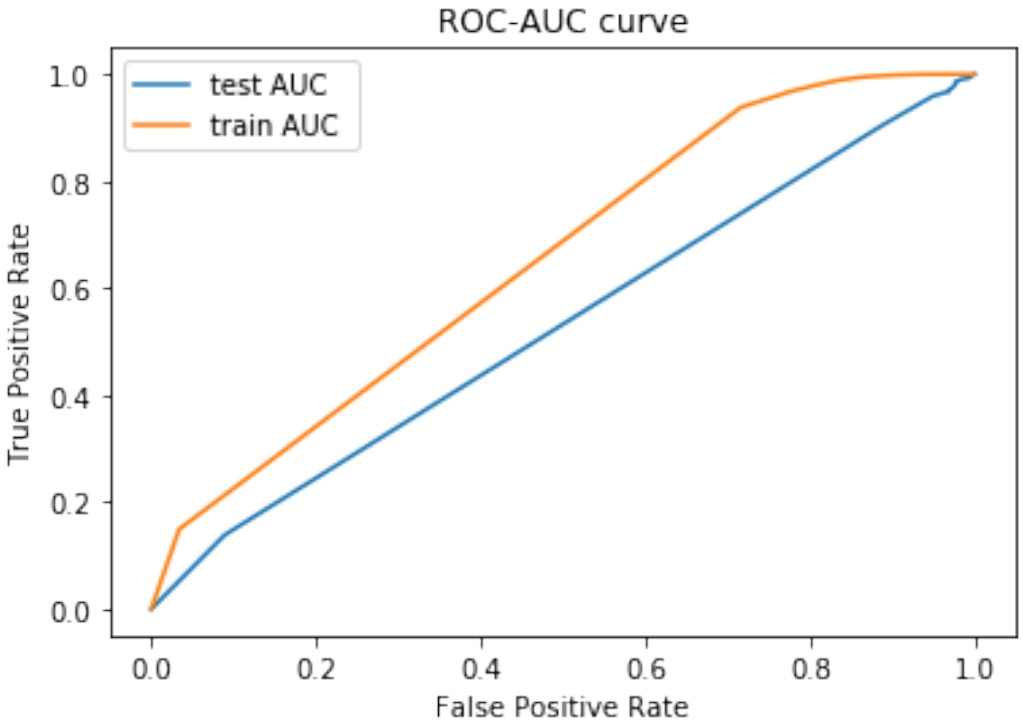

```
# 10. Find the best parameters and fit the model. Plot ROC-AUC
curve(using predict proba method)
# Re-fit the model with best parameters found
clf = DecisionTreeClassifier(max_depth =
best_max_depth ,min_samples_split =best_min_samples_split )
clf.fit(X_train3,y_train3)
```

```
DecisionTreeClassifier(max_depth=30, min_samples_split=500)
```

```
#create ROC curve

y_test_pred_proba = clf.predict_proba(X_test3)[::,1]
print(y_test3.shape,y_test_pred_proba.shape)
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test3,
y_test_pred_proba)

y_train_pred_proba = clf.predict_proba(X_train3)[::,1]
fpr_train, tpr_train, threshold_train = metrics.roc_curve(y_train3,
y_train_pred_proba)
```

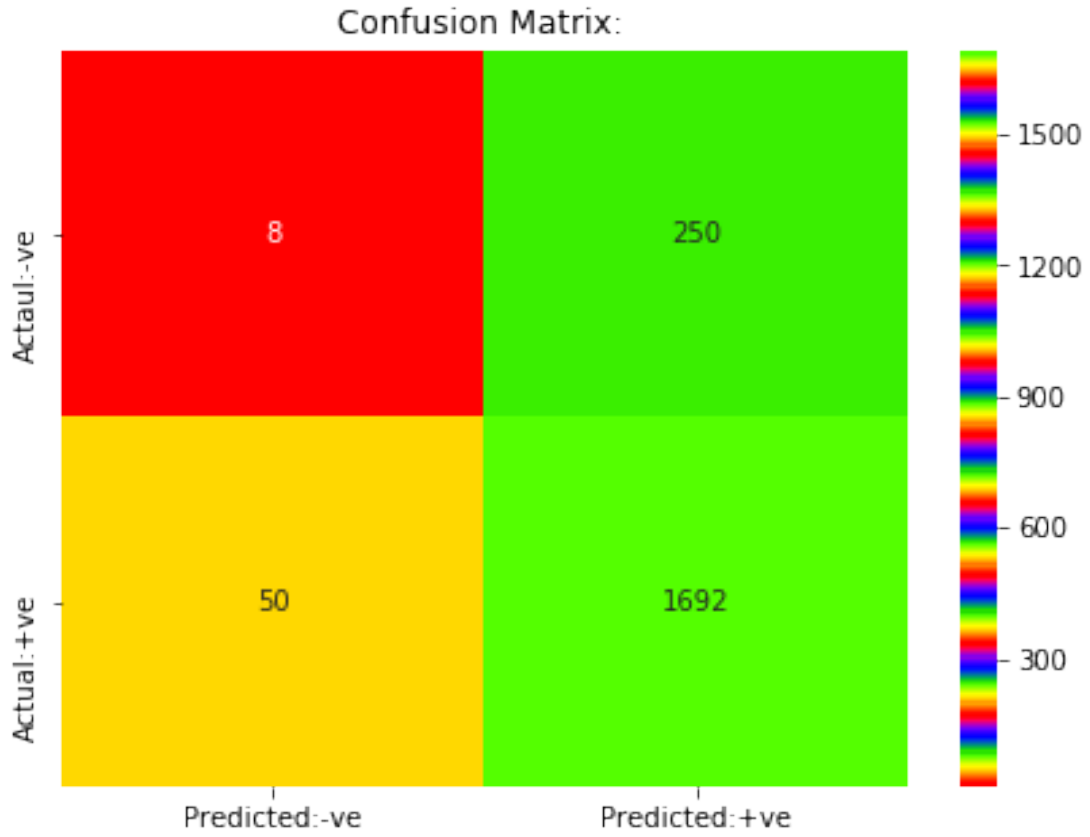
```
#roc_auc = auc(false_positive_rate, true_positive_rate)
plt.plot(fpr_test,tpr_test,label="test AUC ")
plt.plot(fpr_train,tpr_train,label="train AUC ")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title("ROC-AUC curve")
plt.legend()
plt.show()
#print(threshold_test)
#print(y_pred_proba)
```

```
(2000,) (2000,)
```




```
# 11. Plot confusion matrix based on best threshold value
y_pred = clf.predict(X_test3)
cm = confusion_matrix(y_test3,  y_pred)

cm_df =pd.DataFrame(cm,columns=["Predicted:-ve","Predicted:+ve"],index
= ["Actaul:-ve","Actual:+ve"])
print(cm_df)
print("***************confusion matrix***********************")

fig = plt.figure(figsize=(7,5))
ax =sns.heatmap(cm_df,annot =True, xticklabels=True,
yticklabels=True,cmap="prism",fmt="")
plt.title("Confusion Matrix: ")
plt.show()
```

```
            Predicted:-ve  Predicted:+ve
Actaul:-ve              8            250
Actual:+ve             50           1692
***************confusion matrix***********************
```




```
# 12. Find all the false positive data points and plot wordcloud of
essay text and pdf of teacher_number_of_previously_posted_projects.

indx_list = list()
for i in range(y_test3.shape[0]):
    if (y_test3.values[i] == 0) & (y_pred[i] == 1):
        indx_list.append(y_test3.index[i])

#print("actual Y",y_test.values)
#print("pred Y",y_pred)
#print(y_test.index[3])

#print(len(y_test.index))
#print(y_test.index)

#print(len(X_test.index))
```

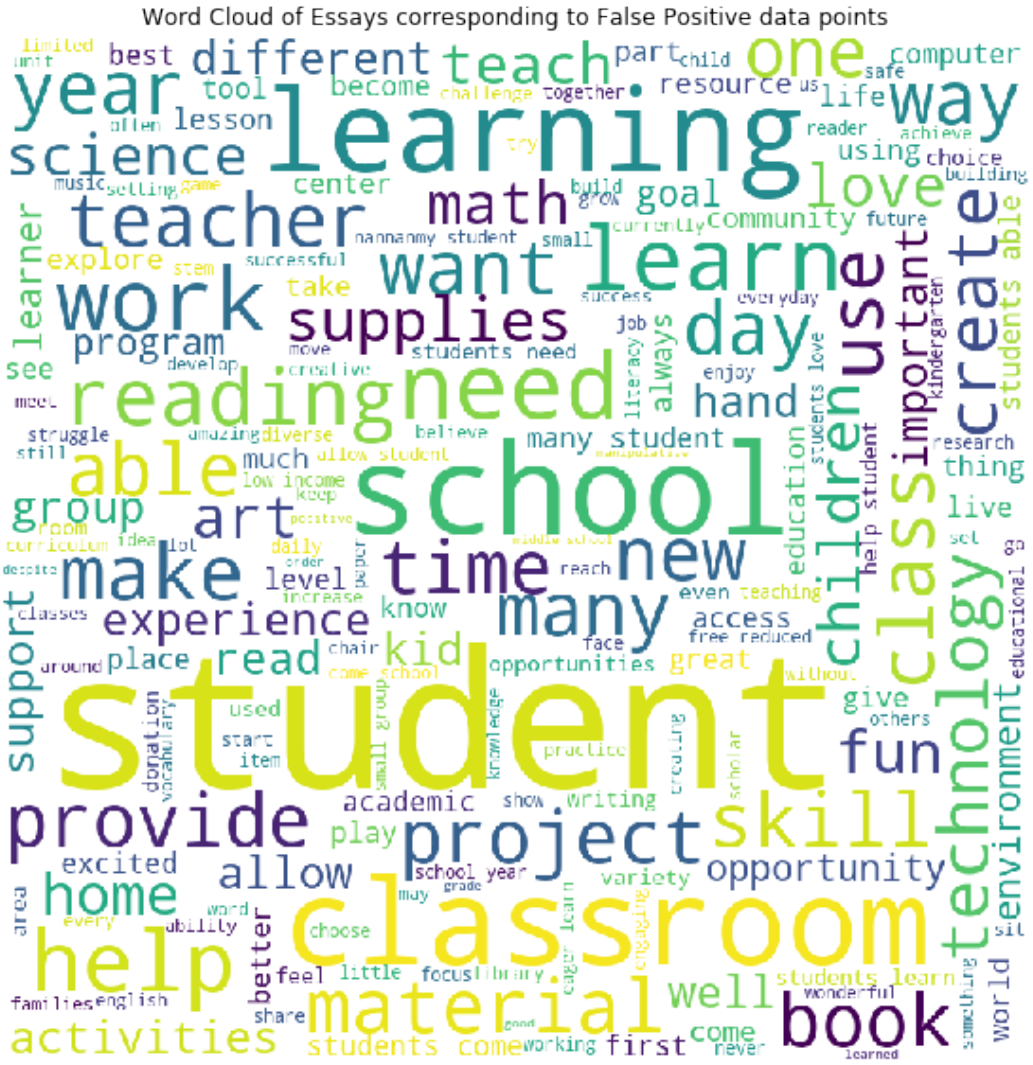
```
#print(X_test.index)
fp_essay = list()
fp_teacher_number_of_previously_posted_projects =list()
print("List of indices for false positive data points: ")
print(indx_list)

for i,ind in enumerate(indx_list):
    essay=data["essay"][data.index==ind].values[0]
    number_of_previously_posted_projects =
data["teacher_number_of_previously_posted_projects"]
[data.index==ind].values[0]
    fp_essay.append(essay)

fp_teacher_number_of_previously_posted_projects.append(number_of_previ
ously_posted_projects)
print(" "*70)
print("List (sample (2)) of essays corresponding to false positive
data poits: ")
print("-"*70)
print(fp_essay[0:2])

#word_count_list = [ len(fp_essay[i]) for i in range(len(fp_essay)) ]
#print()
#print("Word count of essays corresponding to false positive: ")
#print(word_count_list)

#essay_labels = ["essay "+str(i) for i in range(len(fp_essay))]
#fig = plt.figure(figsize = (20, 5))
#plt.bar(essay_labels,word_count_list,width =0.5)
#plt.xlabel("Essays corresponding to false positives")
#plt.ylabel("Word count")
#plt.title("False Positve Data points: Analysis")
#plt.show()
```

```
List of indices for false positive data points:
[5795, 9617, 190, 302, 4124, 8242, 3083, 1693, 4540, 2847, 5876, 1726,
7825, 9806, 286, 8117, 1124, 5256, 1783, 3448, 7620, 9867, 6793, 7529,
6825, 2020, 7582, 4890, 9222, 3184, 2900, 7153, 1379, 8547, 1704,
5098, 964, 9382, 7618, 4413, 1694, 4512, 8181, 213, 1878, 7009, 6975,
2424, 9906, 8983, 6313, 4464, 7775, 5639, 5759, 9989, 4109, 5349, 474,
6853, 1034, 1371, 6940, 2155, 6472, 6732, 7694, 7748, 847, 4057, 16,
9795, 9634, 8698, 2149, 6422, 9975, 668, 6327, 8225, 9968, 4881, 6405,
8282, 268, 3807, 9367, 1282, 4175, 3346, 9350, 9547, 3608, 1329, 5338,
3639, 7709, 2227, 5305, 2792, 4879, 2703, 8655, 2318, 4004, 2333,
7587, 1293, 8054, 8598, 7889, 3043, 4340, 2889, 2691, 2154, 2081,
1467, 4286, 7635, 7897, 8069, 9586, 8298, 1365, 5152, 8468, 6626,
8853, 6703, 640, 5388, 4290, 8209, 9010, 7458, 3265, 2470, 8294, 233,
2361, 1361, 7212, 6258, 1894, 8200, 7824, 5568, 866, 1420, 7771, 7624,
```

```
1103, 3849, 7453, 4782, 4575, 4079, 3416, 2175, 7266, 7539, 9664,
5016, 5187, 7563, 5242, 5522, 4919, 5228, 5627, 6088, 1551, 3472, 870,
7126, 2295, 226, 3056, 3888, 1259, 9168, 3110, 6423, 5741, 9738, 1465,
6796, 7303, 652, 7994, 1797, 7967, 7796, 2917, 8683, 9627, 9912, 7295,
2072, 2953, 1312, 426, 6981, 4343, 5138, 9144, 1626, 4674, 6779, 3119,
4967, 8890, 8965, 6590, 5108, 7875, 8868, 4937, 5205, 4283, 2575,
9230, 1150, 326, 9265, 3023, 4045, 7931, 7729, 3095, 6823, 737, 3228,
6676, 6251, 188, 9893, 3148, 9402, 5122, 3070, 4099, 5756, 2412, 8699,
1346, 5797, 5916, 794]

List (sample (2)) of essays corresponding to false positive data
poits:
------------------------------------------------------------------------
['other students parents work many jobs struggle give things necessary
there students struggle reading grade level distaste reading process
then students grade level struggle think critically students intensive
reading well language arts class true dislike reading kind never
learned love see escape new worlds rather mandatory thing pass state
test students come walks life types backgroundsthese donations help
students see words seeing traditional settings typically inside
textbooks i want able sit together work collaboratively use vocabulary
learning well tapping background knowledge this allow challenge
discussion priceless it understanding experience students ability
learn discussion learning growth deeper knowledge concrete long
lasting they able use knowledge speaking writing listening cross
curricular setting nannan', 'my classroom ppcd prek class means i
could 3 4 5 year olds in order qualify class disability homeless
poverty spanish speaking i try give students experiences i part school
exposure majority get i already working new centers next year order
make sure students every chance learn vocabulary time investigate
explore new materials new prek guidelines to insure ready next school
year experience donations the explorers project would give students
every opportunity learn explore investigate problem solve create it
important able expose students real life situations never even
experience eating going movies take granted i would like create
positive fun learning experience encourage learning well establishing
self confidence positive outcomes i incorporated variety new learning
centers new learning experiences storage new centers well organized
easy access nannan']
```

```
#plot wordcloud of essay text
#Ref: https://www.geeksforgeeks.org/generating-word-cloud-python/?
msclkid=7d386f85ba3811ec95b39b1d46408e15
from wordcloud import WordCloud
#fp_essays_text is to store all words of essays corresponding to false
positive data points in a str so we can find the word cloud of it.

fp_essays_text = str()
for i in fp_essay :
    fp_essays_text = fp_essays_text + i
```

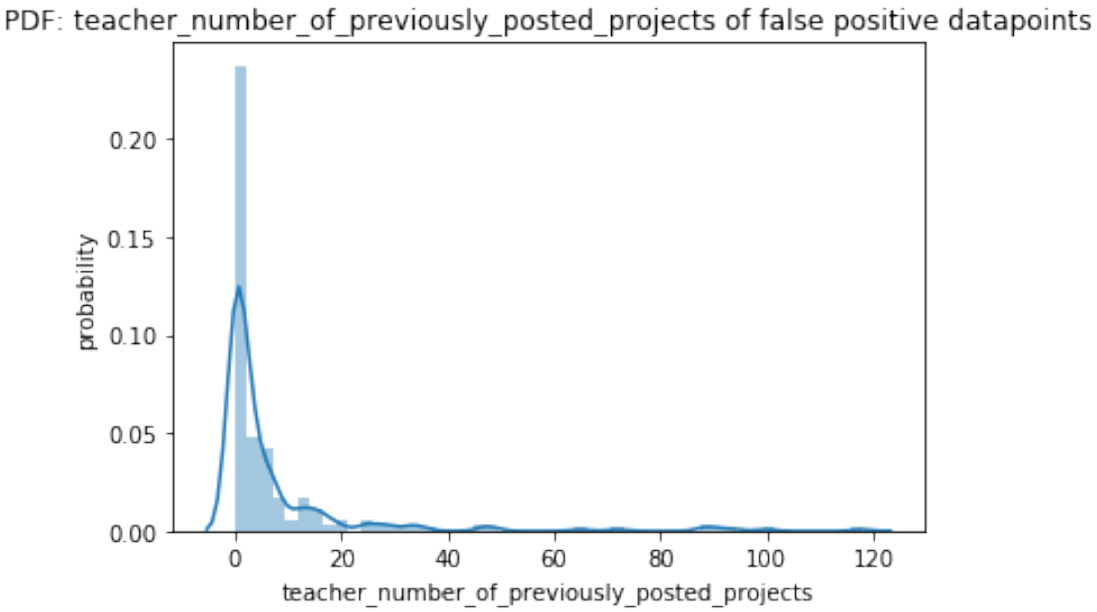
```
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                min_font_size = 10).generate(fp_essays_text)
# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.title("Word Cloud of Essays corresponding to False Positive data
points")
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```




```
print()
print("List of metric:teacher_number_of_previously_posted_projects
corresponding to false positive data poits: ")
print("-"*70)
```

```
print(fp_teacher_number_of_previously_posted_projects)
sns.distplot(fp_teacher_number_of_previously_posted_projects)
plt.title('PDF: teacher_number_of_previously_posted_projects of false
positive datapoints')
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.ylabel('probability')
plt.show()
```

```
List of metric:teacher_number_of_previously_posted_projects
corresponding to false positive data poits:
------------------------------------------------------------------------
[14, 0, 0, 0, 89, 0, 0, 2, 1, 33, 1, 47, 1, 6, 2, 20, 0, 72, 8, 12, 8,
11, 2, 1, 0, 16, 2, 0, 0, 6, 4, 0, 2, 1, 6, 0, 3, 0, 8, 0, 0, 2, 0,
100, 0, 0, 0, 18, 0, 6, 0, 0, 5, 36, 25, 0, 0, 0, 3, 29, 1, 2, 2, 1,
0, 3, 6, 7, 2, 1, 2, 0, 6, 13, 118, 9, 1, 2, 0, 0, 2, 1, 47, 0, 0, 2,
3, 4, 3, 6, 7, 0, 0, 1, 19, 2, 7, 15, 4, 2, 6, 0, 1, 14, 0, 1, 4, 0,
3, 4, 0, 6, 3, 2, 0, 0, 0, 6, 3, 3, 3, 0, 1, 0, 6, 0, 91, 2, 0, 0, 3,
0, 5, 0, 26, 3, 0, 17, 5, 1, 3, 33, 0, 3, 13, 8, 1, 2, 1, 0, 3, 0, 0,
0, 9, 0, 1, 7, 0, 4, 16, 0, 3, 4, 12, 65, 24, 6, 8, 3, 0, 0, 88, 1, 2,
0, 0, 11, 2, 0, 7, 2, 0, 1, 0, 0, 0, 0, 0, 3, 12, 19, 0, 0, 1, 28, 14,
1, 1, 1, 5, 6, 5, 8, 13, 0, 5, 29, 0, 0, 49, 1, 12, 0, 0, 0, 1, 0, 34,
16, 3, 8, 3, 0, 1, 1, 0, 0, 0, 0, 0, 2, 1, 0, 16, 2, 15, 8, 11, 0, 0,
3, 2, 94, 16, 0, 7, 25, 1, 1]
```



13. Write your observations about the wordcloud and pdf.

- {student,classroom,learning,school,...} These words found to be more frequent in the essays corresponded to False Positive data points.

- By analysis of PDF of "teacher_number_of_previously_posted_projects", we found that requests with teacher_number_of_previously_posted_projects = 0 are more likely to be classified as False Positive.

Observation (For all data sets)

- Both tasks, i.e., with TFIDF processed text and TFIDF-W2V processed text (essay) resulted in different results interms of choosing best parameters,confusion matrix, word-cloud.
- TFIDF-W2V processed text gave better AUC Score (0.746) compared to TFIDF AUC score (0.734)
- Also, TFIDF-W2V resulted in less no of vector length(i.e., only 300) after text to vector convertion which has a edge over the other method with regards to memory and time complexity
- {student,classroom,learning,school,...} These words found to be more frequent in the essays corresponded to False Positive data points in both tasks.
- By analysis of PDF of "teacher_number_of_previously_posted_projects", we found that requests with teacher_number_of_previously_posted_projects = 0 are more likely to be classified as False Positive in both tasks.

Regarding non-zero feature set:

- By removing zero features, we arrived at dimensional set for analysis (i.e., 556 zero features removed).Therefore, increasing memory and computationl time
- One main observation TPR improved relative to TFIDF data set

```
#!pip install prettytable
# Tabulate your results
# Please compare all your models using Prettytable library
#Ref: https://pypi.org/project/prettytable/?
msclkid=31b07eccba8911ec909d9465bafb9d1e
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= (" Vectorizer ", " Max_depth ", " Min_sample_split ","
Best -AUC ", "No of False Positives")
tb.add_row([" Tf - Idf", 30 , 500 ,0.734,250 ])
tb.add_row([" AVG-W2V", 10, 500,0.746, 253])
tb.add_row(["Non-Zero Features", 30, 500 ,0.67, 250])
print(tb.get_string(title = "Decision trees- Observations"))

+-------------------------------------------------------------------------
----------------------+
|                                Decision trees- Observations
|
```

Vectorizer	Max_depth	Min_sample_split	Best -AUC	No of False Positives
Tf - Idf	30	500	0.734	250
AVG-W2V	10	500	0.746	253
Non-Zero Features	30	500	0.67	250