Assignment : DT

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

```
Mounted at /content/drive
```

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

import pandas as pd
import numpy as np
import math as m
import matplotlib.pyplot as plt
from scipy import sparse
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from tqdm import tqdm
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import Normalizer
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
import seaborn as sns
import nltk
import re
import pickle
```

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

Please check below video before attempting this assignment

```
from IPython.display import YouTubeVideo
YouTubeVideo('https://youtu.be/Ccy1UuhPY4A', width="10000",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-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

```
#please use below code to load glove vectors
import pickle

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

or else , you can use below code

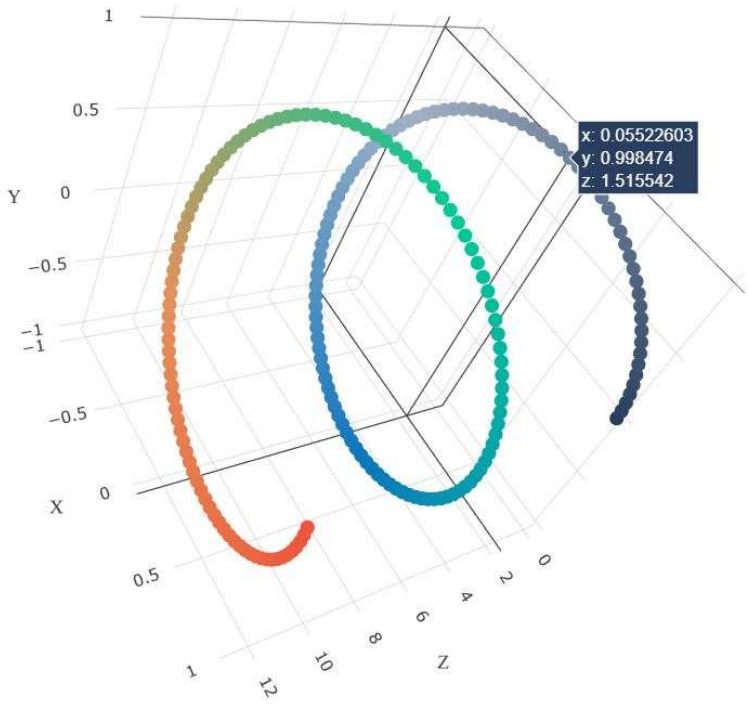
```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        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-
```

```
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(gloveFil
e,\'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) f
or val in splitLine[1:]])\n        model[word] = embedding\n    print ("Done.",len(mode
l)," words loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')
\n\n# ============================\nOutput:\n    \nLoading Glove Model\n1917495it [06:3
```

Task - 1

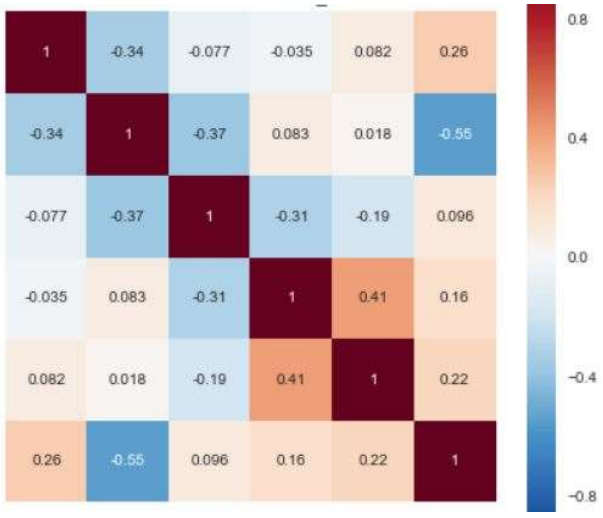
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)
    - Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)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

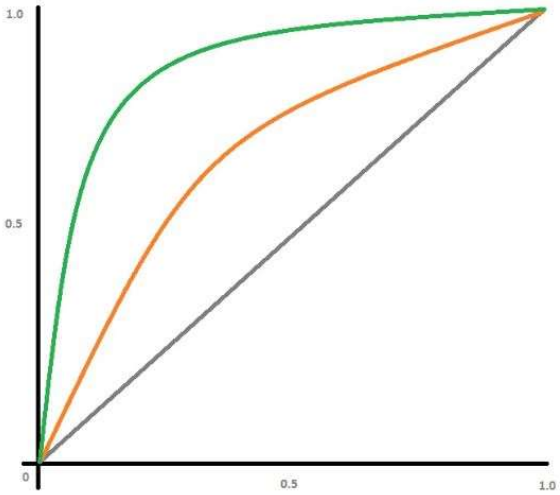
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



[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. 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.



- Along with plotting ROC curve, you need to print the [confusion matrix](https://www.google.com/search?q=confusion+matrix) 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')
```

Decision Tree

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('/content/preprocessed_data (1).csv')
```

```
# write your code in following steps for task 1
# 1. calculate sentiment scores for the essay feature
```

```
# 2. Split your data.
# 3. perform tfidf vectorization of text data.
# 4. perform tfidf w2v vectorization of text data.
# 5. perform encoding of categorical features.
# 6. perform encoding of numerical features
# 7. For task 1 set 1 stack up all the features
# 8. For task 1 set 2 stack up all the features (for stacking dense features you can use np.s
# 9. Perform hyperparameter tuning and plot either heatmap or 3d plot.
# 10. Find the best parameters and fit the model. Plot ROC-AUC curve(using predict proba meth
# 11. Plot confusion matrix based on best threshold value
# 12. Find all the false positive data points and plot wordcloud of essay text and pdf of tea
# 13. Write your observations about the wordcloud and pdf.

# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label


data = pd.read_csv('/content/preprocessed_data (1).csv')


data.head(2)
```

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

```
negative = []
positive = []
neutral = []
compound = []

def update_sentiments(values):
    negative.append(values["neg"])
    positive.append(values["pos"])
    neutral.append(values["neu"])
```

```
    compound.append(values["compound"])


from tqdm import tqdm
for essay in tqdm(data["essay"]):
    update_sentiments(sid.polarity_scores(essay))
```

```
 100%|██████████| 109248/109248 [02:49<00:00, 644.26it/s]
```

```
data["neg"] = negative
data["pos"] = positive
data["neu"] = neutral                # adding new features to dataset based on  Sentiment Int
data["compound"] = compound


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


sid = SentimentIntensityAnalyzer()


sentence= essay
ss_1 = sid.polarity_scores(sentence)
print('sentiment score for sentence 1',ss_1)
```

```
sentiment score for sentence 1 {'neg': 0.038, 'neu': 0.641, 'pos': 0.321, 'compound': 0
```

```
data.head(1)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_p
0	ca	mrs	grades_prek_2	

Splitting Data Into Train And Cross Validation(or test): Stratified Sampling

```
y = data['project_is_approved'].values
```

```
X = data.drop(['project_is_approved'], axis=1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y,random_s
```

```
print("Total data points in Train Dataset =",len(y_train))
print("Total data points in Test Dataset =",len(y_test))
```

```
Total data points in Train Dataset = 73196
Total data points in Test Dataset = 36052
```

Make Data Model Ready: Encoding Eassay(text feature)

TFIDF Vectorizer

```
#  HERE WE ARE INTILIZING THE TFIDF VECTORIZER SAMPLE OF DATASET 50000 AS PER INSTRUCTION

tfidfvectorizer = TfidfVectorizer(min_df=10,max_features=50000)
# FITTING THE  DATA

text_tfidf = tfidfvectorizer.fit(X_train['essay'].values) #fitting
# TRANSFORMING THE BOTH TRAIN AND TEST

X_train_essay_tfidf =tfidfvectorizer.transform(X_train['essay'].values)
X_test_essay_tfidf =tfidfvectorizer.transform(X_test['essay'].values)   # transform

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("==>"*50)
```

```
After vectorizations
(73196, 14266) (73196,)
(36052, 14266) (36052,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>=
```

TFIDF W2V

loading the data in pickle file

```
import pickle
```

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


# REFER : https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.T

tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].values)
# here we are intilizing the tfidf vectorizer on text feature essay

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



#TFIDF W2V for train dataset
train_tfidf_w2v_essays = [] # the tfidf-w2v for each essay is stored in this list

#  PERPOSE FOR USING TQDM LIBRARY IS THAT SEE THE PROGRESS OF CODE
for sentence in tqdm(X_train['essay']):

    # CREATING THE MATRIX OF SIZE 300 USING NUMPY.ZERO
    vector = np.zeros(300)

    # HERE WE WILL INTILIZE THE VARIABLE TFIDF
    tf_idf_weight =0;

    # HERE WE SPILLITED THE WORD IN SENTENCE TO PERFORM CALCULATE THE TERM FREQUENCY
    for word in sentence.split():

      # TAKING THE WORD FROM GLOVE VECTOR FILE

      # REFER : https://stackoverflow.com/questions/37793118/load-pretrained-glove-vectors-in
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]

            # HERE SIMPLE EXTUTION OF FORMULA COMPUTING THE TFIDF
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
# INCREMENTING THE VARIABLE VECTOR
            vector += (vec * tf_idf)

            # COMPUTING THE WEIGHTED TFIDF
            tf_idf_weight += tf_idf
# CHECKING THE tf_idf_weight NOT EQUAL TO 0
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
        # ADDINNG THE VECTOR
    train_tfidf_w2v_essays.append(vector)
X_train_essay_tfidf_w2v= sparse.csr_matrix(train_tfidf_w2v_essays)   # CALCULATING THE VALUE
print("After vectorizations")
```

```
print(X_train_essay_tfidf_w2v.shape, y_train.shape)
print("="*100)
```

```
100%|██████████| 73196/73196 [02:03<00:00, 590.84it/s]
After vectorizations
(73196, 300) (73196,)
==========================================================================================
```

```
#TFIDF W2V for test dataset
test_tfidf_w2v_essays = [] # the tfidf-w2v for each essay is stored in this list

#  PERPOSE FOR USING TQDM LIBRARY IS THAT SEE THE PROGRESS OF CODE
for sentence in tqdm(X_test['essay']):

      # CREATING THE MATRIX OF SIZE 300 USING NUMPY.ZERO
    vector = np.zeros(300)

  # HERE WE WILL INTILIZE THE VARIABLE TFIDF
    tf_idf_weight =0;

    # HERE WE SPILLITED THE WORD IN SENTENCE TO PERFORM CALCULATE THE TERM FREQUENCY
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]

    # HERE SIMPLE EXTUTION OF FORMULA COMPUTING THE TFIDF
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))

        # INCREMENTING THE VARIABLE VECTOR
            vector += (vec * tf_idf)

                # COMPUTING THE WEIGHTED TFIDF

            tf_idf_weight += tf_idf

            # CHECKING THE tf_idf_weight NOT EQUAL TO 0
    if tf_idf_weight != 0:

        vector /= tf_idf_weight
    test_tfidf_w2v_essays.append(vector)

     # CALCULATING THE VALUE TFIDF ON TRAIN DATASET
X_test_essay_tfidf_w2v= sparse.csr_matrix(test_tfidf_w2v_essays)


print("After vectoring")
print(X_test_essay_tfidf_w2v.shape, y_train.shape)
print("="*100)
```

```
100%|██████████| 36052/36052 [01:00<00:00, 594.25it/s]
```

```
After vectoring
(36052, 300) (73196,)
================================================================================
```

Encoding Categorical Features: School State

```
vectorizer_state = CountVectorizer()
vectorizer_state.fit(X_train['school_state'].values) # fitting

X_train_state_ohe = vectorizer_state.transform(X_train['school_state'].values)
X_test_state_ohe = vectorizer_state.transform(X_test['school_state'].values) #transform
print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print("==>"*50)
```

```
After vectorizations
(73196, 51) (73196,)
(36052, 51) (36052,)
==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>:
```

Encoding Categorical Features: teacher_prefix

```
vectorizer_prefix= CountVectorizer()
vectorizer_prefix.fit(X_train['teacher_prefix'].values) # fitting

X_train_teacher_ohe = vectorizer_prefix.transform(X_train['teacher_prefix'].values) #transfor
X_test_teacher_ohe = vectorizer_prefix.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("=>"*50)
```

```
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=:
```

Encoding Categorical Features: project_grade_category

```
vectorizer_grade = CountVectorizer()
vectorizer_grade.fit(X_train['project_grade_category'].values) # fitting

X_train_grade_ohe = vectorizer_grade.transform(X_train['project_grade_category'].values)
X_test_grade_ohe = vectorizer_grade.transform(X_test['project_grade_category'].values) #trans


print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print("=>"*50)
```

```
After vectorizations
(73196, 4) (73196,)
(36052, 4) (36052,)
=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=
```

Encoding Categorical Features: clean_categories

```
vectorizer_category = CountVectorizer()
vectorizer_category.fit(X_train['clean_categories'].values) # fitting

X_train_category_ohe = vectorizer_category.transform(X_train['clean_categories'].values)#tran
X_test_category_ohe = vectorizer_category.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_category_ohe.shape, y_train.shape)
print(X_test_category_ohe.shape, y_test.shape)
print("=>"*50)
```

```
After vectorizations
(73196, 9) (73196,)
(36052, 9) (36052,)
=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=
```

Encoding Categorical Features: clean_subcategories

```
vectorizer_sub = CountVectorizer()
vectorizer_sub.fit(X_train['clean_subcategories'].values) # fitting

X_train_subcategory_ohe = vectorizer_sub.transform(X_train['clean_subcategories'].values)  #t
```

```
X_test_subcategory_ohe = vectorizer_sub.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print("=>"*50)
```

```
After vectorizations
(73196, 30) (73196,)
(36052, 30) (36052,)
=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=:
```

Encoding Numerical Features

Encoding Numerical Feature :price

```
from sklearn.preprocessing import MinMaxScaler
price_scalar = MinMaxScaler()
X_train_price = price_scalar.fit_transform(X_train['price'].values.reshape(-1, 1))
X_test_price = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

```
print("After vectorizations")
print(X_train_price.shape, y_train.shape)
print(X_test_price.shape, y_test.shape)
print("=>"*50)
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=>=:
```

Encoding Numerical Features:teacher_number_of_previously_posted_projects

```
from sklearn.preprocessing import MinMaxScaler       # USING MINMAX SCALER TO AVOID DIMENSION
teacher_number_of_previously_posted_projects_scaler = MinMaxScaler()     # HERE WE ARE SCALLI
```

```
 # USING FIT_TRANFORM TO FITTING THE DATA
X_train_teacher_number_of_previously_posted_projects = teacher_number_of_previously_posted_pr
X_test_teacher_number_of_previously_posted_projects = teacher_number_of_previously_posted_pro

print(" after vecterazation ")
("=>"*100)
print(X_train_teacher_number_of_previously_posted_projects.shape , y_train.shape)       # PRI
print(X_test_teacher_number_of_previously_posted_projects.shape , y_test.shape)
```

```
 after vecterazation
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Numerical Features:neg

```
from sklearn.preprocessing import MinMaxScaler         # USING MINMAX SCALER TO AVOID DIMENSI
neg_scalar = MinMaxScaler()                                          # HERE WE ARE SCALLING T

# USING FIT_TRANFORM TO FITTING THE DATA
X_train_neg = neg_scalar.fit_transform(X_train['neg'].values.reshape(-1, 1))
X_test_neg = neg_scalar.transform(X_test['neg'].values.reshape(-1, 1))
```

```
print("After vectorizations")
print('='* 100)
print(X_train_neg.shape, y_train.shape)  # PRINT THE SHAPE
print(X_test_neg.shape, y_test.shape)
```

```
After vectorizations
====================================================================================================
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Numerical Features:pos

```
pos_scaler = MinMaxScaler()     # # USING MINMAX SCALER TO AVOID DIMENSIONAL ERROR

# HERE WE ARE SCALLING THE FEATURE
X_train_pos = pos_scaler.fit_transform(X_train['pos'].values.reshape(-1,1))
X_test_pos = pos_scaler.transform(X_test['pos'].values.reshape(-1,1))
```

```
print(" after vectorization")

print(X_train_pos.shape , y_train.shape)           # PRINT THE SHAPE
print(X_test_pos.shape , y_test.shape)
```

```
 after vectorization
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Numerical Features:neu

```
neu_scaler = MinMaxScaler()                    # USING MINMAX SCALER TO AVOID DIMENSIONAL ERR

 # HERE WE ARE SCALLING THE FEATURE
X_train_neu = neu_scaler.fit_transform(X_train['neu'].values.reshape(-1,1))
X_test_neu = neu_scaler.transform(X_test['neu'].values.reshape(-1,1))

print(" after vectorization")

print(X_train_neu.shape , y_train.shape)     # PRINT THE SHAPE
print(X_test_neu.shape , y_test.shape)
```

```
 after vectorization
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Numerical Features:compound

```
compound_scaler = MinMaxScaler()             # # USING MINMAX SCALER TO AVOID DIMENSIONAL ERR

# HERE WE ARE SCALLING THE FEATURE
X_train_compound = compound_scaler.fit_transform(X_train['compound'].values.reshape(-1,1))
X_test_compound = compound_scaler.transform(X_test['compound'].values.reshape(-1,1))

print(" after vectorization")

print(X_train_compound.shape , y_train.shape)          # PRINT THE SHAPE
print(X_test_compound.shape , y_test.shape)
```

```
 after vectorization
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Concatinating All The Features

SET-1

Set 1: categorical, numerical features + preprocessed_eassay (TFIDF)

```
from scipy.sparse import hstack
X_tr_set_one = hstack((X_train_essay_tfidf,
                       X_train_state_ohe,
                       X_train_teacher_ohe,
                       X_train_grade_ohe,
                       X_train_price,                    # STACKING THE ALL TRAIN TFIDF FEATUR
                       X_train_category_ohe,
                       X_train_subcategory_ohe,
                       X_train_teacher_number_of_previously_posted_projects,
                       X_train_neg,
                       X_train_pos,
                       X_train_neu,
                       X_train_compound)).tocsr()


X_te_set_one = hstack((X_test_essay_tfidf,
                       X_test_state_ohe,
                       X_test_teacher_ohe,
                       X_test_grade_ohe,
                       X_test_price,                    # STACKING ALL TEST TFIDF FEATURES
                       X_test_category_ohe,
                       X_test_subcategory_ohe,
                       X_test_teacher_number_of_previously_posted_projects,
                       X_test_neg,
                       X_test_pos,
                       X_test_neu,
                       X_test_compound)).tocsr()


print("SHAPE OF TRAIN AND TEST AFTER STACKING")
print(X_tr_set_one.shape, y_train.shape)
print(X_te_set_one.shape, y_test.shape)
print("="*100)
```

```
SHAPE OF TRAIN AND TEST AFTER STACKING
(73196, 14371) (73196,)
(36052, 14371) (36052,)
====================================================================================================
```

```
from scipy.sparse import hstack
```

```
X_tr_set_two = hstack((X_train_essay_tfidf_w2v,
                       X_train_state_ohe,
                       X_train_teacher_ohe,
                       X_train_grade_ohe,
                       X_train_price,                 # STACKING ALL TRAIN TFIDF+W2V FEATURES
                       X_train_category_ohe,
                       X_train_subcategory_ohe,
                       X_train_teacher_number_of_previously_posted_projects,
                       X_train_neg,
                       X_train_pos,
                       X_train_neu,
                       X_train_compound)).tocsr()
```

```
X_te_set_two = hstack((X_test_essay_tfidf_w2v,
                       X_test_state_ohe,
                       X_test_teacher_ohe,               # STACKING ALL TEST  TFIDF+W2V FEATU
                       X_test_grade_ohe,
                       X_test_price,
                       X_test_category_ohe,
                       X_test_subcategory_ohe,
                       X_test_teacher_number_of_previously_posted_projects,
                       X_test_neg,
                       X_test_pos,
                       X_test_neu,
                       X_test_compound)).tocsr()
```

```
print("SHAPE OF TRAIN AND TEST AFTER STACKING")
print(X_tr_set_two.shape,  y_train.shape)
print(X_te_set_two.shape,  y_test.shape)
```

```
SHAPE OF TRAIN AND TEST AFTER STACKING
(73196, 405) (73196,)
(36052, 405) (36052,)
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
tree_parameters = {'max_depth': [1, 5, 10, 50],   # MAX DEPTH WE HAVE TAKEN 50
                   'min_samples_split': [5, 10, 100, 500]}   #  NUMBER OF min_samples_split
```

```
decision_tree= DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(decision_tree, tree_parameters, cv=5, scoring='roc_auc', return_train_scor
clf.fit(X_tr_set_one,y_train)       # FITTING THE STACKED X_TRAIN AND Y_TRAIN INTO OUR ALGO
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(class_weight='balanced'),
             n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc')
```

```
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']
```

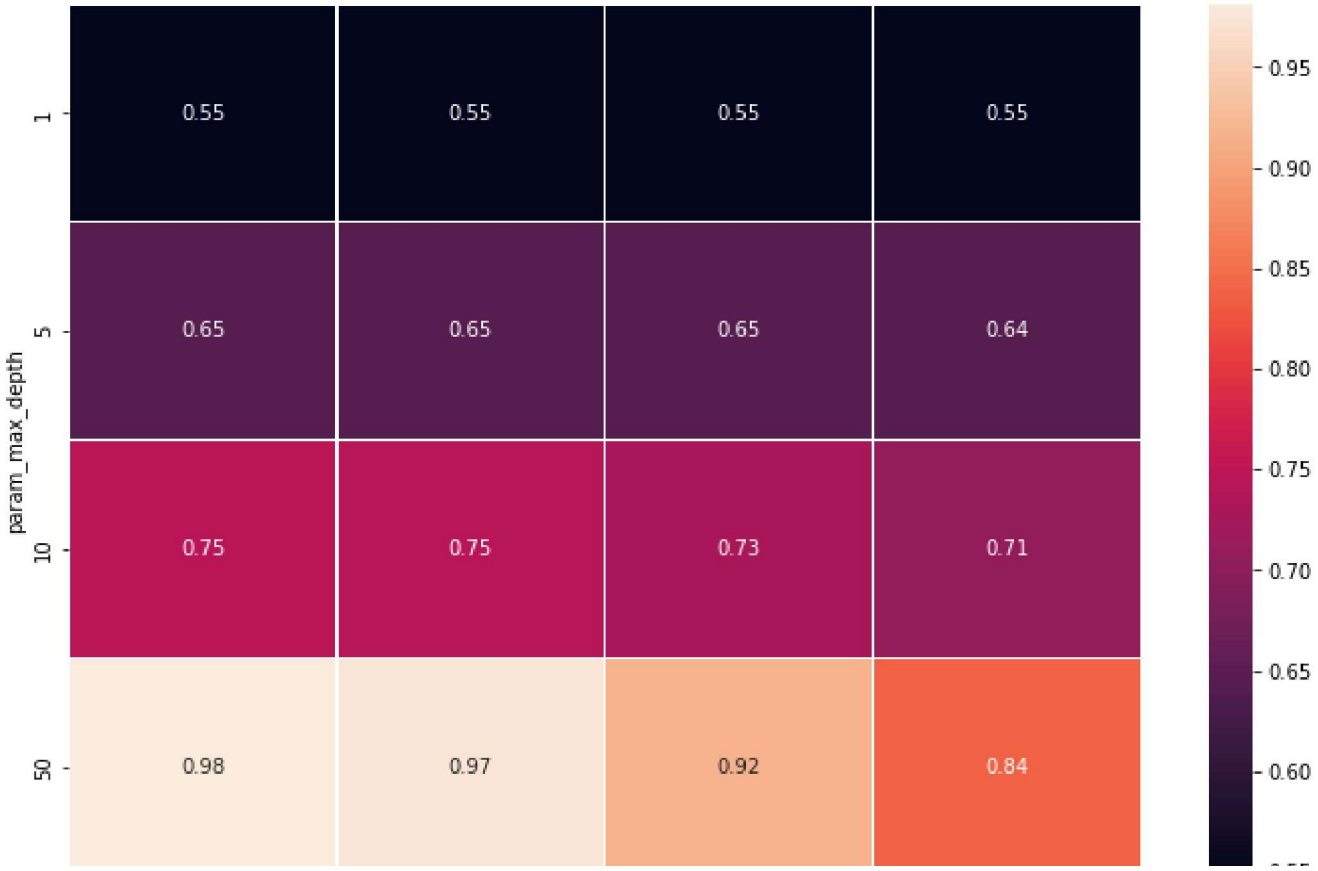
```
#HERE I M TRYING TO GET THE BEST VALUE FOR ALGO
print('Best score: ',clf.best_score_)

# THE I M TRYING TO FINDOUT THAT WHAT SHOULD BE BEST VALUE FOR GBDT PARAMETERS VALUE
print('Best Hyper parameters: ',clf.best_params_)
```

```
Best score:  0.6486284489716533
Best Hyper parameters:  {'max_depth': 10, 'min_samples_split': 500}
```

Plotting Hyperparameter v/s Auc

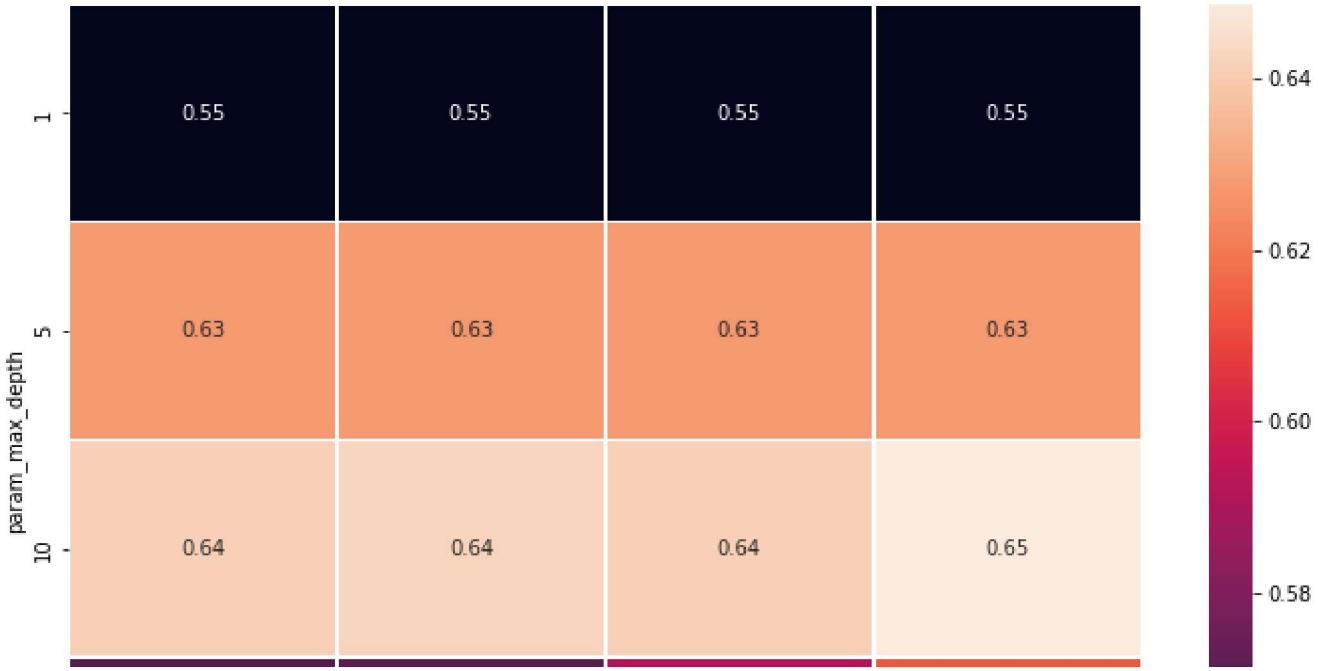
Roc Plot Of Train And Test Data Train data

```
# HERE WE WILL TRY TO SHOW HEAT MAP FOR

# REFER : https://stackoverflow.com/questions/48791709/how-to-plot-a-heat-map-on-pivot-table-

pvt = pd.pivot_table(pd.DataFrame(clf.cv_results_),
    values='mean_train_score', index='param_max_depth',
     columns='param_min_samples_split')

 # GIVING SIZE
plt.figure(figsize=(12,8))
ax=sns.heatmap(pvt,annot=True,linewidths=.10)
```



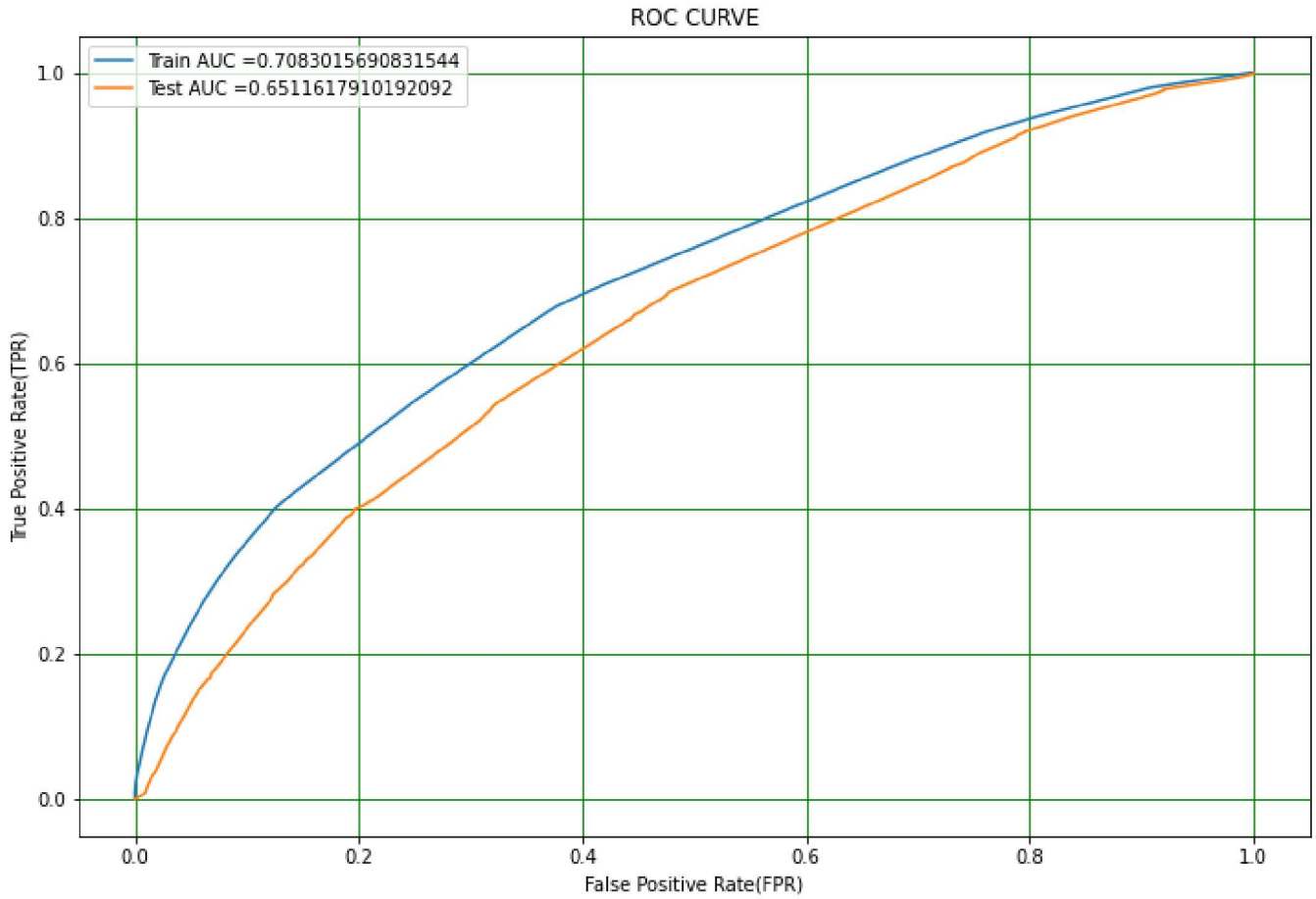
Heat map on test data

```
import pandas as pd

# HERE PLOTTING HEAT MAP FOR TEST DATA


 #https://stackoverflow.com/questions/48791709/how-to-plot-a-heat-map-on-pivot-table-after-gr

pvt = pd.pivot_table(pd.DataFrame(clf.cv_results_),
    values='mean_test_score', index='param_max_depth',
    columns='param_min_samples_split') #https://stackoverflow.com/questions/48791709/how-to-p

    # GIVING SIZE TO HEATMAP
plt.figure(figsize=(12,8))
ax=sns.heatmap(pvt,annot=True,linewidths=.5)
```



Roc Plot Of Train And Test Data

```
model_set1=DecisionTreeClassifier(class_weight='balanced',max_depth = clf.best_params_["max_d
model_set1.fit(X_tr_set_one,y_train)

# converting train and test output into probability

y_train_probs = clf.predict_proba(X_tr_set_one)[:,1] # converting train and test output into
y_test_probs= clf.predict_proba(X_te_set_one )[:,1]

 # storing values of fpr and tpr

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_probs) # storing values of f
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_probs)

# NOW WE LABEL THE PLOT X AND Y
plt.figure(figsize=(12,8))
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(FPR)")
plt.ylabel("True Positive Rate(TPR)")

# TITLE OF THE CURVE IS ROC CURVE
plt.title("ROC CURVE")

# DEFINING THE GRID PARAMETERS
plt.grid(color='green',lw=0.8)
```



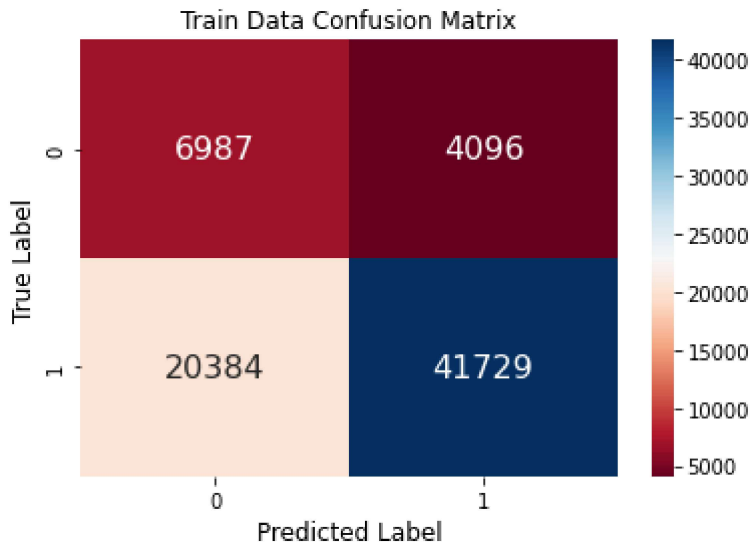
confusion matrix

```
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

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3
    return t

# DEFINING THE THRESHOLD VALUE IF VALUE GREATHER THAN THRESHOLD THEN 1 AND IF ITS LESS THAN T
def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

TRAIN DATA

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
# PLOTTING THE CONFUSION MATRIX OF TRAIN DATA

confusion_matrix=metrics.confusion_matrix(y_train,predict_with_best_t(y_train_probs, best_t))

print("CONFUSION MATRIX OF TRAIN DATA")
print("\n")
print(confusion_matrix)
sns.heatmap(confusion_matrix, annot=True, fmt='d',cmap='RdBu',annot_kws = {"size":16})
plt.ylabel('True Label',size=12)
plt.xlabel('Predicted Label',size=12)
plt.title('Train Data Confusion Matrix',size=12)
```

```
the maximum value of tpr*(1-fpr) 0.4235345867948362 for threshold 0.508
CONFUSION MATRIX OF TRAIN DATA


[[ 6987  4096]
 [20384 41729]]
Text(0.5, 1.0, 'Train Data Confusion Matrix')
```

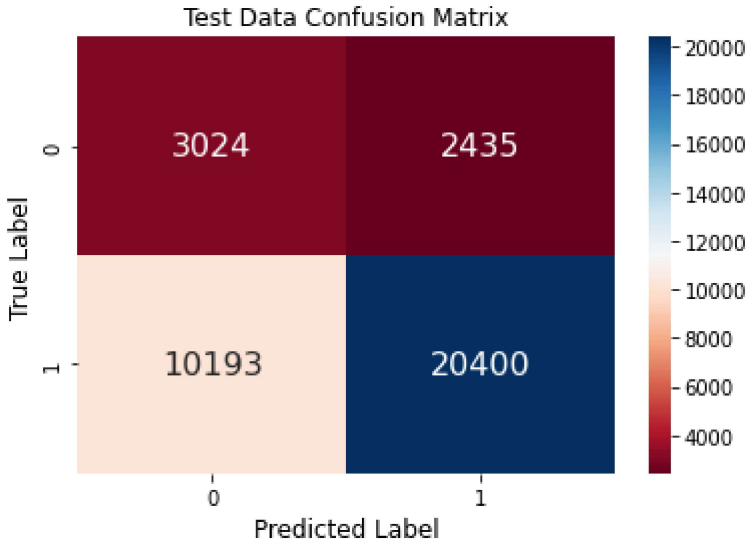


Test Data

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
confusion_matrix=metrics.confusion_matrix(y_test,predict_with_best_t(y_test_probs, best_t))
# PLOTTING THE CONFUSION MATRIX OF TRAIN DATA
```

```
print("CONFUSION MATRIX OF TEST DATA")
print('\n')
print(confusion_matrix)
sns.heatmap(confusion_matrix, annot=True, fmt='d',cmap='RdBu',annot_kws = {"size":16})
plt.ylabel('True Label',size=12)
plt.xlabel('Predicted Label',size=12)
plt.title('Test Data Confusion Matrix',size=12)
```

```
the maximum value of tpr*(1-fpr) 0.3693829056590241 for threshold 0.431
CONFUSION MATRIX OF TEST DATA


[[ 3024  2435]
 [10193 20400]]
Text(0.5, 1.0, 'Test Data Confusion Matrix')
```



Getting All the False Positive Data Points

```
predict=predict_with_best_t(y_test_probs,best_t)
```

```
fpi = []
for i in range(len(y_test)):
    if(y_test[i]==0) & (predict[i] == 1):
        fpi.append(i) # GETTING THE ALL FALSE POSITIVE INDICES
len(fpi)
```

```
2435
```

```
import pandas as pd
cols = X_test.columns
X_test_false_Positive = pd.DataFrame(columns=cols) # MAKING THE FALSE POSITIVE DATAFRAME
X_test_false_Positive=X_test.iloc[fpi]
print(X_test_false_Positive.shape)
```

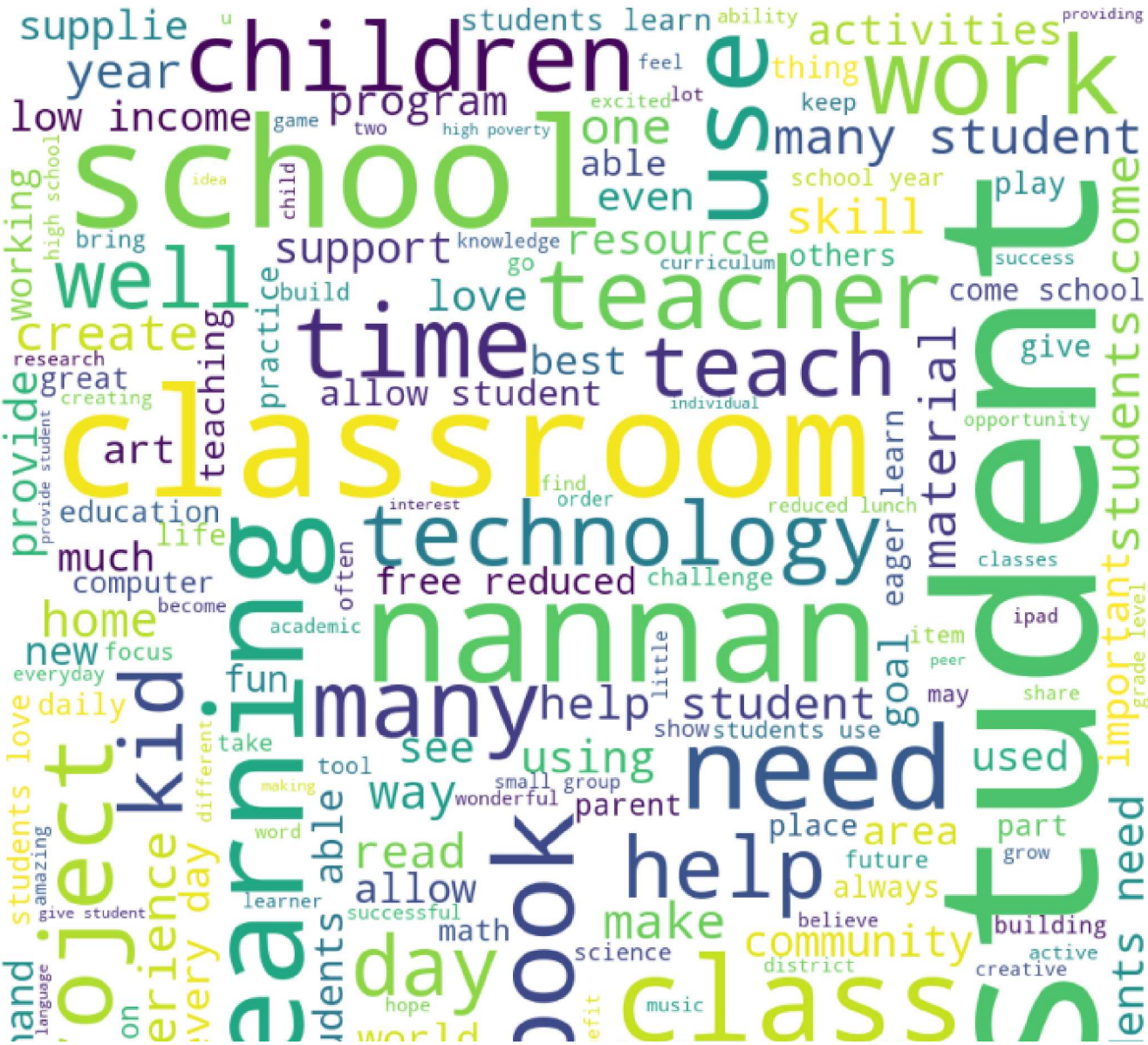
```
(2435, 12)
```

```
X_test_false_Positive.head(1)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previous
69487	sc	mrs	grades_3_5	

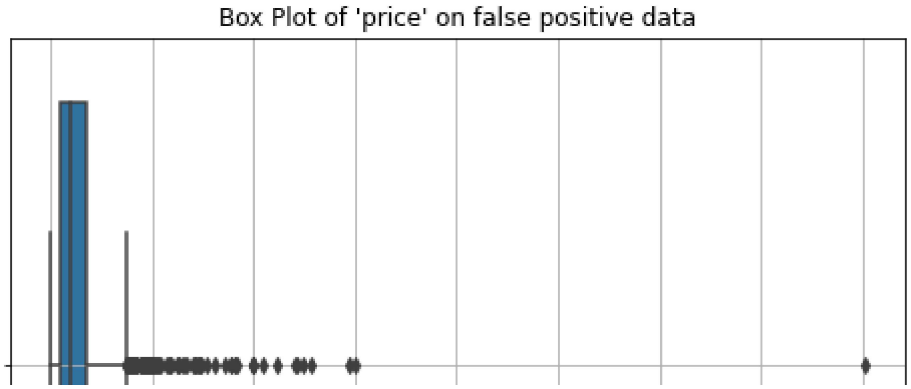
wordcloud Of Essay Text For False Positive Dataset

```
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for word in X_test_false_Positive['essay']:
    val = str(word)                #https://www.geeksforgeeks.org/generating-word-cloud-pytho
    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 = 900, height = 900, background_color ='white', stopwords = stopw



plt.figure(figsize = (10, 12), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

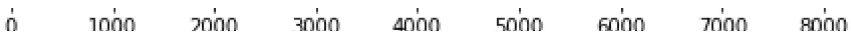


Box Plot With The Price Of These False Positive Data Points

```
plt.figure(figsize=(8,6))
sns.boxplot('price',data=X_test_false_Positive,orient="v").set_title("Box Plot of 'price' on
plt.grid()
```



Pdf Plot With The teacher_number_of_previously_posted_projects Of These False Positive Data Points




```
plt.figure(figsize=(8,6))
plt.grid()
counts, bin_edges = np.histogram(X_test_false_Positive['teacher_number_of_previously_posted_p
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)
plt.plot(bin_edges[1:],pdf,color="green")
plt.title("pdf of   ' teacher_number_of_previously_posted_projects' ")
plt.xlabel('teacher_number_of_previously_posted_projects')
```

```
[8.73511294e-01 7.31006160e-02 2.58726899e-02 9.85626283e-03
 8.62422998e-03 2.87474333e-03 3.28542094e-03 1.64271047e-03
 4.10677618e-04 8.21355236e-04]
[  0.   24.8  49.6  74.4  99.2 124.  148.8 173.6 198.4 223.2 248. ]
Text(0.5, 0, 'teacher_number_of_previously_posted_projects')
```

DECISION TREE USING GRID SEARCH CROSS VALIDATION (SET - 2)


```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
tree_parameters = {'max_depth': [1, 5, 10, 50],
                   'min_samples_split': [5, 10, 100, 500]}

decision_tree= DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(decision_tree, tree_parameters, cv=5, scoring='roc_auc', return_train_scor
clf.fit(X_tr_set_two,y_train)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(class_weight='balanced'),
             n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc')
```

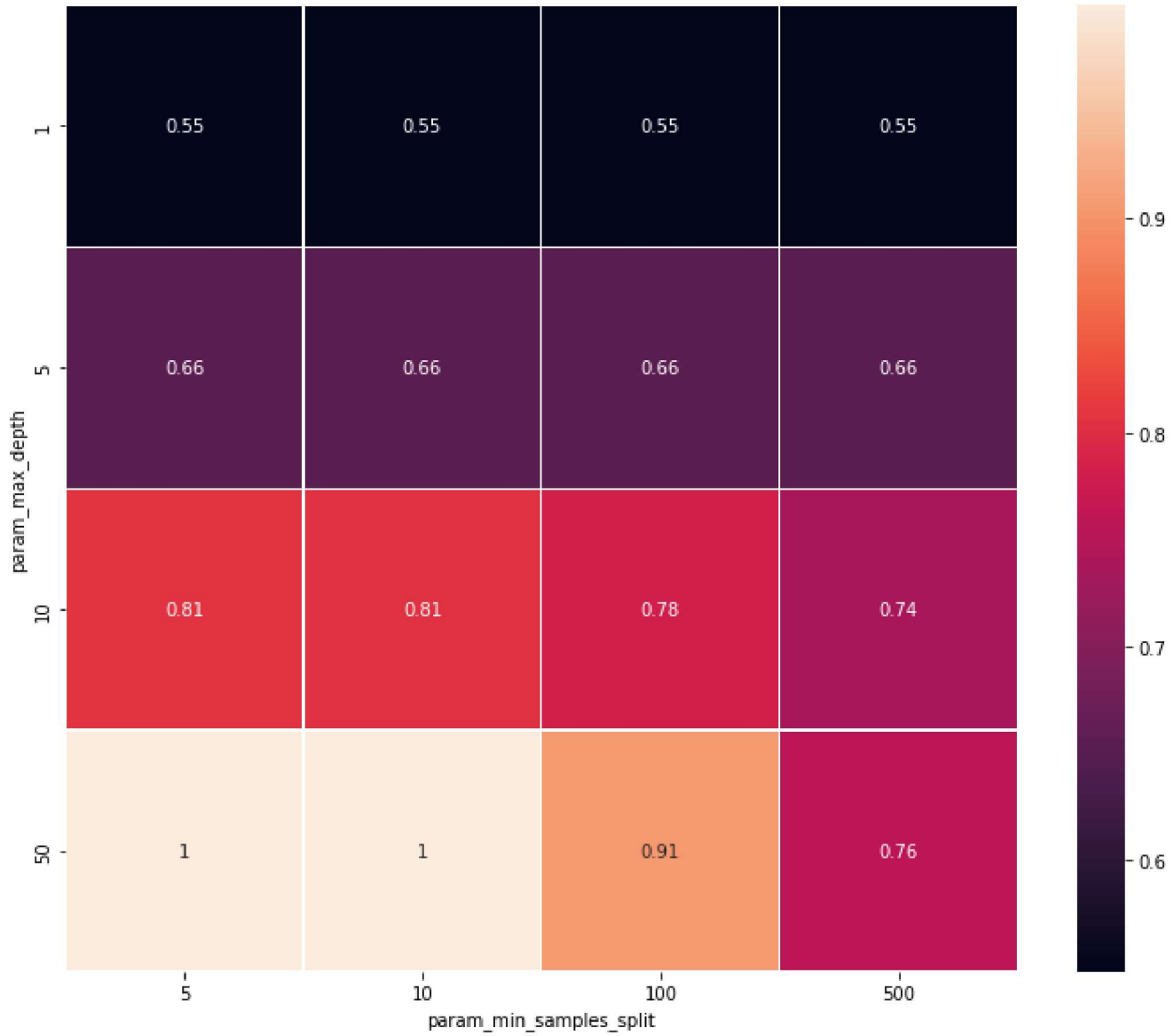
```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']


#HERE I M TRYING TO GET THE BEST VALUE FOR ALGO
print('Best score: ',clf.best_score_)

# THE I M TRYING TO FINDOUT THAT WHAT SHOULD BE BEST VALUE FOR GBDT PARAMETERS VALUE
print('Best Hyper parameters: ',clf.best_params_)
```

```
Best score:  0.6306250399675853
Best Hyper parameters:  {'max_depth': 5, 'min_samples_split': 500}
```

Plotting Hyperparameter v/s Auc

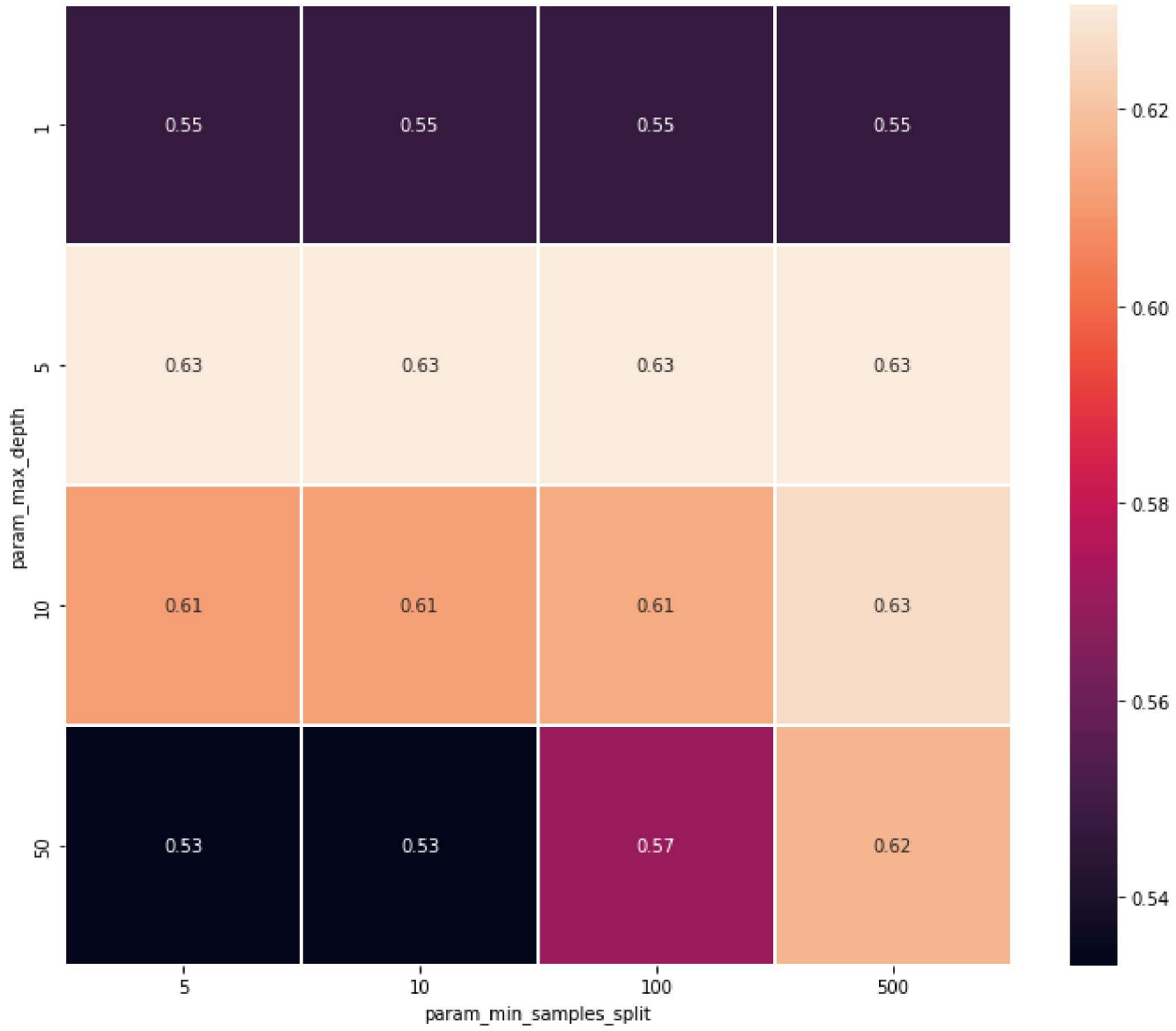
```
# HERE WE WILL TRY TO SHOW HEAT MAP FOR

# REFER : https://stackoverflow.com/questions/48791709/how-to-plot-a-heat-map-on-pivot-table-
```

```
pvt = pd.pivot_table(pd.DataFrame(clf.cv_results_),
    values='mean_train_score', index='param_max_depth',
     columns='param_min_samples_split')

 # GIVING SIZE
plt.figure(figsize=(12,10))
ax=sns.heatmap(pvt,annot=True,linewidths=.10)
```




```
import pandas as pd

# HERE PLOTTING HEAT MAP FOR TEST DATA


 #https://stackoverflow.com/questions/48791709/how-to-plot-a-heat-map-on-pivot-table-after-gr

pvt = pd.pivot_table(pd.DataFrame(clf.cv_results_),
    values='mean_test_score', index='param_max_depth',
    columns='param_min_samples_split') #https://stackoverflow.com/questions/48791709/how-to-p
```

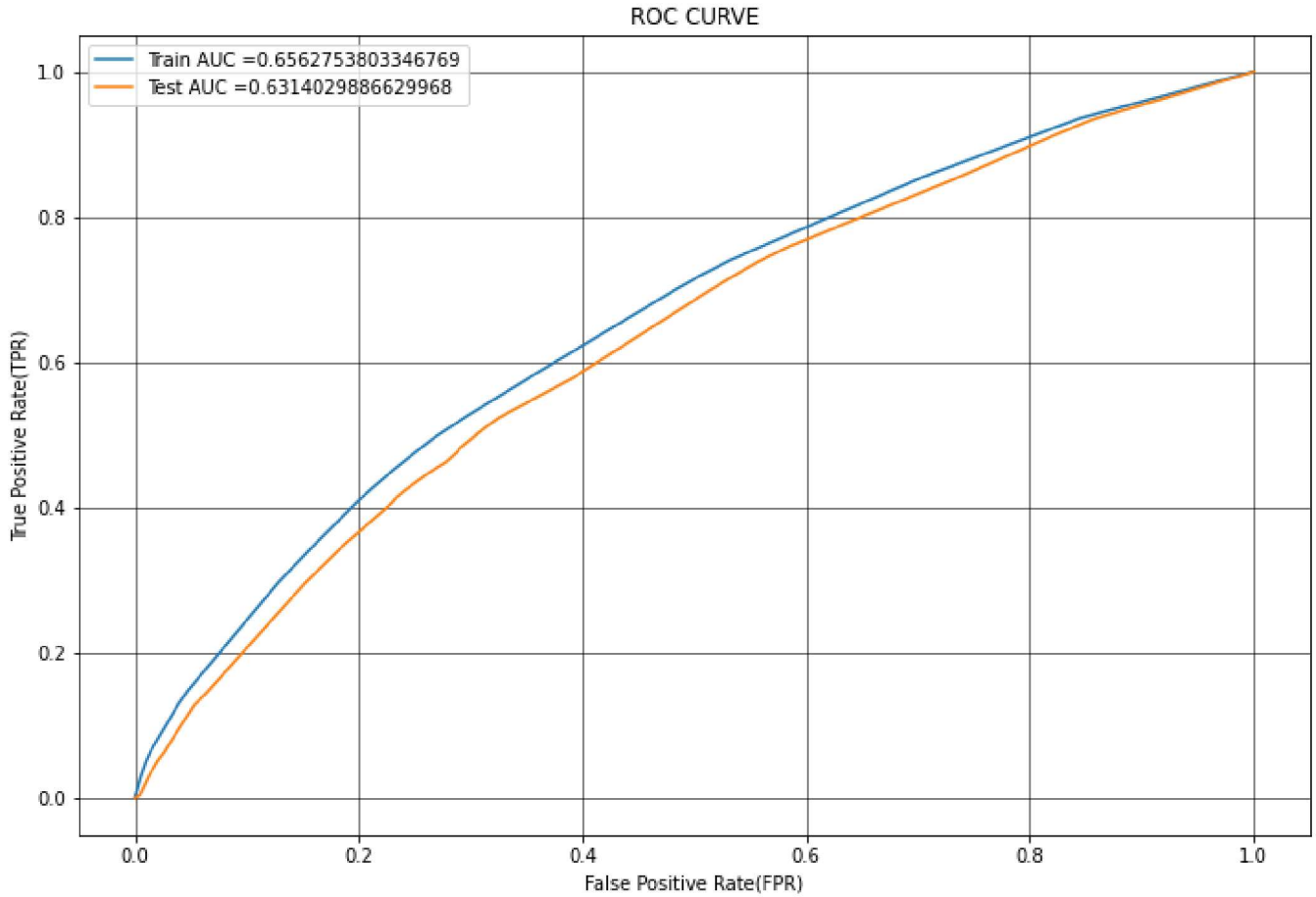
```
    # GIVING SIZE TO HEATMAP
plt.figure(figsize=(12,10))
ax=sns.heatmap(pvt,annot=True,linewidths=.5)
```



Roc Plot Of Train And Test Data

```
# refer : # REFER : https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-pyt

model_set1=DecisionTreeClassifier(class_weight='balanced',max_depth = clf.best_params_["max_d
model set1.fit(X tr set two.v train)
```

```
# converting train and test output into probability

y_train_probs = clf.predict_proba(X_tr_set_two)[:,1] # converting train and test output into
y_test_probs= clf.predict_proba(X_te_set_two )[:,1]

 # storing values of fpr and tpr

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_probs) # storing values of f
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_probs)

plt.figure(figsize=(12,8))
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()

# NOW WE LABEL THE PLOT X AND Y
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")

# TITLE OF THE CURVE IS ROC CURVE
plt.title("ROC CURVE")
plt.grid(color='black',lw=0.5)
```



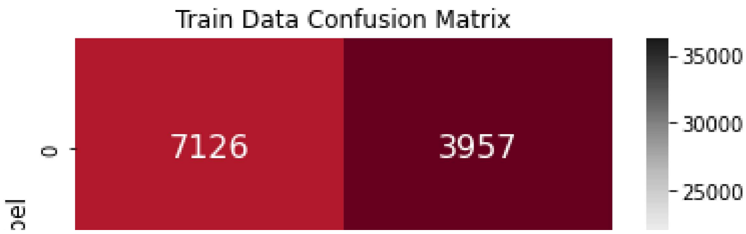
Confusion Matrix

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

# DEFINING THE THRESHOLD VALUE IF VALUE GREATHER THAN THRESHOLD THEN 1 AND IF ITS LESS THAN T
def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Train data

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
CONFUSION_MATRIX=metrics.confusion_matrix(y_train,predict_with_best_t(y_train_probs, best_t))


# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("CONFUSION MATRIX OF TRAIN DATA")
print("\n")
print('==>'*50)
print(CONFUSION_MATRIX)
sns.heatmap(CONFUSION_MATRIX, annot=True, fmt='d',cmap='RdGy',annot_kws = {"size":16})
plt.ylabel('True Label',size=12)
plt.xlabel('Predicted Label',size=12)
plt.title('Train Data Confusion Matrix',size=12)
```

```
the maximum value of tpr*(1-fpr) 0.37494400607820144 for threshold 0.487
CONFUSION MATRIX OF TRAIN DATA


==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>:
[[ 7126  3957]
 [25892 36221]]
Text(0.5, 1.0, 'Train Data Confusion Matrix')
```



test data

```
# REFER : https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
confusion_matrix =metrics.confusion_matrix(y_test,predict_with_best_t(y_test_probs, best_t))


# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
# REFER :  https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

print("CONFUSION MATRIX OF TEST DATA")
print('\n')
print('==>'*50)
print(confusion_matrix)
sns.heatmap(confusion_matrix, annot=True, fmt='d',cmap='RdGy',annot_kws = {"size":16})
plt.ylabel('True Label',size=12)
plt.xlabel('Predicted Label',size=12)
plt.title('Test Data Confusion Matrix',size=12)
```

```
the maximum value of tpr*(1-fpr) 0.3533543080394498 for threshold 0.502
CONFUSION MATRIX OF TEST DATA


==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>==>:
[[ 3683  1776]
 [14570 16023]]
Text(0.5, 1.0, 'Test Data Confusion Matrix')
```

Getting All the False Positive Data Points


```
predict=predict_with_best_t(y_test_probs,best_t)
```

```
fpi = []
for i in range(len(y_test)):
    if(y_test[i]==0) & (predict[i] == 1): #GETTING THE FALSE POSITIVE INDICES
        fpi.append(i)
len(fpi)
```

```
1776
```

```
import pandas as pd
cols = X_test.columns
X_test_false_Positive = pd.DataFrame(columns=cols) # MAKING THE FALSE POSITIVE DATAFRAME
X_test_false_Positive=X_test.iloc[fpi]
print(X_test_false_Positive.shape)
```

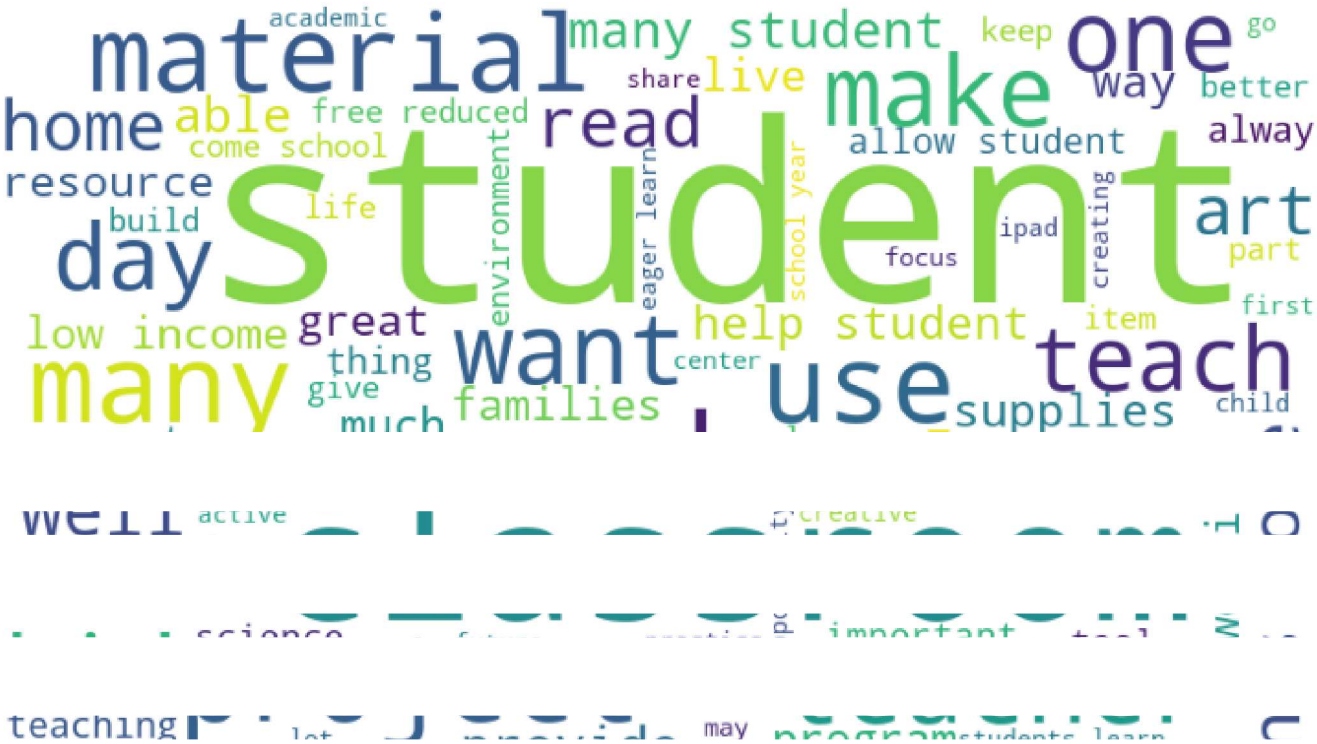
```
(1776, 12)
```

```
X_test_false_Positive.head(1)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previous
69487	sc	mrs	grades_3_5	

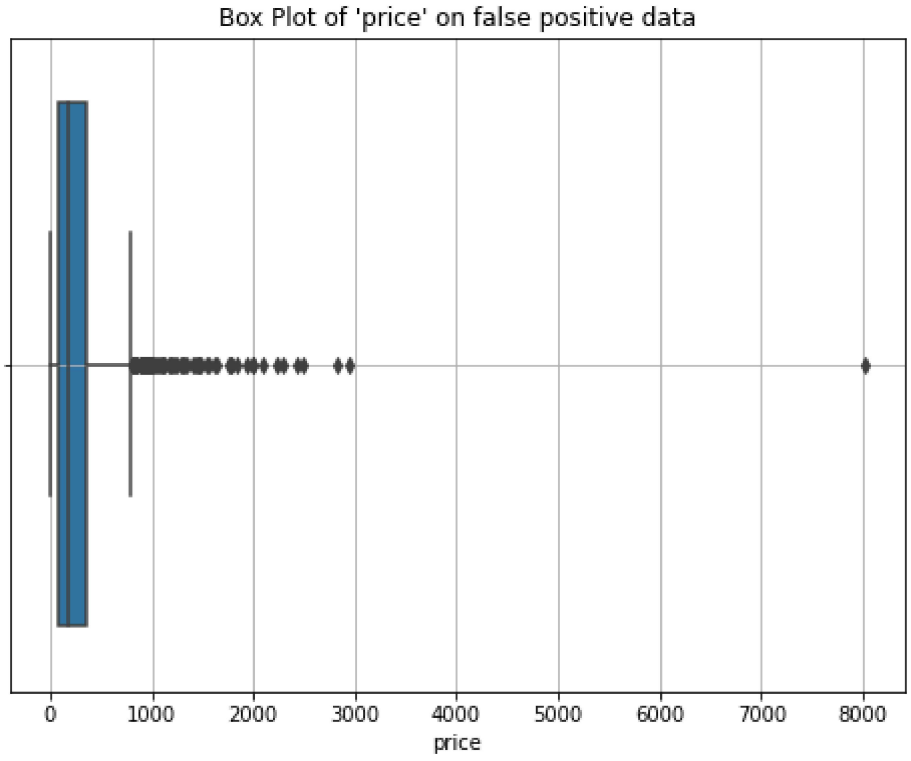
wordcloud Of Essay Text For False Positive Dataset

```
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for word in X_test_false_Positive['essay']:
    val = str(word)                #https://www.geeksforgeeks.org/generating-word-cloud-pytho
    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 = stopw



plt.figure(figsize = (12, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Box Plot With The Price Of These False Positive Data Points

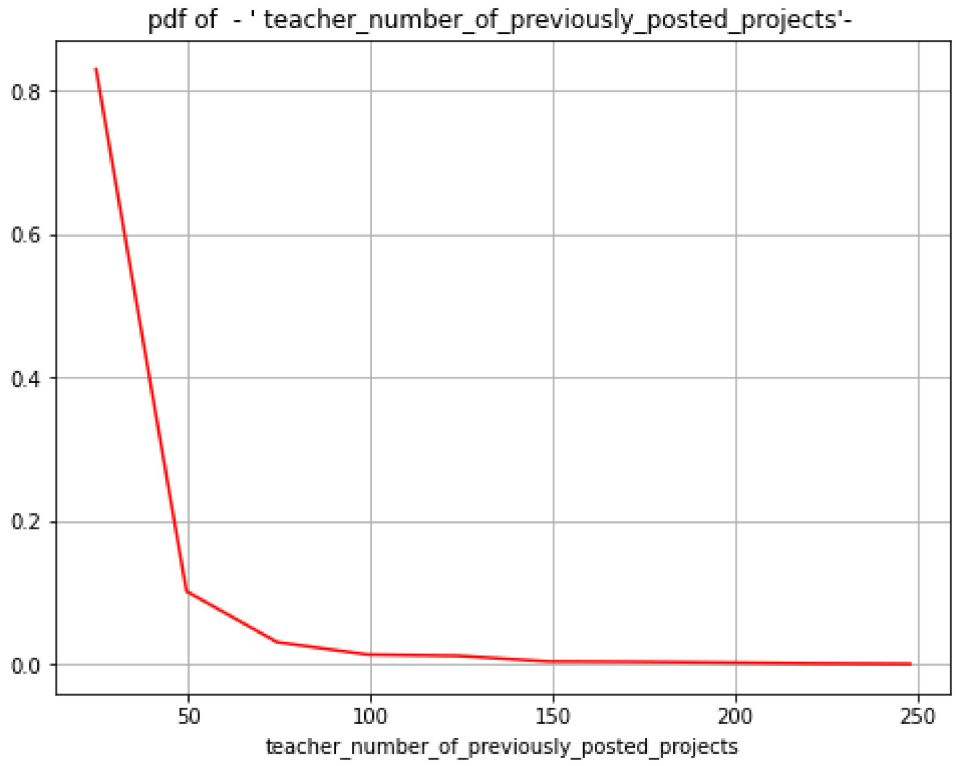
```
plt.figure(figsize=(8,6))
sns.boxplot('price',data=X_test_false_Positive,orient="v").set_title("Box Plot of 'price' on
plt.grid()
```



Pdf Plot With The teacher_number_of_previously_posted_projects Of These False Positive Data Points

```
plt.figure(figsize=(8,6))
plt.grid()
counts, bin_edges = np.histogram(X_test_false_Positive['teacher_number_of_previously_posted_p
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)
plt.plot(bin_edges[1:],pdf,color="red")
plt.title("pdf of  - ' teacher_number_of_previously_posted_projects'- ")
plt.xlabel('teacher_number_of_previously_posted_projects')
```

```
[8.30518018e-01 1.01914414e-01 3.09684685e-02 1.35135135e-02
 1.18243243e-02 3.94144144e-03 3.37837838e-03 2.25225225e-03
 1.12612613e-03 5.63063063e-04]
[  0.   24.8  49.6  74.4  99.2 124.  148.8 173.6 198.4 223.2 248. ]
Text(0.5, 0, 'teacher_number_of_previously_posted_projects')
```



Task - 2

Selecting All The Features Which Are Having Non-Zero Feature Importance(Set -1)

```
# 1. write your code in following steps for task 2
# 2. select all non zero features
# 3. Update your dataset i.e. X_train,X_test and X_cv so that it contains all rows and only n
# 4. perform hyperparameter tuning and plot either heatmap or 3d plot.
# 5. Fit the best model. Plot ROC AUC curve and confusion matrix similar to model 1.


from scipy.sparse import hstack
X_train_set_one = hstack((X_train_essay_tfidf,
                        X_train_state_ohe,
                        X_train_teacher_ohe,
                        X_train_grade_ohe,
                        X_train_price,
                        X_train_category_ohe,
                        X_train_subcategory_ohe,
                        X_train_teacher_number_of_previously_posted_projects,
                        X_train_neg,
                        X_train_pos,
                        X_train_neu,
                        X_train_compound)).tocsr()


X_test_set_one = hstack((X_test_essay_tfidf,
                        X_test_state_ohe,
                        X_test_teacher_ohe,
                        X_test_grade_ohe,
                        X_test_price,
                        X_test_category_ohe,
                        X_test_subcategory_ohe,
                        X_test_teacher_number_of_previously_posted_projects,
                        X_test_neg,
                        X_test_pos,
                        X_test_neu,
                        X_test_compound)).tocsr()


print("SHAPE OF TRAIN AND TEST AFTER STACKING")
print(X_train_set_one.shape, y_train.shape)
print(X_test_set_one.shape, y_test.shape)
print("="*100)
```

```
SHAPE OF TRAIN AND TEST AFTER STACKING
(73196, 14371) (73196,)
(36052, 14371) (36052,)
====================================================================================================
```

Here i m using the hstack task1 data

```
clf_feature= DecisionTreeClassifier(class_weight='balanced',max_depth=None,min_samples_split=
```

```
clf_feature.fit(X_train_set_one,y_train)
```

```
     DecisionTreeClassifier(class_weight='balanced', min_samples_split=500)
```

```
important_features = clf_feature.feature_importances_
# print the important feature
len(important_features)
```

```
     14371
```

```
non_zero_features=[]
for i in range(len(important_features)):
    if important_features[i]>0:             # FILTERING THE NON ZERO FEATURE IMPORTANT FEATUR
        non_zero_features.append(i)
```

```
print("NUMBER OF NON ZERO IMPORTANT FEATURES =" ,len(non_zero_features))
```

```
     NUMBER OF NON ZERO IMPORTANT FEATURES = 1103
```

```
X_train_feature=X_train_set_one[:,non_zero_features]
X_test_feature=X_test_set_one[:,non_zero_features]   # CREATING NON ZERO FEATURE IMPORTANT DA
```

```
print("SHAPE OF TRAIN AND TEST OF NON ZERO IMPORTANT FEATURES")
print(X_train_feature.shape)
print(X_test_feature .shape)
```

```
     SHAPE OF TRAIN AND TEST OF NON ZERO IMPORTANT FEATURES
     (73196, 1103)
     (36052, 1103)
```

```
tree_parameters = {'max_depth': [1, 5, 10, 50],
                   'min_samples_split': [5, 10, 100, 500]}

decision_tree= DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(decision_tree, tree_parameters, cv=5, scoring='roc_auc', return_train_scor
clf.fit(X_train_feature,y_train)
```

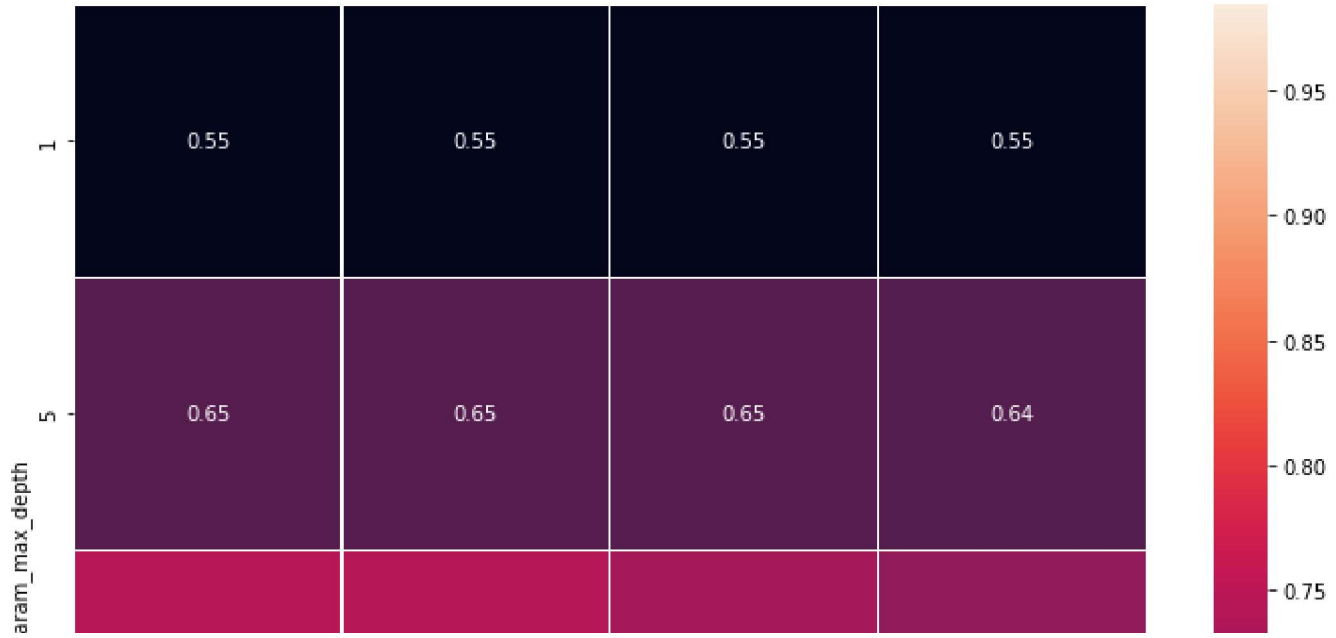
```
      GridSearchCV(cv=5, estimator=DecisionTreeClassifier(class_weight='balanced'),
                   n_jobs=-1,
                   param_grid={'max_depth': [1, 5, 10, 50],
                               'min_samples_split': [5, 10, 100, 500]},
                   return_train_score=True, scoring='roc_auc')
```

```
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']


print('Best score: ',clf.best_score_)
# finding the best parameter for model
print('Best Hyper parameters: ',clf.best_params_)
```

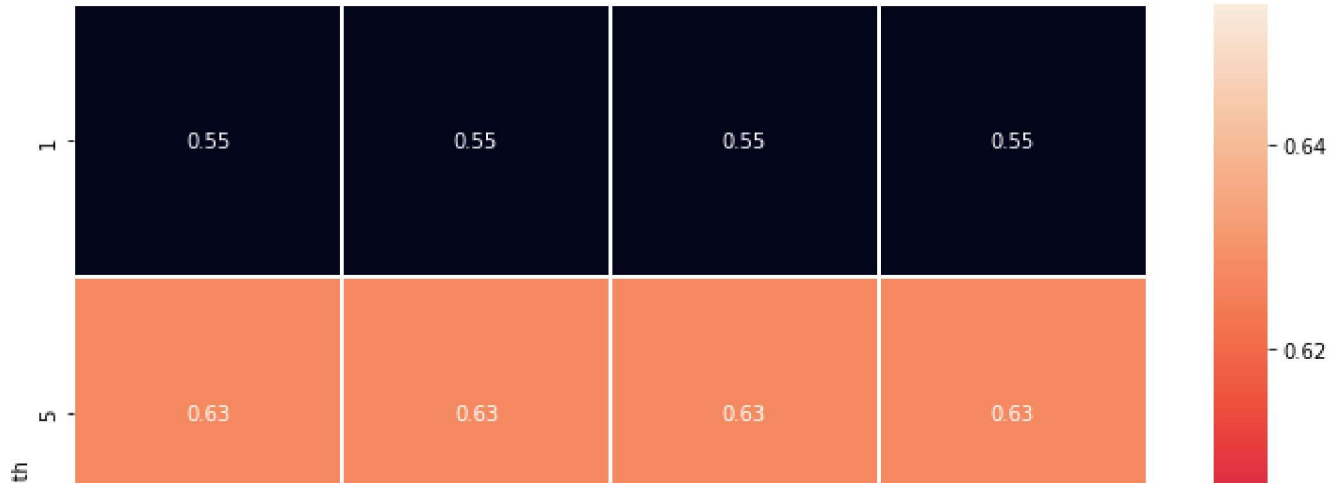
```
     Best score:  0.6537775645707175
     Best Hyper parameters:  {'max_depth': 10, 'min_samples_split': 500}
```

Plotting Hyperparameter v/s Auc

```
# HERE WE WILL TRY TO SHOW HEAT MAP FOR

# REFER : https://stackoverflow.com/questions/48791709/how-to-plot-a-heat-map-on-pivot-table-

pvt = pd.pivot_table(pd.DataFrame(clf.cv_results_),
    values='mean_train_score', index='param_max_depth',
     columns='param_min_samples_split')

 # GIVING SIZE
plt.figure(figsize=(12,10))
ax=sns.heatmap(pvt,annot=True,linewidths=.10)
```

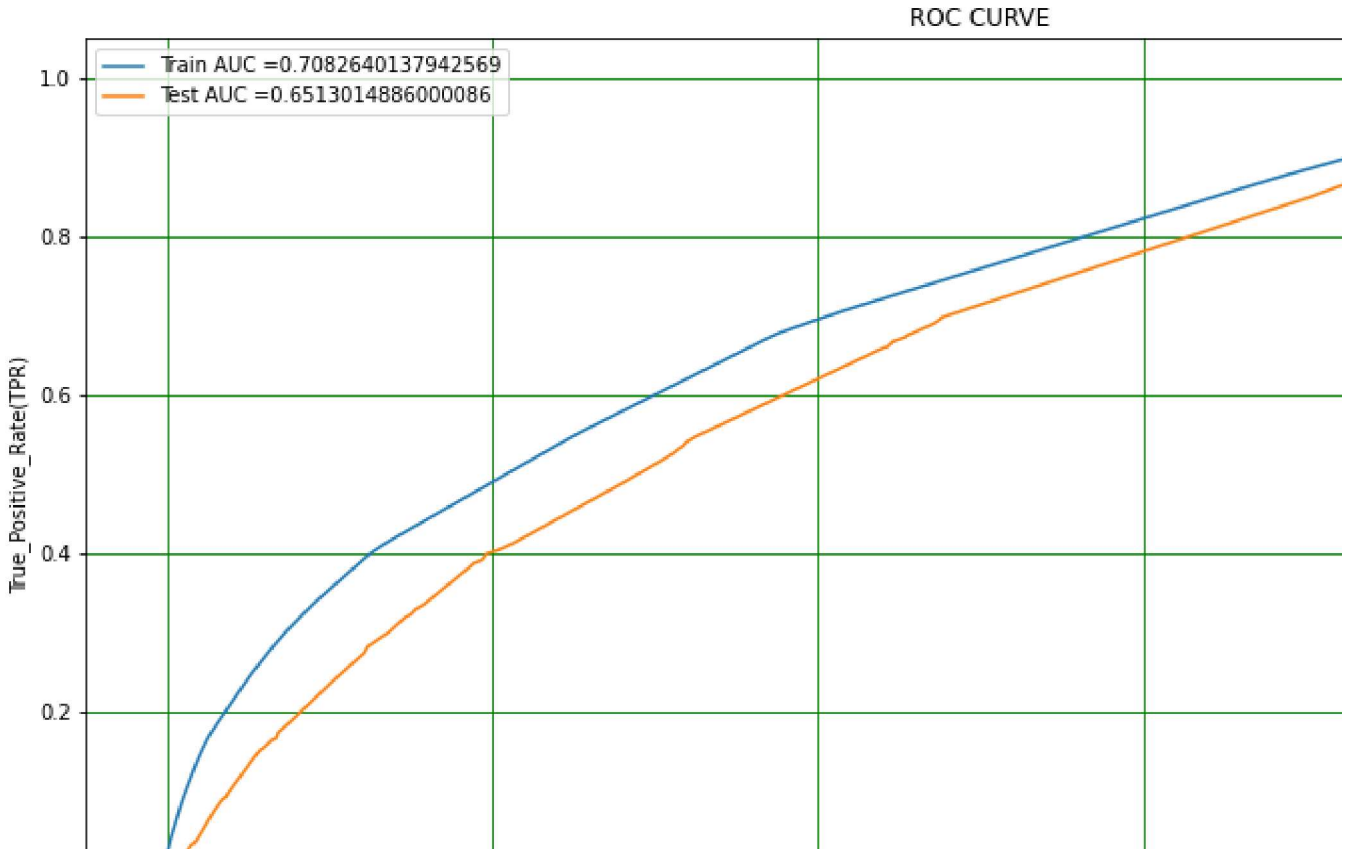



```
import pandas as pd

# HERE PLOTTING HEAT MAP FOR TEST DATA


 #https://stackoverflow.com/questions/48791709/how-to-plot-a-heat-map-on-pivot-table-after-gr

pvt = pd.pivot_table(pd.DataFrame(clf.cv_results_),
    values='mean_test_score', index='param_max_depth',
    columns='param_min_samples_split') #https://stackoverflow.com/questions/48791709/how-to-p

    # GIVING SIZE TO HEATMAP
plt.figure(figsize=(12,10))
ax=sns.heatmap(pvt,annot=True,linewidths=.5)
```



Roc Plot Of Train And Test Data

```
# refer : # REFER : https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-pyt

model_set1=DecisionTreeClassifier(class_weight='balanced',max_depth = clf.best_params_["max_d
model_set1.fit(X_train_set_one,y_train)

# converting train and test output into probability
y_train_probs = clf.predict_proba(X_train_feature)[:,1] # converting train and test output in
y_test_probs= clf.predict_proba(X_test_feature )[:,1]

 # storing values of fpr and tpr
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_probs)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_probs)


# PLOTING THE ROC CURVE
plt.figure(figsize=(16,8))
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()

# NOW WE LABEL THE PLOT X AND Y
plt.xlabel("False_Positive_Rate(FPR)")
plt.ylabel("True_Positive_Rate(TPR)")
# TITLE OF THE CURVE IS ROC CURVE
plt.title("ROC CURVE")

# DEFINING THE GRID PARAMETERS
plt.grid(color='green',lw=0.8)
```



Confusion Matrix

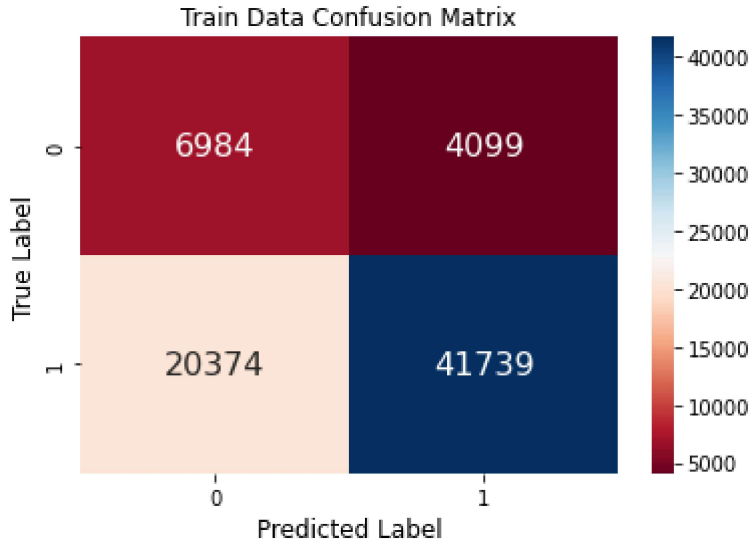
```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3
    return t

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

Train Data

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
confusion_matrix=metrics.confusion_matrix(y_train,predict_with_best_t(y_train_probs, best_t))
```

```
print("CONFUSION MATRIX OF TRAIN DATA")
print("\n")
print(confusion_matrix)
sns.heatmap(confusion_matrix, annot=True, fmt='d',cmap='RdBu',annot_kws = {"size":16})
plt.ylabel('True Label',size=12)
plt.xlabel('Predicted Label',size=12)
plt.title('Train Data Confusion Matrix',size=12)
```

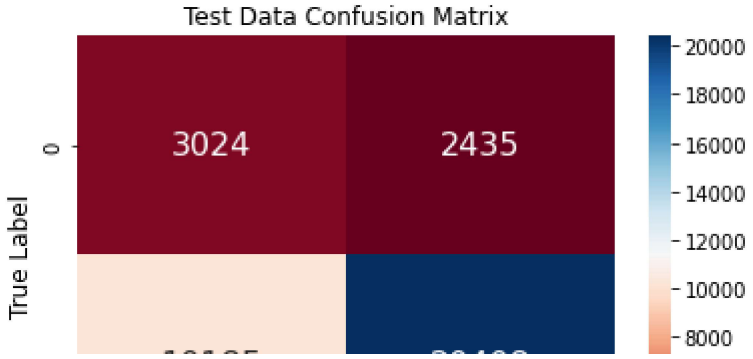
```
the maximum value of tpr*(1-fpr) 0.42345418712846794 for threshold 0.508
CONFUSION MATRIX OF TRAIN DATA


[[ 6984  4099]
 [20374 41739]]
Text(0.5, 1.0, 'Train Data Confusion Matrix')
```



Test Data

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
confusion_matrix=metrics.confusion_matrix(y_test,predict_with_best_t(y_test_probs, best_t))

print("CONFUSION MATRIX OF TEST DATA")
print('\n')
print(confusion_matrix)
sns.heatmap(confusion_matrix, annot=True, fmt='d',cmap='RdBu',annot_kws = {"size":16})
plt.ylabel('True Label',size=12)
plt.xlabel('Predicted Label',size=12)
plt.title('Test Data Confusion Matrix',size=12)
```

```
the maximum value of tpr*(1-fpr) 0.36952776170045903 for threshold 0.431
CONFUSION MATRIX OF TEST DATA


[[ 3024  2435]
 [10185 20408]]
Text(0.5, 1.0, 'Test Data Confusion Matrix')
```



Getting All the False Positive Data Points

```
predict=predict_with_best_t(y_test_probs,best_t)
```

```
false_postive_index = []
for i in range(len(y_test)):
    if(y_test[i]==0) & (predict[i] == 1):
        false_postive_index.append(i)
len(false_postive_index)
```

```
2435
```

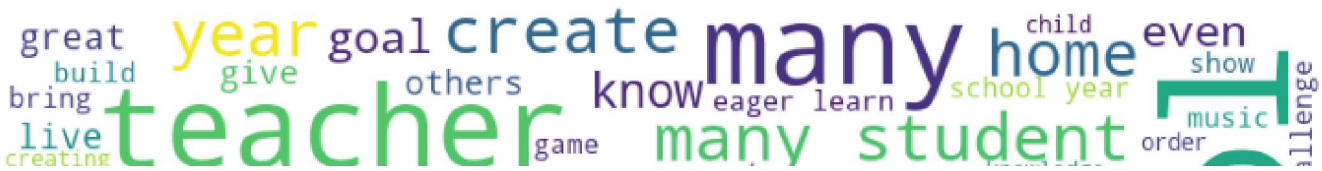
```
import pandas as pd
cols = X_test.columns
X_test_false_Positive = pd.DataFrame(columns=cols) # MAKING THE FALSE POSITIVE DATAFRAME
X_test_false_Positive=X_test.iloc[false_postive_index]
print(X_test_false_Positive.shape)
```

```
(2435, 12)
```

wordcloud Of Essay Text For False Positive Dataset

```
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for word in X_test_false_Positive['essay']:
    val = str(word)                #https://www.geeksforgeeks.org/generating-word-cloud-pytho
```

```
        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 = stopw



    plt.figure(figsize = (12,10), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.show()
```

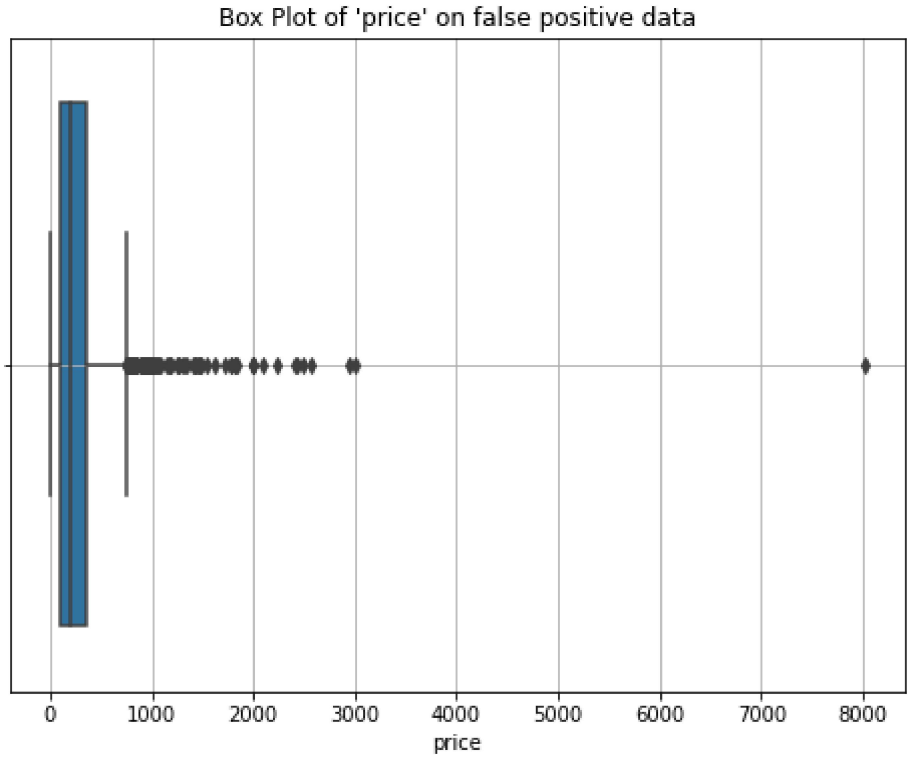


Observation : word_cloud

most word occurs **1.student 2.classroom 3. learning 4. nanan 5. school etc**

Box Plot With The Price Of These False Positive Data Points


```
plt.figure(figsize=(8,6))
sns.boxplot('price',data=X_test_false_Positive,orient="v").set_title("Box Plot of 'price' on
plt.grid()
```

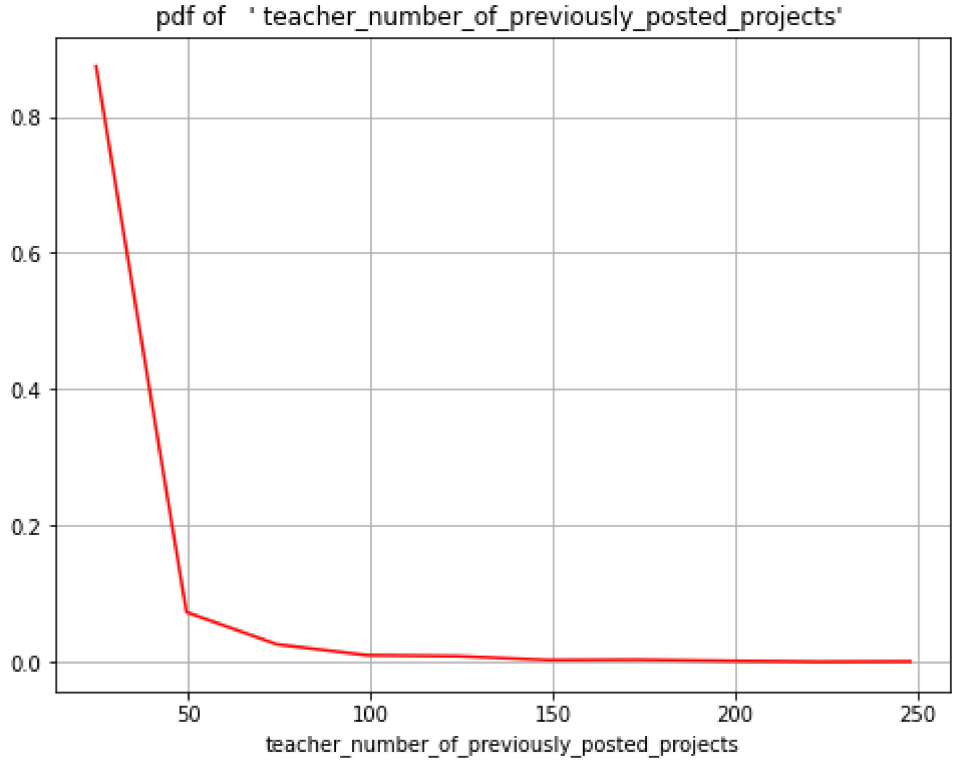


Pdf Plot With The teacher_number_of_previously_posted_projects Of These False Positive Data Points

```
plt.figure(figsize=(8,6))
```

```
plt.grid()
counts, bin_edges = np.histogram(X_test_false_Positive['teacher_number_of_previously_posted_p
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)
plt.plot(bin_edges[1:],pdf,color="red")
plt.title("pdf of   ' teacher_number_of_previously_posted_projects' ")
plt.xlabel('teacher_number_of_previously_posted_projects')
```

```
[8.73511294e-01 7.31006160e-02 2.58726899e-02 9.85626283e-03
 8.62422998e-03 2.87474333e-03 3.28542094e-03 1.64271047e-03
 4.10677618e-04 8.21355236e-04]
[  0.   24.8  49.6  74.4  99.2 124.  148.8 173.6 198.4 223.2 248. ]
Text(0.5, 0, 'teacher_number_of_previously_posted_projects')
```



Summary

```
from prettytable import PrettyTable
from prettytable import ALL as ALL
table=PrettyTable(hrules=ALL)
table.field_names = [ "Sl.N0","Vectorizer", "Model", "Hyper Parameter", "Train Auc" , " Test-
table.add_row([1,"TFIDF", "DECSION TREE", "max_depth =10 , min_samples_split=500",0.7083056,0
table.add_row([2,"TFIDF W2V", "DECSION TREE"," max_depth =5 , min_samples_split=500",0.656275
table.add_row([3,"TFIDF NON ZERO FEATURE IMPORTANCE", "DECSION TREE", "max_depth =10 , min_sa
print(table)
```

```
+-------+----------------------------------+-------------+--------------------------
```

Sl.N0	Vectorizer	Model	Hyper Parameter
1	TFIDF	DECSION TREE	max_depth =10 , min_samples
2	TFIDF W2V	DECSION TREE	max_depth =5 , min_samples
3	TFIDF NON ZERO FEATURE IMPORTANCE	DECSION TREE	max_depth =10 , min_samples

CONCLUSION

NOT A BIG DEFFERENCE BETWEEN TRAIN AND TEST ACCURACY

WE ARE NOT OVERFITTING OR UNDERFITTING ANYMORE HERE

WHEN WE HAVE TAKEN NON ZERO VALUES IN THIRD MODEL NOT MUCH BIG DIFFERENCE

INCREASING THE DEPTH OF TREE IMPROVING THE TRAIN ACCURACY SILIGHTLY BUT NOT MUCH

Colab paid products - Cancel contracts here

✓ 0s completed at 2:52 PM