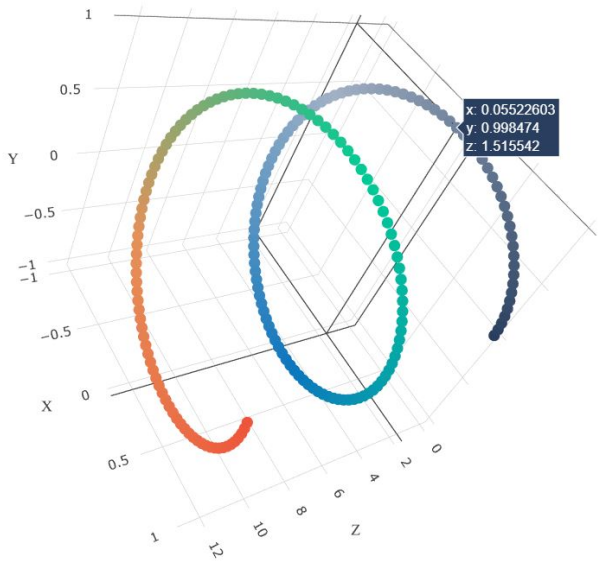
Assignment 8: DT

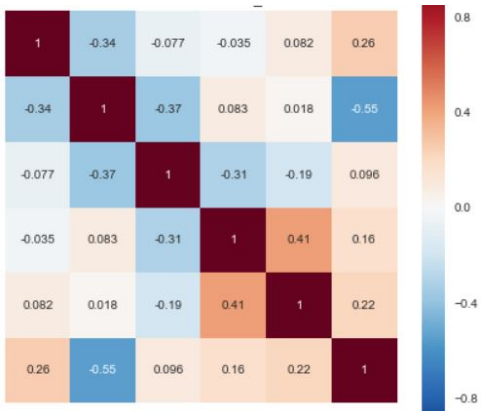
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 2: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper 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 value
    - find the best hyper paramter using k-fold cross validation(use gridsearch cv or randomsearch cv)/simple cross validation data(you can write your own for loops refer sample solution)
3. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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

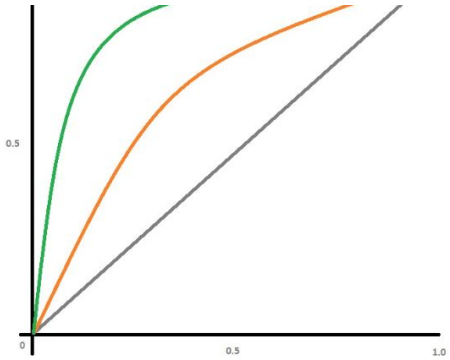
or

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



seaborn heat maps with rows as **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 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

In [96]:

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

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students w
ith the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelli
gences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of differen
t backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a carin
g community of successful \
learners which can be seen through collaborative student project based learning in and out of the
classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice
a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the ki
ndergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role pla
y in our pretend kitchen\
```

```
in the early childhood classroom i have had several kids ask me can we try cooking with real food
i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while co
oking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into maki
ng the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project woul
d expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple
sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cook
books to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoymen
t for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

In [97]:

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

1. Decision Tree

1.1 Loading Data

In [98]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows = 30000)
data.shape
```

Out[98]:

```
(30000, 9)
```

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

In [99]:

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

In [100]:

```
from sklearn.model_selection import train_test_split

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

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

print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(13467, 8) (13467,)
(6633, 8) (6633,)
(9900, 8) (9900,)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [101]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [145]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer_bow_essay = CountVectorizer(min_df=10)
vectorizer_bow_essay.fit(X_train['essay'].values)
```

Out[145]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
```

```
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=None, min_df=10,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [146]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer_bow_essay.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_bow_essay.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_bow_essay.transform(X_test['essay'].values)
bow_essay_feature_names = vectorizer_bow_essay.get_feature_names()
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [104]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [147]:

```
# Encoding School State - OHE
# School State
vectorizer_state = CountVectorizer()
vectorizer_state.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer_state.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer_state.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer_state.transform(X_test['school_state'].values)
school_state_feature_names = vectorizer_state.get_feature_names()
```

In [148]:

```
# Encoding Teacher Prefix OHE
# teacher_prefix
vectorizer_prefix = CountVectorizer()
vectorizer_prefix.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer_prefix.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer_prefix.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer_prefix.transform(X_test['teacher_prefix'].values)
teacher_prefix_feature_names = vectorizer_prefix.get_feature_names()
```

In [149]:

```
# Encoding project_grade_category
vectorizer_grade = CountVectorizer()
vectorizer_grade.fit(X_train['project_grade_category'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer_grade.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer_grade.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer_grade.transform(X_test['project_grade_category'].values)
grade_feature_names = vectorizer_grade.get_feature_names()
```

In [150]:

```
# clean_categories
vectorizer_category = CountVectorizer()
vectorizer_category.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_category_ohe = vectorizer_category.transform(X_train['clean_categories'].values)
X_cv_category_ohe = vectorizer_category.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vectorizer_category.transform(X_test['clean_categories'].values)
category_feature_names = vectorizer_category.get_feature_names()
```

In [151]:

```
# Encoding sub categories
vectorizer_subcategory = CountVectorizer()
vectorizer_subcategory.fit(X_train['clean_subcategories'].values) # fit has to happen only on
train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategory_ohe = vectorizer_subcategory.transform(X_train['clean_subcategories'].values)
X_cv_subcategory_ohe = vectorizer_subcategory.transform(X_cv['clean_subcategories'].values)
X_test_subcategory_ohe = vectorizer_subcategory.transform(X_test['clean_subcategories'].values)
subcategory_feature_names = vectorizer_subcategory.get_feature_names()
```

In [155]:

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

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

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))
```

In [156]:

```
X_train_price_norm = X_train_price_norm.reshape(-1,1)
X_cv_price_norm = X_cv_price_norm.reshape(-1,1)
X_test_price_norm = X_test_price_norm.reshape(-1,1)
```

In [157]:

```
# teacher previously posted projects
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_teach_prev_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_cv_teach_prev_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].v
alues.reshape(1,-1))
X_test_teach_prev_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
```

In [158]:

```
# reshaping the ndarrays post normalization
X_train_teach_prev_norm = X_train_teach_prev_norm.reshape(-1,1)
X_cv_teach_prev_norm = X_cv_teach_prev_norm.reshape(-1,1)
X_test_teach_prev_norm = X_test_teach_prev_norm.reshape(-1,1)
```

1.5 Appling Decision Tree on different kind of featurization as mentioned in the instructions

Apply Decision Tree on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

In [159]:

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

In [160]:

```
# Set1
```

In [161]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train["essay"].values)
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)
print("Shape of Datamatrix after TFIDF Vectorization")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
tfidf_essay_feature_names = vectorizer.get_feature_names()
```

```
Shape of Datamatrix after TFIDF Vectorization
(13467, 5000) (13467,)
(6633, 5000) (6633,)
(9900, 5000) (9900,)
====================================================================================================
```

In [162]:

```
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_price_norm, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_teach_prev_norm)).tocsr()

X_cr = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe, X_cv_category_ohe, X_cv_subcategory_ohe,
               X_cv_price_norm, X_cv_teach_prev_norm)).tocsr()

X_te = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,
               X_test_price_norm, X_test_teach_prev_norm)).tocsr()

tfidf_feature_names_list = []
tfidf_feature_names_list.extend(tfidf_essay_feature_names)
tfidf_feature_names_list.extend(school_state_feature_names)
tfidf_feature_names_list.extend(teacher_prefix_feature_names)
tfidf_feature_names_list.extend(grade_feature_names)
tfidf_feature_names_list.extend("Price")
tfidf_feature_names_list.extend(category_feature_names)
tfidf_feature_names_list.extend(subcategory_feature_names)
tfidf_feature_names_list.extend("Teacher Previously submitted projects")
print (len(tfidf_feature_names_list))
```

```
5137
```

In [163]:

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

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

dt_output = DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(dt_output, tree_parameters, cv=5, scoring='roc_auc', return_train_score=True, n_
jobs=-1)
clf.fit(X_tr,y_train)
```

Out[163]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort=False, random_state=None,
                                              splitter='best'),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [164]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('Best Hyper parameters: ',clf.best_params_)
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.62720332426386
Best Hyper parameters:  {'max_depth': 10, 'min_samples_split': 500}
===========================================================================
Train AUC scores
[0.56108476 0.56108476 0.56108476 0.56108476 0.67758099 0.67740645
 0.67270047 0.66585923 0.81388134 0.81043809 0.77288838 0.72628825
 0.98818525 0.98211011 0.92544917 0.82650262]
CV AUC scores
[0.55590933 0.55590933 0.55590933 0.55590933 0.61787881 0.61818019
 0.61935497 0.62124441 0.61190121 0.60723442 0.60993641 0.62720332
 0.55794457 0.55841942 0.57411404 0.5960995 ]
```

In [165]:

```
from itertools import repeat
x1 = []
y1 = []
max_depth = [1, 5, 10, 50, 100, 500, 100]
min_samples_split = [5, 10, 100, 500]
train_auc_scores = clf.cv_results_['mean_train_score']
cv_auc_scores = clf.cv_results_['mean_test_score']

x1 = [x for item in max_depth for x in repeat(item, 4)]
y1 = [y for item in min_samples_split for y in repeat(item, 7)]
```

In [166]:

```
trace1 = go.Scatter3d(x=x1,y=y1,z=train_auc_scores, name="train auc")
trace2 = go.Scatter3d(x=x1,y=y1,z=cv_auc_scores, name="cv auc")
```

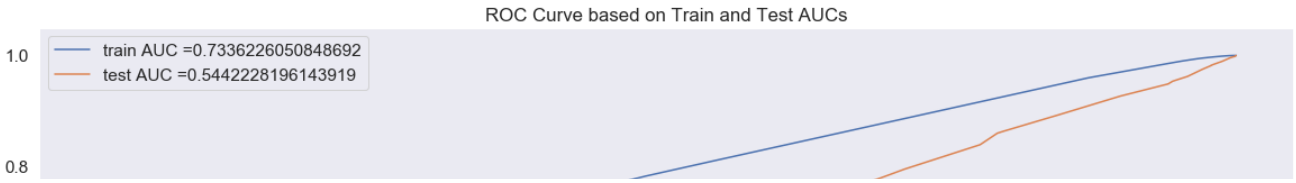
```
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='min_samples_split'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

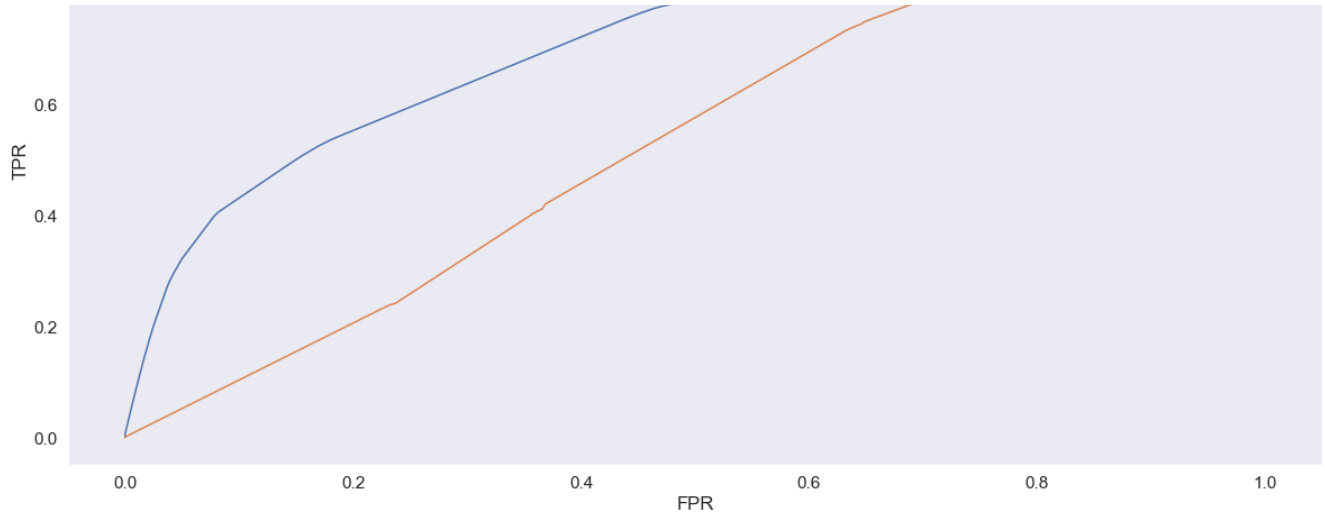
In [167]:

```
dt_output = DecisionTreeClassifier(class_weight='balanced', max_depth =
clf.best_params_["max_depth"],
                                   min_samples_split = clf.best_params_["min_samples_split"])
dt_output.fit(X_tr, y_train)

y_train_pred = dt_output.predict_proba(X_tr)[:,1]  # returning probability estimates of positive c
lass
y_test_pred = dt_output.predict_proba(X_te)[:,1]

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)
```

In [168]:

```
plt.figure(figsize=(20,10))
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("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve based on Train and Test AUCs")
plt.grid()
plt.show()
```





In [169]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [170]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
Train_CM = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
Test_CM = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print("Train confusion matrix")
print(Train_CM)
print("Test confusion matrix")
print(Test_CM)
```

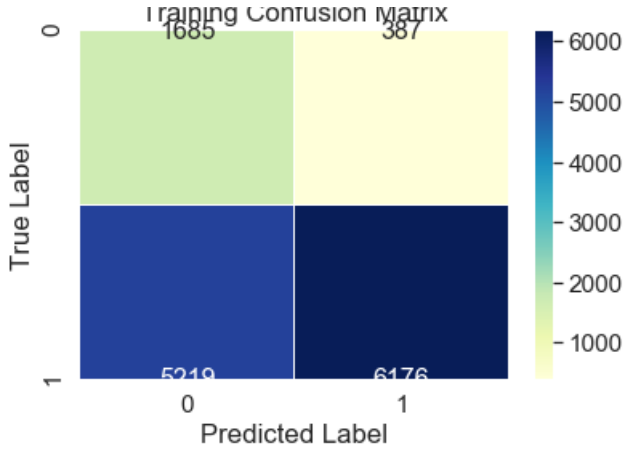
```
the maximum value of tpr*(1-fpr) 0.44076095151128064 for threshold 0.472
Train confusion matrix
[[1685  387]
 [5219 6176]]
Test confusion matrix
[[ 534  993]
 [2088 6285]]
```

In [171]:

```
sns.set(font_scale=1.4)
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="g", annot_kws = {"size":16},linewidths=.5,cmap="YlGn
Bu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[171]:

```
Text(0.5, 1, 'Training Confusion Matrix')
```

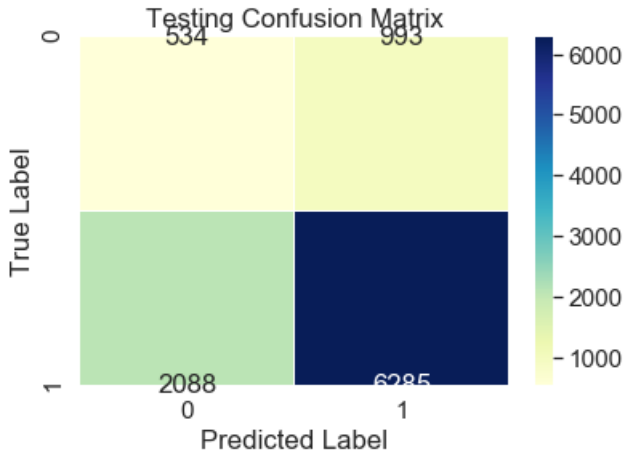


In [172]:

```
sns.heatmap(Test_CM,annot=True,cbar=True,fmt="d", linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Testing Confusion Matrix')
```

Out[172]:

```
Text(0.5, 1, 'Testing Confusion Matrix')
```



In [181]:

```
# Please write all the code with proper documentation
import graphviz
from sklearn import tree
from graphviz import Source
from sklearn.tree import export_graphviz
import pydot
dtree = DecisionTreeClassifier(max_depth=3)
clf = dtree.fit(X_tr,y_train)
_dot_data = tree.export_graphviz(dtree, feature_names=None)
graph = Source(_dot_data)
graph.render("TFIDF Tree", view=True)
```

Out[181]:

```
'TFIDF Tree.pdf'
```

In [182]:

```
false_pos_indices = []
for i in range(len(y_test)):
    if(y_test[i]==0 and _predictions[i] == 1):
        false_pos_indices.append(i)
false_pos_essay = []
for i in false_pos_indices :
```

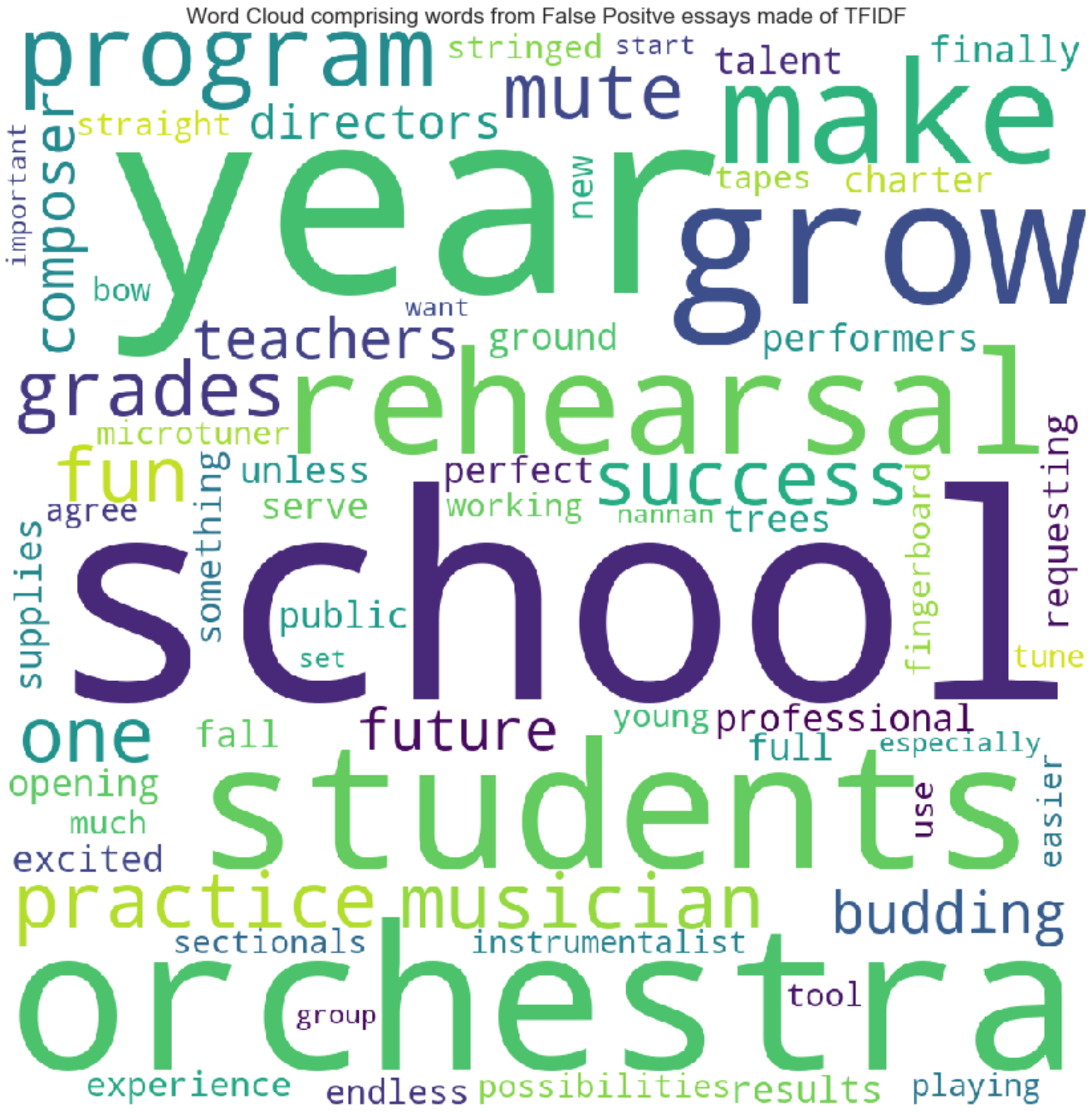
```
    false_pos_essay.append(X_test['essay'].values[i])
```

In [185]:

```
from wordcloud import WordCloud, STOPWORDS
comments = " "
stopwords = set(STOPWORDS)
for _essay in false_pos_essay:
    tokens = str(_essay).lower().split()

for words in tokens:
    comments += words + " "

wordcloud = WordCloud(width=1000, height=1000, background_color="white",stopwords=stopwords,
min_font_size=12).generate(comments)
```

In [186]:

```
plt.figure(figsize=(15,15))
plt.imshow(wordcloud)
plt.axis("off")
plt.title("Word Cloud comprising words from False Positve essays made of TFIDF")
plt.show()
```



In [187]:

```
# Plot the box plot with the `price` of these `false positive data points`
cols = X_test.columns
X_test_falsePos = pd.DataFrame(columns=cols)
for i in false_pos_indices:
    X_test_falsePos = X_test_falsePos.append(X_test.filter(items=[i],axis=0))

X_test_falsePos.head(1)
```

Out[187]:

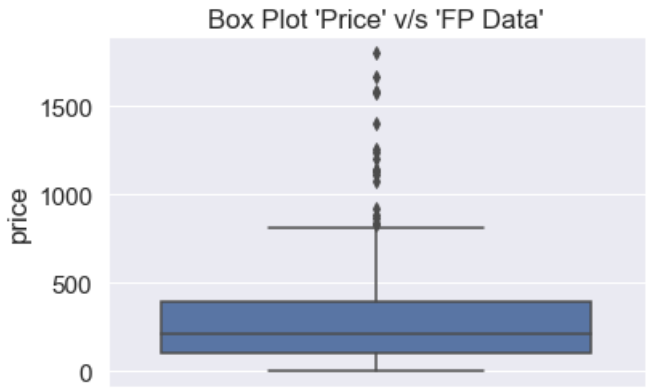
	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcate
1	ut	ms	grades_3_5	4	specialneeds	specia

In [188]:

```
sns.boxplot(y="price",data=X_test_falsePos).set_title("Box Plot 'Price' v/s 'FP Data'")
```

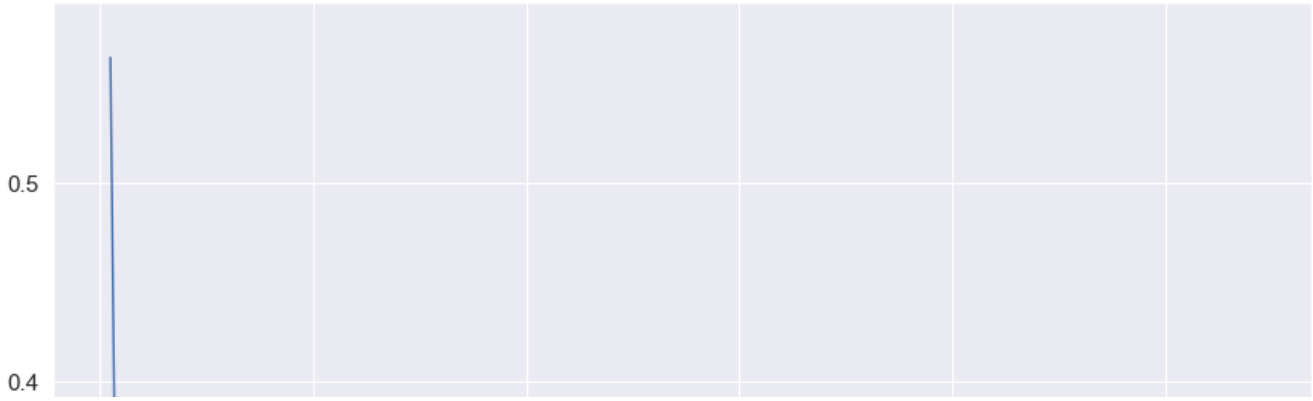
Out[188]:

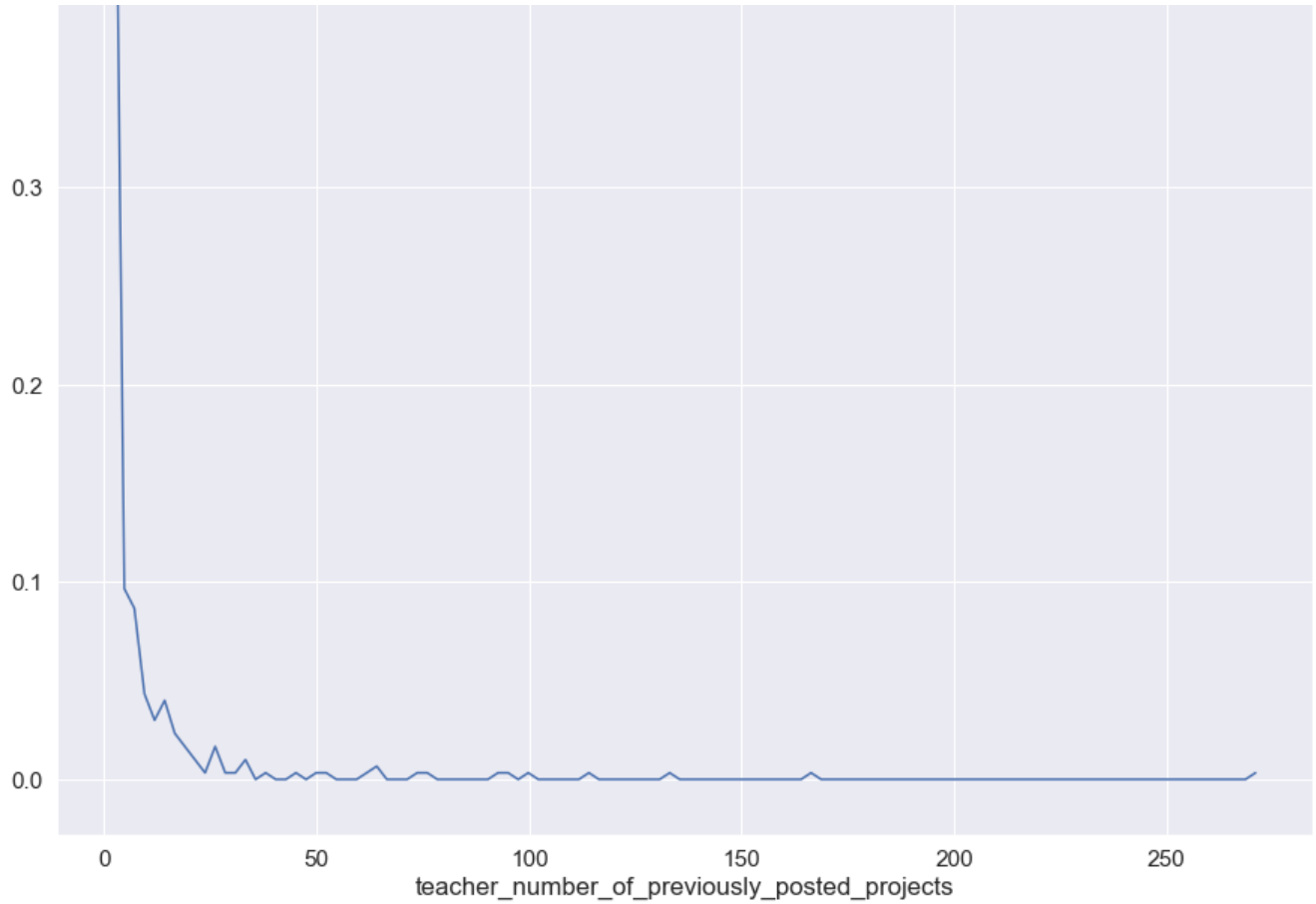
```
Text(0.5, 1.0, "Box Plot 'Price' v/s 'FP Data'")
```



In [189]:

```
# Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive da
ta points`
plt.figure(figsize=(15,15))
counts, bin_edges = np.histogram(X_test_falsePos["teacher_number_of_previously_posted_projects"],
bins="auto",density=True)
pdf = counts/sum(counts)
pdfPoints = plt.plot(bin_edges[1:],pdf)
plt.xlabel("teacher_number_of_previously_posted_projects")
plt.show()
```





In [190]:

```
# set 2
```

In [193]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [195]:

```
# preprocessing project_title and essay with TFIDF W2V Vectorization
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'])
# 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 [196]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
```

```
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 13467/13467 [00:
51<00:00, 262.70it/s]
```

```
13467
300
```

In [197]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))
print(len(tfidf_w2v_vectors_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 6633/6633
[00:24<00:00, 266.41it/s]
```

```
6633
300
```

In [198]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 9900/9900
[00:38<00:00, 259.54it/s]
```

```
9900
```

```
300
```

In [201]:

```
# Concatinating all the features
X_tr = hstack((tfidf_w2v_vectors_train, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_price_norm,
               X_train_teach_prev_norm)).tocsr()

X_cr = hstack((tfidf_w2v_vectors_cv, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe,X_cv_category_ohe,
               X_cv_subcategory_ohe, X_cv_price_norm,
              X_cv_teach_prev_norm)).tocsr()

X_te = hstack((tfidf_w2v_vectors_test, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe,
               X_test_subcategory_ohe, X_test_price_norm,
              X_test_teach_prev_norm)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(13467, 397) (13467,)
(6633, 397) (6633,)
(9900, 397) (9900,)
====================================================================================================
```

In [203]:

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

dt_output = DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(dt_output, tree_parameters, cv=5, scoring='roc_auc', return_train_score=True, n_
jobs=-1)
clf.fit(X_tr,y_train)
```

Out[203]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort=False, random_state=None,
                                              splitter='best'),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [204]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('Best Hyper parameters: ',clf.best_params_)
```

```
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.6086469924899166
Best Hyper parameters:  {'max_depth': 5, 'min_samples_split': 500}
===========================================================================
Train AUC scores
[0.56338611 0.56338611 0.56338611 0.56338611 0.71128928 0.71128928
 0.7084595  0.69246554 0.91033904 0.90613876 0.84612065 0.7392427
 0.9999157  0.99907689 0.8967109  0.74147877]
CV AUC scores
[0.55206053 0.55206053 0.55206053 0.55206053 0.60414161 0.60414161
 0.6034418  0.60864699 0.57575988 0.57528763 0.580043   0.6062949
 0.52769072 0.52430235 0.55638145 0.60347152]
```

In [205]:

```
x1 = []
y1 = []
max_depth = [1, 5, 10, 50, 100, 500, 100]
min_samples_split = [5, 10, 100, 500]
train_auc_scores = clf.cv_results_['mean_train_score']
cv_auc_scores = clf.cv_results_['mean_test_score']

x1 = [x for item in max_depth for x in repeat(item, 4)]
y1 = [y for item in min_samples_split for y in repeat(item, 7)]
```

In [206]:

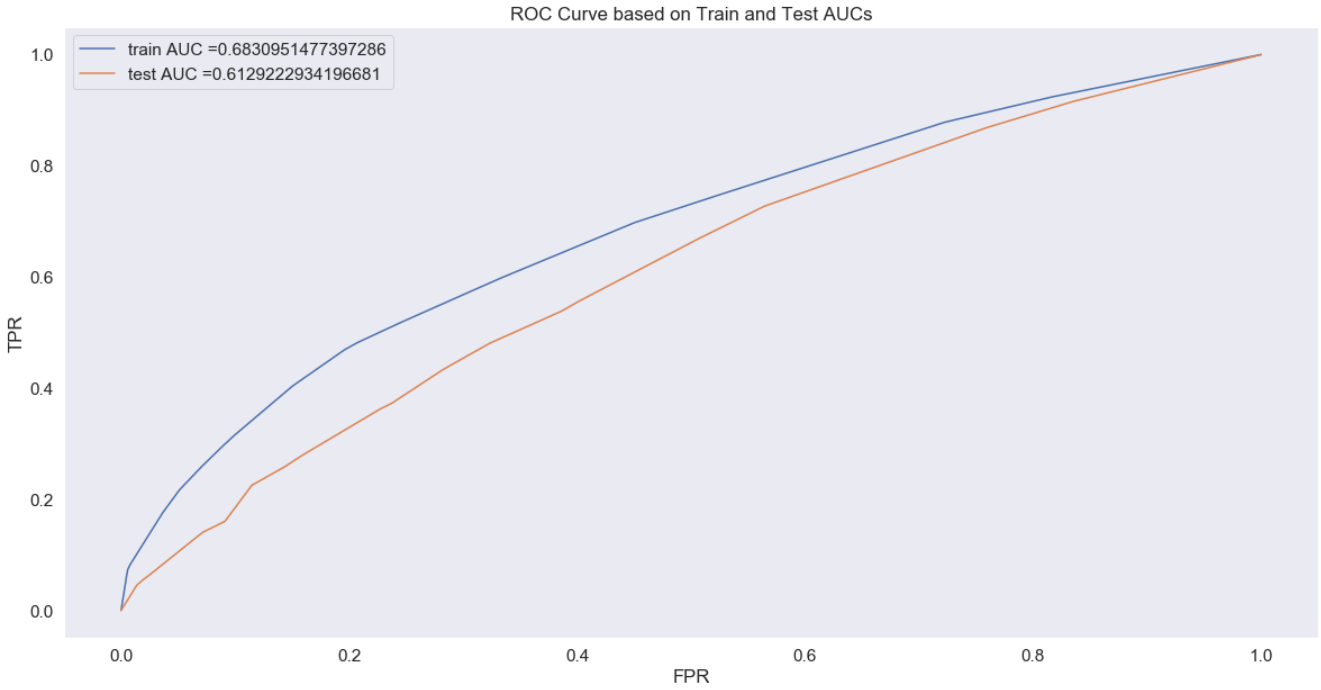
```
trace1 = go.Scatter3d(x=x1,y=y1,z=train_auc_scores, name="train auc")
trace2 = go.Scatter3d(x=x1,y=y1,z=cv_auc_scores, name="cv auc")
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='min_samples_split'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [207]:

```
dt_output = DecisionTreeClassifier(class_weight='balanced', max_depth =
clf.best_params_["max_depth"],
                                   min_samples_split = clf.best_params_["min_samples_split"])
dt_output.fit(X_tr, y_train)

y_train_pred = dt_output.predict_proba(X_tr)[:,1]  # returning probability estimates of positive c
lass
y_test_pred = dt_output.predict_proba(X_te)[:,1]

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)
```

In [208]:

```
plt.figure(figsize=(20,10))
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("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve based on Train and Test AUCs")
plt.grid()
plt.show()
```



In [209]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
Train_CM = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
Test_CM = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print("Train confusion matrix")
print(Train_CM)
print("Test confusion matrix")
print(Test_CM)
```

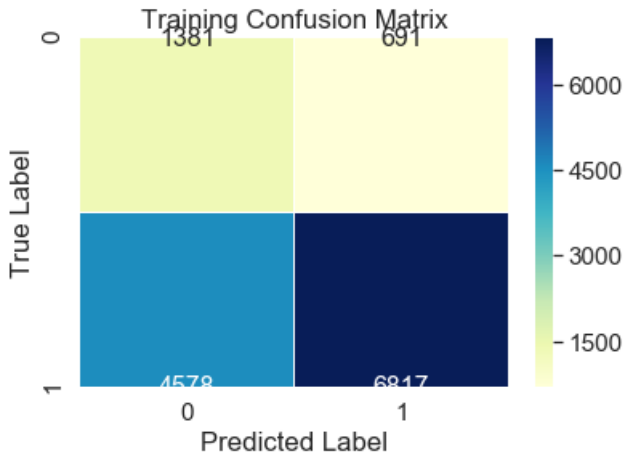
```
the maximum value of tpr*(1-fpr) 0.3987336534177254 for threshold 0.475
Train confusion matrix
[[1381  691]
 [4578 6817]]
Test confusion matrix
[[ 913  614]
 [3711 4662]]
```

In [210]:

```
sns.set(font_scale=1.4)
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="g", annot_kws = {"size":16},linewidths=.5,cmap="YlGn
Bu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[210]:

```
Text(0.5, 1, 'Training Confusion Matrix')
```

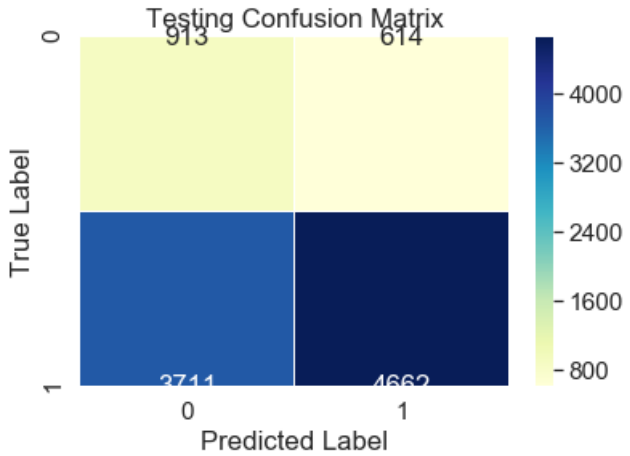


In [211]:

```
sns.heatmap(Test_CM,annot=True,cbar=True,fmt="d", linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Testing Confusion Matrix')
```

Out[211]:

```
Text(0.5, 1, 'Testing Confusion Matrix')
```



In [212]:

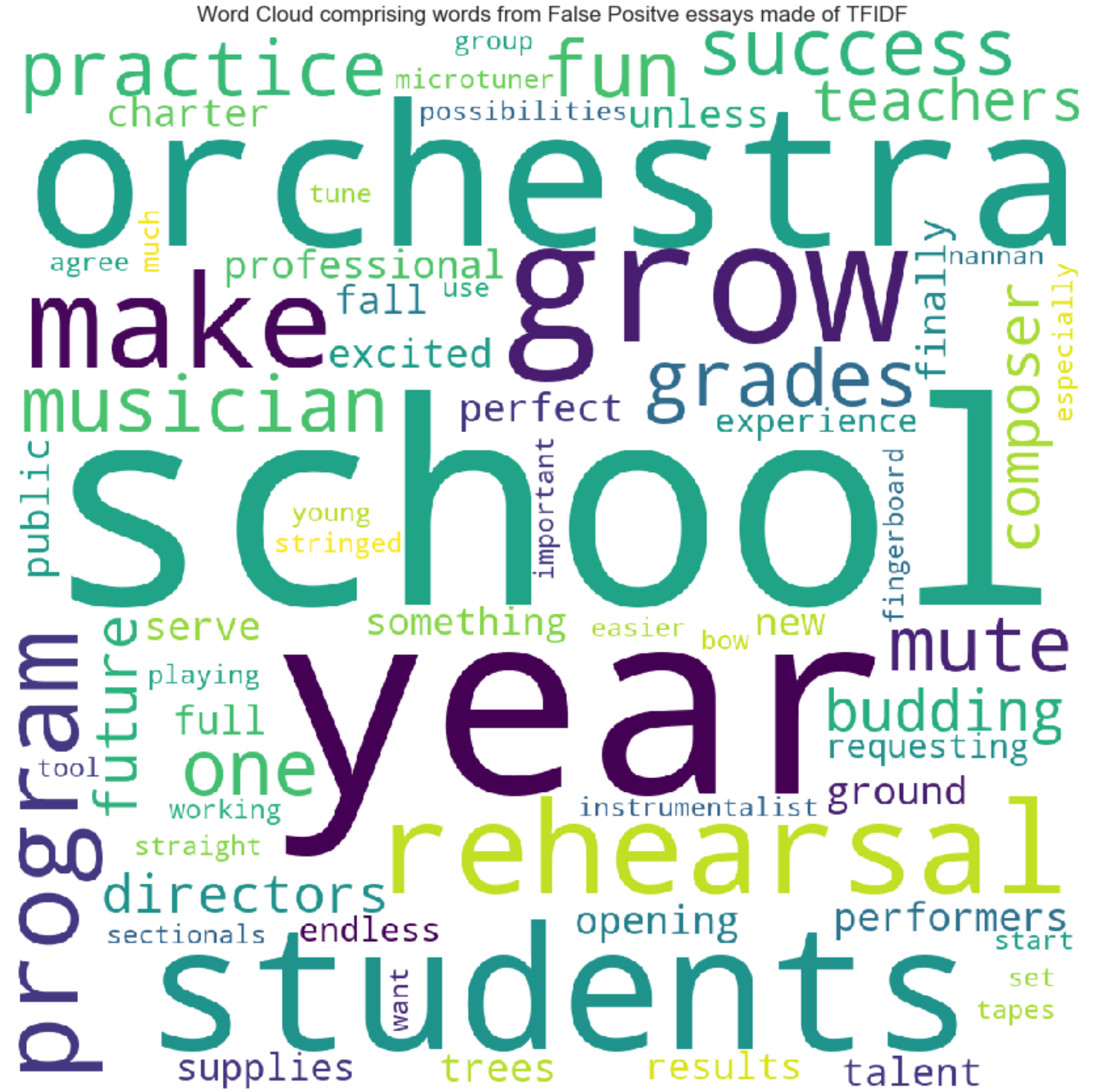
```
false_pos_indices = []
for i in range(len(y_test)):
    if(y_test[i]==0 and _predictions[i] == 1):
        false_pos_indices.append(i)
false_pos_essay = []
for i in false_pos_indices :
  false_pos_essay.append(X_test['essay'].values[i])
```

In [213]:

```
comments = " "
stopwords = set(STOPWORDS)
for _essay in false_pos_essay:
    tokens = str(_essay).lower().split()

for words in tokens:
    comments += words + " "

wordcloud = WordCloud(width=1000, height=1000, background_color="white",stopwords=stopwords,
min_font_size=12).generate(comments)
```

In [214]:

```
plt.figure(figsize=(15,15))
plt.imshow(wordcloud)
plt.axis("off")
plt.title("Word Cloud comprising words from False Positve essays made of TFIDF")
plt.show()
```



In [215]:

```
# Plot the box plot with the `price` of these `false positive data points`
cols = X_test.columns
X_test_falsePos = pd.DataFrame(columns=cols)
```

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

X_test_falsePos.head(1)
```

Out[215]:

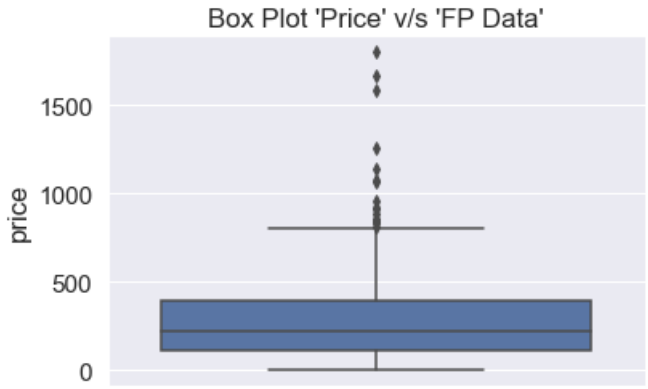
	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcat
19	mo	mrs	grades_9_12	3	literacy_language	literature

In [216]:

```
sns.boxplot(y="price",data=X_test_falsePos).set_title("Box Plot 'Price' v/s 'FP Data'")
```

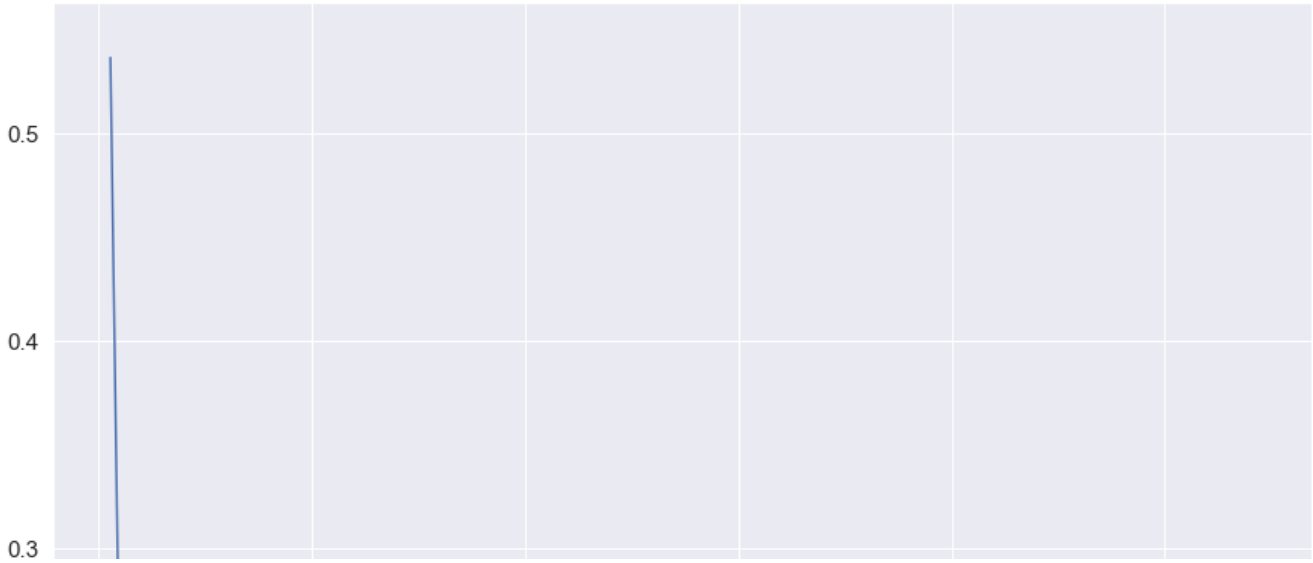
Out[216]:

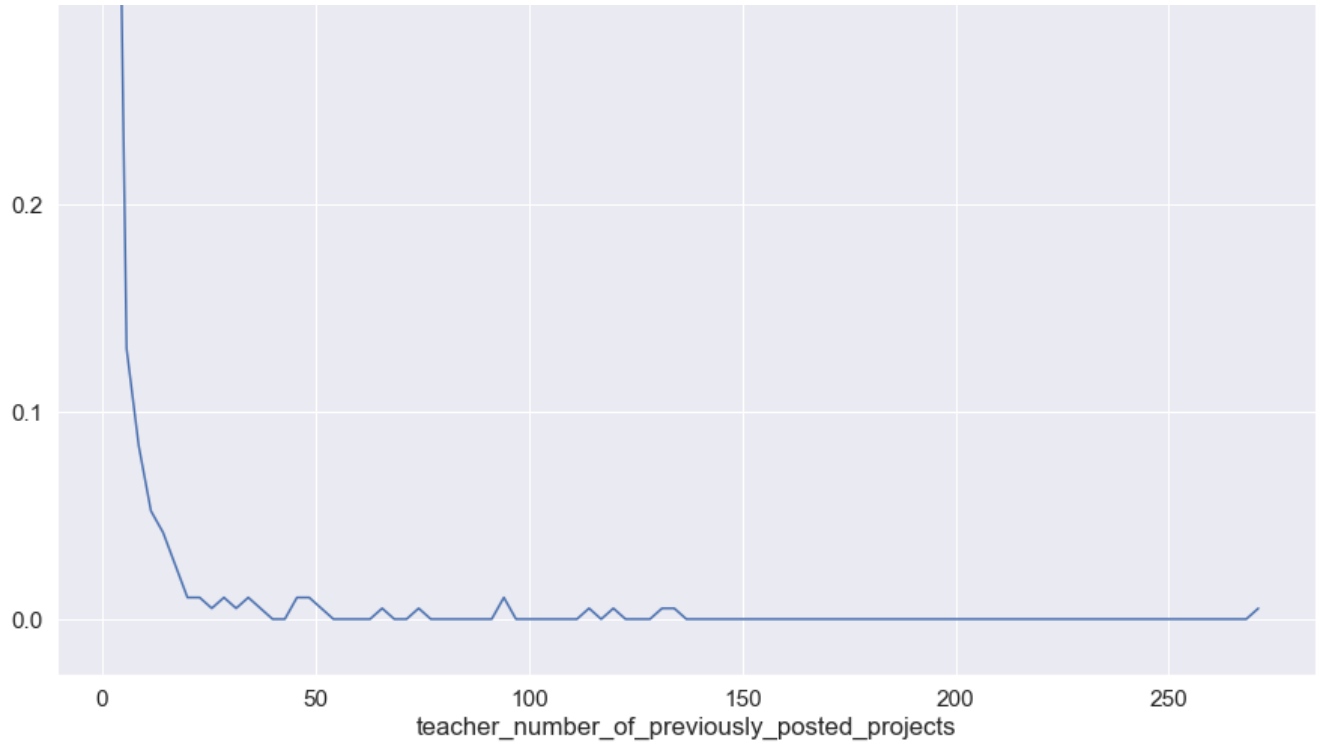
```
Text(0.5, 1.0, "Box Plot 'Price' v/s 'FP Data'")
```



In [217]:

```
# Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive da
ta points`
plt.figure(figsize=(15,15))
counts, bin_edges = np.histogram(X_test_falsePos["teacher_number_of_previously_posted_projects"],
bins="auto",density=True)
pdf = counts/sum(counts)
pdfPoints = plt.plot(bin_edges[1:],pdf)
plt.xlabel("teacher_number_of_previously_posted_projects")
plt.show()
```





1.6 Getting top features using `feature_importances_`

In [0]:

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

In [219]:

```
# Concatinating all the features for Set 1

X_tr = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_price_norm, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_teach_prev_norm)).tocsr()

X_cr = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe, X_cv_category_ohe, X_cv_subcategory_ohe,
               X_cv_price_norm, X_cv_teach_prev_norm)).tocsr()

X_te = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,
               X_test_price_norm, X_test_teach_prev_norm)).tocsr()
```

In [220]:

```
# https://stackoverflow.com/questions/47111434/randomforestregressor-and-feature-importances-error
def selectTopImportance(model, X, top=1):
    # model -> base classifier
    # X - Training Dataset
    # k - No of best features
    return X[:,model.best_estimator_.feature_importances_.argsort()[::-1][:top]]
```

In [221]:

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

```
dt_output = DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(dt_output, tree_parameters, cv=5, scoring='roc_auc', return_train_score=True, n_
jobs=-1)
clf.fit(X_tr,y_train)
```

Out[221]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort=False, random_state=None,
                                              splitter='best'),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [222]:

```
X_final_train = selectTopImportance(clf,X_tr, top=5000)
X_final_test = selectTopImportance(clf, X_te, top=5000)
```

In [223]:

```
print(X_final_train.shape)
print(X_final_test.shape)
```

```
(13467, 5000)
(9900, 5000)
```

In [224]:

```
dtree = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10, 100, 500]}
clf = GridSearchCV(dtree, parameters, cv=5, scoring='roc_auc',return_train_score=True)
finalset= clf.fit(X_final_train, y_train)
```

In [225]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('Best Hyper parameters: ',clf.best_params_)
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.6277725628104964
Best Hyper parameters:  {'max_depth': 10, 'min_samples_split': 500}
===========================================================================
Train AUC scores
[0.56108476 0.56108476 0.56108476 0.56108476 0.67758146 0.67738507
 0.67270047 0.66585923 0.81493931 0.81174346 0.77331275 0.72631164
 0.98885536 0.98207984 0.92795654 0.82659987]
CV AUC scores
[0.55590933 0.55590933 0.55590933 0.55590933 0.61900684 0.61791905
 0.61935497 0.62124441 0.61192864 0.61007091 0.61037762 0.62777256
```

```
 0.61955497 0.62124441 0.61192864 0.61007091 0.61057762 0.62777256
 0.55767495 0.55643592 0.57156239 0.60080686]
```

In [226]:

```
x1 = []
y1 = []
max_depth = [1, 5, 10, 50, 100, 500, 100]
min_samples_split = [5, 10, 100, 500]
train_auc_scores = clf.cv_results_['mean_train_score']
cv_auc_scores = clf.cv_results_['mean_test_score']

x1 = [x for item in max_depth for x in repeat(item, 4)]
y1 = [y for item in min_samples_split for y in repeat(item, 7)]
```

In [227]:

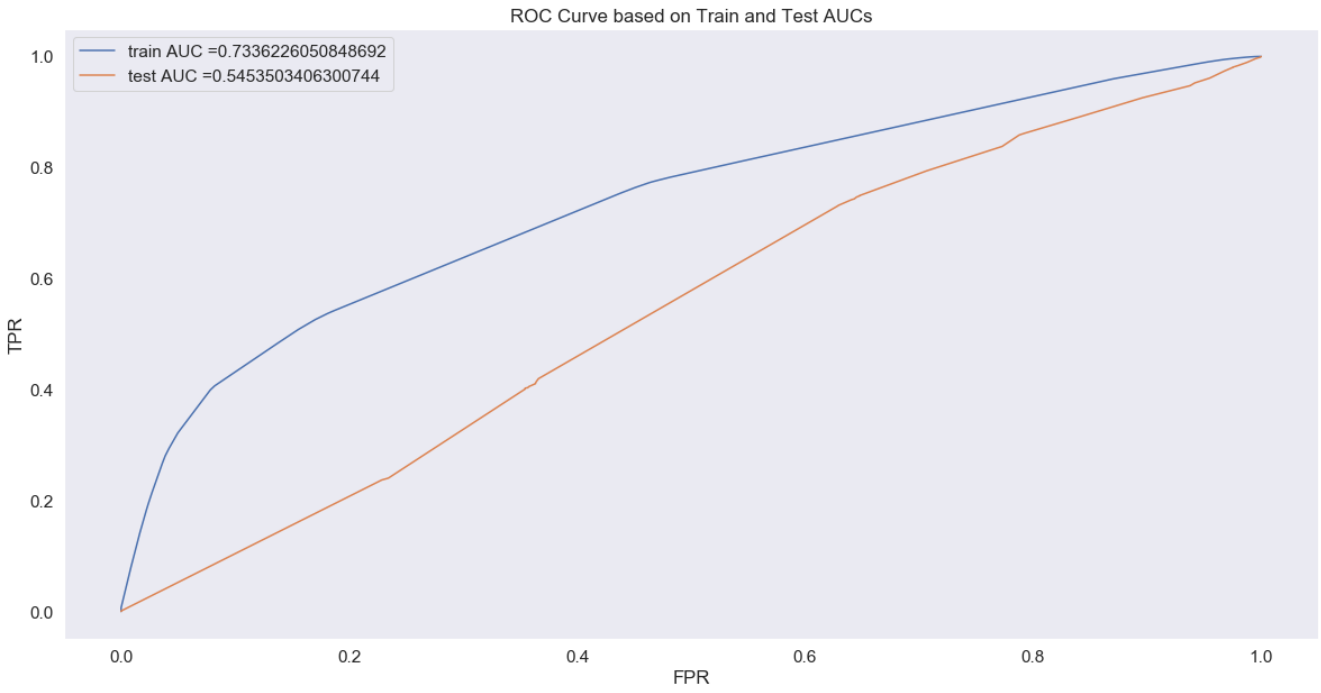
```
trace1 = go.Scatter3d(x=x1,y=y1,z=train_auc_scores, name="train auc")
trace2 = go.Scatter3d(x=x1,y=y1,z=cv_auc_scores, name="cv auc")
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='min_samples_split'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [228]:

```
dt_output = DecisionTreeClassifier(class_weight='balanced', max_depth =
clf.best_params_["max_depth"],
                                   min_samples_split = clf.best_params_["min_samples_split"])
dt_output.fit(X_tr, y_train)

y_train_pred = dt_output.predict_proba(X_tr)[:,1]  # returning probability estimates of positive c
lass
y_test_pred = dt_output.predict_proba(X_te)[:,1]

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)
```

In [229]:

```
plt.figure(figsize=(20,10))
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("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve based on Train and Test AUCs")
plt.grid()
plt.show()
```



In [230]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
Train_CM = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
Test_CM = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print("Train confusion matrix")
print(Train_CM)
print("Test confusion matrix")
print(Test_CM)
```

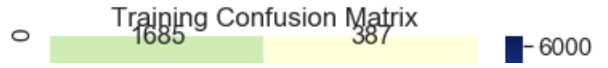
```
the maximum value of tpr*(1-fpr) 0.44076095151128064 for threshold 0.472
Train confusion matrix
[[1685  387]
 [5219 6176]]
Test confusion matrix
[[ 538  989]
 [2099 6274]]
```

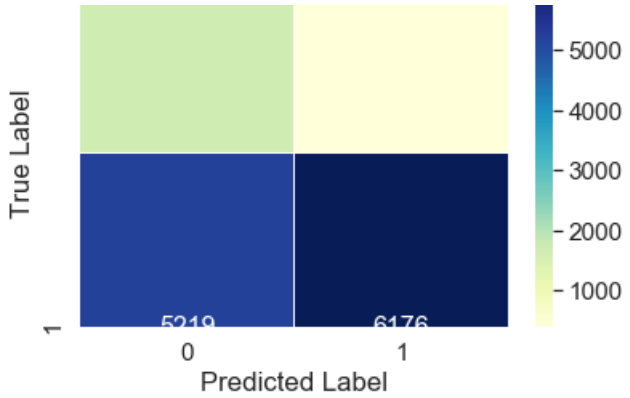
In [231]:

```
sns.set(font_scale=1.4)
sns.heatmap(Train_CM,annot=True,cbar=True,fmt="g", annot_kws = {"size":16},linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Training Confusion Matrix')
```

Out[231]:

```
Text(0.5, 1, 'Training Confusion Matrix')
```



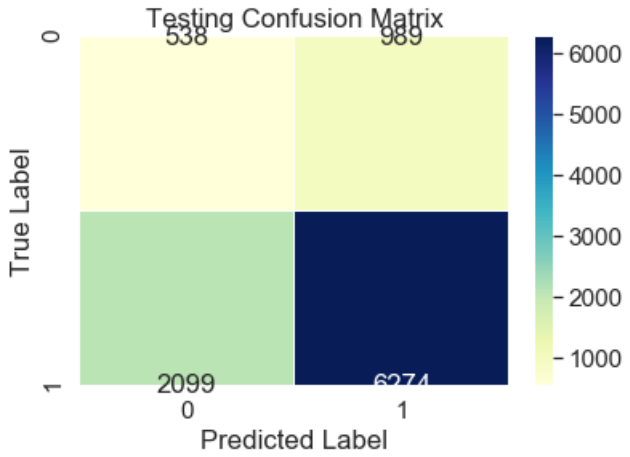


In [232]:

```
sns.heatmap(Test_CM,annot=True,cbar=True,fmt="d", linewidths=.5,cmap="YlGnBu")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Testing Confusion Matrix')
```

Out[232]:

```
Text(0.5, 1, 'Testing Confusion Matrix')
```



In [235]:

```
# Please write all the code with proper documentation
dtree = DecisionTreeClassifier(max_depth=3)
clf = dtree.fit(X_tr,y_train)
_dot_data = tree.export_graphviz(dtree, feature_names=None)
graph = Source(_dot_data)
graph.render("Best Features TFIDF Tree", view=True)
```

Out[235]:

```
'Best Features TFIDF Tree.pdf'
```

In [236]:

```
false_pos_indices = []
for i in range(len(y_test)):
    if(y_test[i]==0 and _predictions[i] == 1):
        false_pos_indices.append(i)
false_pos_essay = []
for i in false_pos_indices :
  false_pos_essay.append(X_test['essay'].values[i])
```

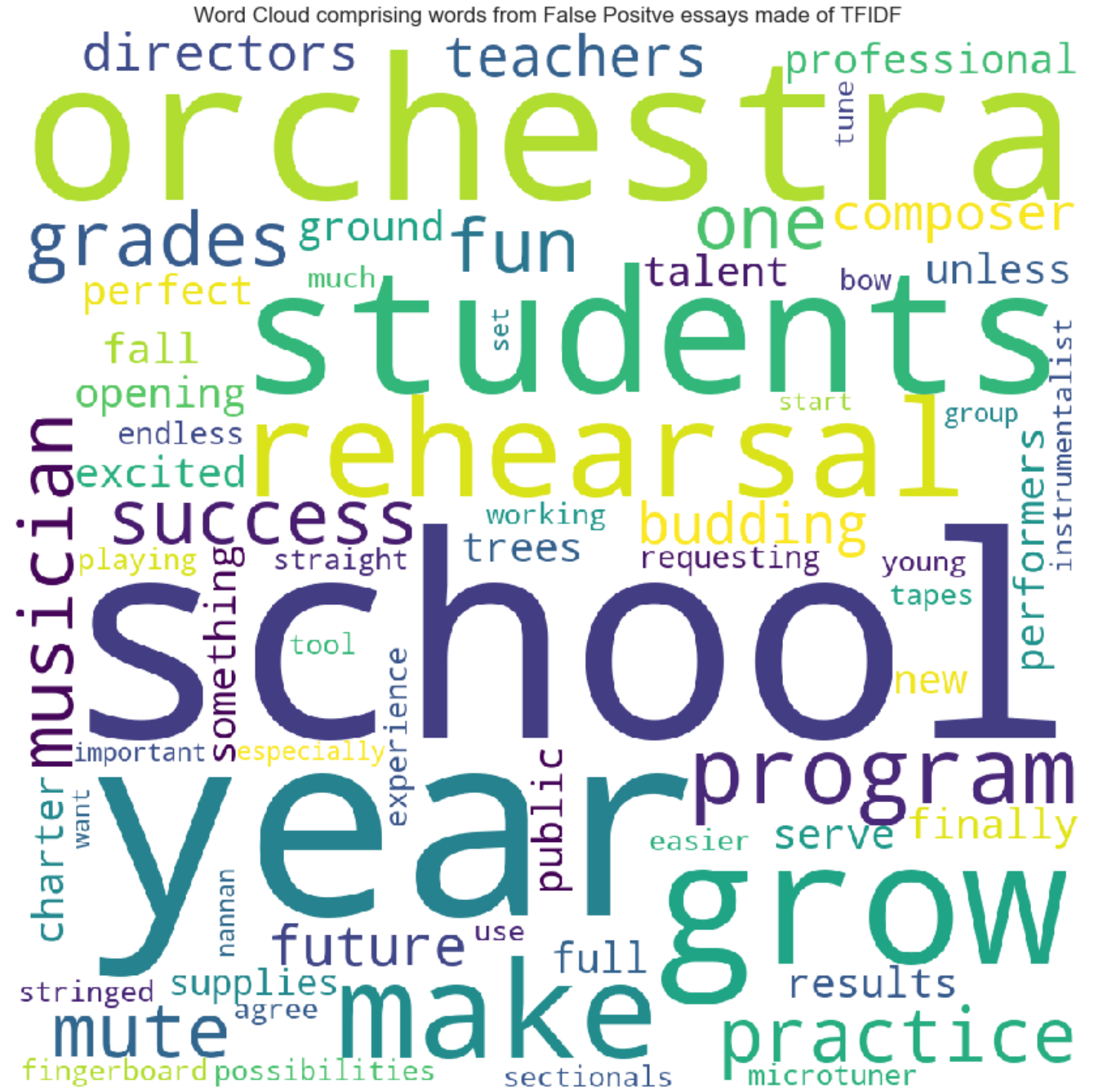
In [237]:

```
comments = " "
stopwords = set(STOPWORDS)
```

```
for _essay in false_pos_essay:
    tokens = str(_essay).lower().split()

for words in tokens:
    comments += words + " "

wordcloud = WordCloud(width=1000, height=1000, background_color="white",stopwords=stopwords,
min_font_size=12).generate(comments)
```

In [238]:

```
plt.figure(figsize=(15,15))
plt.imshow(wordcloud)
plt.axis("off")
plt.title("Word Cloud comprising words from False Positve essays made of TFIDF")
plt.show()
```



In [239]:

```
# Plot the box plot with the `price` of these `false positive data points`
cols = X_test.columns
X_test_falsePos = pd.DataFrame(columns=cols)
for i in false_pos_indices:
    X_test_falsePos = X_test_falsePos.append(X_test.filter(items=[i],axis=0))
```

```
X_test_falsePos.head(1)
```

Out[239]:

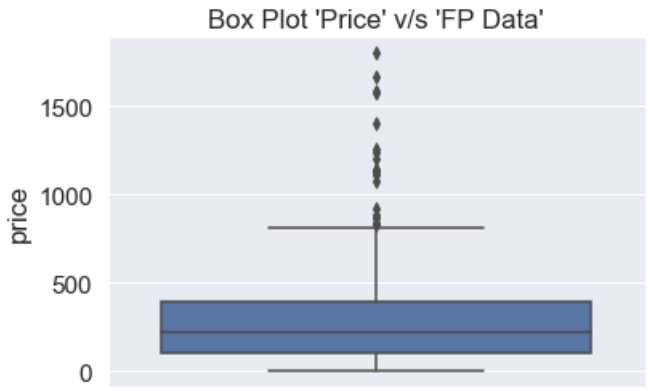
	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcate
1	ut	ms	grades_3_5	4	specialneeds	specia

In [240]:

```
sns.boxplot(y="price",data=X_test_falsePos).set_title("Box Plot 'Price' v/s 'FP Data'")
```

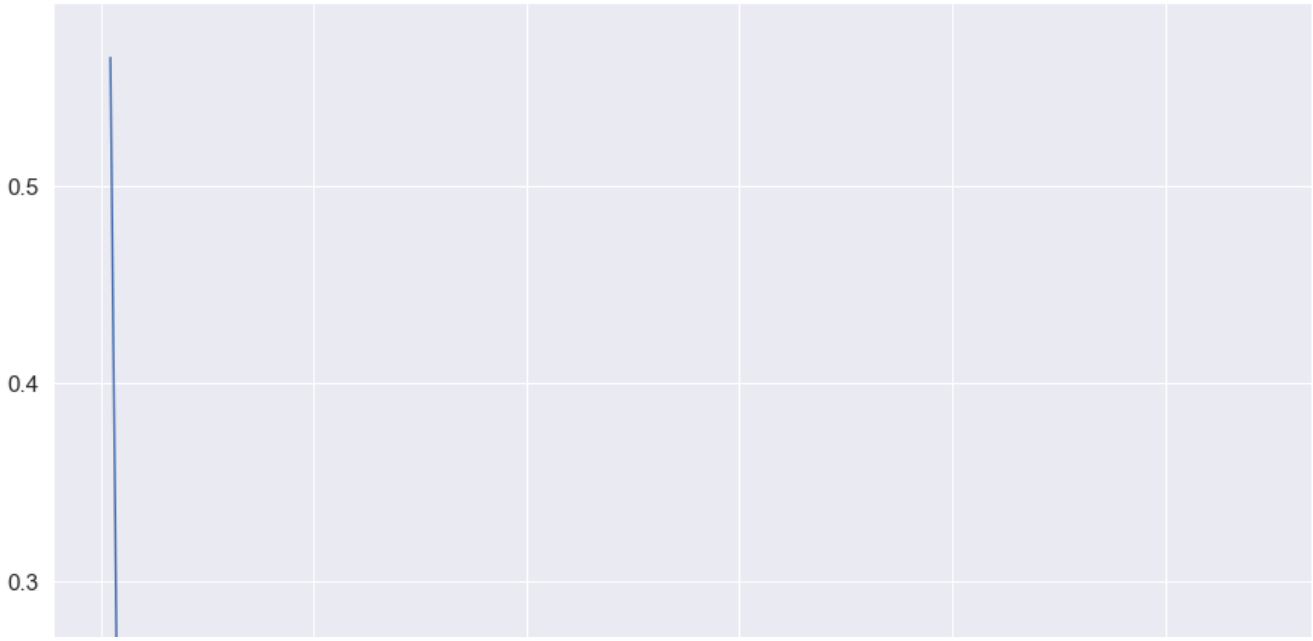
Out[240]:

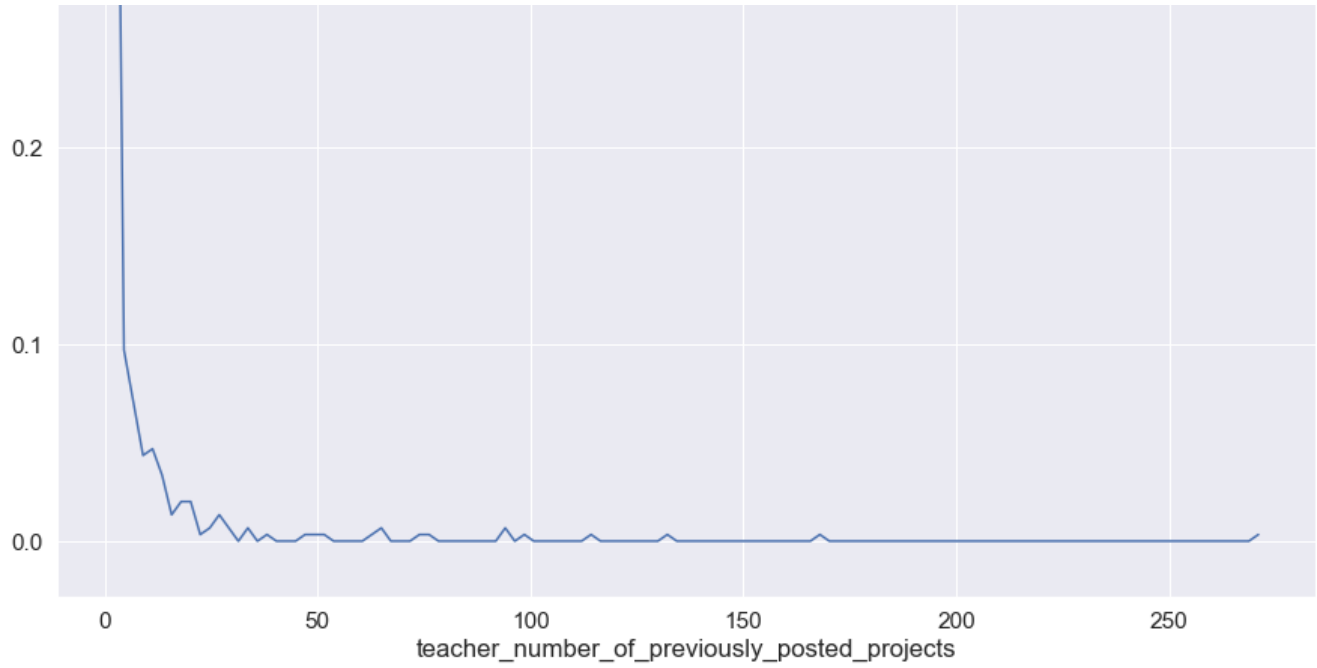
```
Text(0.5, 1.0, "Box Plot 'Price' v/s 'FP Data'")
```



In [241]:

```
# Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive da
ta points`
plt.figure(figsize=(15,15))
counts, bin_edges = np.histogram(X_test_falsePos["teacher_number_of_previously_posted_projects"],
bins="auto",density=True)
pdf = counts/sum(counts)
pdfPoints = plt.plot(bin_edges[1:],pdf)
plt.xlabel("teacher_number_of_previously_posted_projects")
plt.show()
```





2. Summary

In [242]:

```
from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter-max depth", "min samples split", "Train AUC
", "Test AUC"]
x.add_row(["TFIDF","DT",10,500,0.70,0.624])
x.add_row(["TFIDF W2V","DT",5,500,0.664,0.613])
x.add_row(["Best TFIDF", "DT", 10,500, 0.70, 0.62])
print(x)
```

```
+------------+-------+--------------------------+-------------------+-----------+----------+
| Vectorizer | Model | Hyperparameter-max depth | min samples split | Train AUC | Test AUC |
+------------+-------+--------------------------+-------------------+-----------+----------+
|   TFIDF    |   DT  |            10            |        500        |    0.7    |  0.624   |
| TFIDF W2V  |   DT  |            5             |        500        |   0.664   |  0.613   |
| Best TFIDF |   DT  |            10            |        500        |    0.7    |   0.62   |
+------------+-------+--------------------------+-------------------+-----------+----------+
```

In [ ]: