In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import chart_studio.plotly
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

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

In [2]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
project_data.project_is_approved.value_counts()
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'scho
ol_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

Out[3]:
```
1    92706
0    16542
Name: project_is_approved, dtype: int64
```

In [4]:
```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[4]:

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

### Preprocessing of project_subject_categories

In [5]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.co

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

In [6]:
```
project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
project_data
```

| | | | | | | |
|---|---|---|---|---|---|---|
| **109239** | 156548 | p103958 | 8b9a9dc5bd4aa0301b0ff416e2ed29f6 | Mrs. | MN | |
| **109240** | 93971 | p257729 | 58c112dcb2f1634a4d4236bf0dcdcb31 | Mrs. | MD | |
| **109241** | 36517 | p180358 | 3e5c98480f4f39d465837b2955df6ae0 | Mrs. | MD | |
| **109242** | 34811 | p080323 | fe10e79b7aeb570dfac87eeea7e9a8f1 | Mrs. | SC | |
| **109243** | 38267 | p048540 | fadf72d6cd83ce6074f9be78a6fcd374 | Mr. | MO | |

In [7]:
```
from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

## Preprocessing of project_subject_subcategories

In [8]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.co

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science",
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trail
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())
```

In [9]:
```
project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
project_data.tail()
```

Out[9]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | school_state | proj |
|---|---|---|---|---|---|---|
| **109243** | 38267 | p048540 | fadf72d6cd83ce6074f9be78a6fcd374 | Mr. | MO | |
| **109244** | 169142 | p166281 | 1984d915cc8b91aa16b4d1e6e39296c6 | Ms. | NJ | |
| **109245** | 143653 | p155633 | cdbfd04aa041dc6739e9e576b1fb1478 | Mrs. | NJ | |
| **109246** | 164599 | p206114 | 6d5675dbfafa1371f0e2f6f1b716fe2d | Mrs. | NY | |
| **109247** | 128381 | p191189 | ca25d5573f2bd2660f7850a886395927 | Ms. | VA | |

In [10]:
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

### Grade conversion

In [11]:
```
#reference :https://github.com/robinsones/Predicting-Sucess-on-DonorsChoose/blob/

conv_grade_dict = {'Grades PreK-2' : 1.,'Grades 3-5' : 2.,'Grades 6-8':3.,'Grades
project_data['grade_level'] = project_data["project_grade_category"].apply(conv_g


project_data['project_grade_category'] = [ gr_ca.replace(' ','_') for gr_ca in pr
project_data['project_grade_category'] = [ gr_ca.replace('-','_') for gr_ca in pr
```

In [12]:
```
project_data['project_grade_category'].value_counts()
```

Out[12]:
```
Grades_PreK_2    44225
Grades_3_5       37137
Grades_6_8       16923
Grades_9_12      10963
Name: project_grade_category, dtype: int64
```

### Teacher prefix

In [13]:
```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')
```

In [14]:
```
def replace_cate(lst):          # Removing (.) in Mrs.
    return lst.replace('.','')


project_data['teacher_prefix']= project_data['teacher_prefix'].astype(str).apply
```

In [15]:
```
preprocessed_teacher_prefix = []

for teach_prefix in tqdm(project_data["teacher_prefix"]):
    preprocessed_teacher_prefix.append(teach_prefix.strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 109
248/109248 [00:00<00:00, 1309692.01it/s]
```

## Feature Engineering

In [16]:
```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [17]:
```
project_data.head(2)
```

Out[17]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | school_state | project_sul |
|---|---|---|---|---|---|---|
| **0** | 160221 | p253737 | c90749f5d961ff158d4b4d1e7dc665fc | Mrs | IN | 20 |
| **1** | 140945 | p258326 | 897464ce9ddc600bced1151f324dd63a | Mr | FL | 20 |

In [18]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [19]:
```
# https://gist.github.com/sebleier/554280
# Removing stopwords
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 't
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than'
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn
            'won', "won't", 'wouldn', "wouldn't"]
```

In [20]:
```
# https://stackoverflow.com/a/47091490/4084039
def feature_engineering(data):
    from tqdm import tqdm
    preprocessed_essays = []
    # tqdm is for printing the status bar
    for sentance in tqdm(data.values):
        pharse = decontracted(sentance)
        pharse = pharse.replace('\\r', ' ')
        pharse = pharse.replace('\\"', ' ')
        pharse = pharse.replace('\\n', ' ')
        pharse = pharse.replace('nan',' ')
        pharse = re.sub('[^A-Za-z0-9]+', ' ', pharse)
        # https://gist.github.com/sebleier/554280
        pharse = ' '.join(e for e in pharse.split() if e.lower() not in stopwords
        preprocessed_essays.append(pharse.lower().strip())
    return preprocessed_essays
```

In [21]:
```
preprocessed_essays=feature_engineering(project_data['essay'])
```

```
100%|████████████████████████████████████████████████████████████████████████|
109248/109248 [01:19<00:00, 1374.36it/s]
```

In [22]:
```
# Fearturing the Project_title as well
preprocessed_project_title=feature_engineering(project_data['project_title'])
```

```
100%|████████████████████████████████████████████████████████████████████████| 1
09248/109248 [00:02<00:00, 39446.08it/s]
```

In [23]:
```
project_data['featured_title']=preprocessed_project_title
project_data.drop(['project_title'], axis=1, inplace=True)
```

**Merging the data**

In [24]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [25]:
```
project_data.drop(['project_resource_summary'], axis=1, inplace=True)
project_data.drop(['Unnamed: 0'], axis=1, inplace=True)
project_data.drop(['id'], axis=1, inplace=True)
project_data.drop(['teacher_id'], axis=1, inplace=True)
```

In [26]:
```
project_data.head()
```

Out[26]:

| acher_number_of_previously_posted_projects | project_is_approved | clean_categories | clean_subcategor |
|---|---|---|---|
| 0 | 0 | Literacy_Language | ESL Liter |
| 7 | 1 | History_Civics Health_Sports | Civics_Governm TeamSp |
| 1 | 0 | Health_Sports | Health_Welln TeamSp |
| 4 | 1 | Literacy_Language Math_Science | Literacy Mathema |
| 1 | 1 | Math_Science | Mathema |

### Splitting data

In [27]:
```
#https://stackoverflow.com/questions/29763620
X=project_data.loc[:, project_data.columns != 'project_is_approved']
y=project_data['project_is_approved']
X.shape
```

Out[27]: (109248, 16)

In [28]:
```
from sklearn.model_selection import train_test_split

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

print(X_train.shape)
print(X_test.shape)

print(y_train.shape)
print(y_test.shape)
```

```
(73196, 16)
(36052, 16)
(73196,)
(36052,)
```

### Vectorizing the data

### Categorical data

In [29]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['clean_categories'].values)
feature_names_bow=[]
feature_names_tfidf=[]
# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories=vectorizer.transform(X_train['clean_categories'].values
X_test_clean_categories=vectorizer.transform(X_test['clean_categories'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_clean_categories.shape)
feature_names_bow.extend(vectorizer.get_feature_names())
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
['AppliedLearning', 'Care_Hunger', 'Health_Sports', 'History_Civics', 'Literacy
_Language', 'Math_Science', 'Music_Arts', 'SpecialNeeds', 'Warmth']
Shape of matrix after one hot encodig  (73196, 9)
```

In [30]:
```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_sub_categories=vectorizer.transform(X_train['clean_subcategories']
X_test_clean_sub_categories=vectorizer.transform(X_test['clean_subcategories'].va


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_clean_sub_categories.shape
feature_names_bow.extend(vectorizer.get_feature_names())
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
['AppliedSciences', 'Care_Hunger', 'CharacterEducation', 'Civics_Government',
'College_CareerPrep', 'CommunityService', 'ESL', 'EarlyDevelopment', 'Economic
s', 'EnvironmentalScience', 'Extracurricular', 'FinancialLiteracy', 'ForeignLan
guages', 'Gym_Fitness', 'Health_LifeScience', 'Health_Wellness', 'History_Geogr
aphy', 'Literacy', 'Literature_Writing', 'Mathematics', 'Music', 'NutritionEduc
ation', 'Other', 'ParentInvolvement', 'PerformingArts', 'SocialSciences', 'Spec
ialNeeds', 'TeamSports', 'VisualArts', 'Warmth']
Shape of matrix after one hot encodig  (73196, 30)
```

In [31]:
```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['school_state'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_skl_state=vectorizer.transform(X_train['school_state'].values)
X_test_skl_state=vectorizer.transform(X_test['school_state'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_skl_state.shape)
feature_names_bow.extend(vectorizer.get_feature_names())
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA',
'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS',
'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH', 'OK', 'OR', 'PA',
'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV', 'WY']
Shape of matrix after one hot encodig  (73196, 51)
```

In [32]:
```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['teacher_prefix'].values.astype("U"))

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix=vectorizer.transform(X_train['teacher_prefix'].values.asty
X_test_teacher_prefix=vectorizer.transform(X_test['teacher_prefix'].values.astype

print("Shape of matrix after one hot encodig ",X_train_teacher_prefix.shape)
feature_names_bow.extend(vectorizer.get_feature_names())
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
Shape of matrix after one hot encodig  (73196, 6)
```

In [33]:
```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_level=vectorizer.transform(X_train['project_grade_category'].value
X_test_grade_level=vectorizer.transform(X_test['project_grade_category'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_grade_level.shape)
feature_names_bow.extend(vectorizer.get_feature_names())
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
Shape of matrix after one hot encodig  (73196, 4)
```

**Text data**

**BOW**

In [34]:
```
bow_essay_vectorizer = CountVectorizer(min_df=15)
bow_essay_vectorizer.fit(X_train['essay'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow=bow_essay_vectorizer.transform(X_train['essay'].values)
X_test_essay_bow=bow_essay_vectorizer.transform(X_test['essay'].values)


print("Shape of matrix after one hot encodig ",X_train_essay_bow.shape)
print("Shape of matrix after one hot encodig ",X_test_essay_bow.shape)

feature_names_bow.extend(bow_essay_vectorizer.get_feature_names())
```

```
Shape of matrix after one hot encodig  (73196, 12378)
Shape of matrix after one hot encodig  (36052, 12378)
```

In [35]:
```
bow_title_vectorizer = CountVectorizer(min_df=15)
bow_title_vectorizer.fit(X_train['featured_title'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_featured_title_bow=bow_title_vectorizer.transform(X_train['featured_titl
X_test_featured_title_bow=bow_title_vectorizer.transform(X_test['featured_title'


print(" after one hot encodig ",X_train_featured_title_bow.shape)
feature_names_bow.extend(bow_title_vectorizer.get_feature_names())
```

```
 after one hot encodig  (73196, 1962)
```

**TFIDF vectorizer**

In [36]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=15)
vectorizer.fit(X_train['essay'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf=vectorizer.transform(X_train['essay'].values)
X_test_essay_tfidf=vectorizer.transform(X_test['essay'].values)


print("Shape of matrix after one hot encodig ",X_train_essay_tfidf.shape)
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
Shape of matrix after one hot encodig  (73196, 12378)
```

In [37]:
```
vectorizer = TfidfVectorizer(min_df=15)

# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer.fit(X_train['featured_title'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_featured_title_tfidf=vectorizer.transform(X_train['featured_title'].valu
X_test_featured_title_tfidf=vectorizer.transform(X_test['featured_title'].values


print("Shape of matrix after one hot encodig ",X_train_featured_title_tfidf.shap
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
Shape of matrix after one hot encodig  (73196, 1962)
```

In [38]:
```
from sklearn.preprocessing import Normalizer

def normalizer(a,b):
    normalizer = Normalizer()
    normalizer.fit(a[b].values.reshape(1,-1))
    out=normalizer.transform(a[b].values.reshape(1,-1))
    return out

X_train_price_standardized=normalizer(X_train,'price')
X_test_price_standardized=normalizer(X_test,'price')

print(X_train_price_standardized.shape, y_train.shape)
print(X_test_price_standardized.shape, y_test.shape)
```

```
(1, 73196) (73196,)
(1, 36052) (36052,)
```

In [39]:
```
X_train_price_standardized = X_train_price_standardized.reshape(-1,1)
X_test_price_standardized = X_test_price_standardized.reshape(-1,1)
```

In [40]:
```
X_train_prev_proj=normalizer(X_train,'teacher_number_of_previously_posted_projec
X_test_prev_proj=normalizer(X_test,'teacher_number_of_previously_posted_projects
```

In [41]:
```
X_train_prev_proj = X_train_prev_proj.reshape(-1,1)
X_test_prev_proj = X_test_prev_proj.reshape(-1,1)
```

### AVG W2V

In [42]:
```
#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
```

In [43]:
```
model = loadGloveModel('glove.42B.300d.txt')
```

```
Loading Glove Model

1917494it [04:37, 6921.81it/s]

Done. 1917494  words loaded!
```

In [44]:
```
glove_words = set(model.keys())
```

In [45]:
```
#for essay
# average Word2Vec
# compute average word2vec for each review.
def func(wordlist):

  train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored i
  for sentence in tqdm(wordlist): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length # we are taking 
    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
    train_avg_w2v_vectors.append(vector)

  print(len(train_avg_w2v_vectors))
  print(len(train_avg_w2v_vectors[0]))
  return train_avg_w2v_vectors
```

In [46]:
```
train_avg_w2v_vectors=func(X_train['essay'].values)
test_avg_w2v_vectors=func(X_test['essay'].values)

test_avg_w2v_vectors_title=func(X_train['featured_title'].values)
train_avg_w2v_vectors_title=func(X_test['featured_title'].values)
```

```
100%|████████████████████████████████████████████████████████████████
██| 73196/73196 [03:46<00:00, 322.57it/s]

73196
300

100%|████████████████████████████████████████████████████████████████
██| 36052/36052 [00:47<00:00, 753.98it/s]

36052
300

100%|████████████████████████████████████████████████████████████████
| 73196/73196 [00:12<00:00, 5702.03it/s]

73196
300

100%|████████████████████████████████████████████████████████████████
| 36052/36052 [00:05<00:00, 6536.08it/s]

36052
300
```

**TFIDF weighted W2V**

In [47]:
```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [48]:
```
# computing average word2vec
def tf_idf_done(word_list):
    train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is
    for sentence in tqdm(word_list): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/rev
        for word in sentence.split():#.split(): # for each word in a review/sente
            if (word in glove_words) and (word in tfidf_words):
              #vec = model.wv[word]
                vec = model[word] # getting the vector for each word
# here we are multiplying idf value(dictionary[word]) and the tf value((sentence
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        train_title_tfidf_w2v_vectors.append(vector)
    print(len(train_title_tfidf_w2v_vectors))
    print(len(train_title_tfidf_w2v_vectors[0]))
    return train_title_tfidf_w2v_vectors
```

In [49]:
```
train_tfidf_w2v_vectors=tf_idf_done(X_train['essay'].values)
test_tfidf_w2v_vectors=tf_idf_done(X_test['essay'].values)
```

```
100%|████████████████████████████████████████████████████████████████████
██| 73196/73196 [05:41<00:00, 214.51it/s]

73196
300

100%|████████████████████████████████████████████████████████████████████
██| 36052/36052 [03:36<00:00, 166.51it/s]

36052
300
```

In [50]:
```
train_title_tfidf_w2v_vectors=tf_idf_done(X_train['featured_title'].values)
test_title_tfidf_w2v_vectors=tf_idf_done(X_test['featured_title'].values)
```

```
100%|████████████████████████████████████████████████████████████████████████████|
73196/73196 [00:03<00:00, 22798.09it/s]

73196
300

100%|████████████████████████████████████████████████████████████████████████████|
36052/36052 [00:01<00:00, 22946.84it/s]

36052
300
```

**Stacking the data**

**TFIDF**

In [51]:
```
from scipy.sparse import hstack
X_train_tfidf = hstack((X_train_clean_categories, X_train_clean_sub_categories,X_
            X_train_grade_level,X_train_prev_proj,X_train_price_standardized,X_t


X_test_tfidf = hstack((X_test_clean_categories, X_test_clean_sub_categories,X_te
            X_test_grade_level,X_test_prev_proj,X_test_price_standardized,X_test_
```

**TFIDF weighted W2V**

In [52]:
```
X_train_tfidf_avg = hstack((X_train_clean_categories, X_train_clean_sub_categorie
            X_train_grade_level,X_train_prev_proj,X_train_price_standardized,
            train_tfidf_w2v_vectors,train_title_tfidf_w2v_vectors
            )).tocsr()


X_test_tfidf_avg = hstack((X_test_clean_categories, X_test_clean_sub_categories,X
            X_test_grade_level,X_test_prev_proj,X_test_price_standardized,
            test_tfidf_w2v_vectors,test_title_tfidf_w2v_vectors
            )).tocsr()
```

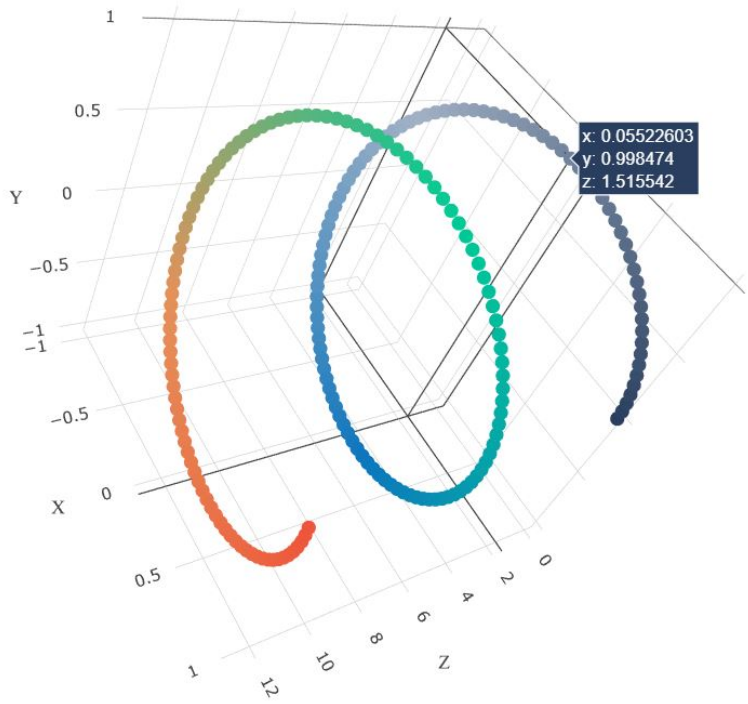
# Assignment 8: DT

1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_eassay (TFIDF)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_samples_split` in range [5, 10, 100, 500])**

- Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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**

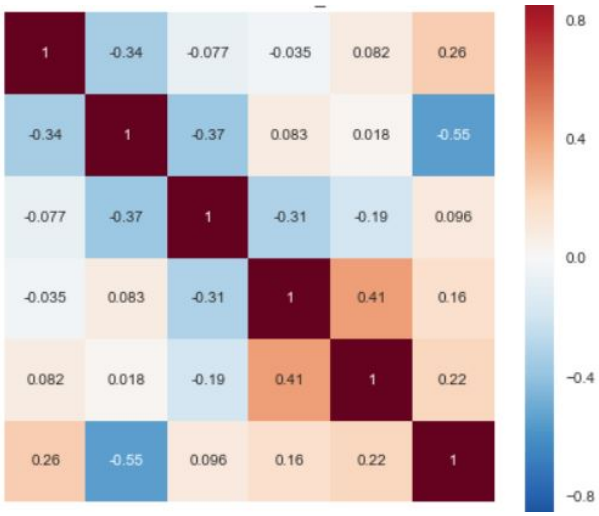
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure




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*

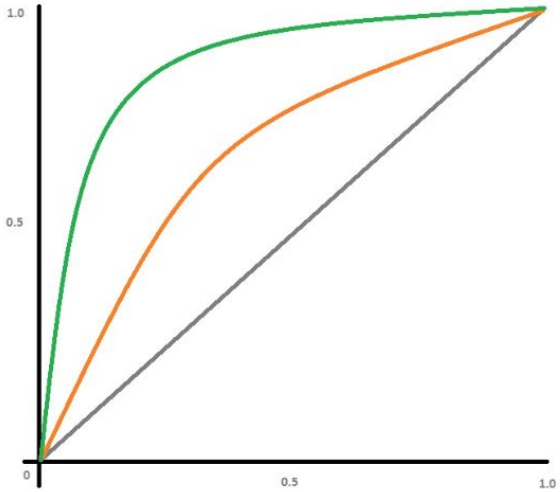
**or**

- 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.




- Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) 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`

4. **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.

5. 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 |

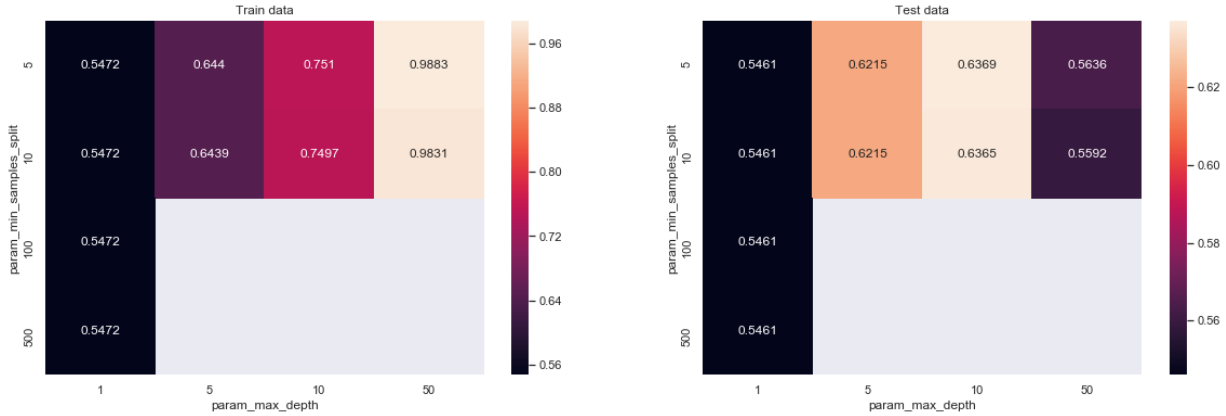
# 1. Decision Tree

**TFIDF**

In [53]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
```

In [85]:
```
dt2 = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10,100,500]}
clff = RandomizedSearchCV(dt2, parameters, cv=3, scoring='roc_auc',return_train_s
tfifc_DT = clff.fit(X_train_tfidf, y_train)
```

In [86]:
```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clff.cv_results_).groupby(['param_min_samples_split',

fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train data')
ax[1].set_title('Test data')
plt.show()
```




***Best parameter***

In [87]:
```
print(clff.best_estimator_)
print(clff.score(X_train_tfidf, y_train))
print(clff.score(X_test_tfidf, y_test))
```

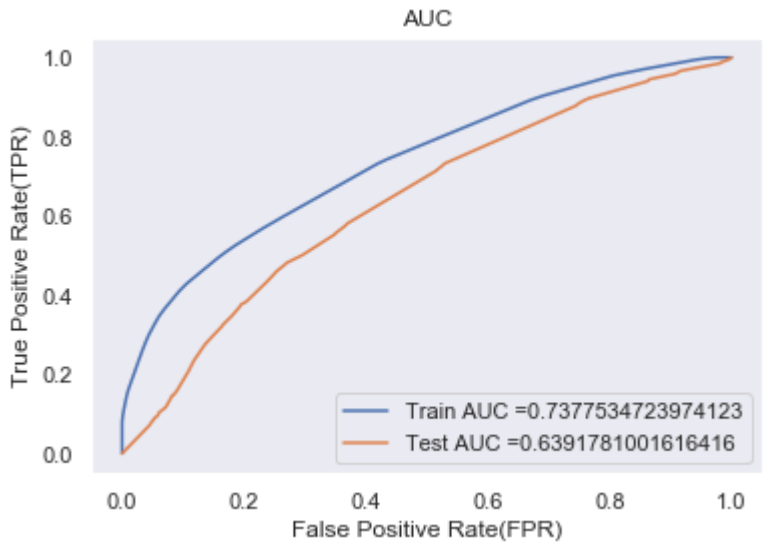
```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=10, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
0.7377534723974123
0.6385569023445679
```

In [88]:
```
clff.best_estimator_.fit(X_train_tfidf, y_train)
```

Out[88]:
```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=10, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [89]:
```
def pred_prob(clff, data):
    y_pred = []
    y_pred = clff.predict_proba(data)[:,1]
    return y_pred
```

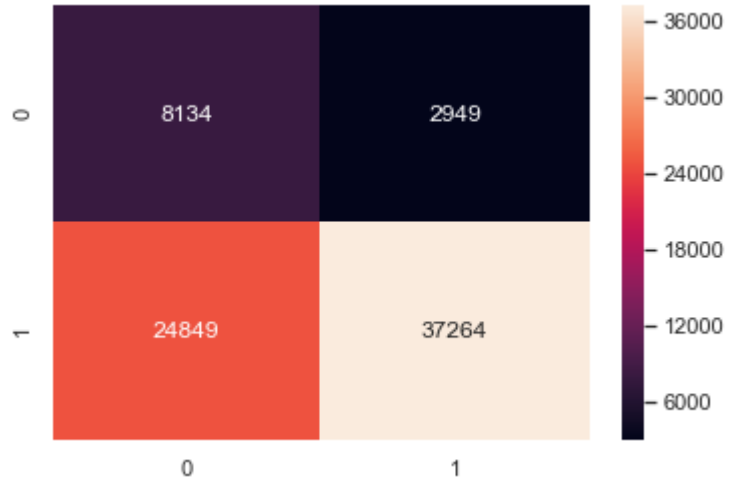
In [90]:
```
from sklearn.metrics import roc_curve, auc

# bow = MultinomialNB(alpha = clff.best_params_['alpha'],class_prior = [0.5,0.5],
tfidf= clff.best_estimator_
# tfidf=DecisionTreeClassifier (class_weight = 'balanced',max_depth=10,min_sample
tfidf.fit(X_train_tfidf, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
# not the predicted outputs

y_train_pred = pred_prob(tfidf,X_train_tfidf)
y_test_pred = pred_prob(tfidf,X_test_tfidf)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

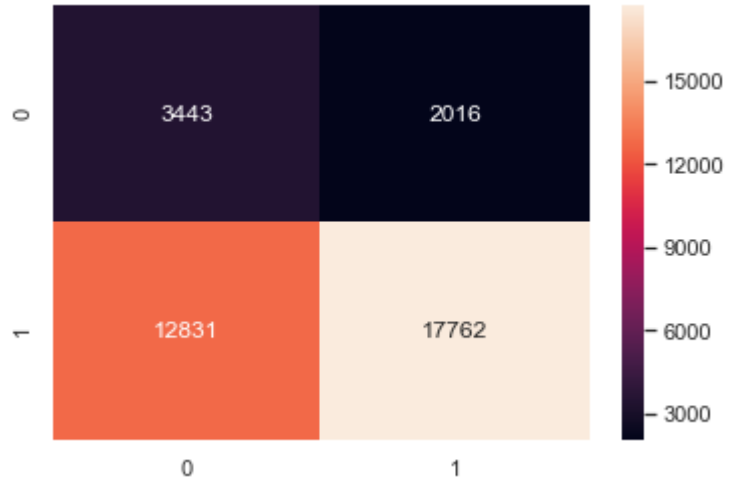


In [91]:
```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print("tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [92]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
tpr*(1-fpr) 0.44030518555302994 for threshold 0.468
```



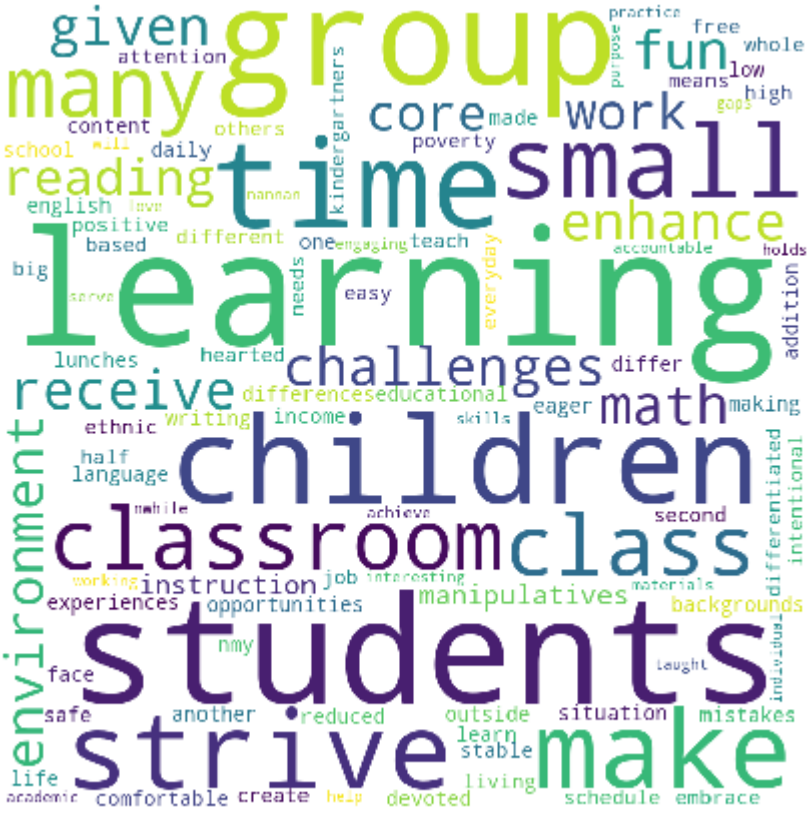




## False positive data point

In [93]:
```
predictions=predict_with_best_t(y_train_pred, best_t)
```

In [94]:
```
#https://www.google.com/search?q=geeks+for+geeks+false+positive&rlz=1C1SQJL_enIN&
#https://github.com/pskadasi/DecisionTrees_DonorsChoose/blob/master/Copy_of_8_Don
fpi = []
for i in range(len(y_test)) :
    if (y_test.values[i] == 0) & (predictions[i] == 1) :
        fpi.append(i)
fp_essay1 = []
for i in fpi :
    fp_essay1.append(X_test['essay'].values[i])
```

In [95]:
```
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', stopw

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



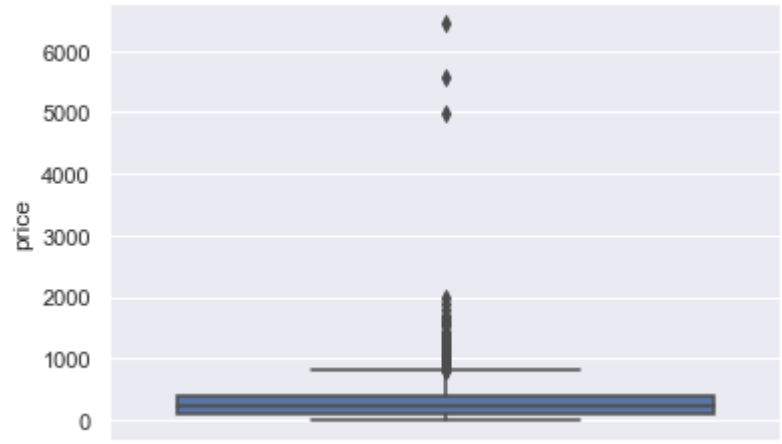

## Box plot with the `price`

In [96]:
```
cols = X_test.columns
X_test_falsePos1 = pd.DataFrame(columns=cols)


for i in fpi : # (in fpi all the false positives data points indexes)
    X_test_falsePos1 = X_test_falsePos1.append(X_test.filter(items=[i], axis=0))

X_test_falsePos1.head(1)
len(X_test_falsePos1)
```

Out[96]: 974

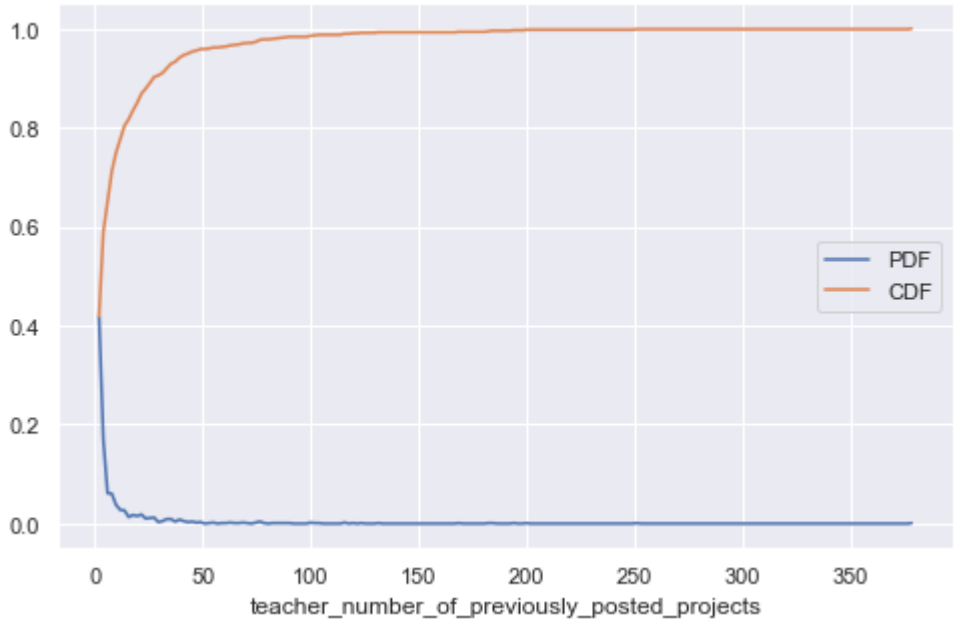
In [97]:
```
sns.boxplot(y='price', data=X_test_falsePos1)
```

Out[97]: <matplotlib.axes._subplots.AxesSubplot at 0x254efa6def0>




## Pdf with the `teacher_number_of_previously_posted_projects`

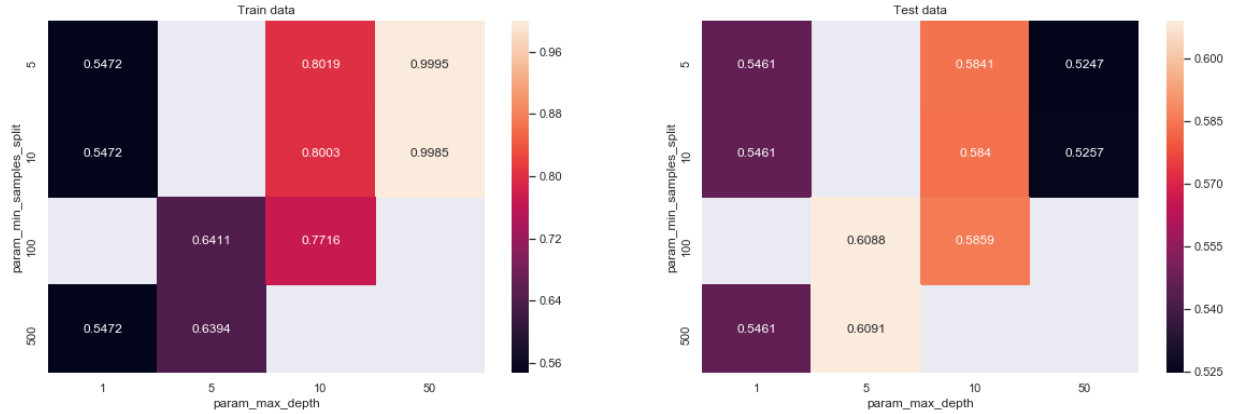
In [98]:
```
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_test_falsePos1['teacher_number_of_previously_
bins='auto', density=True)
pdf = counts/sum(counts)
cdf = np.cumsum(pdf)
pdfP, = plt.plot(bin_edges[1:], pdf)
cdfP, = plt.plot(bin_edges[1:], cdf)
plt.legend([pdfP, cdfP], ["PDF", "CDF"])
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.show()
```




**TFIDF weighted W2V**

In [144]:
```
dt2 = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10,100,500]}
clf = RandomizedSearchCV(dt2, parameters, cv=3, scoring='roc_auc',return_train_s
tfifc_DT = clf.fit(X_train_tfidf_avg, y_train)
```

In [145]:
```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_min_samples_split',

fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train data')
ax[1].set_title('Test data')
plt.show()
```

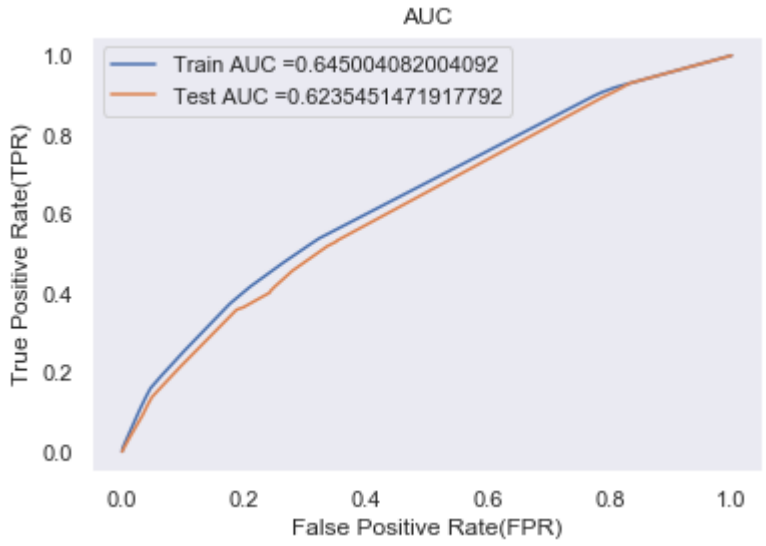



In [146]:
```
print(clf.best_estimator_)
```

```
DecisionTreeClassifier(class_weight='balanced', 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=False, random_state=None,
            splitter='best')
```

In [147]:
```
def pred_prob(clf, data):
    y_pred = []
    y_pred = clf.predict_proba(data)[:,1]
    return y_pred
```

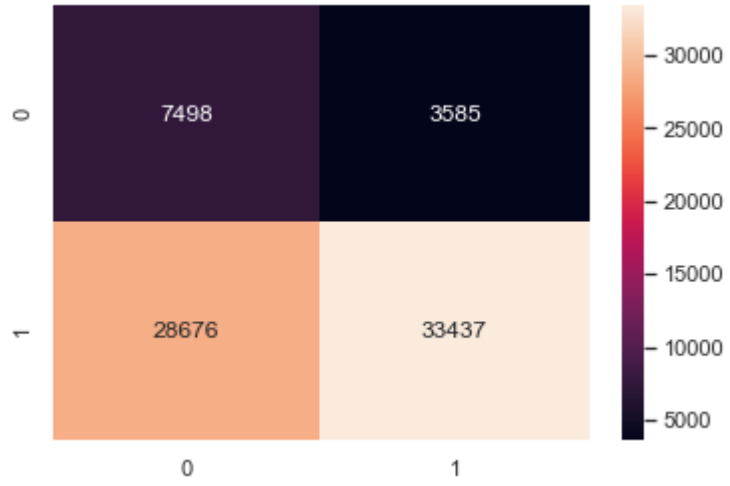
In [148]:
```
from sklearn.metrics import roc_curve, auc

# bow = MultinomialNB(alpha = clf.best_params_['alpha'],class_prior = [0.5,0.5])
tfidf= clf.best_estimator_
# tfidf=DecisionTreeClassifier (class_weight = 'balanced',max_depth=10,min_sample
tfidf.fit(X_train_tfidf, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
# not the predicted outputs

y_train_pred = pred_prob(tfidf,X_train_tfidf)
y_test_pred = pred_prob(tfidf,X_test_tfidf)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

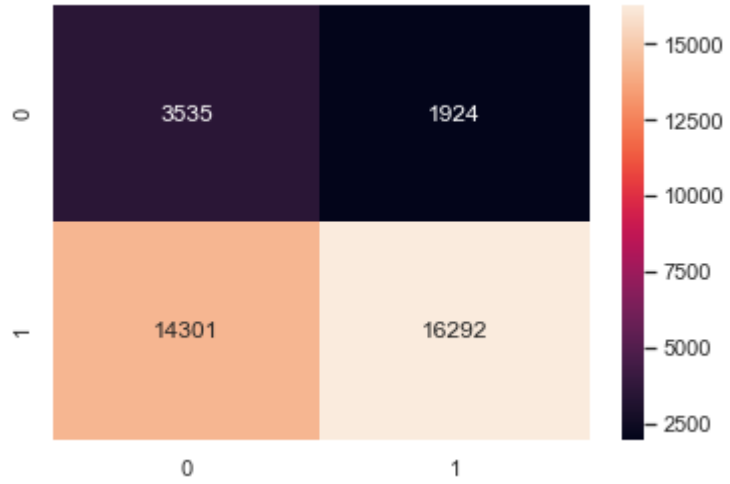


In [149]:
```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print("tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [150]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
tpr*(1-fpr) 0.36419409697651245 for threshold 0.491
```



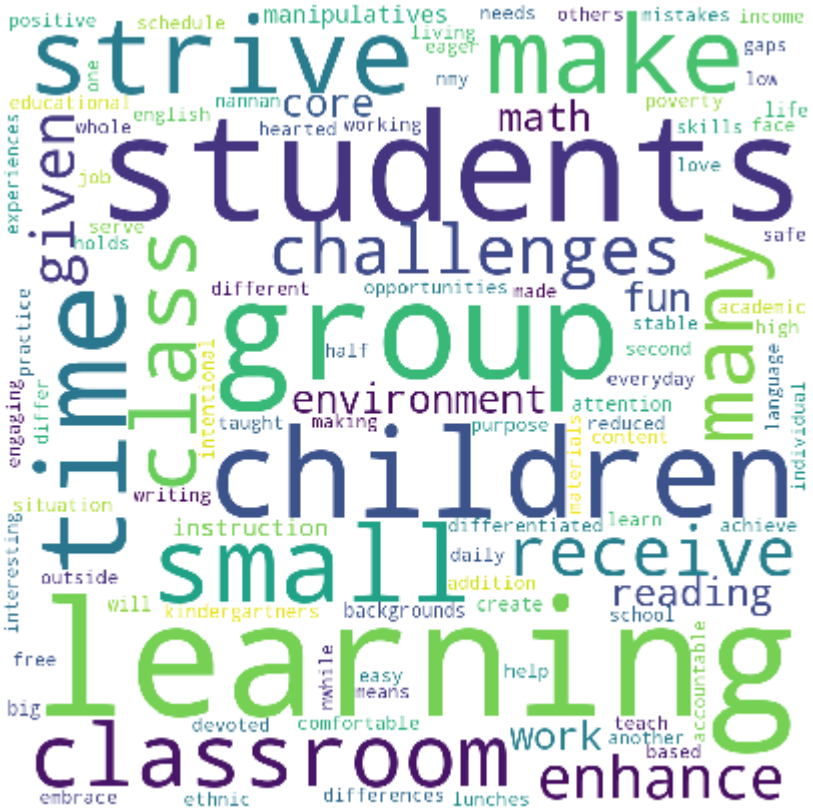




**False positive data point**

In [151]:
```
predictions=predict_with_best_t(y_train_pred, best_t)
```

In [152]:
```
#https://www.google.com/search?q=geeks+for+geeks+false+positive&rlz=1C1SQJL_enIN&
#https://github.com/pskadasi/DecisionTrees_DonorsChoose/blob/master/Copy_of_8_Don
fpi = []
for i in range(len(y_test)) :
    if (y_test.values[i] == 0) & (predictions[i] == 1) :
        fpi.append(i)
fp_essay1 = []
for i in fpi :
    fp_essay1.append(X_test['essay'].values[i])
```

In [154]:
```
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
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', stopw

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



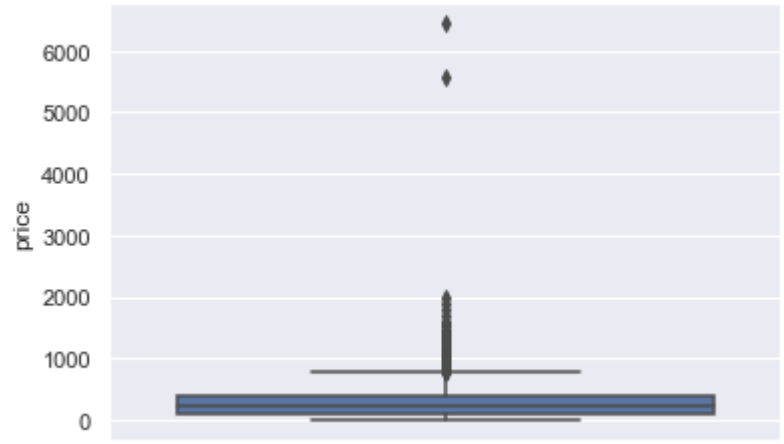

## Box plot with the price

In [155]:
```
cols = X_test.columns
X_test_falsePos1 = pd.DataFrame(columns=cols)


for i in fpi : # (in fpi all the false positives data points indexes)
    X_test_falsePos1 = X_test_falsePos1.append(X_test.filter(items=[i], axis=0))

X_test_falsePos1.head(1)
len(X_test_falsePos1)
```

Out[155]: 894

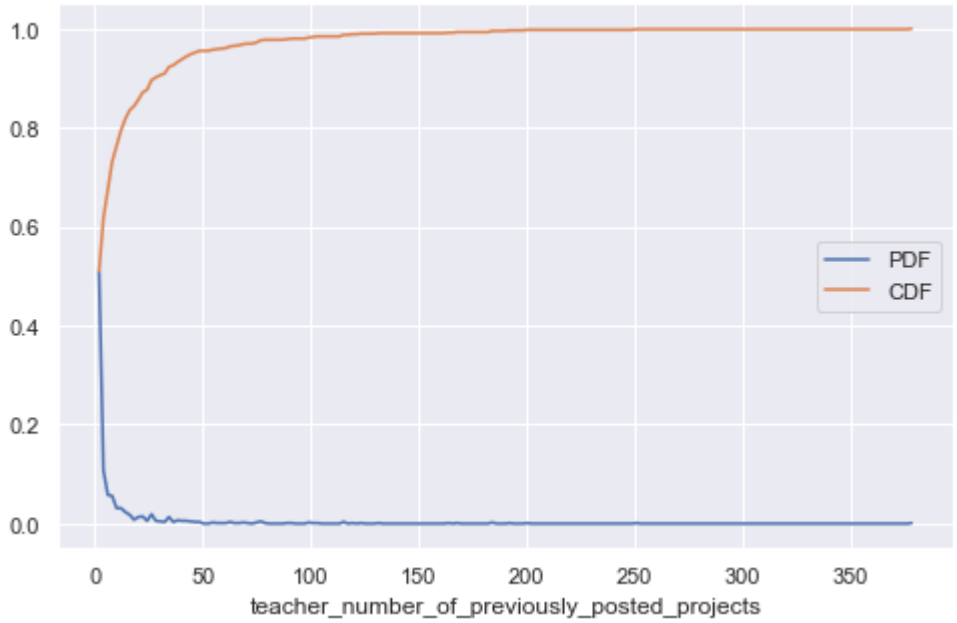
In [156]:
```
sns.boxplot(y='price', data=X_test_falsePos1)
```

Out[156]: <matplotlib.axes._subplots.AxesSubplot at 0x254ef79f978>




**Pdf with the teacher_number_of_previously_posted_projects**

In [157]:
```
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_test_falsePos1['teacher_number_of_previously_
bins='auto', density=True)
pdf = counts/sum(counts)
cdf = np.cumsum(pdf)
pdfP, = plt.plot(bin_edges[1:], pdf)
cdfP, = plt.plot(bin_edges[1:], cdf)
plt.legend([pdfP, cdfP], ["PDF", "CDF"])
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.show()
```




## feature_importances

In [99]:
```
#https://stackoverflow.com/questions/47111434/randomforestregressor-and-feature-
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
def selectKImportance(model, X):
    return X[:,model.best_estimator_.feature_importances_.argsort()[::-1]]
```

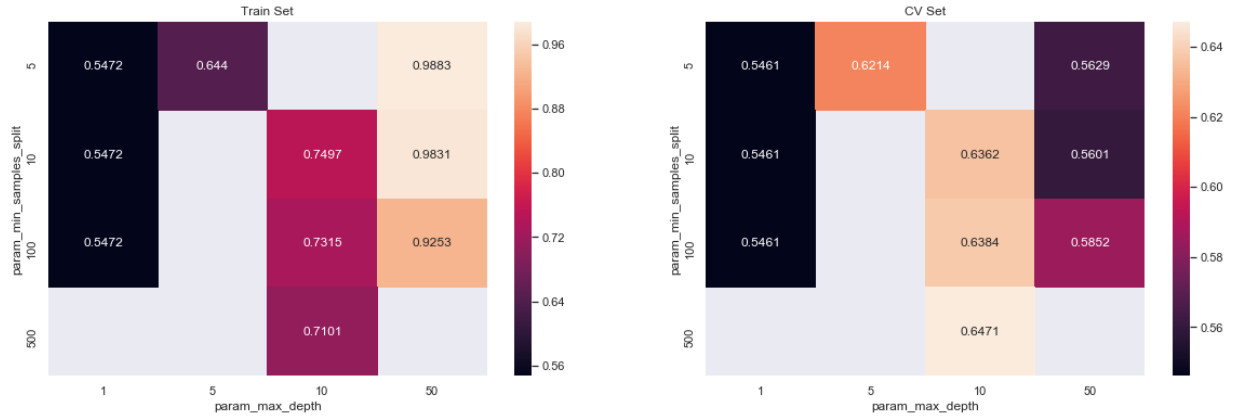
In [116]:
```
X__train = selectKImportance(clff, X_train_tfidf)
X__test = selectKImportance(clff, X_test_tfidf)
```

In [109]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense
X__train = hstack((X_train_essay_tfidf,X_train_featured_title_tfidf,
                   X_train_teacher_prefix,X_train_clean_categories,X_train_c
                   X_train_grade_level,X_train_skl_state,
                   X_train_price_standardized,X_train_prev_proj))
print(X__train.shape, y_train.shape)
```

```
(73196, 14442) (73196,)
```

In [112]:
```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
dt5= DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10,100,500]}
clf_f=RandomizedSearchCV(dt2, parameters, cv=3, scoring='roc_auc',return_train_s
set= clf_f.fit(X__train, y_train)
```

In [111]:
```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf_f.cv_results_).groupby(['param_min_samples_split'
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```

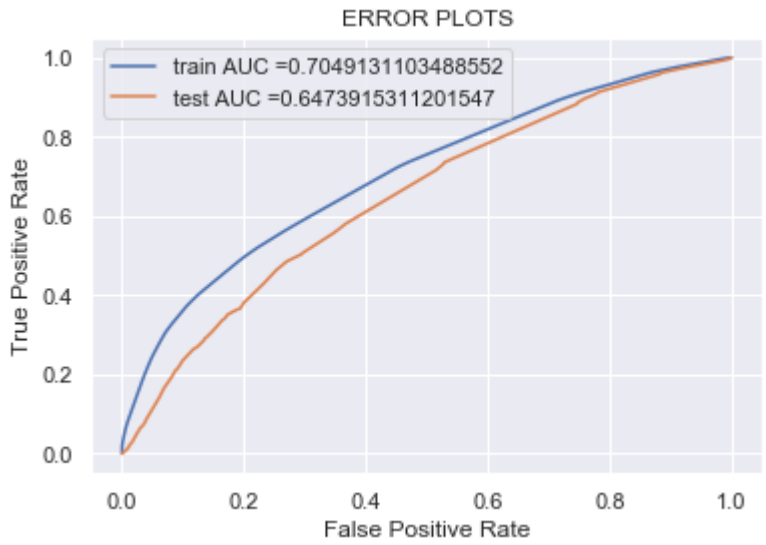



In [117]:
```
#Best Estimator and Best tune parameters
print(clf_f.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf_f.score(X__train,y_train))
print(clf_f.score(X__test,y_test))
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=10, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
0.49930007824727896
0.500616266891556
```

In [121]:
```
best_tune_parameters=[{'max_depth': [10], 'min_samples_split':[500] } ]
```

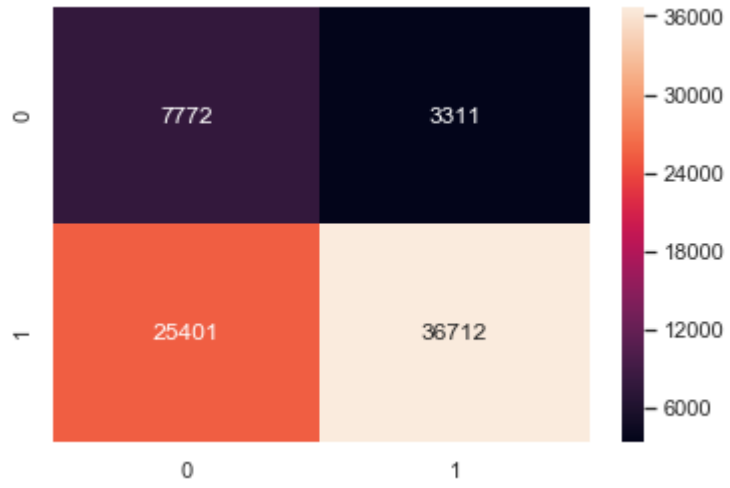
In [124]:
```
n sklearn.metrics import roc_curve, auc
n sklearn.tree import DecisionTreeClassifier
n sklearn.metrics import roc_curve, auc
l1= GridSearchCV( DecisionTreeClassifier(class_weight = 'balanced'),best_tune_para
/1=DecisionTreeClassifier (class_weight = 'balanced',max_depth=10,min_samples_spli
l1.fit(X__train, y_train)
or visulation
/1.fit(X__train, y_train)
tps://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.
rain_pred1 = clf11.predict_proba(X__train) [:,1]
est_pred1 = clf11.predict_proba(X__test) [:,1]
in_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1))
plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
legend()
xlabel("False Positive Rate")
ylabel("True Positive Rate")
title("ERROR PLOTS")
grid(True)
show()
```

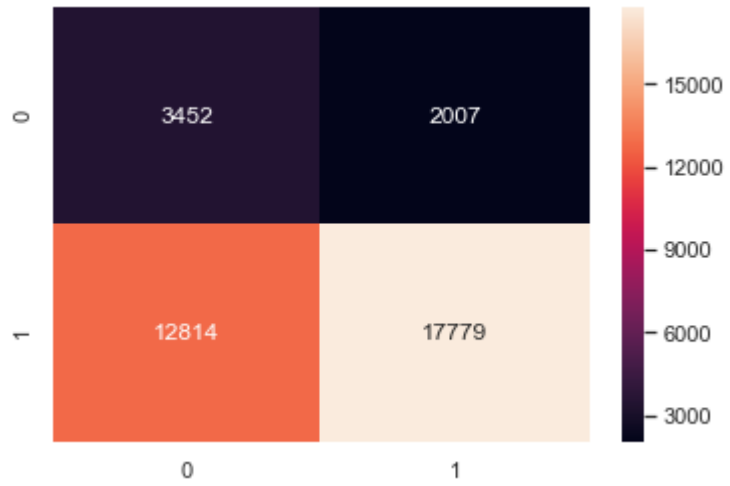


In [125]:
```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print("tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [128]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds1, train_fpr1, train_tpr1)
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred1, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred1, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
tpr*(1-fpr) 0.4144775361244713 for threshold 0.471
```

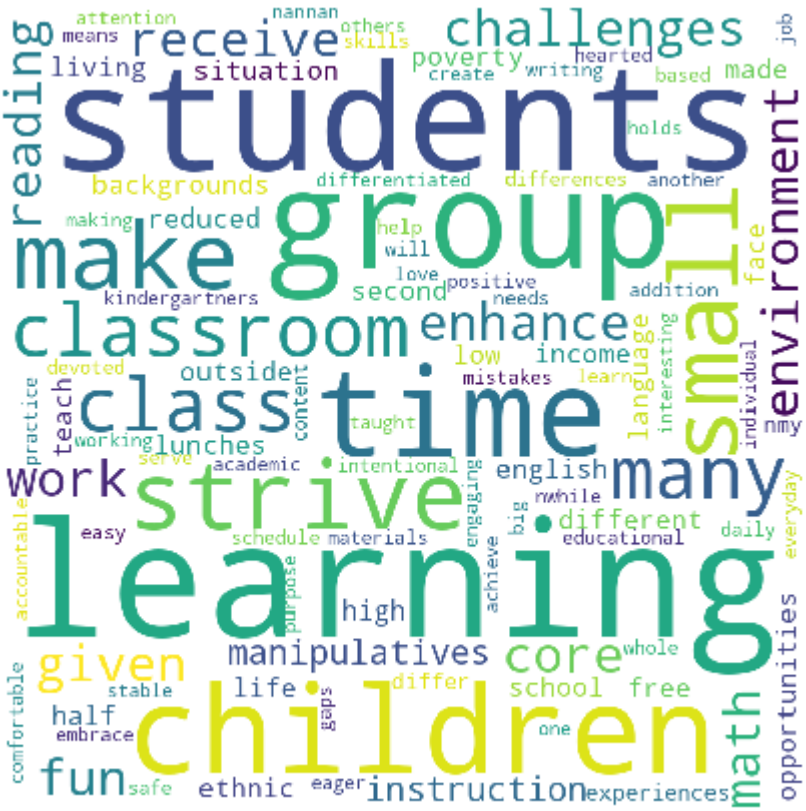



In [130]:
```
predictions=predict_with_best_t(y_train_pred1, best_t)
```

In [132]:
```
#false positive analysis

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

In [136]:
```
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '

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', stopw
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()
```



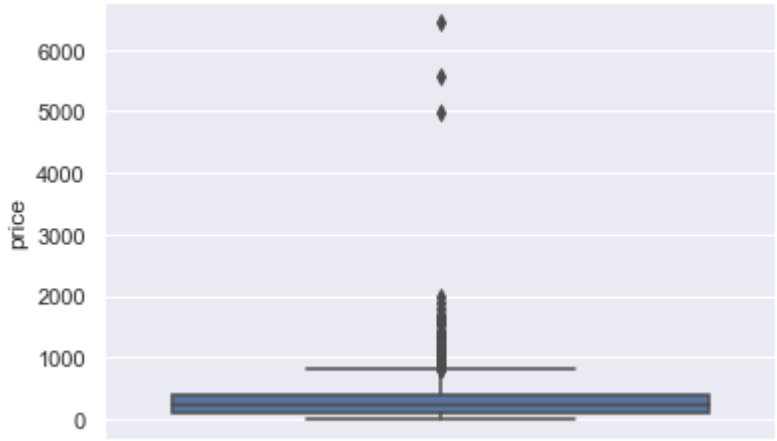

**Box plot**

In [137]:
```
cols = X_test.columns
X_test_falsePos1 = pd.DataFrame(columns=cols)


for i in fpi : # (in fpi all the false positives data points indexes)
    X_test_falsePos1 = X_test_falsePos1.append(X_test.filter(items=[i], axis=0))

X_test_falsePos1.head(1)
len(X_test_falsePos1)
```

Out[137]: 969

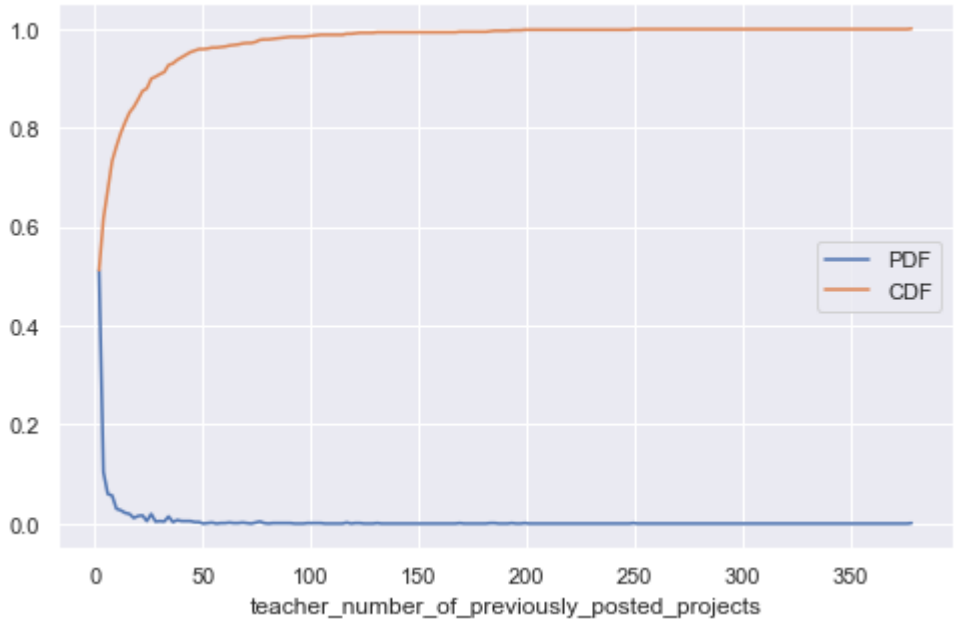
In [138]:
```
sns.boxplot(y='price', data=X_test_falsePos1)
```

Out[138]: <matplotlib.axes._subplots.AxesSubplot at 0x254ec5f4358>




**Pdf with the teacher_number_of_previously_posted_projects**

In [139]:
```
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_test_falsePos1['teacher_number_of_previously_
bins='auto', density=True)
pdf = counts/sum(counts)
cdf = np.cumsum(pdf)
pdfP, = plt.plot(bin_edges[1:], pdf)
cdfP, = plt.plot(bin_edges[1:], cdf)
plt.legend([pdfP, cdfP], ["PDF", "CDF"])
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.show()
```




## Observation

In [158]:
```
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= (" Vectorizer ", " Max_depth ", " Min_sample_split "," Test -AUC
tb.add_row([" Tf - Idf", 10 , 500 ,63.9 ])
tb.add_row(["A VG - Tf - Idf", 5 , 500 ,62.2])
tb.add_row(["Non-Zero Features", 10, 500 ,64.7 ])
print(tb.get_string(titles = "Decision trees- Observations"))
```

```
+-------------------+-------------+--------------------+-------------+
|     Vectorizer    |  Max_depth  |  Min_sample_split  |  Test -AUC  |
+-------------------+-------------+--------------------+-------------+
|      Tf - Idf     |      10     |        500         |     63.9    |
|  A VG - Tf - Idf  |      5      |        500         |     62.2    |
| Non-Zero Features |      10     |        500         |     64.7    |
+-------------------+-------------+--------------------+-------------+
```

In [ ]:
```
```