Assignment : DT

Please check below video before attempting this assignment

In [115]:

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

Out[115]:

In [116]:

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

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
from sklearn.metrics import roc_curve, auc
import pickle
from tqdm import tqdm
import os
```

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

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](https://en.wikipedia.org/wiki/GloVe_(machine_learning)) and [this](https://en.wikipedia.org/wiki/GloVe_(machine_learning)) for more details.

Download glove vectors from this link

In [117]:

```
#please use below code to load glove vectors
with open('/content/drive/MyDrive/temp/9.donar choose/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

or else , you can use below code

In [118]:

```
'''
# 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-pic
kle-to-save-and-load-variables-in-python/

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


'''
```

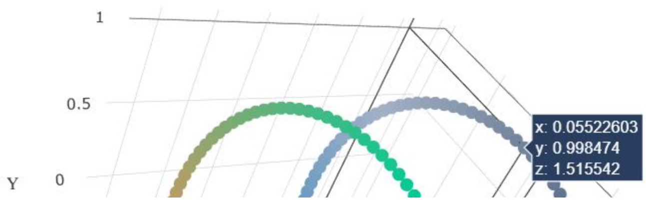
Out[118]:

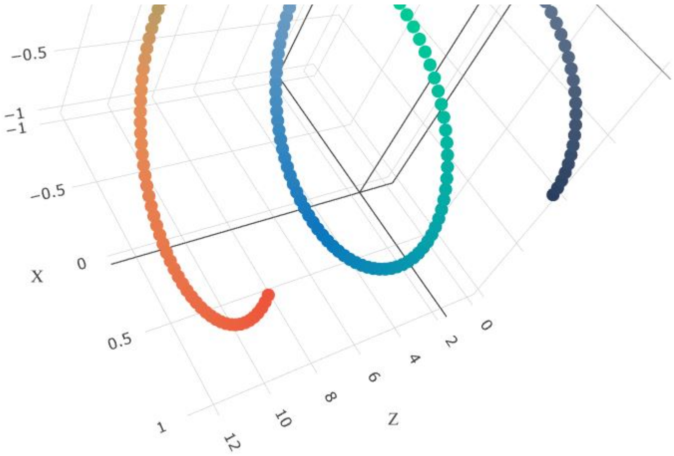
```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef l
oadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\
', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.s
plit()\n        word = splitLine[0]\n        embedding = np.array([float(val) for val in
splitLine[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," words
loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ========
====================\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\
nDone. 1917495  words loaded!\n\n# ============================\n\nwords = []\nfor i in p
reproced_texts:\n    words.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    word
s.extend(i.split(\' \'))\nprint("all the words in the coupus", len(words))\nwords = set(w
ords)\nprint("the unique words in the coupus", len(words))\n\ninter_words = set(model.key
s()).intersection(words)\nprint("The number of words that are present in both glove vecto
rs and our coupus",      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3
),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i
in words_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length", len(word
s_courpus))\n\n\n# stronging variables into pickle files python: http://www.jessicayung.c
om/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pickle\nwith open(\'
glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

Task - 1

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)**
     ** **
   - **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)**
       ** **
     - **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**

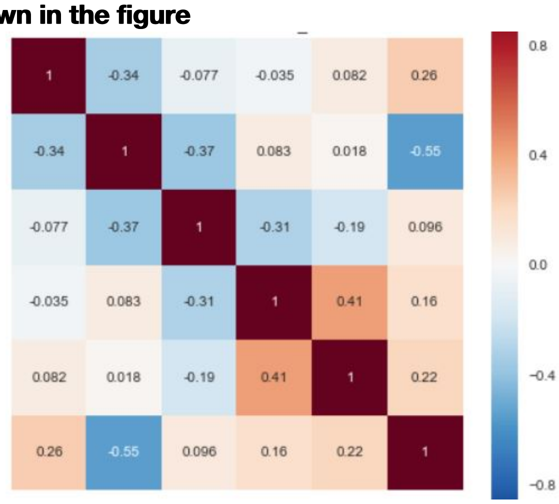




with X-axis as min_sample_split, Y-axis as max_depth, and Z-axis as AUC Score , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive ***3d_scatter_plot.ipynb***

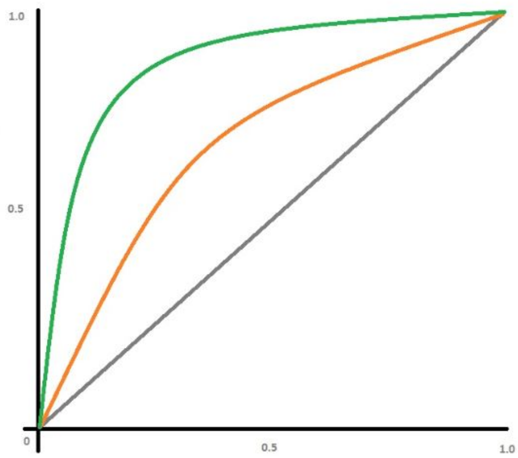
or

- **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 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`**
    ** **

In [119]:

```
data  = pd.read_csv('/content/drive/MyDrive/temp/9.donar choose/preprocessed_data.csv', n
rows=50000)
# data  = pd.read_csv('preprocessed_data.csv', nrows=50000) # you can take less number of
rows like this
print(data.columns)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, data['project_is_approved'],st
ratify=data['project_is_approved'], test_size=0.33)
X_train,X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratif
y=y_train)

X_train.drop(["project_is_approved"], axis = 1, inplace = True)
X_test.drop(["project_is_approved"], axis = 1, inplace = True)
X_cv.drop(["project_is_approved"], axis = 1, inplace = True)
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

In [120]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
X_train_essay = X_train['essay'].values
X_test_essay = X_test['essay'].values
X_cv_essay = X_cv['essay'].values

vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train_essay)
X_train_essay_tfidf= vectorizer.transform(X_train_essay)
X_test_essay_tfidf= vectorizer.transform(X_test_essay)
X_cv_essay_tfidf= vectorizer.transform(X_cv_essay)

print("Shape of matrix after one hot encodig ",X_train_essay_tfidf.shape,X_test_essay_tfi
df.shape,X_cv_essay_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (22445, 8865) (16500, 8865) (11055, 8865)
```

In [121]:

```
# average Word2Vec
# compute average word2vec for each review.
def avg_w2v_vec(array):
   avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
   for sentence in tqdm(array): # for each review/sentence
       vector = np.zeros(300) # as word vectors are of zero length
       cnt_words =0; # num of words with a valid vector in the sentence/review
       for word in sentence.split(): # for each word in a review/sentence
          if word in glove_words:
            vector += model[word]
            cnt_words += 1
       if cnt_words != 0:
         vector /= cnt_words
       avg_w2v_vectors.append(vector)
```

```
    print(len(avg_w2v_vectors))
    print(len(avg_w2v_vectors[0]))
    return avg_w2v_vectors


X_train_essay_W2V= avg_w2v_vec(X_train_essay)
X_test_essay_W2V= avg_w2v_vec(X_test_essay)
X_cv_essay_W2V= avg_w2v_vec(X_cv_essay)
```

```
100%|██████████| 22445/22445 [00:07<00:00, 3125.03it/s]
```

```
22445
300
```

```
100%|██████████| 16500/16500 [00:05<00:00, 3126.33it/s]
```

```
16500
300
```

```
100%|██████████| 11055/11055 [00:03<00:00, 2985.82it/s]
```

```
11055
300
```

In [122]:

```
vectorizer1 = CountVectorizer()
vectorizer1.fit(X_train['school_state'])
X_train_School_state_bow= vectorizer1.transform(X_train['school_state'])
X_test_School_state_bow= vectorizer1.transform(X_test['school_state'])
X_cv_School_state_bow= vectorizer1.transform(X_cv['school_state'])
print("Shape of matrix after one hot encodig ",X_train_School_state_bow.shape,X_test_School_state_bow.shape,X_cv_School_state_bow.shape)

vectorizer1.fit(X_train['teacher_prefix'])
X_train_teacher_pre_bow= vectorizer1.transform(X_train['teacher_prefix'])
X_test_teacher_pre_bow= vectorizer1.transform(X_test['teacher_prefix'])
X_cv_teacher_pre_bow= vectorizer1.transform(X_cv['teacher_prefix'])
print(X_train_teacher_pre_bow.shape,X_test_teacher_pre_bow.shape)

vectorizer1.fit(X_train['project_grade_category'])
X_train_proj_cate_bow= vectorizer1.transform(X_train['project_grade_category'])
X_test_proj_cate_bow= vectorizer1.transform(X_test['project_grade_category'])
X_cv_proj_cate_bow= vectorizer1.transform(X_cv['project_grade_category'])
print(X_train_proj_cate_bow.shape,X_test_proj_cate_bow.shape,X_cv_proj_cate_bow.shape)


vectorizer1.fit(X_train['clean_categories'])
X_train_cln_catgy_bow= vectorizer1.transform(X_train['clean_categories'])
X_test_cln_catgy_bow= vectorizer1.transform(X_test['clean_categories'])
X_cv_cln_catgy_bow= vectorizer1.transform(X_cv['clean_categories'])
print(X_train_cln_catgy_bow.shape,X_test_cln_catgy_bow.shape,X_cv_cln_catgy_bow.shape)



vectorizer1.fit(X_train['clean_subcategories'])
X_train_cln_subcatgy_bow= vectorizer1.transform(X_train['clean_subcategories'])
X_test_cln_subcatgy_bow= vectorizer1.transform(X_test['clean_subcategories'])
X_cv_cln_subcatgy_bow= vectorizer1.transform(X_cv['clean_subcategories'])
print(X_train_cln_subcatgy_bow.shape,X_test_cln_subcatgy_bow.shape,X_cv_cln_subcatgy_bow.shape)
```

```
Shape of matrix after one hot encodig  (22445, 51) (16500, 51) (11055, 51)
(22445, 5) (16500, 5)
(22445, 4) (16500, 4) (11055, 4)
(22445, 7) (16500, 7) (11055, 7)
(22445, 28) (16500, 28) (11055, 28)
```

In [123]:

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

```
scalar = StandardScaler()
scalar.fit(X_train['price'].values.reshape(-1,1))
X_train_price_norm = scalar.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm= scalar.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm= scalar.transform(X_test['price'].values.reshape(-1,1))
print("X_train_price_norm:",(X_train_price_norm.shape))
print("X_cv_price_norm",(X_cv_price_norm.shape))
print("X_test_price_norm",(X_test_price_norm.shape))

scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_train_posted_proj_norm = scalar.transform(X_train['teacher_number_of_previously_posted_
projects'].values.reshape(-1,1))
X_cv_posted_proj_norm= scalar.transform(X_cv['teacher_number_of_previously_posted_project
s'].values.reshape(-1,1))
X_test_posted_proj_norm= scalar.transform(X_test['teacher_number_of_previously_posted_pro
jects'].values.reshape(-1,1))
print("X_train_posted_proj_norm:",(X_train_posted_proj_norm.shape))
print("X_cv_posted_proj_norm",(X_cv_posted_proj_norm.shape))
print("X_test_posted_proj_norm",(X_test_posted_proj_norm.shape))
```

```
X_train_price_norm: (22445, 1)
X_cv_price_norm (11055, 1)
X_test_price_norm (16500, 1)
X_train_posted_proj_norm: (22445, 1)
X_cv_posted_proj_norm (11055, 1)
X_test_posted_proj_norm (16500, 1)
```

In [124]:

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

def senti(array):
 sentiment=[]
 sid=SentimentIntensityAnalyzer()
 for i in array:
   score=sid.polarity_scores(i)
   value=sum(score.values())
   sentiment.append(value)
 return sentiment

X_train_senti=np.array(senti(X_train['essay'].tolist()))
X_test_senti=np.array(senti(X_test['essay'].tolist()))
X_cv_senti=np.array(senti(X_cv['essay'].tolist()))

X_train_sentiment=np.reshape(X_train_senti,(len(X_train_senti),1))
X_test_sentiment=np.reshape(X_test_senti,(len(X_test_senti),1))
X_cv_sentiment=np.reshape(X_cv_senti,(len(X_cv_senti),1))

print((X_train_senti.shape))
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
(22445,)
```

stacking vectorised features

In [125]:

```
from scipy.sparse import hstack

X_train_set1=hstack((X_train_School_state_bow,X_train_teacher_pre_bow,X_train_proj_cate_b
ow,
       X_train_posted_proj_norm, X_train_cln_catgy_bow,X_train_cln_subcatgy_bow,X_train_e
ssay_tfidf
       ,X_train_price_norm,X_train_sentiment)).tocsr()

X_cv_set1=hstack((X_cv_School_state_bow,X_cv_teacher_pre_bow,X_cv_proj_cate_bow,
       X_cv_posted_proj_norm, X_cv_cln_catgy_bow,X_cv_cln_subcatgy_bow,X_cv_essay_tfidf
       ,X_cv_price_norm,X_cv_sentiment)).tocsr()
```

```
X_test_set1=hstack((X_test_School_state_bow,X_test_teacher_pre_bow,X_test_proj_cate_bow,
       X_test_posted_proj_norm, X_test_cln_catgy_bow,X_test_cln_subcatgy_bow,X_test_essay
_tfidf
       ,X_test_price_norm,X_test_sentiment)).tocsr()


X_train_set2=hstack((X_train_School_state_bow,X_train_teacher_pre_bow,X_train_proj_cate_b
ow,
       X_train_posted_proj_norm, X_train_cln_catgy_bow,X_train_cln_subcatgy_bow,X_train_e
ssay_W2V
       ,X_train_price_norm,X_train_sentiment)).tocsr()

X_cv_set2=hstack((X_cv_School_state_bow,X_cv_teacher_pre_bow,X_cv_proj_cate_bow,
       X_cv_posted_proj_norm, X_cv_cln_catgy_bow,X_cv_cln_subcatgy_bow,X_cv_essay_W2V
       ,X_cv_price_norm,X_cv_sentiment)).tocsr()
X_test_set2=hstack((X_test_School_state_bow,X_test_teacher_pre_bow,X_test_proj_cate_bow,
       X_test_posted_proj_norm, X_test_cln_catgy_bow,X_test_cln_subcatgy_bow,X_test_essay
_W2V
       ,X_test_price_norm,X_test_sentiment)).tocsr()

print(X_train_set1.shape)
print(X_train_set2.shape)
```

```
(22445, 8963)
(22445, 398)
```

FOR SET1

parameter tuning for SET1

In [126]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

param_grid = {
              'min_samples_split': [5, 10, 100, 500],
              'max_depth' : [1, 5, 10, 50],
             }
DT = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator=DT, param_grid=param_grid, scoring='roc_auc', cv=4,
n_jobs=-1,return_train_score=True, verbose=True)
grid_search.fit(X_train_set1,y_train)

print(grid_search.best_estimator_)
print('train_accuracy:',grid_search.score(X_train_set1,y_train))
print('train_accuracy:',grid_search.score(X_test_set1,y_test))
```

```
Fitting 4 folds for each of 16 candidates, totalling 64 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   53.1s
[Parallel(n_jobs=-1)]: Done  64 out of  64 | elapsed:  3.6min finished
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, 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')
train_accuracy: 0.664815146294404
train_accuracy: 0.6044335243519513
{'mean_fit_time': array([ 0.44313681,  0.43262756,  0.42967695,  0.43049753,  2.04404444,
        2.01333332,  1.98088026,  1.91414291,  4.77224505,  4.662512  ,
        4.20022219,  3.77433372, 21.43146825, 20.95247483, 19.50948632,
       16.75023186]), 'std_fit_time': array([0.01768212, 0.00629435, 0.00651265, 0.008997
78, 0.02189398,
       0.01846545, 0.00760787, 0.00272687, 0.12604121, 0.12092405,
       0.03093538, 0.03286194, 0.53218316, 0.43660423, 0.75310467,
       1.07941433]), 'mean_score_time': array([0.01239145, 0.01116294, 0.01092362, 0.0115
```

```
0352, 0.01198345,
       0.01135278, 0.01163167, 0.01190966, 0.01128978, 0.01107949,
       0.01096398, 0.01109183, 0.01241434, 0.01253885, 0.01254368,
       0.0115937 ]), 'std_score_time': array([1.54996113e-03, 6.83827595e-04, 5.95860357e
-04, 1.13014977e-03,
       8.99724772e-04, 3.40299032e-04, 5.31398280e-04, 9.92373057e-04,
       8.49052003e-05, 1.78433918e-04, 1.30767470e-04, 3.35715456e-05,
       5.00126559e-04, 1.16868689e-04, 1.16705834e-04, 1.46168791e-03]), 'param_max_depth
': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50, 50],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_min_samples_split': masked_array(data=[5, 10, 100, 500,
5, 10, 100, 500, 5, 10, 100, 500, 5,
                   10, 100, 500],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'max_depth': 1, 'min_samples_split': 5}, {'max_dep
th': 1, 'min_samples_split': 10}, {'max_depth': 1, 'min_samples_split': 100}, {'max_depth
': 1, 'min_samples_split': 500}, {'max_depth': 5, 'min_samples_split': 5}, {'max_depth':
5, 'min_samples_split': 10}, {'max_depth': 5, 'min_samples_split': 100}, {'max_depth': 5,
'min_samples_split': 500}, {'max_depth': 10, 'min_samples_split': 5}, {'max_depth': 10, '
min_samples_split': 10}, {'max_depth': 10, 'min_samples_split': 100}, {'max_depth': 10, '
min_samples_split': 500}, {'max_depth': 50, 'min_samples_split': 5}, {'max_depth': 50, 'm
in_samples_split': 10}, {'max_depth': 50, 'min_samples_split': 100}, {'max_depth': 50, 'm
in_samples_split': 500}], 'split0_test_score': array([0.5431867 , 0.5431867 , 0.5431867 ,
0.5431867 , 0.61182121,
       0.61182121, 0.61164502, 0.61119411, 0.59342335, 0.59039537,
       0.60452168, 0.61557682, 0.51039583, 0.51398517, 0.55333684,
       0.59070431]), 'split1_test_score': array([0.55483671, 0.55483671, 0.55483671, 0.55
483671, 0.59965565,
       0.60008012, 0.59954979, 0.60014805, 0.6010471 , 0.60312565,
       0.60463536, 0.61051695, 0.54641524, 0.53277269, 0.54975068,
       0.59226494]), 'split2_test_score': array([0.55212663, 0.55212663, 0.55212663, 0.55
212663, 0.59558843,
       0.59574518, 0.59430942, 0.59474012, 0.57279039, 0.57429744,
       0.59057614, 0.60671627, 0.5044894 , 0.53325639, 0.55209193,
       0.60744087]), 'split3_test_score': array([0.55979597, 0.55979597, 0.55979597, 0.55
979597, 0.61791835,
       0.6179174 , 0.61779913, 0.62020088, 0.58289936, 0.58541052,
       0.59928913, 0.62265562, 0.52837783, 0.52167082, 0.57041686,
       0.60165004]), 'mean_test_score': array([0.55248651, 0.55248651, 0.55248651, 0.5524
8651, 0.60624591,
       0.60639098, 0.60582584, 0.60657079, 0.58754005, 0.58830725,
       0.59975558, 0.61386641, 0.52241958, 0.52542127, 0.55639908,
       0.59801504]), 'std_test_score': array([0.00603257, 0.00603257, 0.00603257, 0.00603
257, 0.00900477,
       0.00888124, 0.00934397, 0.00985356, 0.01067874, 0.01035146,
       0.00572293, 0.00596907, 0.01641154, 0.00806664, 0.00819494,
       0.00686604]), 'rank_test_score': array([11, 11, 11, 11,  4,  3,  5,  2,  9,  8,  6
,  1, 16, 15, 10,  7],
      dtype=int32), 'split0_train_score': array([0.55764915, 0.55764915, 0.55764915, 0.55
764915, 0.64409745,
       0.64409745, 0.64389355, 0.64065386, 0.70419397, 0.7014034 ,
       0.68441099, 0.66938357, 0.9013306 , 0.88964332, 0.85693553,
       0.79920036]), 'split1_train_score': array([0.56453002, 0.56453002, 0.56453002, 0.5
6453002, 0.65014814,
       0.65014814, 0.64947488, 0.64816862, 0.7123637 , 0.71079121,
       0.70046418, 0.6921228 , 0.93360946, 0.92806698, 0.91963519,
       0.86445631]), 'split2_train_score': array([0.56094625, 0.56094625, 0.56094625, 0.5
6094625, 0.63857692,
       0.63857692, 0.63718028, 0.63647525, 0.70380531, 0.70363843,
       0.68798932, 0.67602847, 0.89038595, 0.88922375, 0.85296481,
       0.79901568]), 'split3_train_score': array([0.56227023, 0.56227023, 0.56227023, 0.5
6227023, 0.65060088,
       0.65053135, 0.65035626, 0.64834414, 0.71590556, 0.71412299,
       0.69481798, 0.67892268, 0.90187297, 0.89334432, 0.86856039,
       0.82048391]), 'mean_train_score': array([0.56134891, 0.56134891, 0.56134891, 0.561
34891, 0.64585585,
       0.64583846, 0.64522624, 0.64341047, 0.70906713, 0.707489  ,
       0.69192062, 0.67911438, 0.90679975, 0.90006959, 0.87452398,
```

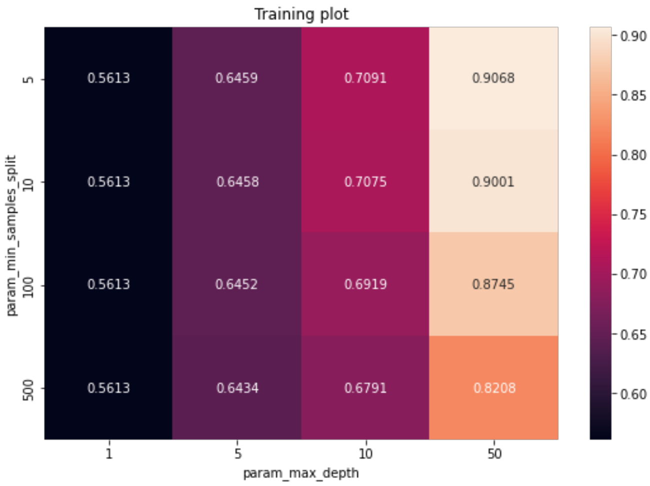
```
       0.82078906]), 'std_train_score': array([0.00249091, 0.00249091, 0.00249091, 0.0024
9091, 0.00492478,
       0.00490809, 0.00526502, 0.00506649, 0.00522173, 0.00516662,
       0.00618944, 0.00826842, 0.01614279, 0.01624363, 0.02666804,
       0.02667899])}
```

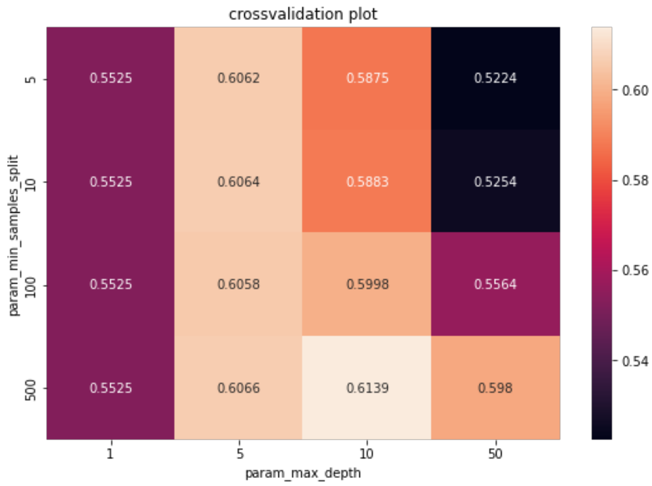
HEATMAP for SET1

In [127]:

```
#https://blog.quantinsti.com/creating-heatmap-using-python-seaborn/
import seaborn as sns
score = pd.DataFrame(grid_search.cv_results_).groupby(['param_min_samples_split', 'param
_max_depth']).max().unstack()[['mean_test_score','mean_train_score']]
print(score)
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(score.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(score.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Training plot')
ax[1].set_title('crossvalidation plot')
plt.show()
```

```
                        mean_test_score            ... mean_train_score
param_max_depth                       1         5  ...               10         50
param_min_samples_split                            ...
5                              0.552487  0.606246  ...         0.709067   0.906800
10                             0.552487  0.606391  ...         0.707489   0.900070
100                            0.552487  0.605826  ...         0.691921   0.874524
500                            0.552487  0.606571  ...         0.679114   0.820789

[4 rows x 8 columns]
```





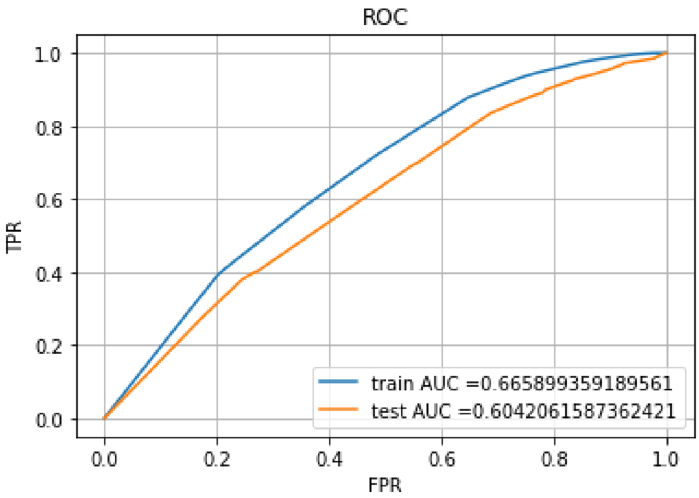
Using Best Estimator(params) and Plotting ROC for SET1

In [149]:

```
clf_best=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, 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')
clf_best.fit(X_train_set1,y_train)

y_train_predt_set1 = clf_best.predict_proba(X_train_set1) [:,1]
y_test_predt_set1= clf_best.predict_proba(X_test_set1) [:,1]

FPR_train_s1,TPR_train_s1,thres_train_s1=roc_curve(y_train,y_train_predt_set1)
FPR_test_s1,TPR_test_s1,thres_test_s1=roc_curve(y_test,y_test_predt_set1)

print("BEST train AUC score :",auc(FPR_train_s1, TPR_train_s1))
print("BEST test AUC score :",auc(FPR_test_s1, TPR_test_s1))
```

```
plt.plot(FPR_train_s1, TPR_train_s1, label="train AUC ="+str(auc(FPR_train_s1, TPR_train_
s1)))
plt.plot(FPR_test_s1, TPR_test_s1, label="test AUC ="+str(auc(FPR_test_s1, TPR_test_s1)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC")
plt.grid(True)
plt.show()
```

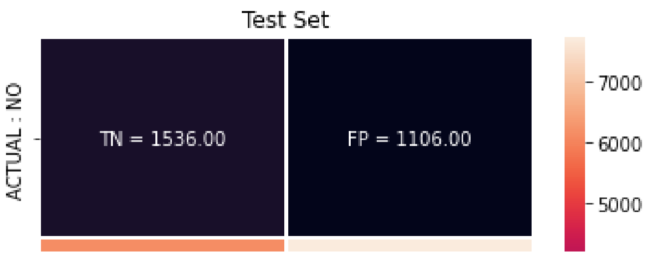
```
BEST train AUC score : 0.665899359189561
BEST test AUC score : 0.6042061587362421
```

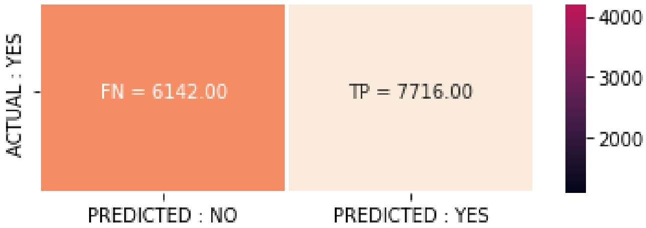


Confusion Matrix for SET1 for test data

In [150]:

```
def predict(proba, threshould, fpr, tpr):
  temp = threshould[np.argmax(fpr*(1-tpr))]
  predictions = []
  global y_testpred_s1
  for i in proba:
    if i>=temp:
     predictions.append(1)
    else:
     predictions.append(0)
  y_testpred_s1= predictions
  return predictions


#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn

con_m_test = confusion_matrix(y_test, predict(y_test_predt_set1, thres_test_s1, FPR_test
_s1,TPR_test_s1))
key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
#fig, ax = plt.subplots(1,2, figsize=(5,5))
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key
.flatten(),con_m_test.flatten())])).reshape(2,2)

sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES']
,yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '')

plt.title('Test Set')
plt.show()
```





Word Cloud for set1 test data

In [151]:

```
#false positive data point gathering
#https://github.com/pskadasi/DecisionTrees_DonorsChoose/blob/master/

index1= []
for i in range(len(y_test)) :
  if (y_test.iloc[i] == 0) & (y_testpred_s1[i] == 1) :
    index1.append(i)

fp_essay1 = []
for i in index1 :
  fp_essay1.append(X_test['essay'].iloc[i])

fp_price1=[]
for i in index1:
    fp_price1.append(X_test['price'].iloc[i])

fp_teacher_number_of_previously_posted_projects1=[]
for i in index1:
    fp_teacher_number_of_previously_posted_projects1.append(X_test['teacher_number_of_pre
viously_posted_projects'].iloc[i])

print(len(fp_teacher_number_of_previously_posted_projects1))


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 =
stopwords, min_font_size = 10).generate(comment_words)


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

plt.show()
```

1106

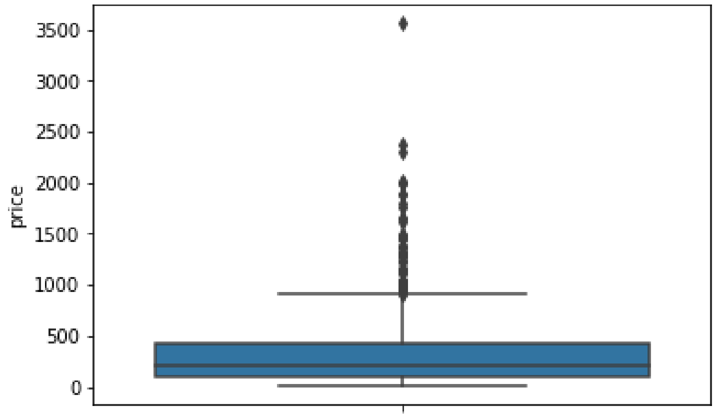
BOX PLOT of price feature of set1 test data

In [152]:

```
df=pd.DataFrame(fp_price1)
df.columns=['price']
sns.boxplot(data=df,y='price')
```

Out[152]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f10ce19bc50>
```



PDF of teacher_number_of_previously_posted_projects for FP dataset of SET1

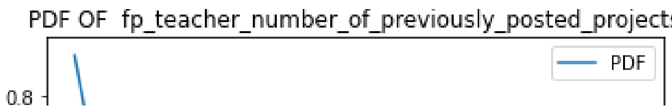
In [153]:

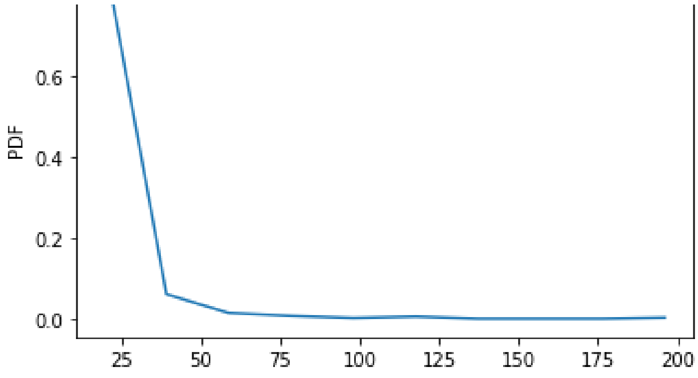
```
count,bin=np.histogram(fp_teacher_number_of_previously_posted_projects1,bins=10,density=True)
PDF=count/sum(count)
print(PDF)
plt.plot(bin[1:],PDF,label="PDF")
plt.ylabel("PDF")
plt.legend()
plt.title("PDF OF  fp_teacher_number_of_previously_posted_projects")
```

```
[0.89963834 0.06148282 0.01537071 0.00813743 0.00271248 0.00632911
 0.00090416 0.00090416 0.00090416 0.00361664]
```

Out[153]:

```
Text(0.5, 1.0, 'PDF OF  fp_teacher_number_of_previously_posted_projects')
```





FOR SET2

In [133]:

```
print((X_train_set2.shape))
print((X_train_set1.shape))
print(len(y_train))
```

```
(22445, 398)
(22445, 8963)
22445
```

parameter tuning for SET2

In [155]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

param_grid1 = {
              'min_samples_split': [5, 10, 100, 500],
              'max_depth' : [1, 5, 10, 50],
             }
DT1 = DecisionTreeClassifier()
grid_search1= GridSearchCV(estimator=DT1, param_grid=param_grid1, scoring='roc_auc', cv=
4,n_jobs=-1,return_train_score=True, verbose=True)
grid_search1.fit(X_train_set2,y_train)

print(grid_search1.best_estimator_)
print('train_accoracy:',grid_search1.score(X_train_set2,y_train))
print('train_accuracy:',grid_search1.score(X_test_set2,y_test))
print(grid_search1.cv_results_)
```

```
Fitting 4 folds for each of 16 candidates, totalling 64 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done  64 out of  64 | elapsed:  8.6min finished
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
train_accoracy: 0.6538394985667538
train_accuracy: 0.6173909472623208
{'mean_fit_time': array([ 1.27969462,  1.27211857,  1.26842481,  1.27257919,  6.09734184,
        6.07519323,  6.05166793,  5.91376108, 14.21640396, 14.12194687,
       13.45545655, 11.10469031, 50.18937659, 49.87025541, 46.67042357,
       27.94684738]), 'std_fit_time': array([0.00446219, 0.01540865, 0.00922675, 0.006198
44, 0.06891569,
       0.05670858, 0.07249975, 0.11271141, 0.14121706, 0.14235279,
       0.18616164, 0.29880878, 0.74488845, 1.02198821, 1.01180484,
       2.44748899]), 'mean_score_time': array([0.01871181, 0.01754099, 0.02158409, 0.0231
1492, 0.01759684,
       0.01792818, 0.02017581, 0.01789904, 0.01764524, 0.01770169,
```

```
       0.01778138, 0.01788878, 0.01847231, 0.01853973, 0.01868188,
       0.01681519]), 'std_score_time': array([0.00102064, 0.00095593, 0.00547744, 0.00796
856, 0.00151492,
       0.00136498, 0.00164471, 0.0006795 , 0.00015945, 0.00035862,
       0.00028487, 0.00043439, 0.00021986, 0.00053868, 0.00033683,
       0.00241146]), 'param_max_depth': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10, 10
, 10, 10, 50, 50, 50, 50],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_min_samples_split': masked_array(data=[5, 10, 100, 500,
5, 10, 100, 500, 5, 10, 100, 500, 5,
                   10, 100, 500],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'max_depth': 1, 'min_samples_split': 5}, {'max_dep
th': 1, 'min_samples_split': 10}, {'max_depth': 1, 'min_samples_split': 100}, {'max_depth
': 1, 'min_samples_split': 500}, {'max_depth': 5, 'min_samples_split': 5}, {'max_depth':
5, 'min_samples_split': 10}, {'max_depth': 5, 'min_samples_split': 100}, {'max_depth': 5,
'min_samples_split': 500}, {'max_depth': 10, 'min_samples_split': 5}, {'max_depth': 10, '
min_samples_split': 10}, {'max_depth': 10, 'min_samples_split': 100}, {'max_depth': 10, '
min_samples_split': 500}, {'max_depth': 50, 'min_samples_split': 5}, {'max_depth': 50, 'm
in_samples_split': 10}, {'max_depth': 50, 'min_samples_split': 100}, {'max_depth': 50, 'm
in_samples_split': 500}], 'split0_test_score': array([0.55464355, 0.55464355, 0.55464355,
0.55464355, 0.60139422,
       0.59993517, 0.60166057, 0.60096656, 0.58797183, 0.58805951,
       0.60614878, 0.60636155, 0.53794052, 0.54384271, 0.57824676,
       0.60715398]), 'split1_test_score': array([0.55622793, 0.55622793, 0.55622793, 0.55
622793, 0.61320534,
       0.61242136, 0.61326712, 0.6081512 , 0.56347002, 0.56924658,
       0.58827477, 0.59485125, 0.51486116, 0.51675506, 0.55327231,
       0.58644915]), 'split2_test_score': array([0.55404031, 0.55404031, 0.55404031, 0.55
404031, 0.60161345,
       0.60161345, 0.60138364, 0.60108843, 0.56037327, 0.55898001,
       0.57610442, 0.59237261, 0.52210908, 0.5302201 , 0.54136257,
       0.57734506]), 'split3_test_score': array([0.52985042, 0.52985042, 0.52985042, 0.52
985042, 0.61509275,
       0.61503172, 0.61362701, 0.61334739, 0.59157116, 0.58831226,
       0.60029206, 0.61200051, 0.51706539, 0.51865707, 0.54752297,
       0.59427295]), 'mean_test_score': array([0.54869055, 0.54869055, 0.54869055, 0.5486
9055, 0.60782644,
       0.60725042, 0.60748458, 0.60588839, 0.57584657, 0.57614959,
       0.59270501, 0.60139648, 0.52299404, 0.52736874, 0.55510115,
       0.59130528]), 'std_test_score': array([0.01090666, 0.01090666, 0.01090666, 0.01090
666, 0.00635819,
       0.0065684 , 0.00596464, 0.00519666, 0.01402575, 0.01257202,
       0.01154886, 0.00808343, 0.00902044, 0.01081741, 0.01401108,
       0.01093684]), 'rank_test_score': array([11, 11, 11, 11,  1,  3,  2,  4,  9,  8,  6
,  5, 16, 15, 10,  7],
      dtype=int32), 'split0_train_score': array([0.56830587, 0.56830587, 0.56830587, 0.56
830587, 0.66176416,
       0.66169442, 0.66030204, 0.66030204, 0.77442514, 0.76930606,
       0.74032489, 0.71831116, 0.99944367, 0.99502018, 0.93415773,
       0.78803866]), 'split1_train_score': array([0.56766152, 0.56766152, 0.56766152, 0.5
6766152, 0.67491328,
       0.67459423, 0.67363371, 0.66992127, 0.80210488, 0.79868867,
       0.76653439, 0.73284704, 0.99740779, 0.99369005, 0.92794725,
       0.81750482]), 'split2_train_score': array([0.56549373, 0.56549373, 0.56549373, 0.5
6549373, 0.66420382,
       0.66420382, 0.66391486, 0.66276152, 0.76944436, 0.76710409,
       0.7298822 , 0.70572312, 0.99893222, 0.99427346, 0.9333881 ,
       0.82762069]), 'split3_train_score': array([0.54640499, 0.54640499, 0.54640499, 0.5
4640499, 0.66734985,
       0.66734985, 0.66570085, 0.66216167, 0.78051417, 0.77531376,
       0.74892212, 0.70647928, 0.99855763, 0.99474922, 0.93797724,
       0.79587729]), 'mean_train_score': array([0.56196653, 0.56196653, 0.56196653, 0.561
96653, 0.66705778,
       0.66696058, 0.66588786, 0.66378663, 0.78162214, 0.77760315,
       0.7464159 , 0.71584015, 0.99858533, 0.99443323, 0.93336758,
       0.80726037]), 'std_train_score': array([0.00904465, 0.00904465, 0.00904465, 0.0090
4465, 0.00494878,
```

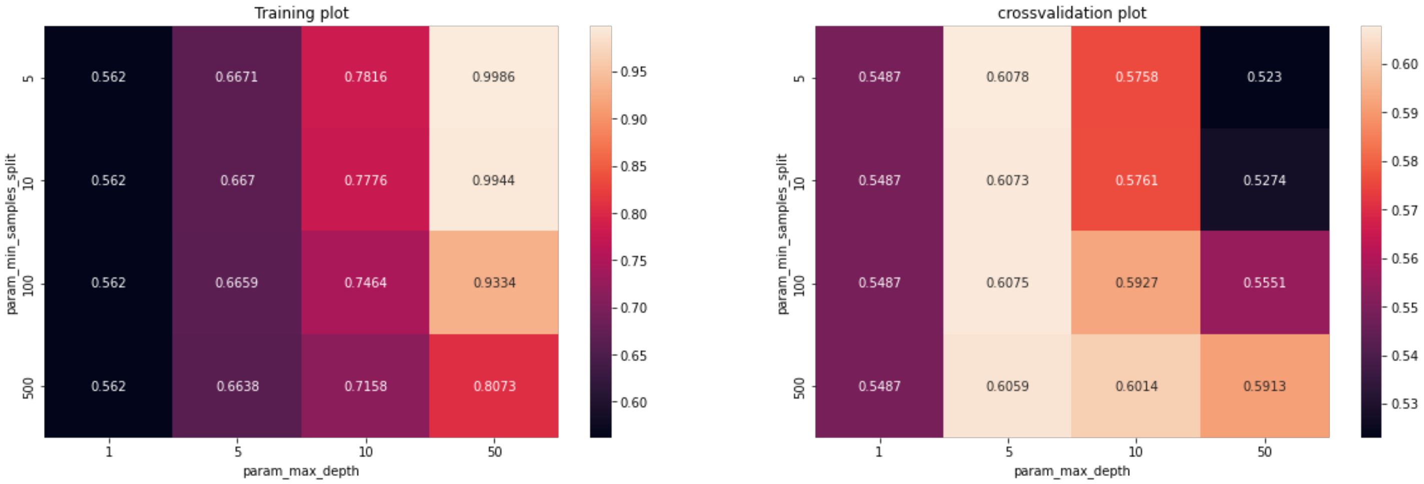
```
       0.00484139, 0.00487666, 0.00365608, 0.01245859, 0.01253906,
       0.01343036, 0.01101499, 0.00074908, 0.00050552, 0.00357959,
       0.01595729])}
```

HEATMAP for SET2

In [156]:

```
#https://blog.quantinsti.com/creating-heatmap-using-python-seaborn/
import seaborn as sns
score1 = pd.DataFrame(grid_search1.cv_results_).groupby(['param_min_samples_split', 'param_max_depth']).max().unstack()[['mean_test_score','mean_train_score']]
print(score1)
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(score1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(score1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Training plot')
ax[1].set_title('crossvalidation plot')
plt.show()
```

```
                        mean_test_score            ... mean_train_score          
param_max_depth                       1         5  ...               10        50
param_min_samples_split                            ...                           
5                              0.548691  0.607826  ...         0.781622  0.998585
10                             0.548691  0.607250  ...         0.777603  0.994433
100                            0.548691  0.607485  ...         0.746416  0.933368
500                            0.548691  0.605888  ...         0.715840  0.807260

[4 rows x 8 columns]
```



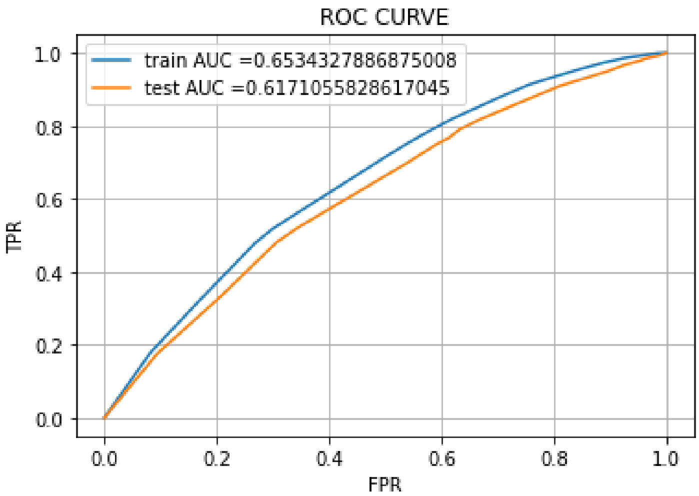
Using Best Estimator(params) and Plotting ROC for SET2 for test data

In [157]:

```
clf_best1=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, 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')
clf_best1.fit(X_train_set2,y_train)

y_train_predt_set2 = clf_best1.predict_proba(X_train_set2) [:,1]
y_test_predt_set2= clf_best1.predict_proba(X_test_set2) [:,1]

FPR_train_s2,TPR_train_s2,thres_train_s2=roc_curve(y_train,y_train_predt_set2)
FPR_test_s2,TPR_test_s2,thres_test_s2=roc_curve(y_test,y_test_predt_set2)

print("BEST train AUC score :",auc(FPR_train_s2, TPR_train_s2))
print("BEST test AUC score :",auc(FPR_test_s2, TPR_test_s2))
```

```
plt.plot(FPR_train_s2, TPR_train_s2, label="train AUC ="+str(auc(FPR_train_s2, TPR_train_
s2)))
plt.plot(FPR_test_s2, TPR_test_s2, label="test AUC ="+str(auc(FPR_test_s2, TPR_test_s2)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC CURVE")
plt.grid(True)
plt.show()
```

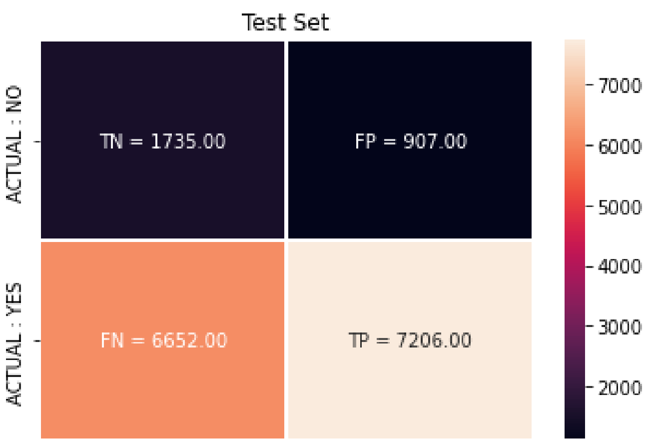
```
BEST train AUC score : 0.6534327886875008
BEST test AUC score : 0.6171055828617045
```



Confusion Matrix for SET2

In [158]:

```
def predict1(proba, threshould, fpr, tpr):
  temp = threshould[np.argmax(fpr*(1-tpr))]
  predictions = []
  global y_testpred_s2
  for i in proba:
    if i>=temp:
      predictions.append(1)
    else:
      predictions.append(0)
  y_testpred_s2= predictions
  return predictions

confusion_mat_test = confusion_matrix(y_test, predict1(y_test_predt_set2, thres_test_s2,
FPR_test_s2,TPR_test_s2))
key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key
.flatten(),confusion_mat_test.flatten())])).reshape(2,2)
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES']
,yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '')

plt.title('Test Set')
plt.show()
```

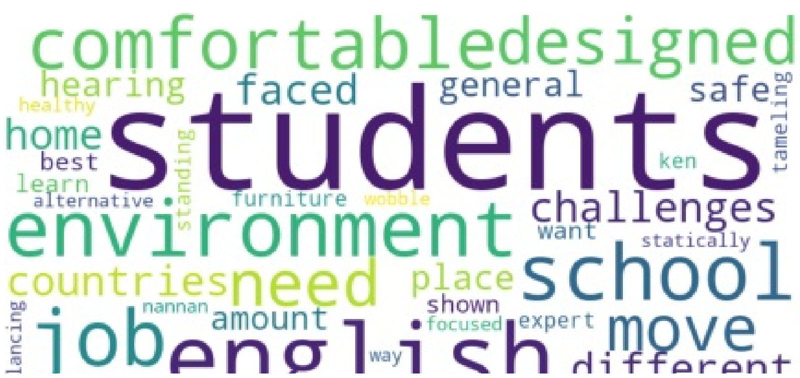


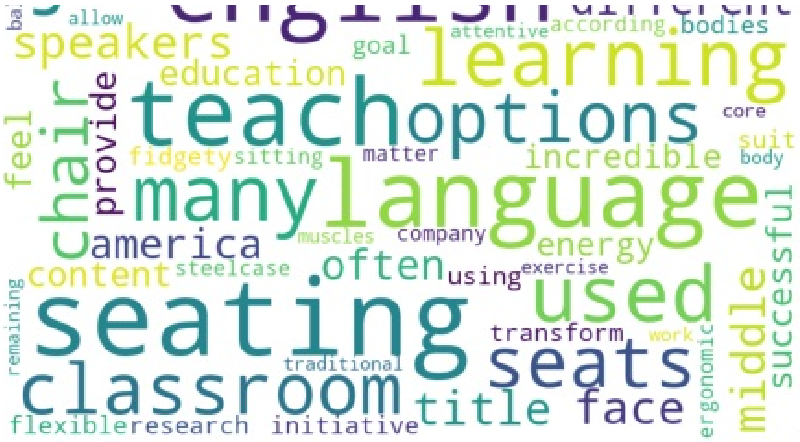
Word Cloud for set2 test data

In [159]:

```
#false positive data point gathering
#https://github.com/pskadasi/DecisionTrees_DonorsChoose/blob/master/

index2= []
for i in range(len(y_test)) :
  if (y_test.iloc[i] == 0) & (y_testpred_s2[i] == 1) :
    index2.append(i)

fp_essay2 = []
for i in index2 :
  fp_essay2.append(X_test['essay'].iloc[i])

fp_price2=[]
for i in index2:
    fp_price2.append(X_test['price'].iloc[i])

fp_teacher_number_of_previously_posted_projects2=[]
for i in index2:
    fp_teacher_number_of_previously_posted_projects2.append(X_test['teacher_number_of_pre
viously_posted_projects'].iloc[i])

print(len(fp_teacher_number_of_previously_posted_projects2))


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 =
stopwords, min_font_size = 10).generate(comment_words)


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

plt.show()
```

907





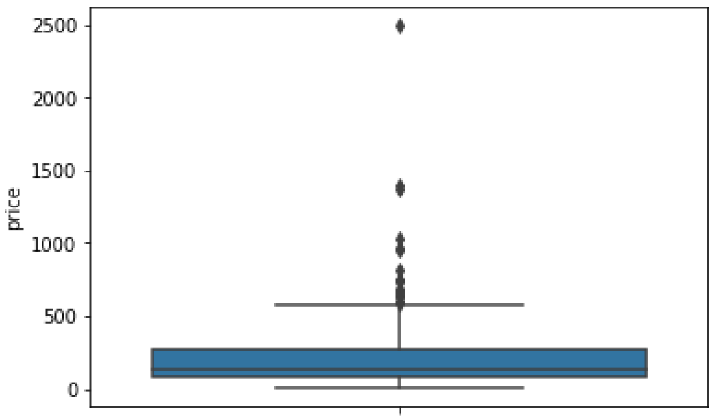
BOX PLOT of price feature of set2 test data

In [160]:

```
#boxplot of price of FPs
df2=pd.DataFrame(fp_price2)
df2.columns=['price']
sns.boxplot(data=df2,y='price')
```

Out[160]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7f10cf03d4d0&gt;
```



PDF of teacher_number_of_previously_posted_projects for FP of SET2

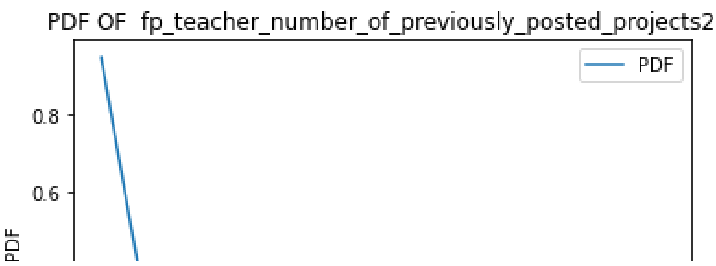
In [140]:

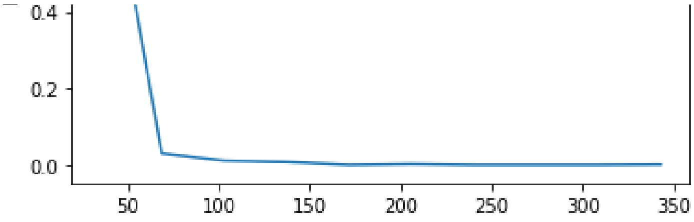
```
count1,bin1=np.histogram(fp_teacher_number_of_previously_posted_projects2,bins=10,density
=True)
PDF1=count1/sum(count1)
print(PDF1)
plt.plot(bin1[1:],PDF1,label="PDF")
plt.ylabel("PDF")
plt.legend()
plt.title("PDF OF  fp_teacher_number_of_previously_posted_projects2")
```

```
[0.94818082 0.02976847 0.01102536 0.00771775 0.         0.00220507
 0.         0.         0.         0.00110254]
```

Out[140]:

```
Text(0.5, 1.0, 'PDF OF  fp_teacher_number_of_previously_posted_projects2')
```





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

```
<img src='http://i.imgur.com/YVpIGGE.jpg' width=400px>
```

  ** **

Creating DATSET of top features

In [141]:

```
clf_bestfeature=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini'
,
                       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')
clf_bestfeature.fit(X_train_set1,y_train)

featurelist=clf_bestfeature.feature_importances_
indexoffeature=[]
for i in range(len(clf_bestfeature.feature_importances_)):
    if featurelist[i]!=0:
        indexoffeature.append(i)
print(len(indexoffeature))
print(len(featurelist))

X_train_bst_feature=X_train_set1[:,indexoffeature]
X_test_bst_feature=X_test_set1[:,indexoffeature]

print(X_train_bst_feature.shape)
```

```
851
8963
(22445, 851)
```

HYPERPARAMETER tuning for top feature datset

In [142]:

```
param_grid3 = {
              'min_samples_split': [5, 10, 100, 500],
              'max_depth' : [1, 5, 10, 50],
             }
```

```python
DT3 = DecisionTreeClassifier()
grid_search3= GridSearchCV(estimator=DT3, param_grid=param_grid3, scoring='roc_auc', cv=
4,n_jobs=-1,return_train_score=True, verbose=True)
grid_search3.fit(X_train_bst_feature,y_train)

print(grid_search3.best_estimator_)
print('train_accoracy:',grid_search3.score(X_train_bst_feature,y_train))
print('train_accuracy:',grid_search3.score(X_test_bst_feature,y_test))
print(grid_search3.cv_results_)
```

```
Fitting 4 folds for each of 16 candidates, totalling 64 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  64 out of  64 | elapsed:   51.6s finished
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=50, 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')
train_accoracy: 0.7861172568738634
train_accuracy: 0.6029367951720538
{'mean_fit_time': array([0.10815489, 0.1072011 , 0.10352957, 0.10559916, 0.5016796 ,
       0.48323309, 0.48504055, 0.45927739, 1.13022149, 1.10398018,
       0.99850094, 0.89655727, 5.13526404, 5.04049593, 4.6516872 ,
       3.83590734]), 'std_fit_time': array([0.00447141, 0.00634111, 0.004169  , 0.0034114
4, 0.02054307,
       0.00965015, 0.00608466, 0.00538351, 0.03688834, 0.03396085,
       0.0103167 , 0.02268204, 0.08982655, 0.08184183, 0.05574456,
       0.20858516]), 'mean_score_time': array([0.00983852, 0.00615674, 0.00639123, 0.0063
3287, 0.00690031,
       0.00683379, 0.00710154, 0.00669175, 0.00705868, 0.00694132,
       0.00695884, 0.00707263, 0.00810748, 0.0080964 , 0.00815189,
       0.00740498]), 'std_score_time': array([3.67765819e-03, 3.30438017e-04, 1.30521529e
-04, 8.23709450e-05,
       1.46897869e-04, 9.43087888e-05, 4.80508795e-04, 6.68034595e-05,
       1.20452286e-04, 8.70004063e-05, 9.35890108e-05, 2.57587326e-04,
       1.12754014e-04, 8.14021181e-05, 1.94568661e-04, 1.10884109e-03]), 'param_max_depth
': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50, 50],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'param_min_samples_split': masked_array(data=[5, 10, 100, 500,
5, 10, 100, 500, 5, 10, 100, 500, 5,
                   10, 100, 500],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'max_depth': 1, 'min_samples_split': 5}, {'max_dep
th': 1, 'min_samples_split': 10}, {'max_depth': 1, 'min_samples_split': 100}, {'max_depth
': 1, 'min_samples_split': 500}, {'max_depth': 5, 'min_samples_split': 5}, {'max_depth':
5, 'min_samples_split': 10}, {'max_depth': 5, 'min_samples_split': 100}, {'max_depth': 5,
'min_samples_split': 500}, {'max_depth': 10, 'min_samples_split': 5}, {'max_depth': 10, '
min_samples_split': 10}, {'max_depth': 10, 'min_samples_split': 100}, {'max_depth': 10, '
min_samples_split': 500}, {'max_depth': 50, 'min_samples_split': 5}, {'max_depth': 50, 'm
in_samples_split': 10}, {'max_depth': 50, 'min_samples_split': 100}, {'max_depth': 50, 'm
in_samples_split': 500}], 'split0_test_score': array([0.5431867 , 0.5431867 , 0.5431867 ,
0.5431867 , 0.613272  ,
       0.61331814, 0.61365435, 0.61246919, 0.58761131, 0.59155633,
       0.60787619, 0.61716911, 0.5145954 , 0.53827602, 0.60082047,
       0.63753925]), 'split1_test_score': array([0.55483671, 0.55483671, 0.55483671, 0.55
483671, 0.60022224,
       0.60042887, 0.60043383, 0.60159586, 0.58387051, 0.59354179,
       0.59842664, 0.61105897, 0.52775872, 0.51856543, 0.55794958,
       0.61414845]), 'split2_test_score': array([0.55212663, 0.55212663, 0.55212663, 0.55
212663, 0.59058334,
       0.59058381, 0.59128269, 0.59609302, 0.58036825, 0.58503305,
       0.59475795, 0.6089373 , 0.51170514, 0.52605966, 0.57220494,
       0.62534879]), 'split3_test_score': array([0.55979597, 0.55979597, 0.55979597, 0.55
979597, 0.61210674,
       0.61210744, 0.61222819, 0.61448157, 0.58187129, 0.58146077,
       0.61156296, 0.62409858, 0.52850779, 0.53894253, 0.59006128,
```

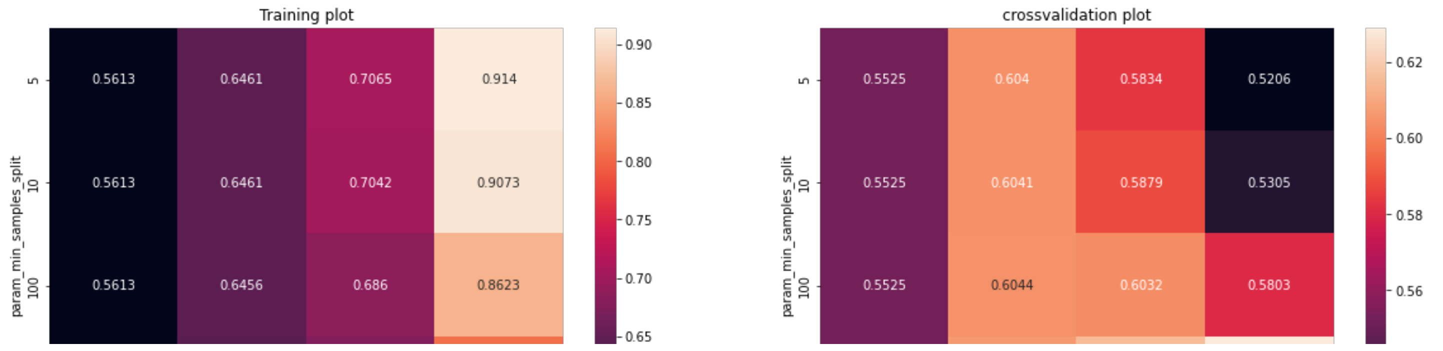
```
        0.63851459]), 'mean_test_score': array([0.55248651, 0.55248651, 0.55248651, 0.5524
8651, 0.60404608,
        0.60410957, 0.60439977, 0.60615991, 0.58343034, 0.58789799,
        0.60315593, 0.61531599, 0.52064176, 0.53046091, 0.58025907,
        0.62888777]), 'std_test_score': array([0.00603257, 0.00603257, 0.00603257, 0.00603
257, 0.00929999,
        0.00929055, 0.00914765, 0.00760312, 0.00271483, 0.00487036,
        0.00681636, 0.00590292, 0.0075655 , 0.00857157, 0.0164426 ,
        0.00996614]), 'rank_test_score': array([11, 11, 11, 11,  6,  5,  4,  3,  9,  8,  7
,  2, 16, 15, 10,  1],
      dtype=int32), 'split0_train_score': array([0.55764915, 0.55764915, 0.55764915, 0.55
764915, 0.64664129,
        0.64664129, 0.6462355 , 0.64110783, 0.70787666, 0.70679597,
        0.68433817, 0.66970086, 0.92528832, 0.92149867, 0.86714906,
        0.79774453]), 'split1_train_score': array([0.56453002, 0.56453002, 0.56453002, 0.5
6453002, 0.64626037,
        0.6462437 , 0.64618396, 0.64483245, 0.7093146 , 0.70712288,
        0.69737658, 0.68641305, 0.93242778, 0.92888673, 0.87916732,
        0.82717522]), 'split2_train_score': array([0.56094625, 0.56094625, 0.56094625, 0.5
6094625, 0.64540681,
        0.64533674, 0.64390938, 0.6401605 , 0.70022749, 0.69760372,
        0.67893872, 0.66894522, 0.89303226, 0.88222267, 0.83473839,
        0.78642552]), 'split3_train_score': array([0.56227023, 0.56227023, 0.56227023, 0.5
6227023, 0.64624734,
        0.64617741, 0.64598474, 0.64326017, 0.70839464, 0.70512546,
        0.68336695, 0.67256949, 0.90543248, 0.89665899, 0.86810655,
        0.81956486]), 'mean_train_score': array([0.56134891, 0.56134891, 0.56134891, 0.561
34891, 0.64613895,
        0.64609978, 0.6455784 , 0.64234024, 0.70645335, 0.70416201,
        0.68600511, 0.67440715, 0.91404521, 0.90731676, 0.86229033,
        0.80772753]), 'std_train_score': array([0.00249091, 0.00249091, 0.00249091, 0.0024
9091, 0.00045135,
        0.00047492, 0.00096814, 0.00182534, 0.0036312 , 0.00386147,
        0.00687355, 0.00706223, 0.01565287, 0.01877293, 0.01659351,
        0.01636895])}
```

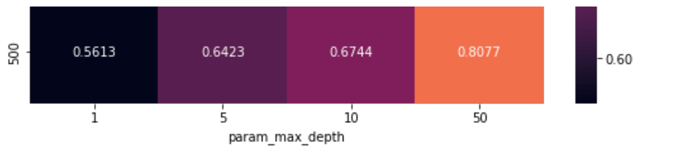
HEATMAP for hyperparameter

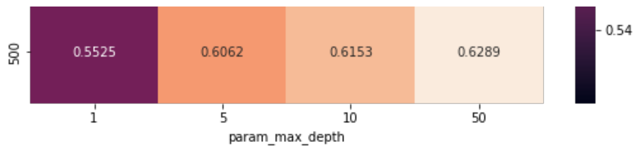
In [143]:

```python
score2 = pd.DataFrame(grid_search3.cv_results_).groupby(['param_min_samples_split', 'par
am_max_depth']).max().unstack()[['mean_test_score','mean_train_score']]
print(score2)
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(score2.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(score2.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Training plot')
ax[1].set_title('crossvalidation plot')
plt.show()
```

```
                        mean_test_score            ... mean_train_score
param_max_depth                       1         5  ...               10         50
param_min_samples_split                            ...
5                              0.552487  0.604046  ...         0.706453  0.914045
10                             0.552487  0.604110  ...         0.704162  0.907317
100                            0.552487  0.604400  ...         0.686005  0.862290
500                            0.552487  0.606160  ...         0.674407  0.807728

[4 rows x 8 columns]
```





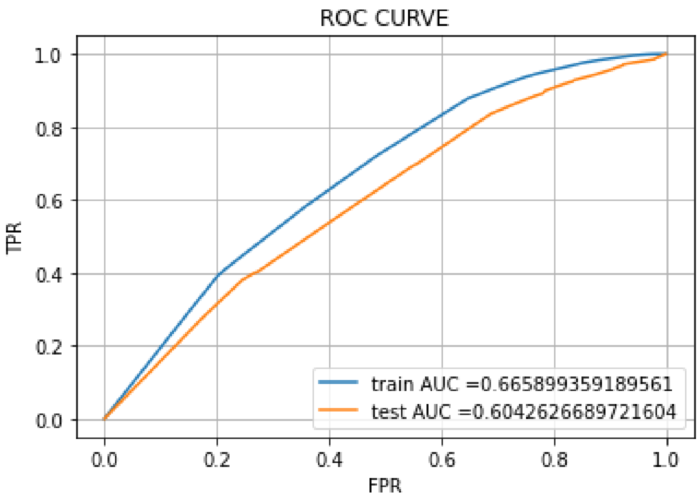


Using Best Estimator(params) and Plotting ROC for top features

In [144]:

```
clf_best_topfeature=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='g
ini',
                       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')
clf_best_topfeature.fit(X_train_bst_feature,y_train)

y_train_predt_bst_fea = clf_best_topfeature.predict_proba(X_train_bst_feature) [:,1]
y_test_predt_bst_fea= clf_best_topfeature.predict_proba(X_test_bst_feature) [:,1]

FPR_train_bst_fea,TPR_train_bst_fea,thres_train_bst_fea=roc_curve(y_train,y_train_predt_b
st_fea)
FPR_test_bst_fea,TPR_test_bst_fea,thres_test_bst_fea=roc_curve(y_test,y_test_predt_bst_fe
a)

print("BEST train AUC score :",auc(FPR_train_bst_fea, TPR_train_bst_fea))
print("BEST test AUC score :",auc(FPR_test_bst_fea, TPR_test_bst_fea))



plt.plot(FPR_train_bst_fea, TPR_train_bst_fea, label="train AUC ="+str(auc(FPR_train_bst_
fea, TPR_train_bst_fea)))
plt.plot(FPR_test_bst_fea, TPR_test_bst_fea, label="test AUC ="+str(auc(FPR_test_bst_fea,
TPR_test_bst_fea)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC CURVE")
plt.grid(True)
plt.show()
```

```
BEST train AUC score : 0.665899359189561
BEST test AUC score : 0.6042626689721604
```

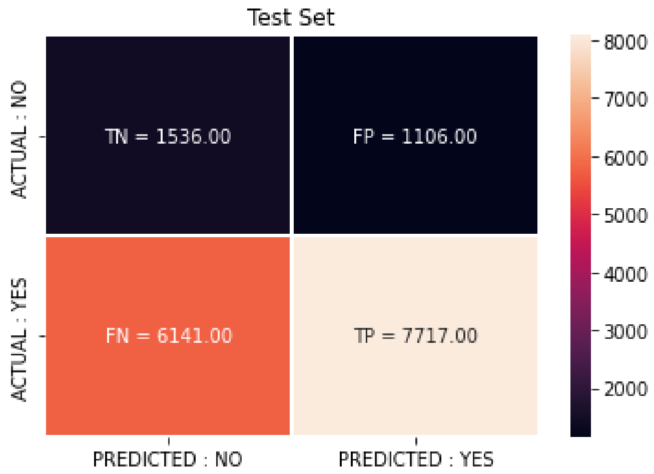


Confusion Matrix for top feature set for test data

In [145]:

```
def predict3(proba, threshould, fpr, tpr):
  temp = threshould[np.argmax(fpr*(1-tpr))]
  predictions = []
```

```
  global y_testpred_bst_feature
  for i in proba:
    if i>=temp:
      predictions.append(1)
    else:
      predictions.append(0)
  y_testpred_bst_feature= predictions
  return predictions

confusion_mat_test = confusion_matrix(y_test, predict3(y_test_predt_bst_fea, thres_test_b
st_fea, FPR_test_bst_fea,TPR_test_bst_fea))
key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key
.flatten(),confusion_mat_test.flatten())])).reshape(2,2)
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES']
,yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '')

plt.title('Test Set')
plt.show()
```



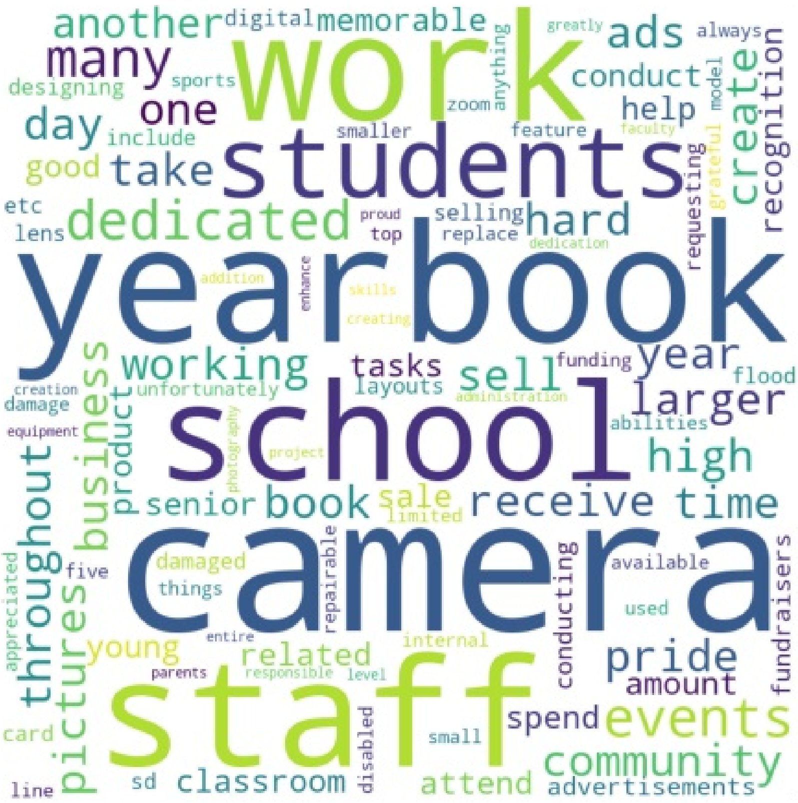
Word Cloud for FP test data of top feature datset

In [146]:

```
#false positive data point gathering
#https://github.com/pskadasi/DecisionTrees_DonorsChoose/blob/master/

index3= []
for i in range(len(y_test)) :
  if (y_test.iloc[i] == 0) & (y_testpred_bst_feature[i] == 1) :
    index3.append(i)

fp_essay3 = []
for i in index3 :
  fp_essay3.append(X_test['essay'].iloc[i])

fp_price3=[]
for i in index3:
    fp_price3.append(X_test['price'].iloc[i])

fp_teacher_number_of_previously_posted_projects3=[]
for i in index3:
    fp_teacher_number_of_previously_posted_projects3.append(X_test['teacher_number_of_pre
viously_posted_projects'].iloc[i])

print(len(fp_teacher_number_of_previously_posted_projects3))


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 =
stopwords, min_font_size = 10).generate(comment_words)


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

plt.show()
```

1106



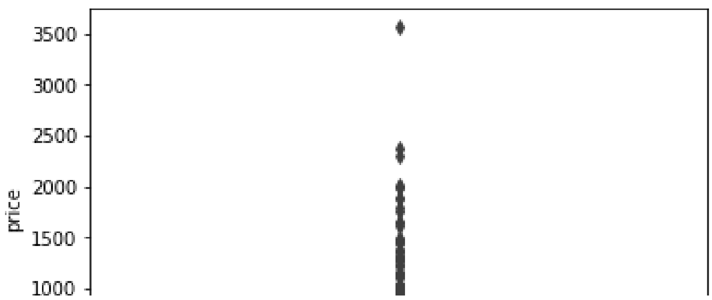
BOX PLOT for price of FP dataset of top feature

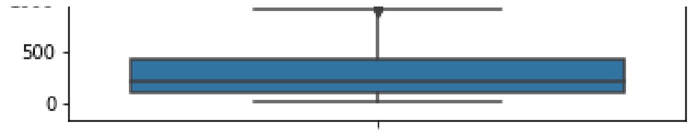
In [147]:

```
df3=pd.DataFrame(fp_price3)
df3.columns=['price']
sns.boxplot(data=df3,y='price')
```

Out[147]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f10ce4f5f90>





PDF of teacher_number_of_previously_posted_projects for FP dataset of top feature

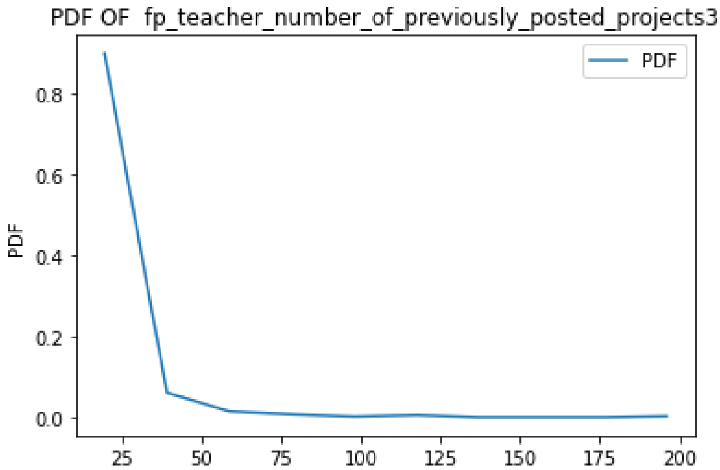
In [148]:

```
count3,bin3=np.histogram(fp_teacher_number_of_previously_posted_projects3,bins=10,density
=True)
PDF3=count3/sum(count3)
print(PDF3)
plt.plot(bin3[1:],PDF3,label="PDF")
plt.ylabel("PDF")
plt.legend()
plt.title("PDF OF  fp_teacher_number_of_previously_posted_projects3")
```

```
[0.89963834 0.06148282 0.01537071 0.00813743 0.00271248 0.00632911
 0.00090416 0.00090416 0.00090416 0.00361664]
```

Out[148]:

```
Text(0.5, 1.0, 'PDF OF  fp_teacher_number_of_previously_posted_projects3')
```



SUMMERY

In [164]:

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

x = PrettyTable()

x.field_names = ["Vectorizor used", "MAX DEPTH", "MIN -SAMPLE", " TRAIN AUC", "TEST AUC"
]

x.add_row(["TFIDF", 10, 500,0.665,0.604])
x.add_row(["W2V",5,500, 0.653,0.617])
x.add_row(["TOP FEATURE", 10, 500, 0.665,0.604])


print(x)
```

```
+-----------------+-----------+-------------+------------+----------+
| Vectorizor used | MAX DEPTH | MIN -SAMPLE |  TRAIN AUC | TEST AUC |
+-----------------+-----------+-------------+------------+----------+
|      TFIDF      |     10    |     500     |   0.665    |  0.604   |
|       W2V       |     5     |     500     |   0.653    |  0.617   |
|   TOP FEATURE   |     10    |     500     |   0.665    |  0.604   |
+-----------------+-----------+-------------+------------+----------+
```