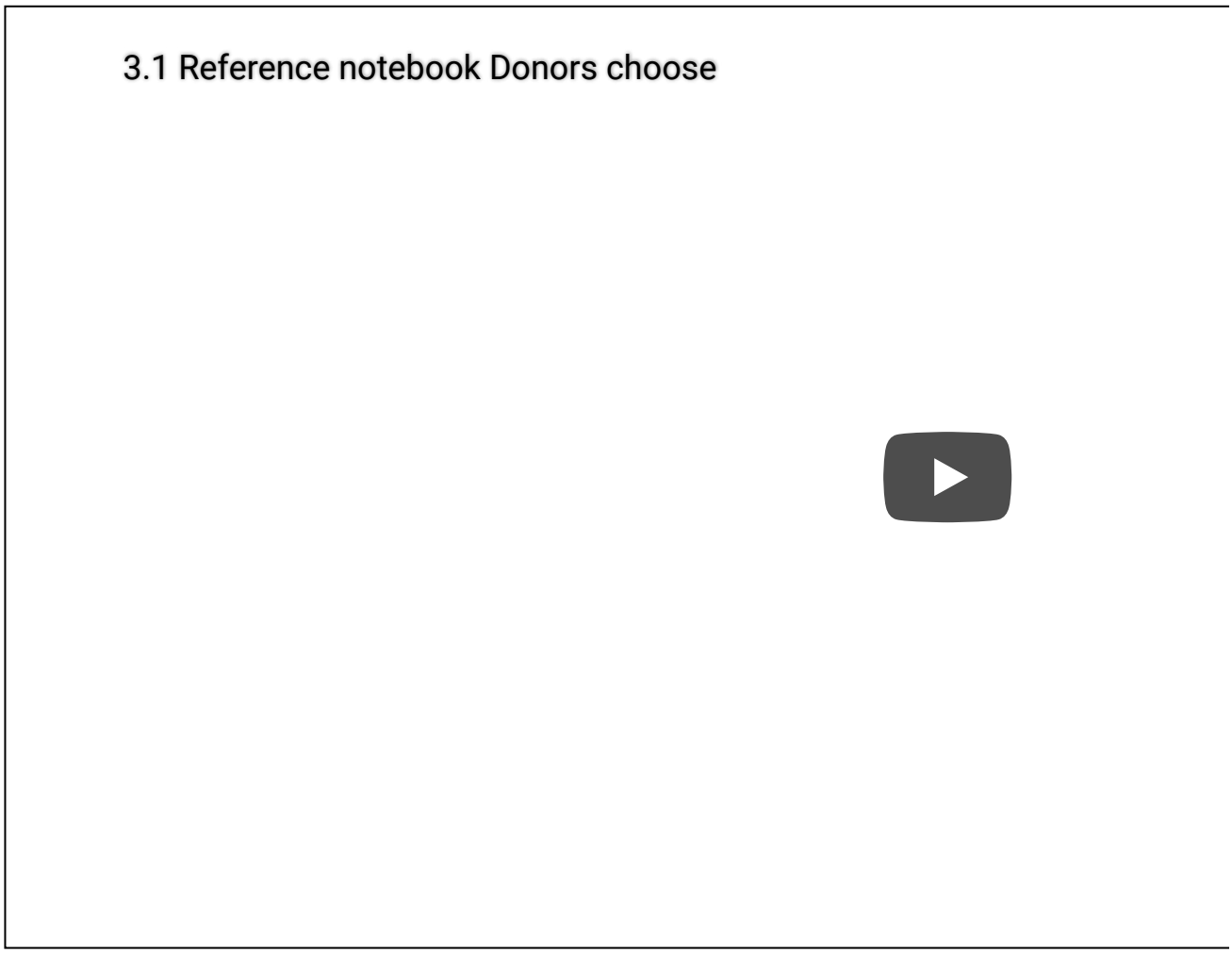
Assignment : DT

Please check below video before attempting this assignment

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




```
from google.colab import drive
drive.mount('drive')
```

```
Mounted at drive
```

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-IDFW2V for more details.

Glove vectors

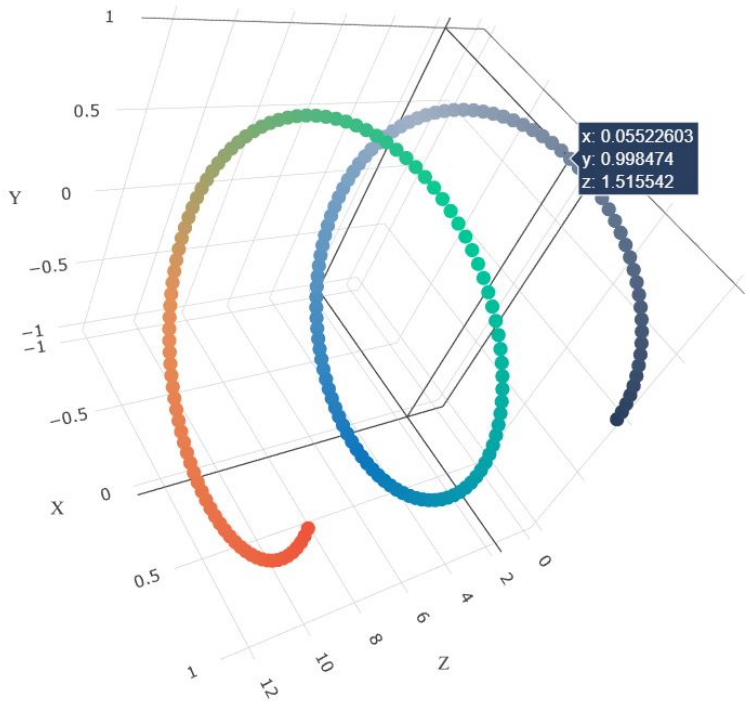
In this assignment you will be working with glove vectors , please check this and this for more details.

Download glove vectors from this link

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

Task - 1

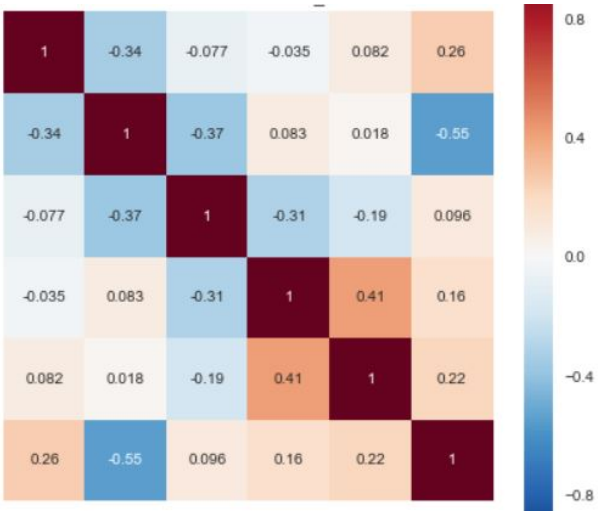
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)
    - Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)
2. **The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], 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)
3. **Representation of results**
    - 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



with X-axis as **min_sample_split**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

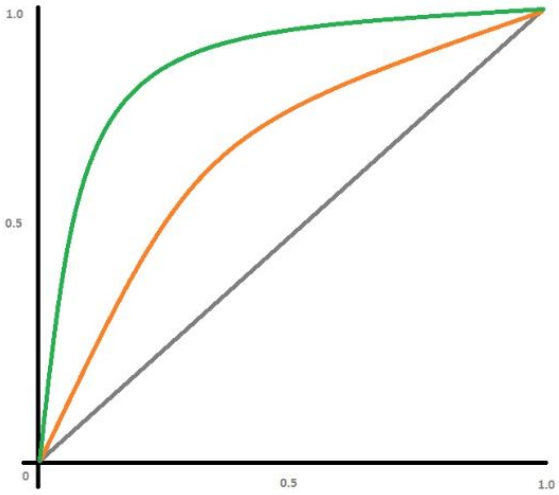
or

- 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



seaborn heat maps with rows as **min_sample_split**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- 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.



- Along with plotting ROC curve, you need to print the [confusion matrix](https://) with predicted and original labels of test data points

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

- Once after you plot the confusion matrix with the test data, get all the `false positive data points`
  - Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these `false positive data points`
  - Plot the box plot with the `price` of these `false positive data points`
  - Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

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

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

Hint for calculating Sentiment scores

```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
True
```

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest st
for learning my students learn in many different ways using all of our senses and multiple
of techniques to help all my students succeed students in my class come from a variety of
for wonderful sharing of experiences and cultures including native americans our school is
learners which can be seen through collaborative student project based learning in and out
in my class love to work with hands on materials and have many different opportunities to
mastered having the social skills to work cooperatively with friends is a crucial aspect o
montana is the perfect place to learn about agriculture and nutrition my students love to
in the early childhood classroom i have had several kids ask me can we try cooking with re
and create common core cooking lessons where we learn important math and writing concepts
food for snack time my students will have a grounded appreciation for the work that went i
of where the ingredients came from as well as how it is healthy for their bodies this proj
nutrition and agricultural cooking recipes by having us peel our own apples to make homema
and mix up healthy plants from our classroom garden in the spring we will also create our
shared with families students will gain math and literature skills as well as a life long
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975, /usr/local/lib/python3.6/dist-pa
  warnings.warn("The twython library has not been installed. "
```

1. Decision Tree

1.1 Loading Data

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

import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer


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

from nltk.corpus import stopwords
import pickle

from tqdm import tqdm
import os


train_path="/content/drive/My Drive/train_data.csv"
resource_path="/content/drive/My Drive/resources.csv"


import pandas
project_data = pd.read_csv(train_path,nrows=50000)
resource_data=pd.read_csv(resource_path)


print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (50000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_sta
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

```
print("Number of data points in train data", resource_data.shape)
```

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

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

Preprocessing categorical and numerical data

```
project_data['project_grade_category'].value_counts()
```

```
Grades PreK-2    20316
Grades 3-5       16968
Grades 6-8        7750
Grades 9-12       4966
Name: project_grade_category, dtype: int64
```

```
#Preprocessing project grade category
# https://stackoverflow.com/questions/36383821/pandas-dataframe-apply-function-to-column-s
project_data['project_grade_category'] = project_data['project_grade_category'].str.replac
project_data['project_grade_category'] = project_data['project_grade_category'].str.replac
project_data['project_grade_category'] = project_data['project_grade_category'].str.lower(
project_data['project_grade_category'].value_counts()
```

```
grades_prek_2    20316
grades_3_5       16968
grades_6_8        7750
grades_9_12       4966
Name: project_grade_category, dtype: int64
```

```
project_data['project_subject_categories'].value_counts()
```

```
Literacy & Language                           10927
Math & Science                                 7695
Literacy & Language, Math & Science            6705
Health & Sports                                4700
Music & The Arts                               2358
Special Needs                                  1913
Literacy & Language, Special Needs             1814
Applied Learning                               1719
Math & Science, Literacy & Language            1041
Applied Learning, Literacy & Language          1018
Math & Science, Special Needs                   871
History & Civics                                839
Literacy & Language, Music & The Arts           794
Math & Science, Music & The Arts                755
Applied Learning, Special Needs                 672
History & Civics, Literacy & Language           651
Health & Sports, Special Needs                  633
Warmth, Care & Hunger                           606
Math & Science, Applied Learning                565
```

```
Applied Learning, Math & Science                477
Health & Sports, Literacy & Language            369
Literacy & Language, History & Civics           363
Applied Learning, Music & The Arts              360
Math & Science, History & Civics                282
Literacy & Language, Applied Learning           280
Applied Learning, Health & Sports               264
Math & Science, Health & Sports                 187
History & Civics, Math & Science                171
Special Needs, Music & The Arts                 140
History & Civics, Music & The Arts              135
Health & Sports, Math & Science                 118
History & Civics, Special Needs                 103
Health & Sports, Applied Learning                99
Applied Learning, History & Civics               78
Music & The Arts, Special Needs                  67
Health & Sports, Music & The Arts                66
Literacy & Language, Health & Sports             33
Health & Sports, History & Civics                25
History & Civics, Applied Learning               25
Special Needs, Health & Sports                   14
Health & Sports, Warmth, Care & Hunger           12
Music & The Arts, Health & Sports                10
Music & The Arts, History & Civics                9
History & Civics, Health & Sports                 8
Applied Learning, Warmth, Care & Hunger           8
Math & Science, Warmth, Care & Hunger             7
Special Needs, Warmth, Care & Hunger              6
Music & The Arts, Applied Learning                4
Literacy & Language, Warmth, Care & Hunger        3
Music & The Arts, Warmth, Care & Hunger           1
Name: project_subject_categories, dtype: int64
```

```
#Preprocessing project_subject_categories
project_data['project_subject_categories'] = project_data['project_subject_categories'].st
project_data['project_subject_categories'] = project_data['project_subject_categories'].st
project_data['project_subject_categories'] = project_data['project_subject_categories'].st
project_data['project_subject_categories'] = project_data['project_subject_categories'].st
project_data['project_subject_categories'] = project_data['project_subject_categories'].st
project_data['project_subject_categories'].value_counts()
```

```
literacy_language                       10927
math_science                             7695
literacy_language_math_science           6705
health_sports                            4700
music_arts                               2358
specialneeds                             1913
literacy_language_specialneeds           1814
appliedlearning                          1719
math_science_literacy_language           1041
appliedlearning_literacy_language        1018
math_science_specialneeds                 871
history_civics                            839
literacy_language_music_arts              794
math_science_music_arts                   755
appliedlearning_specialneeds              672
history_civics_literacy_language          651
health_sports_specialneeds                633
warmth_care_hunger                        606
```

```
math_science_appliedlearning              565
appliedlearning_math_science              477
health_sports_literacy_language           369
literacy_language_history_civics          363
appliedlearning_music_arts                360
math_science_history_civics               282
literacy_language_appliedlearning         280
appliedlearning_health_sports             264
math_science_health_sports                187
history_civics_math_science               171
specialneeds_music_arts                   140
history_civics_music_arts                 135
health_sports_math_science                118
history_civics_specialneeds               103
health_sports_appliedlearning              99
appliedlearning_history_civics             78
music_arts_specialneeds                    67
health_sports_music_arts                   66
literacy_language_health_sports            33
health_sports_history_civics               25
history_civics_appliedlearning             25
specialneeds_health_sports                 14
health_sports_warmth_care_hunger           12
music_arts_health_sports                   10
music_arts_history_civics                   9
history_civics_health_sports                8
appliedlearning_warmth_care_hunger          8
math_science_warmth_care_hunger             7
specialneeds_warmth_care_hunger             6
music_arts_appliedlearning                  4
literacy_language_warmth_care_hunger        3
music_arts_warmth_care_hunger               1
Name: project_subject_categories, dtype: int64
```

```
project_data['teacher_prefix'].value_counts()
```

```
Mrs.       26140
Ms.        17936
Mr.         4859
Teacher     1061
Dr.            2
Name: teacher_prefix, dtype: int64
```

```
#Preprocessing teacher_prefix
# check if we have any nan values are there
print(project_data['teacher_prefix'].isnull().values.any())
print("number of nan values",project_data['teacher_prefix'].isnull().values.sum())
```

```
True
number of nan values 2
```

```
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('Mrs.')
project_data['teacher_prefix'].value_counts()
```

```
Mrs.       26142
Ms.        17936
Mr.         4859
```

```
Teacher     1061
Dr.            2
Name: teacher_prefix, dtype: int64
```

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.replace('.','')
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.lower()
project_data['teacher_prefix'].value_counts()
```

```
mrs        26142
ms         17936
mr          4859
teacher     1061
dr             2
Name: teacher_prefix, dtype: int64
```

```
project_data['project_subject_subcategories'].value_counts()
```

```
Literacy                                     4434
Literacy, Mathematics                        3833
Literature & Writing, Mathematics            2705
Literacy, Literature & Writing               2570
Mathematics                                  2441
                                             ...
Civics & Government, Nutrition Education        1
Character Education, Financial Literacy         1
Character Education, Economics                  1
Extracurricular, Foreign Languages              1
Financial Literacy, Parent Involvement          1
Name: project_subject_subcategories, Length: 384, dtype: int64
```

```
#Preprocessing project_subject_subcategories
project_data['project_subject_subcategories'] = project_data['project_subject_subcategorie
project_data['project_subject_subcategories'] = project_data['project_subject_subcategorie
project_data['project_subject_subcategories'] = project_data['project_subject_subcategorie
project_data['project_subject_subcategories'] = project_data['project_subject_subcategorie
project_data['project_subject_subcategories'] = project_data['project_subject_subcategorie
project_data['project_subject_subcategories'].value_counts()
```

```
literacy                                   4434
literacy_mathematics                       3833
literature_writing_mathematics             2705
literacy_literature_writing                2570
mathematics                                2441
                                           ...
visualarts_warmth_care_hunger                 1
civics_government_nutritioneducation          1
charactereducation_economics                  1
appliedsciences_financialliteracy             1
environmentalscience_financialliteracy        1
Name: project_subject_subcategories, Length: 384, dtype: int64
```

```
project_data['school_state'].value_counts()
```

```
CA    7024
NY    3393
TX    3320
```

```
FL    2839
NC    2340
IL    1967
SC    1830
GA    1828
MI    1468
PA    1419
OH    1180
IN    1171
MO    1166
WA    1103
LA    1094
MA    1076
OK    1074
NJ    1005
AZ     994
VA     916
WI     833
UT     792
AL     790
TN     774
CT     774
MD     668
NV     665
KY     614
MS     598
OR     577
MN     556
CO     538
AR     446
IA     306
ID     302
KS     285
DC     247
HI     239
NM     236
ME     222
WV     218
DE     155
AK     153
NE     144
SD     142
NH     141
RI     126
MT     106
ND      63
WY      51
VT      32
Name: school_state, dtype: int64
```

```
#preprocessing school_state
project_data['school_state'] = project_data['school_state'].str.lower()
project_data['school_state'].value_counts()
```

```
ca    7024
ny    3393
tx    3320
fl    2839
nc    2340
il    1967
```

```
sc    1830
ga    1828
mi    1468
pa    1419
oh    1180
in    1171
mo    1166
wa    1103
la    1094
ma    1076
ok    1074
nj    1005
az     994
va     916
wi     833
ut     792
al     790
ct     774
tn     774
md     668
nv     665
ky     614
ms     598
or     577
mn     556
co     538
ar     446
ia     306
id     302
ks     285
dc     247
hi     239
nm     236
me     222
wv     218
de     155
ak     153
ne     144
sd     142
nh     141
ri     126
mt     106
nd      63
wy      51
vt      32
Name: school_state, dtype: int64
```

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

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

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
```

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


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


project_data['project_title'].head(5)

    0     Educational Support for English Learners at Home
    1                Wanted: Projector for Hungry Learners
    2    Soccer Equipment for AWESOME Middle School Stu...
    3                               Techie Kindergarteners
    4                               Interactive Math Tools
    Name: project_title, dtype: object


print("printing some random reviews")
print(9, project_data['project_title'].values[9])
print(34, project_data['project_title'].values[34])
print(147, project_data['project_title'].values[147])

    printing some random reviews
    9 Just For the Love of Reading--\r\nPure Pleasure
    34 \"Have A Ball!!!\"
    147 Who needs a Chromebook?\r\nWE DO!!


# Combining all the above stundents
from tqdm import tqdm
def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
```

```
            sent = sent.replace('\\r', ' ')
            sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
            # https://gist.github.com/sebleier/554280
            sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
            preprocessed_text.append(sent.lower().strip())
        return preprocessed_text


    #preprocessing project_title
    preprocessed_titles = preprocess_text(project_data['project_title'].values)
```

```
    100%|██████████| 50000/50000 [00:01<00:00, 41990.84it/s]
```

```
    print("printing some random reviews")
    print(9, preprocessed_titles[9])
    print(34, preprocessed_titles[34])
    print(147, preprocessed_titles[147])
```

```
    printing some random reviews
    9 love reading pure pleasure
    34 ball
    147 needs chromebook
```

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


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

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

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


    sent = decontracted(project_data['essay'].values[20000])
    print(sent)
    print("="*50)
```

```
    My kindergarten students have varied disabilities ranging from speech and language de
    ==================================================
```

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-pytho
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
     My kindergarten students have varied disabilities ranging from speech and language de
```

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
     My kindergarten students have varied disabilities ranging from speech and language de
```

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

```
#convert all the words to lower case first and then remove the stopwords
for i in range(len(project_data['essay'].values)):
    project_data['essay'].values[i] = project_data['essay'].values[i].lower()
```

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.replace('nan',' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
```

```
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 50000/50000 [00:28<00:00, 1780.69it/s]
```

```
#creating a new column with the preprocessed essays and replacing it with the original col
project_data['preprocessed_essays'] = preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

```
#convert all the words to lower case first and then remove the stopwords
for i in range(len(project_data['project_title'].values)):
    project_data['project_title'].values[i] = project_data['project_title'].values[i].lowe
```

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

```
100%|██████████| 50000/50000 [00:01<00:00, 41283.92it/s]
```

```
#creating a new column with the preprocessed titles,useful for analysis
project_data['preprocessed_titles'] = preprocessed_titles
```

Performing sentiment analysis using nltk vader lexicon to calculate values

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()
neg=[]
pos=[]
neu=[]
compound=[]
for_sentiment = project_data['essay'].tolist()
for i in for_sentiment:
    j=sid.polarity_scores(i)['neg']
```

```
    k=sid.polarity_scores(i)['pos']
    l=sid.polarity_scores(i)['neu']
    m=sid.polarity_scores(i)['compound']
    neg.append(j)
    pos.append(k)
    neu.append(l)
    compound.append(m)
project_data['neg']=neg
project_data['pos']=pos
project_data['neu']=neu
project_data['compound']=compound


y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)
X.head(1)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	mrs	in

1.2 Splitting data into Train and cross validation(or test): Stratified Sampling

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
print("Split ratio")
print('-'*50)
print('Train dataset:',len(X_train)/len(X)*100,'%\n','size:',len(X_train))
print('Test dataset:',len(X_test)/len(X)*100,'%\n','size:',len(X_test))
```

```
Split ratio
--------------------------------------------------
Train dataset: 67.0 %
 size: 33500
Test dataset: 33.0 %
 size: 16500
```

1.3 Make Data Model Ready: encoding eassay

TFIDF

```
#Converting essay in to TFIDF representation using count vectorizer.
```

```
vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)
vectorizer_tfidf_essay.fit(X_train['preprocessed_essays'])      #Fitting has to be on Trai

X_train_essay_tfidf = vectorizer_tfidf_essay.transform(X_train['preprocessed_essays'].valu

X_test_essay_tfidf = vectorizer_tfidf_essay.transform(X_test['preprocessed_essays'].values

print("Shape of train data matrix after one hot encoding ",X_train_essay_tfidf.shape)

print("Shape of test data matrix after one hot encoding ",X_test_essay_tfidf.shape)
```

```
Shape of train data matrix after one hot encoding  (33500, 10330)
Shape of test data matrix after one hot encoding  (16500, 10330)
```

TFIDF WEIGHTED W2V

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_essays'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())


# converting essay to tfidf_w2v
# compute average word2vec for each review.
def tfidf_w2v(words):
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(words): # 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((senten
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # g
                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)

    print(len(tfidf_w2v_vectors))
    print(len(tfidf_w2v_vectors[0]))
    return tfidf_w2v_vectors


train_tfidf_w2v=tfidf_w2v(X_train['preprocessed_essays'])
test_tfidf_w2v=tfidf_w2v(X_test['preprocessed_essays'])
```

```
100%|██████████| 33500/33500 [01:04<00:00, 518.82it/s]
  0%|          | 56/16500 [00:00<00:30, 532.83it/s]33500
300
100%|██████████| 16500/16500 [00:30<00:00, 534.51it/s]16500
```

```
300
```

1.4 Make Data Model Ready: encoding numerical, categorical features

```
#one hot encoding school_state
vectorizer1 = CountVectorizer(binary=True)
vectorizer1.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer1.transform(X_train['school_state'].values)
X_test_state_ohe = vectorizer1.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer1.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 51) (33500,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id',
====================================================================================
```

```
#one hot encoding teacher_prefix
vectorizer2 = CountVectorizer()
vectorizer2.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer2.transform(X_train['teacher_prefix'].values)

X_test_teacher_ohe = vectorizer2.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)

print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer2.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 5) (33500,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================
```

```
#One hot coding project_grade_category
vectorizer3 = CountVectorizer()
vectorizer3.fit(X_train['project_grade_category'].values) # fit has to happen only on trai

# we use the fitted CountVectorizer to convert the text to vector
```

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer3.transform(X_train['project_grade_category'].values)

X_test_grade_ohe = vectorizer3.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)

print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer3.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 4) (33500,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

```
#One hot encoding project_subject_categories
vectorizer4 = CountVectorizer()
vectorizer4.fit(X_train['project_subject_categories'].values) # fit has to happen only on

# we use the fitted CountVectorizer to convert the text to vector
X_train_subject_ohe = vectorizer4.transform(X_train['project_subject_categories'].values)

X_test_subject_ohe = vectorizer4.transform(X_test['project_subject_categories'].values)

print("After vectorizations")
print(X_train_subject_ohe.shape, y_train.shape)

print(X_test_subject_ohe.shape, y_test.shape)
print(vectorizer4.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 50) (33500,)
(16500, 50) (16500,)
['appliedlearning', 'appliedlearning_health_sports', 'appliedlearning_history_civics
====================================================================================================
```

```
#One hot encoding project_subject_subcategories
vectorizer5 = CountVectorizer()
vectorizer5.fit(X_train['project_subject_subcategories'].values) # fit has to happen only

# we use the fitted CountVectorizer to convert the text to vector
X_train_subject_sub_ohe = vectorizer5.transform(X_train['project_subject_subcategories'].v

X_test_subject_sub_ohe = vectorizer5.transform(X_test['project_subject_subcategories'].val

print("After vectorizations")
print(X_train_subject_sub_ohe.shape, y_train.shape)

print(X_test_subject_sub_ohe.shape, y_test.shape)
print(vectorizer5.get_feature_names())
```

```
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 374) (33500,)
(16500, 374) (16500,)
['appliedsciences', 'appliedsciences_charactereducation', 'appliedsciences_civics_gov
==========================================================================================
```

```
#Normalizing price
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-g
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_inde
price_data.head(2)
```

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

```
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
```

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(-1,1))
X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)

print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================================
```

```
#Normalizing teacher_number_of_previously_posted_projects
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
```

```
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1

X_train_post_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_proj

X_test_post_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projec



print("After vectorizations")
print(X_train_post_norm.shape, y_train.shape)

print(X_test_post_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

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

X_train_neg_norm = normalizer.transform(X_train['neg'].values.reshape(-1,1))

X_test_neg_norm = normalizer.transform(X_test['neg'].values.reshape(-1,1))



print("After vectorizations")
print(X_train_neg_norm.shape, y_train.shape)

print(X_test_neg_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['pos'].values.reshape(-1,1))

X_train_pos_norm = normalizer.transform(X_train['pos'].values.reshape(-1,1))
```

```
X_test_pos_norm = normalizer.transform(X_test['pos'].values.reshape(-1,1))



print("After vectorizations")
print(X_train_pos_norm.shape, y_train.shape)

print(X_test_pos_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

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

X_train_neu_norm = normalizer.transform(X_train['neu'].values.reshape(-1,1))

X_test_neu_norm = normalizer.transform(X_test['neu'].values.reshape(-1,1))



print("After vectorizations")
print(X_train_neu_norm.shape, y_train.shape)

print(X_test_neu_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['compound'].values.reshape(-1,1))

X_train_com_norm = normalizer.transform(X_train['compound'].values.reshape(-1,1))

X_test_com_norm = normalizer.transform(X_test['compound'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_com_norm.shape, y_train.shape)

print(X_test_com_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

SET-1

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr1 = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade

X_te1 = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe

print("Final Data matrix")
print(X_tr1.shape, y_train.shape)

print(X_te1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 10820) (33500,)
(16500, 10820) (16500,)
====================================================================================================
```

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 490
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

Perfoming grid search for hyper parameter tuning

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.h
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import roc_auc_score

DT = DecisionTreeClassifier(class_weight = 'balanced')
params = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10, 100,500]}
clf = GridSearchCV(DT, params, cv= 3, scoring='roc_auc',verbose=1,return_train_score=True)
clf.fit(X_tr1, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
bestMaxDepth_1=clf.best_params_['max_depth']
bestMinSampleSplit_1=clf.best_params_['min_samples_split']
bestScore_1=clf.best_score_
print("BEST MAX DEPTH: ",clf.best_params_['max_depth']," BEST SCORE: ",clf.best_score_,"BE
```

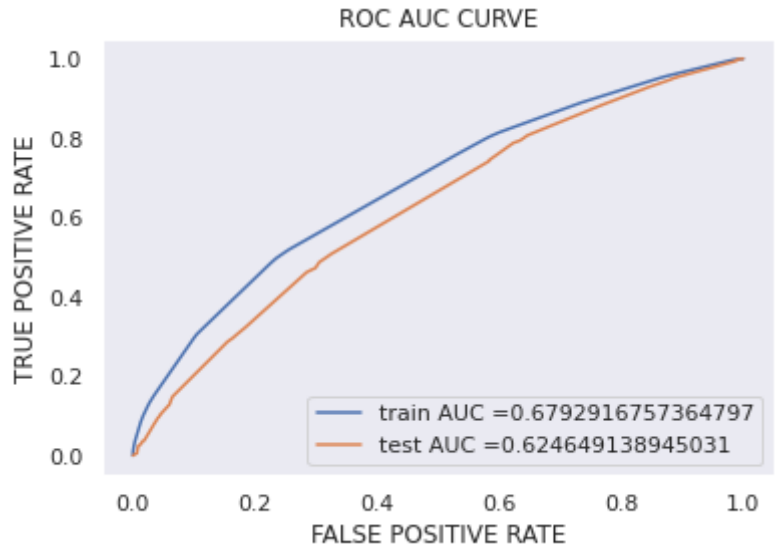
```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  48 out of  48 | elapsed:  4.4min finished
BEST MAX DEPTH:  10  BEST SCORE:  0.6136014521264089 BEST MIN SAMPLE SPLIT:  500
```

```
print(clf.best_estimator_)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
# Best tune parameters
best_tune_parameters=[{'max_depth':[10], 'min_samples_split':[500] } ]


# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc

clf=DecisionTreeClassifier (class_weight = 'balanced',max_depth=10,min_samples_split=500)
clf.fit(X_tr1, y_train)
# for visulation
clf.fit(X_tr1, y_train)
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#
y_train_pred1 = clf.predict_proba(X_tr1) [:,1]
y_test_pred1 = clf.predict_proba(X_te1) [:,1]
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred1)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
```

```
plt.plot(train_fpr, train_tpr, label="train AUC = "+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC AUC CURVE")
plt.grid()
plt.show()
```




```
import seaborn as sns; sns.set()
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_min_samples_split', 'param_max
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```

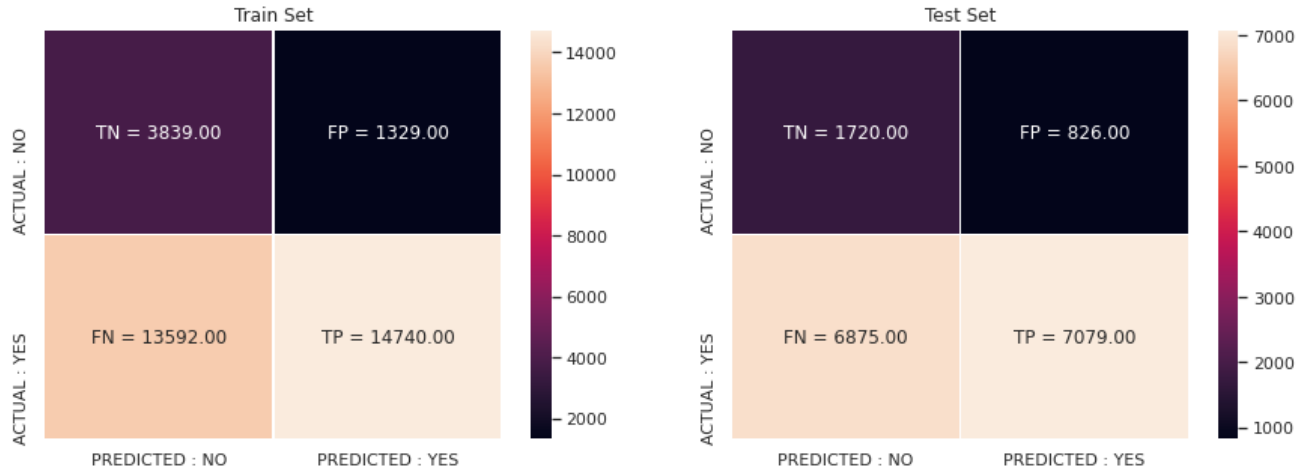

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high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

def predict_with_best_t(proba, threshould):
    predictions = []
    global predictions1
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1=predictions
    return predictions


print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred1, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred1, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.3864700626232347 for threshold 0.495
Train confusion matrix
[[ 3839  1329]
 [13592 14740]]
Test confusion matrix
[[1720  826]
 [6875 7079]]
```

```
print("="*100)
from sklearn.metrics import confusion_matrix
import seaborn as sns; sns.set()
con_m_train=confusion_matrix(y_train, predict_with_best_t(y_train_pred1, best_t))
con_m_test=confusion_matrix(y_test, predict_with_best_t(y_test_pred1, best_t))
key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.f
sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],y
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
```

```
plt.show()
```

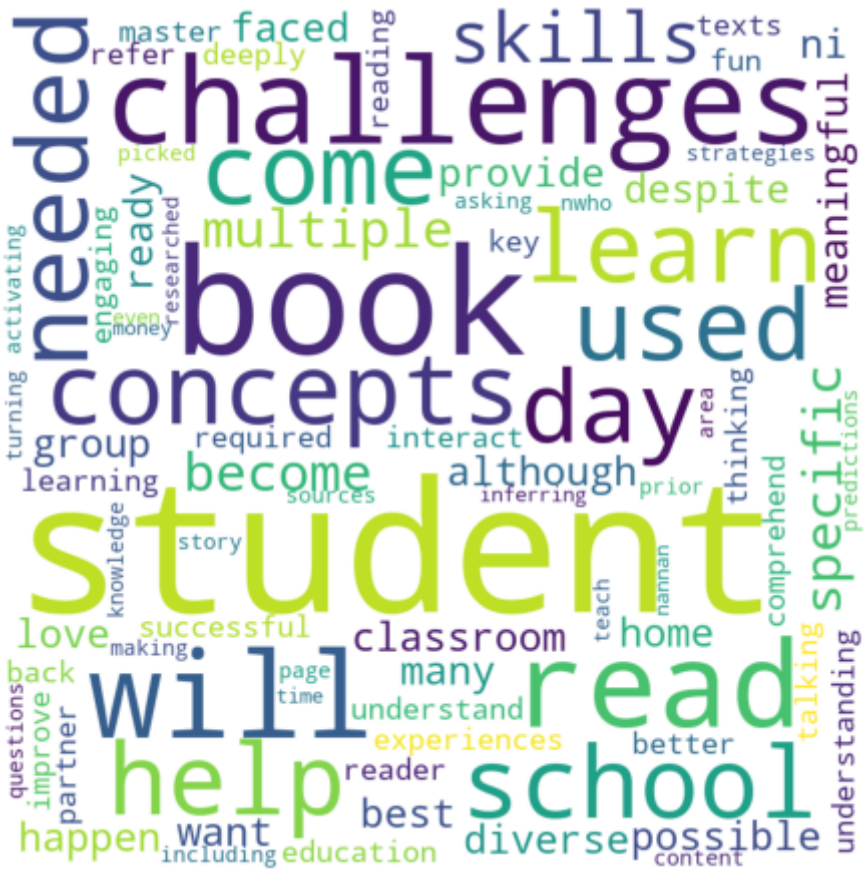
```
=====================================================================================
the maximum value of tpr*(1-fpr) 0.3864700626232347 for threshold 0.495
```



Plotting the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these `false positive data points`

```
false_positives = []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions1[i] == 1) :
    false_positives.append(i)
fp_essay1 = []
for i in false_positives :
  fp_essay1.append(X_test['essay'].values[i])


#COPIED FROM:https://www.geeksforgeeks.org/generating-word-cloud-python/

from wordcloud import WordCloud, STOPWORDS

comment_words = ' '
stopwords = set(STOPWORDS)

for val in fp_essay1 :
  val = str(val)
  tokens = val.split()

for i in range(len(tokens)):
  tokens[i] = tokens[i].lower()

for words in tokens :
  comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = st
```

```
plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Creating Dataframe with falsepositive points

```
cols = X_test.columns
X_test_falsePositives = pd.DataFrame(columns=cols)
```

```
for i in false_positives :#getting all false positives
  X_test_falsePositives = X_test_falsePositives.append(X_test.filter(items=[i], axis=0))
```

```
X_test_falsePositives.head(1)
```

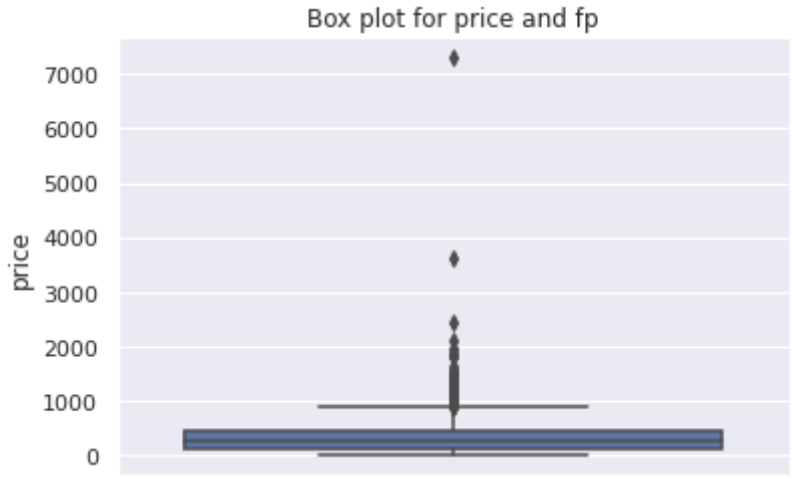
```
Unnamed: 0        id                            teacher_id  teacher_prefix  school_state
```

```
len(X_test_falsePositives)
```

```
826
```

Plotting the box plot with the `price` of these `false positive data points`

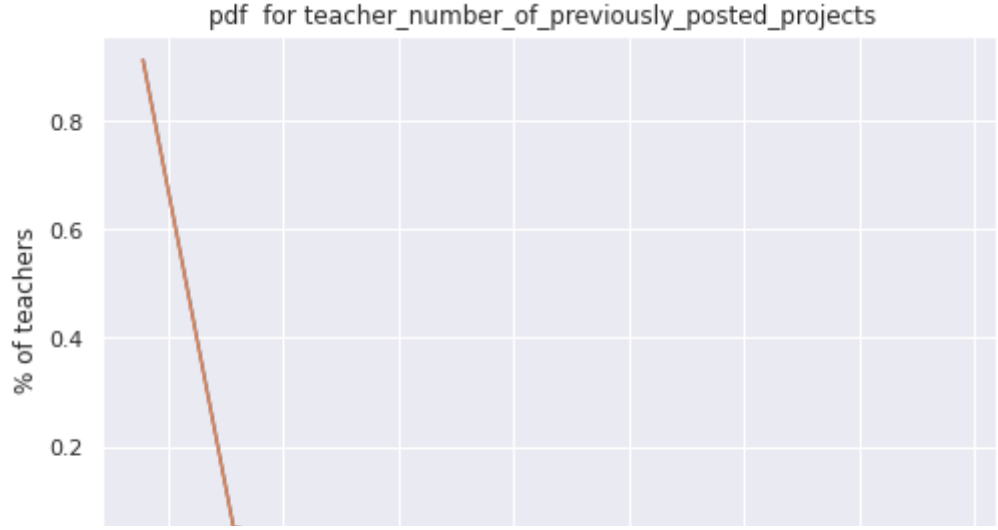
```
sns.boxplot(y='price', data=X_test_falsePositives).set_title("Box plot for price and fp")
plt.show()
```



Plotting the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

```
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_test_falsePositives['teacher_number_of_previously_poste
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)

plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:],pdf)



plt.title("pdf  for teacher_number_of_previously_posted_projects  ")
plt.xlabel("teacher_number_of_previously_posted_projects")
plt.ylabel("% of teachers")


plt.show();
```

```
[0.91041162 0.05326877 0.00605327 0.00726392 0.00726392 0.00363196
 0.00363196 0.00605327 0.         0.00242131]
[0.0 19.6 39.2 58.800000000000004 78.4 98.0 117.60000000000001
 137.20000000000002 156.8 176.4 196.0]
```



SET-2

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr2 = hstack((train_tfidf_w2v, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe

X_te2 = hstack((test_tfidf_w2v, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,X_t

print("Final Data matrix")
print(X_tr2.shape, y_train.shape)

print(X_te2.shape, y_test.shape)
print("="*100)
```

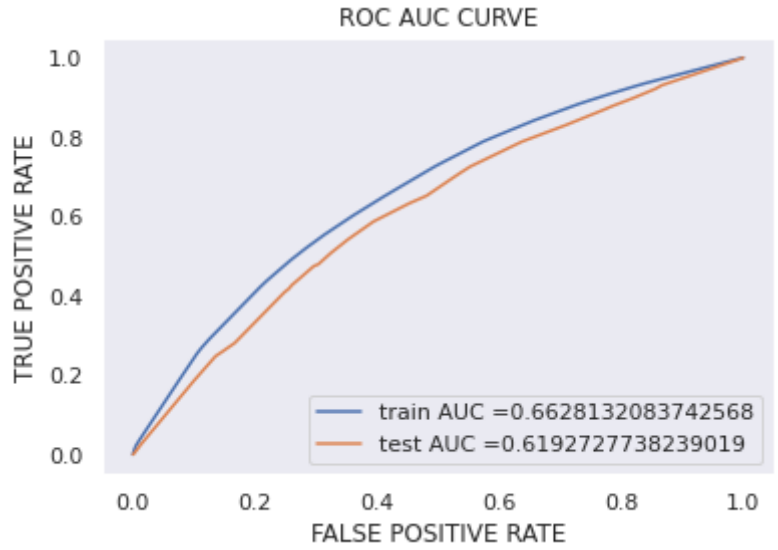
```
Final Data matrix
(33500, 790) (33500,)
(16500, 790) (16500,)
====================================================================================================
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.h
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import roc_auc_score

DT = DecisionTreeClassifier(class_weight = 'balanced')
params = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10, 100,500]}
clf1 = GridSearchCV(DT, params, cv= 3, scoring='roc_auc',verbose=1,return_train_score=True
clf1.fit(X_tr2, y_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']
```

```
cv_auc_std= clf1.cv_results_['std_test_score']
bestMaxDepth_2=clf1.best_params_['max_depth']
bestMinSampleSplit_2=clf1.best_params_['min_samples_split']
bestScore_2=clf1.best_score_
print("BEST MAX DEPTH: ",clf1.best_params_['max_depth']," BEST SCORE: ",clf1.best_score_,"
```

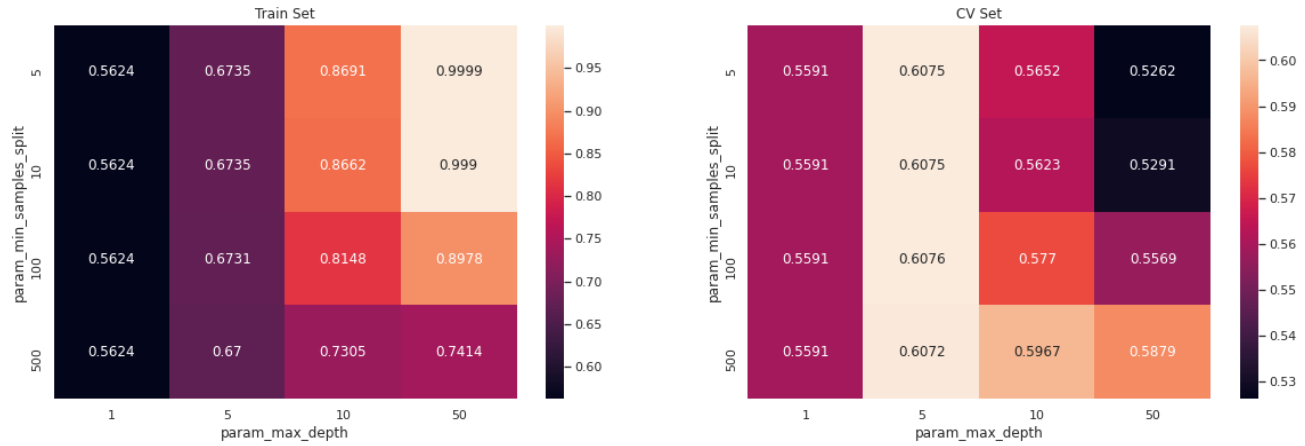
```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  48 out of  48 | elapsed:  9.9min finished
BEST MAX DEPTH:  5  BEST SCORE:  0.6075672291665991 BEST MIN SAMPLE SPLIT:  100
```

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc

clf1=DecisionTreeClassifier (class_weight = 'balanced',max_depth=5,min_samples_split=100)
clf1.fit(X_tr2, y_train)
# for visulation
clf1.fit(X_tr2, y_train)
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#
y_train_pred2 = clf1.predict_proba(X_tr2) [:,1]
y_test_pred2 = clf1.predict_proba(X_te2) [:,1]
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred2)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred2)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC AUC CURVE")
plt.grid()
plt.show()
```




```
import seaborn as sns; sns.set()
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import GridSearchCV
max_scores2 = pd.DataFrame(clf1.cv_results_).groupby(['param_min_samples_split', 'param_ma
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores2.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores2.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



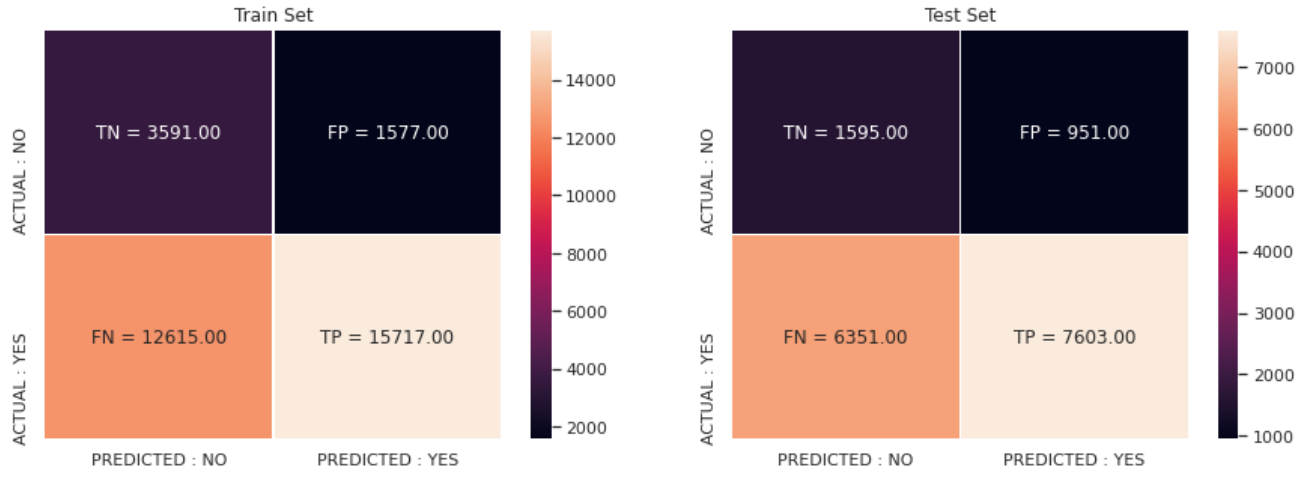

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high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

def predict_with_best_t(proba, threshould):
    predictions2 = []
    global predictions3
    for i in proba:
        if i>=threshould:
            predictions2.append(1)
        else:
            predictions2.append(0)
    predictions3=predictions2
    return predictions2


print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
```

```
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred2, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred2, best_t)))
```

```
========================================================================================================
the maximum value of tpr*(1-fpr) 0.3864700626232347 for threshold 0.495
Train confusion matrix
[[ 3591  1577]
 [12615 15717]]
Test confusion matrix
[[1595  951]
 [6351 7603]]
```

```
print("="*100)
from sklearn.metrics import confusion_matrix
import seaborn as sns; sns.set()
con_m_train=confusion_matrix(y_train, predict_with_best_t(y_train_pred2, best_t))
con_m_test=confusion_matrix(y_test, predict_with_best_t(y_test_pred2, best_t))
key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.f
sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],y
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```

```
========================================================================================================
the maximum value of tpr*(1-fpr) 0.3864700626232347 for threshold 0.495
```




```
false_positives1 = []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions3[i] == 1) :
    false_positives1.append(i)
fp_essay2 = []
```

```
for i in false_positives1 :
  fp_essay2.append(X_test['essay'].values[i])
```

Plotting the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these false positive data points

```
#COPIED FROM:https://www.geeksforgeeks.org/generating-word-cloud-python/

from wordcloud import WordCloud, STOPWORDS

comment_words = ' '
stopwords = set(STOPWORDS)

for val in fp_essay2 :
  val = str(val)
  tokens = val.split()

for i in range(len(tokens)):
  tokens[i] = tokens[i].lower()

for words in tokens :
  comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = st


plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

Creating Dataframe for Falsepositives


```
cols = X_test.columns
X_test_falsePositives1 = pd.DataFrame(columns=cols)
```


```
for i in false_positives1 :
  X_test_falsePositives1 = X_test_falsePositives1.append(X_test.filter(items=[i], axis=0))
```

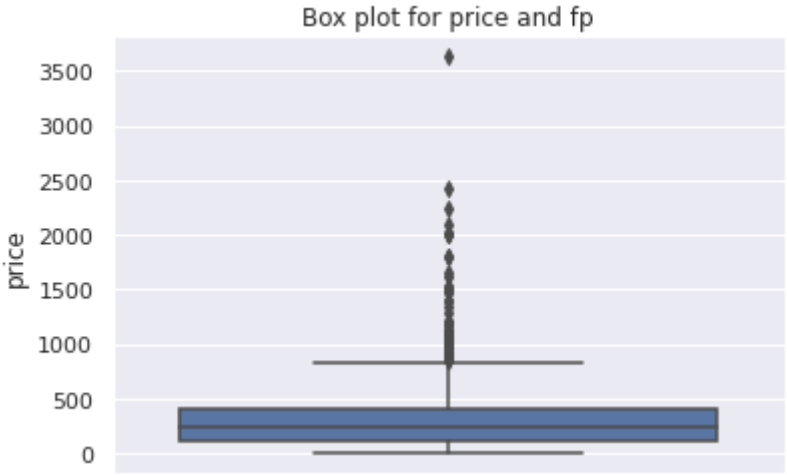


```
len(X_test_falsePositives1)
```

```
951
```

Plotting the box plot with the `price` of these `false positive data points`


```
sns.boxplot(y='price', data=X_test_falsePositives1).set_title("Box plot for price and fp")
plt.show()
```

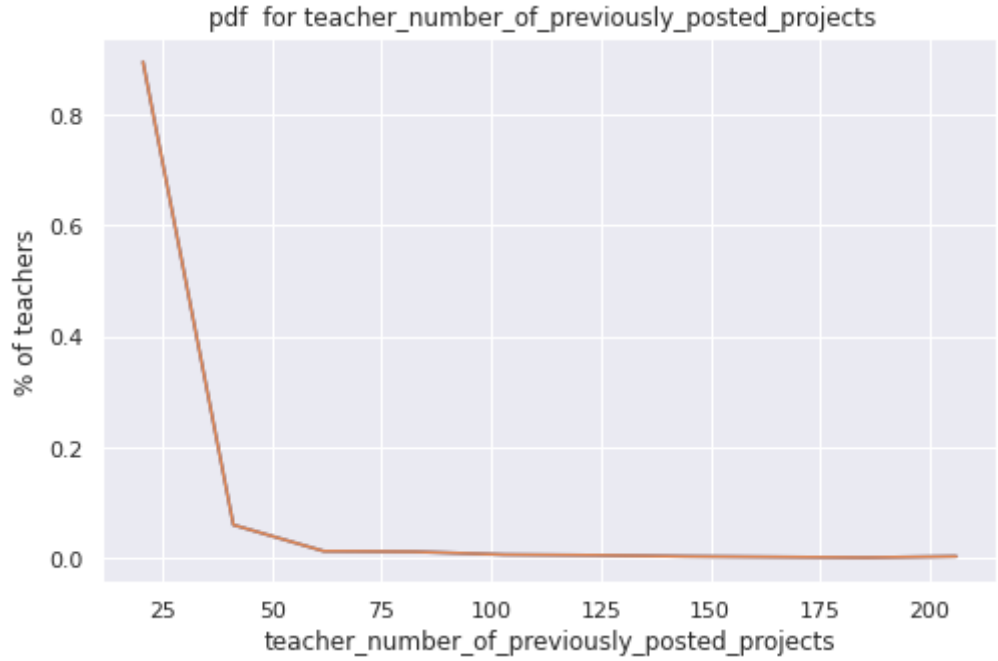


Plotting the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

```
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_test_falsePositives1['teacher_number_of_previously_post
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)

plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:],pdf)
```

```
plt.title("pdf  for teacher_number_of_previously_posted_projects  ")
plt.xlabel("teacher_number_of_previously_posted_projects")
plt.ylabel("% of teachers")


plt.show();
```

```
[0.89484753 0.05993691 0.0126183  0.01156677 0.00630915 0.00525762
 0.00315457 0.00210305 0.00105152 0.00315457]
[0.0 20.6 41.2 61.800000000000004 82.4 103.0 123.60000000000001
 144.20000000000002 164.8 185.4 206.0]
```



TASK-2

SELECTING ALL THE FEATURES BY KEEPING MAX_DEPTH DEFAULT NONE FOR SET-1 AND DOING HYPER PARAMETER TUNING

```
DT5 = DecisionTreeClassifier(class_weight = 'balanced')
params = { 'min_samples_split': [5, 10, 100,500]}
clf5 = GridSearchCV(DT, params, cv= 3, scoring='roc_auc',verbose=1,return_train_score=True
clf5.fit(X_tr1, y_train)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:  4.6min finished
GridSearchCV(cv=3, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
```

```
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             presort='deprecated',
                                             random_state=None,
                                             splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'min_samples_split': [5, 10, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=1)
```

Getting the feature importance using 'feature_importances_'

```
#https://datascience.stackexchange.com/questions/6683/feature-selection-using-feature-impo
#https://stackoverflow.com/questions/51682470/how-to-get-feature-importance-in-decision-tr
def selectKImportance(model,X,k=5):
    model.fit(X_tr1,y_train)
    return X[:,model.best_estimator_.feature_importances_.argsort()[::-1][:k]]
```

```
X_set5_train = selectKImportance(clf5, X_tr1,5000)
X_set5_test = selectKImportance(clf5, X_te1, 5000)
print(X_set5_train.shape)
print(X_set5_test.shape)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:  4.7min finished
Fitting 3 folds for each of 4 candidates, totalling 12 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:  4.6min finished
(33500, 5000)
(16500, 5000)
```

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr3 = hstack((X_set5_train, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,X_

X_te3 = hstack((X_set5_test, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,X_test

print("Final Data matrix")
print(X_tr3.shape, y_train.shape)

print(X_te3.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 5490) (33500,)
(16500, 5490) (16500,)
====================================================================================================
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.h
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import roc_auc_score

DT5 = DecisionTreeClassifier(class_weight = 'balanced')
params = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10, 100,500]}
clf3 = GridSearchCV(DT5, params, cv= 3, scoring='roc_auc',verbose=1,return_train_score=Tru
clf3.fit(X_tr3, y_train)

train_auc= clf3.cv_results_['mean_train_score']
train_auc_std= clf3.cv_results_['std_train_score']
cv_auc = clf3.cv_results_['mean_test_score']
cv_auc_std= clf3.cv_results_['std_test_score']
bestMaxDepth_3=clf3.best_params_['max_depth']
bestMinSampleSplit_3=clf3.best_params_['min_samples_split']
bestScore_3=clf3.best_score_
print("BEST MAX DEPTH: ",clf3.best_params_['max_depth']," BEST SCORE: ",clf3.best_score_,"
```

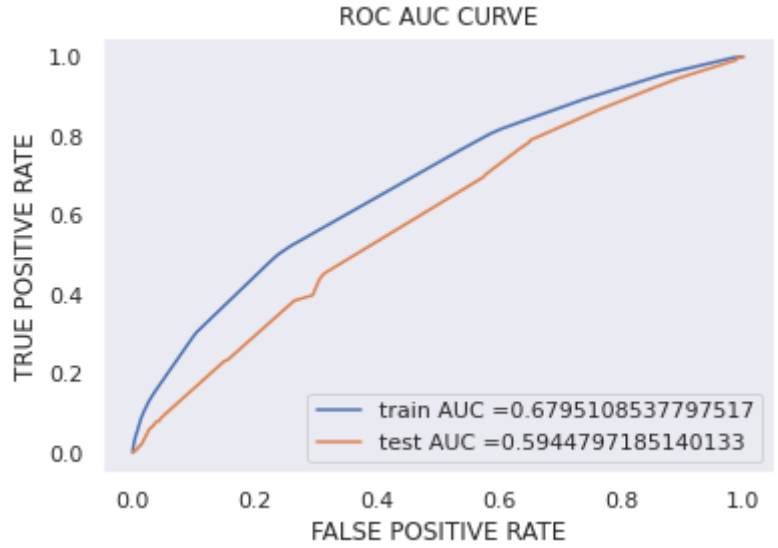
```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  48 out of  48 | elapsed:  2.8min finished
BEST MAX DEPTH:  10  BEST SCORE:  0.616748728717313 BEST MIN SAMPLE SPLIT:  500
```

```
print(clf3.best_estimator_)
```

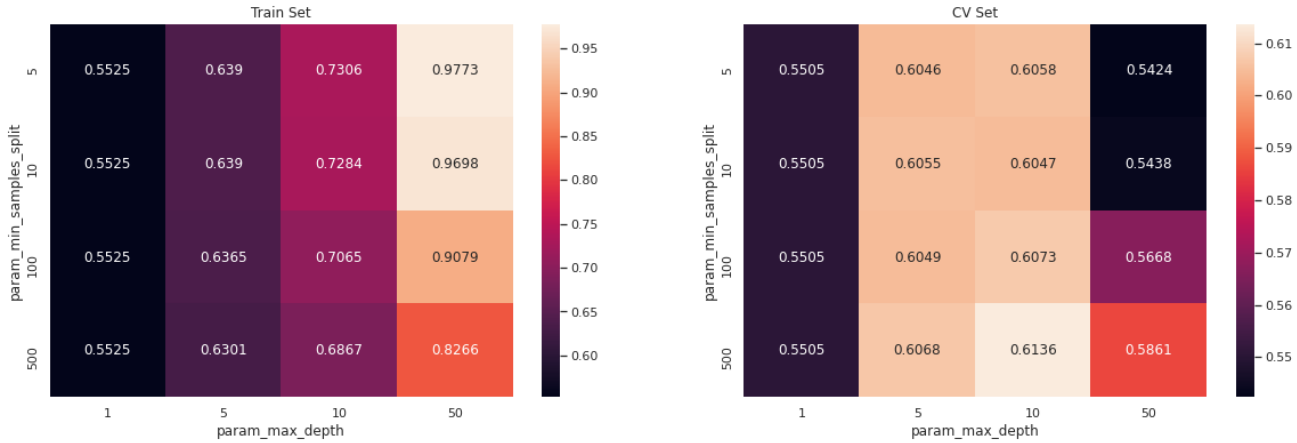
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc

clf3=DecisionTreeClassifier (class_weight = 'balanced',max_depth=10,min_samples_split=500)
clf3.fit(X_tr3, y_train)
# for visulation
clf3.fit(X_tr3, y_train)
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#
y_train_pred3 = clf3.predict_proba(X_tr3) [:,1]
y_test_pred3 = clf3.predict_proba(X_te3) [:,1]
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred3)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred3)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE")
plt.ylabel("TRUE POSITIVE RATE")
plt.title("ROC AUC CURVE")
```

```
plt.title("ROC AUC CURVE")
plt.grid()
plt.show()
```




```
import seaborn as sns; sns.set()
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
max_scores3 = pd.DataFrame(clf3.cv_results_).groupby(['param_min_samples_split', 'param_ma
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



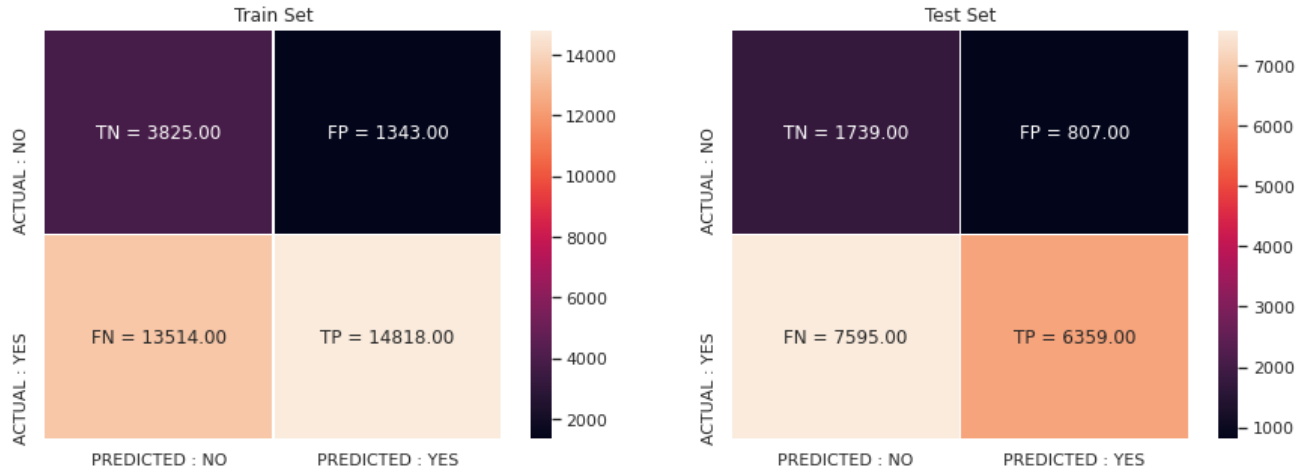

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
```

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

def predict_with_best_t(proba, threshould):
    predictions5 = []
    global predictions6
    for i in proba:
        if i>=threshould:
            predictions5.append(1)
        else:
            predictions5.append(0)
    predictions6=predictions5
    return predictions5


print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred3, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred3, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.3870983247508861 for threshold 0.495
Train confusion matrix
[[ 3825  1343]
 [13514 14818]]
Test confusion matrix
[[1739  807]
 [7595 6359]]
```

```
print("="*100)
from sklearn.metrics import confusion_matrix
import seaborn as sns; sns.set()
con_m_train=confusion_matrix(y_train, predict_with_best_t(y_train_pred3, best_t))
con_m_test=confusion_matrix(y_test, predict_with_best_t(y_test_pred3, best_t))
key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.f
sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],y
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```

```
==========================================================================================
the maximum value of tpr*(1-fpr) 0.3870983247508861 for threshold 0.495
```

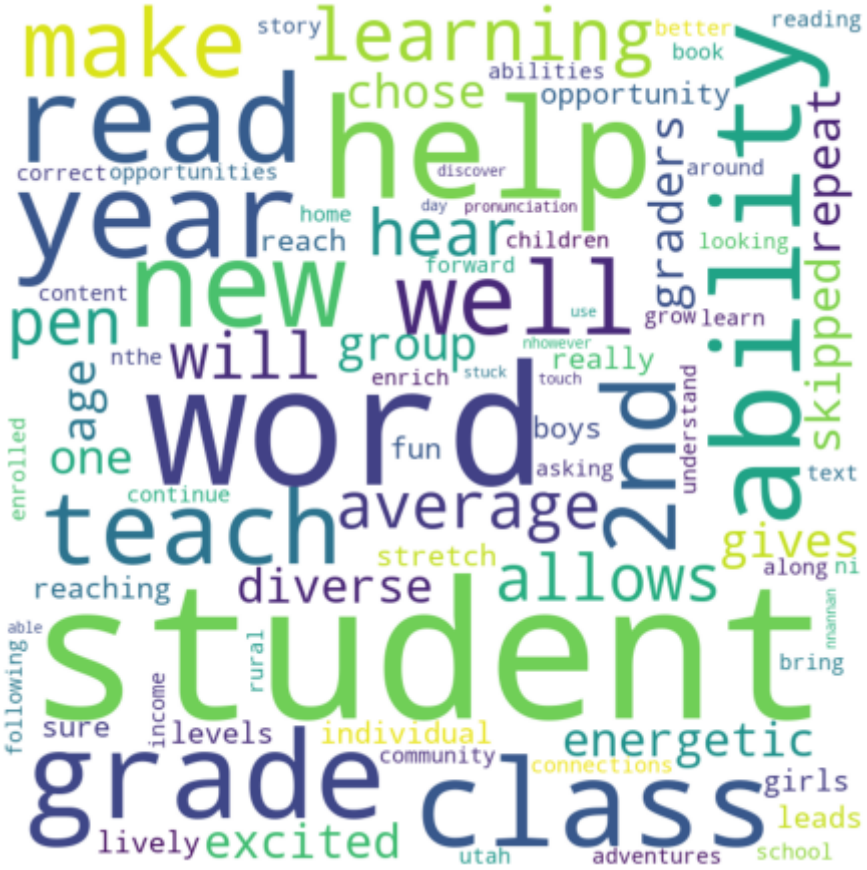



```
false_positives2 = []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions6[i] == 1) :
    false_positives2.append(i)
fp_essay3 = []
for i in false_positives2 :
  fp_essay3.append(X_test['essay'].values[i])
```

Plotting the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these `false positive data points`

```
#COPIED FROM:https://www.geeksforgeeks.org/generating-word-cloud-python/

from wordcloud import WordCloud, STOPWORDS

comment_words = ' '
stopwords = set(STOPWORDS)

for val in fp_essay3 :
  val = str(val)
  tokens = val.split()

for i in range(len(tokens)):
  tokens[i] = tokens[i].lower()

for words in tokens :
  comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = st


plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
```

```
plt.show()
```



Creating Dataframe with false positives

```
cols = X_test.columns
X_test_falsePositives2 = pd.DataFrame(columns=cols)
```

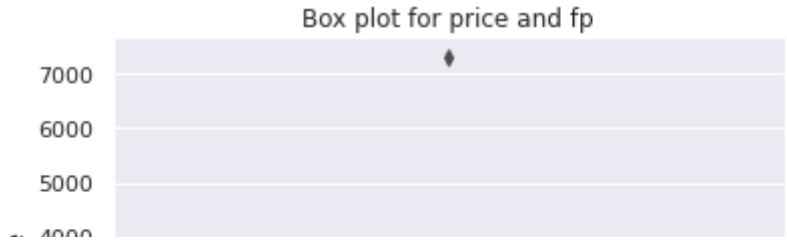
```
for i in false_positives2 :
  X_test_falsePositives2 = X_test_falsePositives2.append(X_test.filter(items=[i], axis=0))
```

```
len(X_test_falsePositives2)
```

```
807
```

Plot the box plot with the `price` of these `false positive data points`

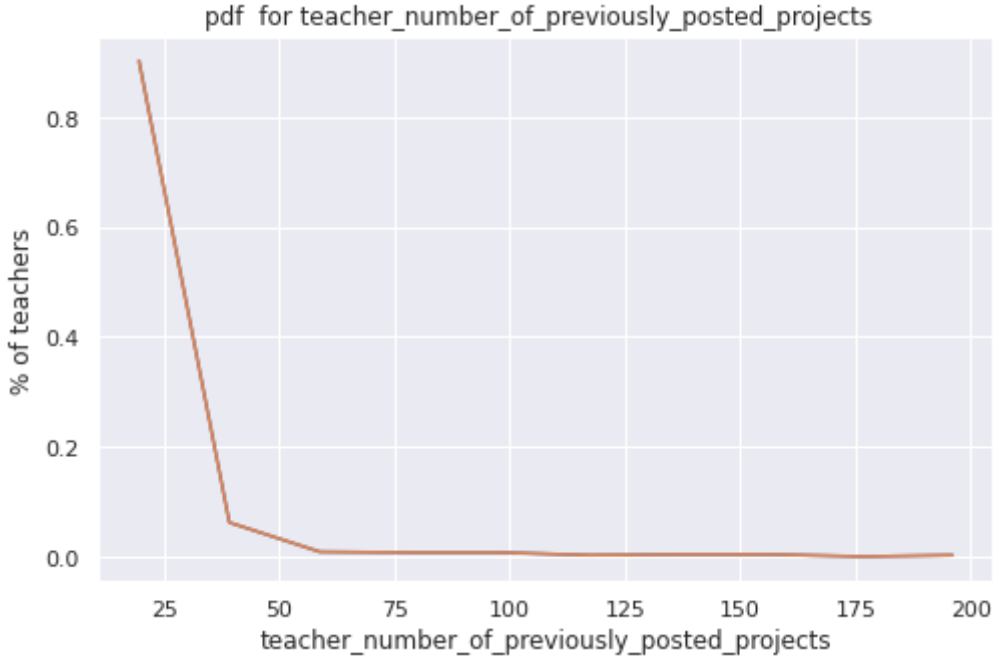
```
sns.boxplot(y='price', data=X_test_falsePositives2).set_title("Box plot for price and fp")
plt.show()
```



Plotting the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

```
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_test_falsePositives2['teacher_number_of_previously_post
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)

plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:],pdf)



plt.title("pdf  for teacher_number_of_previously_posted_projects  ")
plt.xlabel("teacher_number_of_previously_posted_projects")
plt.ylabel("% of teachers")


plt.show();
```

```
[0.90210657 0.06195787 0.0086741  0.00743494 0.00743494 0.00247831
 0.00371747 0.00371747 0.         0.00247831]
[0.0 19.6 39.2 58.800000000000004 78.4 98.0 117.60000000000001
 137.20000000000002 156.8 176.4 196.0]
```



SUMMARY

```
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer","MAX_DEPTH", "MIN_SAMPLE_SPLIT", "Test AUC"]

x.add_row(["TFIDF","10","500", 0.624])
x.add_row(["TFIDF_W2V", "5", "100",0.619])
x.add_row(["FEATURE_IMPORTANCE", "10", "500",0.594])
print(x)
```

```
+--------------------+-----------+------------------+----------+
|     Vectorizer     | MAX_DEPTH | MIN_SAMPLE_SPLIT | Test AUC |
+--------------------+-----------+------------------+----------+
|       TFIDF        |     10    |       500        |  0.624   |
|     TFIDF_W2V      |     5     |       100        |  0.619   |
| FEATURE_IMPORTANCE |     10    |       500        |  0.594   |
+--------------------+-----------+------------------+----------+
```