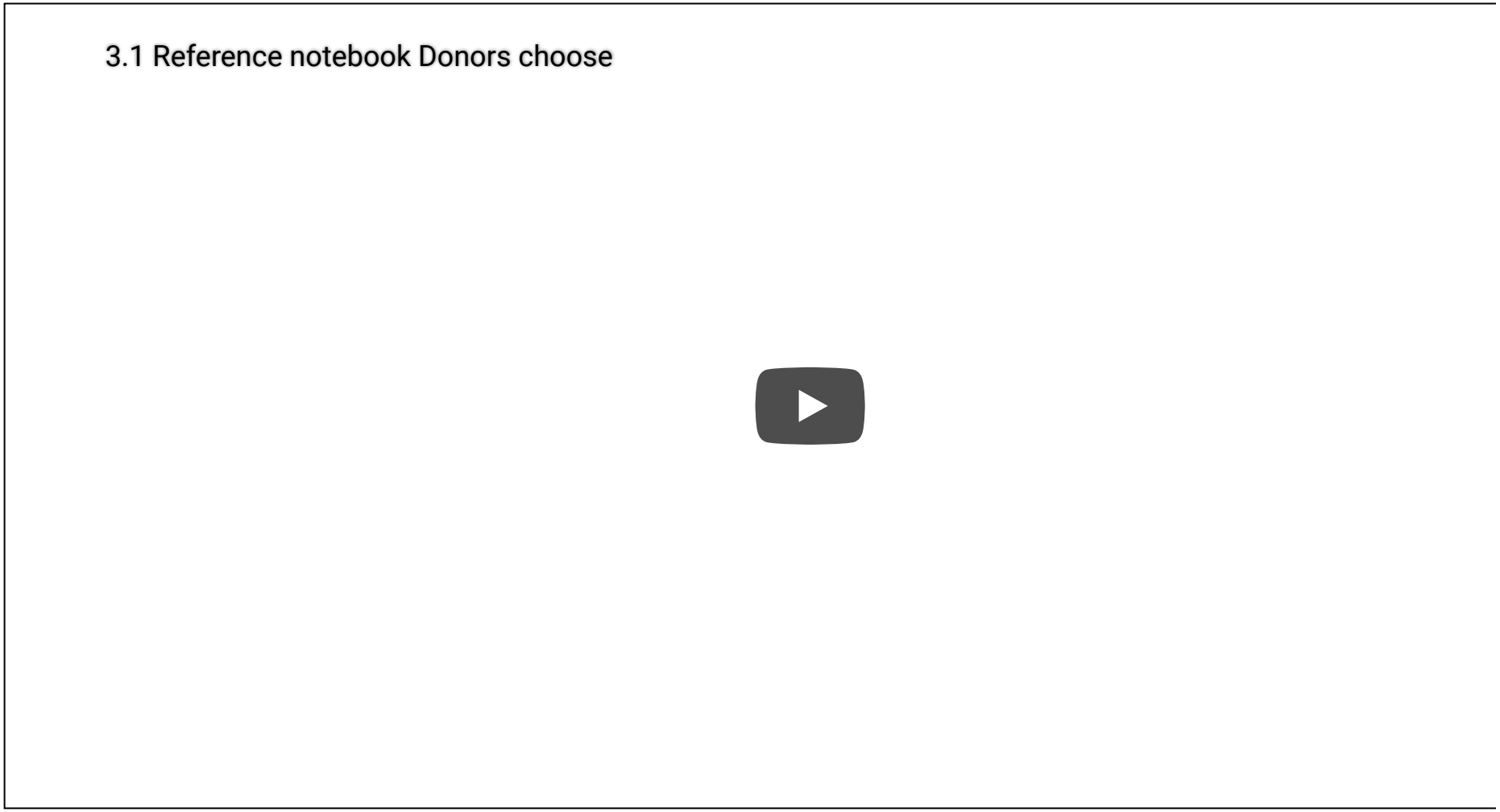
Assignment : DT

Please check below video before attempting this assignment

In [1]:
```
from IPython.display import YouTubeVideo
YouTubeVideo('ZhLXULFjIjQ', width="1000",height="500")
```

Out[1]:



TF-IDFW2V

Tfidf w2v (w1,w2..) = (tfidf(w1) * w2v(w1) + tfidf(w2) * w2v(w2) + …) / (tfidf(w1) + tfidf(w2) + …)

(Optional) Please check course video on AVgw2V and TF-IDFW2V (https://www.appliedaicourse.com/lecture/11/applied-machine-learning-

online-course/2916/avg-word2vec-tf-idf-weighted-word2vec/3/module-3-foundations-of-natural-language-processing-and-machine-learning)for more details.

Glove vectors

In this assignment you will be working with glove vectors , please check this (https://en.wikipedia.org/wiki/GloVe_(machine_learning)) and this (https://en.wikipedia.org/wiki/GloVe_(machine_learning)) for more details.

Download glove vectors from this link (https://drive.google.com/file/d/1lDca_ge-GYO0iQ6_XDLWePQFMdAA2b8f/view?usp=sharing)

In [2]:
```
#please use below code to load glove vectors
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-3d8008fb8a4a> in <module>
      1 #please use below code to load glove vectors
      2 with open('glove_vectors', 'rb') as f:
----> 3     model = pickle.load(f)
      4     glove_words =  set(model.keys())

NameError: name 'pickle' is not defined
```

or else , you can use below code

In [3]:
```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
```

```
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-

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


'''
```

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

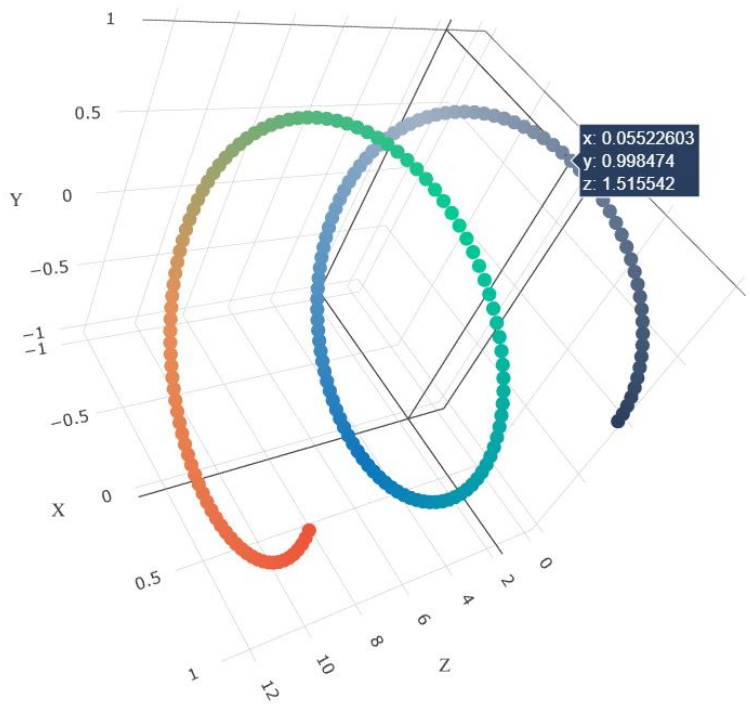
Task - 1

1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)
    - Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)
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**

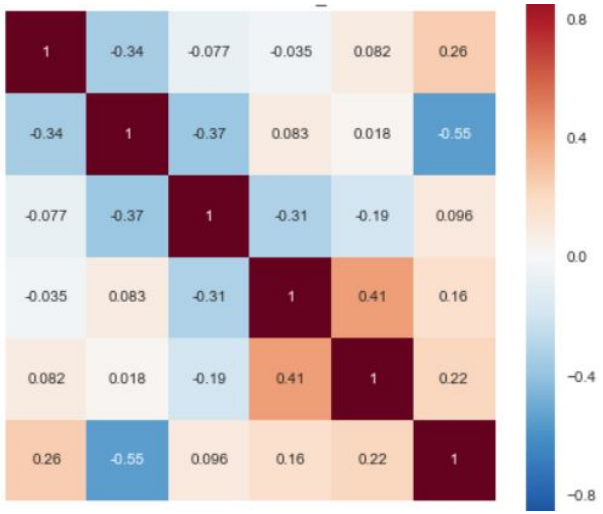
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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*

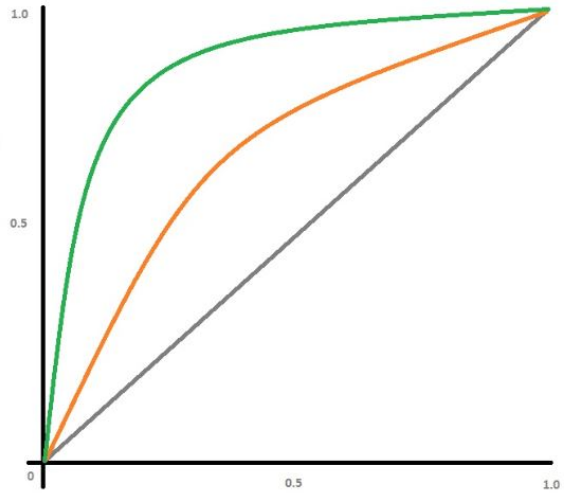
or

- 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 **min_sample_split**, columns as **max_depth**, and values inside the cell representing **AUC Score**

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



- Along with plotting ROC curve, you need to print the confusion matrix (https://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`

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) (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)), discard the all other remaining features and then apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM).
- You need to do hyperparameter tuning corresponding to the model you selected and procedure in step 2 and step 3
  Note: when you want to find the feature importance make sure you don't use max_depth parameter keep it None.

  You need to summarize the results at the end of the notebook, summarize it in the table format

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

Hint for calculating Sentiment scores

In [4]:
```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to C:\Users\disha
[nltk_data]     d\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[4]: True

In [5]:
```
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 with the biggest enthus
for learning my students learn in many different ways using all of our senses and multiple intelligences i use a wide ra
of techniques to help all my students succeed students in my class come from a variety of different backgrounds which ma
for wonderful sharing of experiences and cultures including native americans our school is a caring community of success
learners which can be seen through collaborative student project based learning in and out of the classroom kindergarten
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 kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our pretend kitch
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 cooking delicious health
food for snack time my students will have a grounded appreciation for the work that went into making the food and knowle
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our learning
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our own bre
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be printed an
shared with families students will gain math and literature skills as well as a life long enjoyment 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,
```

1. Decision Tree

1.1 Loading Data

In [6]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows=50000)
```

In [7]:
```
data.head(2)
```

Out[7]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_s
0	ca	mrs	grades_prek_2	53	1	math_science	a hea
1	ut	ms	grades_3_5	4	1	specialneeds	

In [8]:
```
data.shape
```

Out[8]:
```
(50000, 9)
```

splitting the data

In [9]:
```
y = data['project_is_approved'].values
x = data.drop(['project_is_approved'], axis = 1)
x.head(2)
```

Out[9]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	ess
0	ca	mrs	grades_prek_2	53	math_science	appliedsciences health_lifescience	i fortuna enou use fa tale st kits c
1	ut	ms	grades_3_5	4	specialneeds	specialneeds	imagin 9 ye old y th gra classro

In [10]:
```
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)
```

TF-IDF : vectorizing text data

In [11]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = TfidfVectorizer(ngram_range=(1,4), min_df=10, max_features = 5000)
x_train_essay_tfidf = vectorizer.fit_transform(x_train['essay'].values)
x_test_essay_tfidf = vectorizer.transform(x_test['essay'].values)

print("after vectorization")
print(x_train_essay_tfidf.shape, y_train.shape)
print(x_test_essay_tfidf.shape, y_test.shape)
```

```
after vectorization
(33500, 5000) (33500,)
(16500, 5000) (16500,)
```

vectorizing categorical features

In [12]:
```
model = CountVectorizer()
x_train_state_ohe = model.fit_transform(x_train['school_state'].values)
x_test_state_ohe = model.transform(x_test['school_state'].values)

print("after vectorization")
print(x_train_state_ohe.shape, y_train.shape)
print(x_test_state_ohe.shape, y_test.shape)
print(model.get_feature_names())
```

```
after vectorization
(33500, 51) (33500,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'm
a', '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']
```

In [13]:
```
model = CountVectorizer()
x_train_teacher_ohe = model.fit_transform(x_train['teacher_prefix'].values)
x_test_teacher_ohe = model.transform(x_test['teacher_prefix'].values)

print("after vectorization")
print(x_train_teacher_ohe.shape, y_train.shape)
print(x_test_teacher_ohe.shape, y_test.shape)
print(model.get_feature_names())
```

```
after vectorization
(33500, 5) (33500,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

In [14]:
```
model = CountVectorizer()
x_train_project_ohe = model.fit_transform(x_train['project_grade_category'].values)
x_test_project_ohe = model.transform(x_test['project_grade_category'].values)

print("after vectorization")
print(x_train_project_ohe.shape, y_train.shape)
print(x_test_project_ohe.shape, y_test.shape)
print(model.get_feature_names())
```

```
after vectorization
(33500, 4) (33500,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

In [15]:
```
model = CountVectorizer()
x_train_clean_categories_ohe = model.fit_transform(x_train['clean_categories'].values)
x_test_clean_categories_ohe = model.transform(x_test['clean_categories'].values)

print("after vectorization")
print(x_train_clean_categories_ohe.shape, y_train.shape)
print(x_test_clean_categories_ohe.shape, y_test.shape)
print(model.get_feature_names())
```

```
after vectorization
(33500, 9) (33500,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_art
s', 'specialneeds', 'warmth']
```

In [16]:
```
model = CountVectorizer()
x_train_clean_subcategories_ohe = model.fit_transform(x_train['clean_subcategories'].values)
x_test_clean_subcategories_ohe = model.transform(x_test['clean_subcategories'].values)

print("after vectorization")
print(x_train_clean_categories_ohe.shape, y_train.shape)
print(x_test_clean_categories_ohe.shape, y_test.shape)
print(model.get_feature_names())
```

```
after vectorization
(33500, 9) (33500,)
(16500, 9) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice',
'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguag
es', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'ma
thematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialne
eds', 'teamsports', 'visualarts', 'warmth']
```

In [17]:
```
x_train_price = x_train['price'].values.reshape(-1,1)
x_test_price = x_test['price'].values.reshape(-1,1)
```

In [18]:
```
x_train_prev_project = x_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
x_test_prev_project = x_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
```

Sentiment scores of essay

In [19]:
```
essay = list(x_train['essay'])
sentiment_scores = []
for i in essay:
    ss = sid.polarity_scores(i)
    sentiment_scores.append(ss)
```

In [20]:
```
neg_train = []
neu_train = []
pos_train = []
compound_train = []
for i in sentiment_scores:
    neg_train.append(i['neg'])
    neu_train.append(i['neu'])
    pos_train.append(i['pos'])
    compound_train.append(i['compound'])
```

In [21]:
```
essay = list(x_test['essay'])
sentiment_scores = []
for i in essay:
    ss = sid.polarity_scores(i)
    sentiment_scores.append(ss)
```

In [22]:
```
neg_test = []
neu_test = []
pos_test = []
compound_test = []
for i in sentiment_scores:
    neg_test.append(i['neg'])
    neu_test.append(i['neu'])
    pos_test.append(i['pos'])
    compound_test.append(i['compound'])
```

In [23]:
```
import numpy as np
neg_train = np.array(neg_train).reshape(-1,1)
pos_train = np.array(pos_train).reshape(-1,1)
neu_train = np.array(neu_train).reshape(-1,1)
compound_train = np.array(compound_train).reshape(-1,1)
neg_test = np.array(neg_test).reshape(-1,1)
pos_test = np.array(pos_test).reshape(-1,1)
neu_test = np.array(neu_test).reshape(-1,1)
compound_test = np.array(compound_test).reshape(-1,1)
```

In [24]:
```
from scipy.sparse import hstack
x_tr = hstack((x_train_essay_tfidf, x_train_state_ohe, x_train_teacher_ohe,x_train_project_ohe,x_train_clean_categories_
x_te = hstack((x_test_essay_tfidf, x_test_state_ohe, x_test_teacher_ohe, x_test_project_ohe, x_test_clean_categories_ohe

print("Final Data matrix")
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
Final Data matrix
(33500, 8) (33500,)
(16500, 8) (16500,)
```

In [25]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, auc
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(class_weight='balanced')
clf.fit(x_tr,y_train)
```

Out[25]:
```
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')
```

In [26]:
```
y_train_pred = clf.predict(x_tr)
```

In [27]:
```
train_auc = roc_auc_score(y_train,y_train_pred)
```

In [28]:
```
train_auc
```

Out[28]:
```
0.9999822285409632
```

In [29]:
```
hyperparameters = {'max_depth':[1,5,10,50],'min_samples_split':[5,10,100,500]}
grid_search = GridSearchCV(clf,hyperparameters, scoring = 'roc_auc', cv = 5, verbose = 1,n_jobs = -1,return_train_score=
```

In [30]:
```
grid_search.fit(x_tr,y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:  8.5min finished
```

Out[30]:
```
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'),
       fit_params=None, 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=1)
```

In [31]:
```
best_depth = grid_search.best_params_['max_depth']
best_min_samples_split = grid_search.best_params_['min_samples_split']
```

In [32]:
```
auc = grid_search.score(x_te,y_test)
```

In [33]:
```
auc
```

Out[33]:
```
0.6380193683985583
```

In [34]:
```
clf = DecisionTreeClassifier(class_weight = 'balanced',max_depth = best_depth, min_samples_split = best_min_samples_spli
clf.fit(x_tr,y_train)
```

Out[34]:
```
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')
```

In [35]:
```
y_test_pred = clf.predict(x_te)
```

In [36]:
```
test_auc = roc_auc_score(y_test,y_test_pred)
```

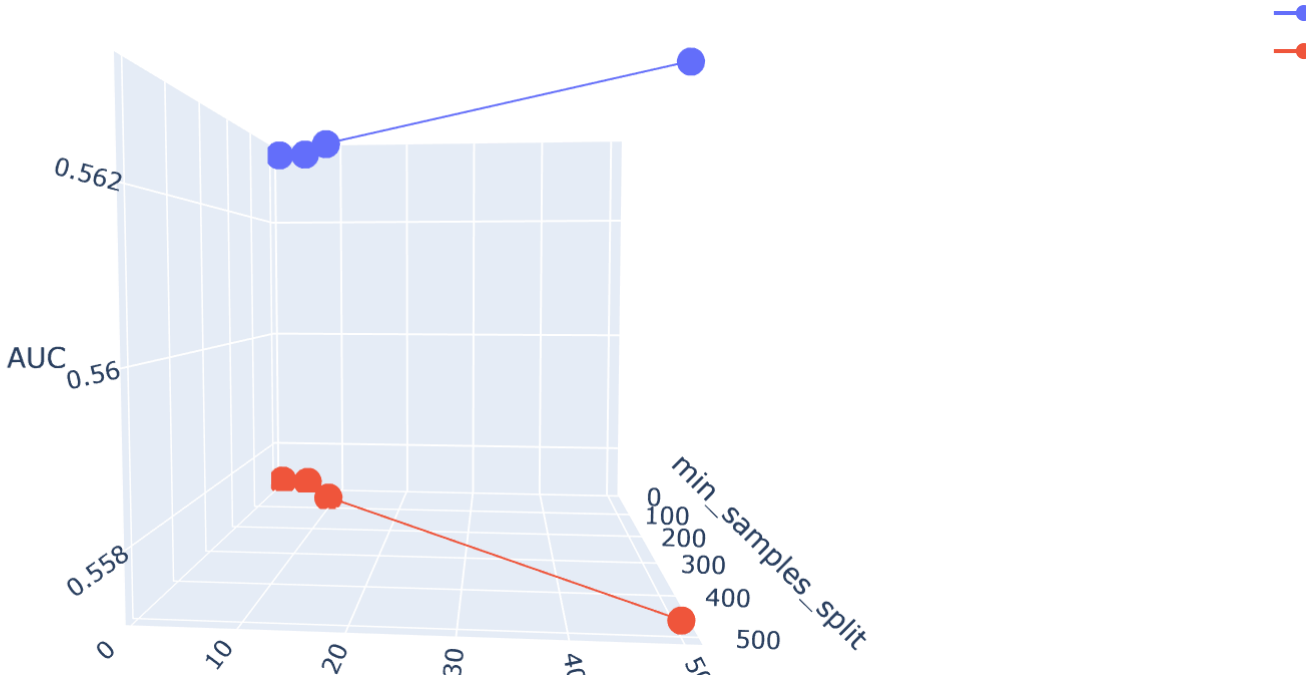
In [37]:
```
test_auc
```

Out[37]:
```
0.6016087363459088
```

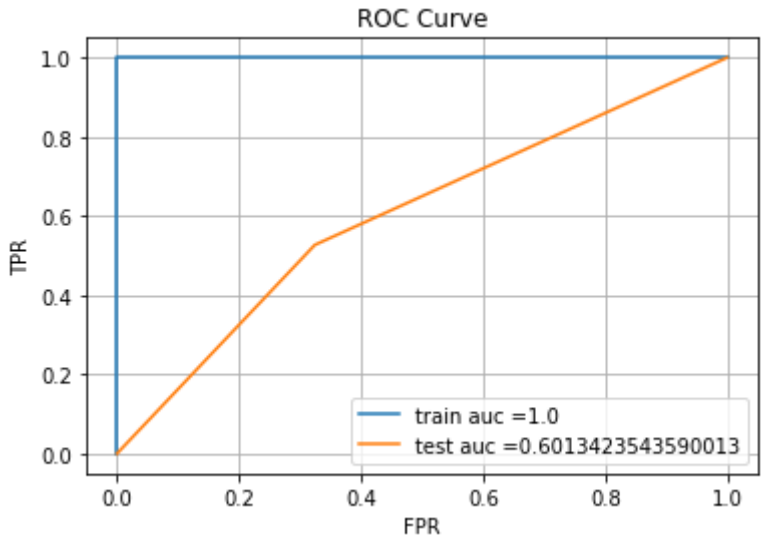
In [38]:
```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
```

In [39]:
```
x1 = [5,10,100,500]
y1 = [1,5,10,50]
z1 = grid_search.cv_results_['mean_train_score']
x2 = [5,10,100,500]
y2 = [1,5,10,50]
z2 = grid_search.cv_results_['mean_test_score']
```

In [40]:
```
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'test')
data = [trace1, trace2]

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

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

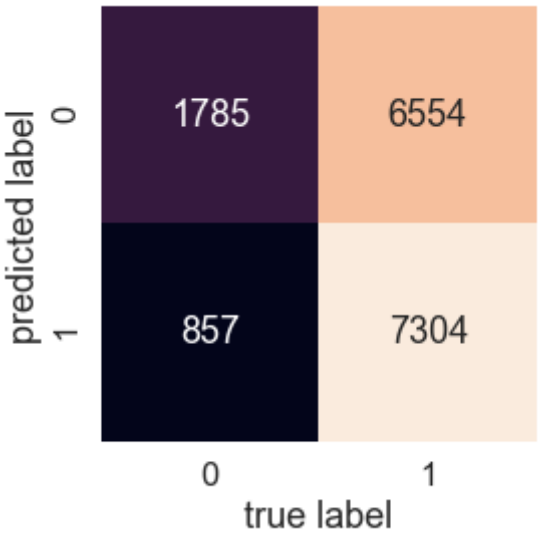


In [48]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

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)


auc_train=auc(train_fpr, train_tpr)
auc_test=auc(test_fpr, test_tpr)


plt.plot(train_fpr, train_tpr, label="train auc ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test auc ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



In [66]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

con_mat = confusion_matrix(y_test, y_test_pred)
sns.heatmap(con_mat.T, square=True, annot=True, fmt='d', cbar=False)
sns.set(font_scale=1.5)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[66]:
```
Text(84.18, 0.5, 'predicted label')
```



In [41]:
```
false_positives = np.logical_and(y_test == 0, y_test_pred == 1)

x_te_false_pos = x_test[false_positives]['essay']
x_test_false_positive = x_test[false_positives]
```

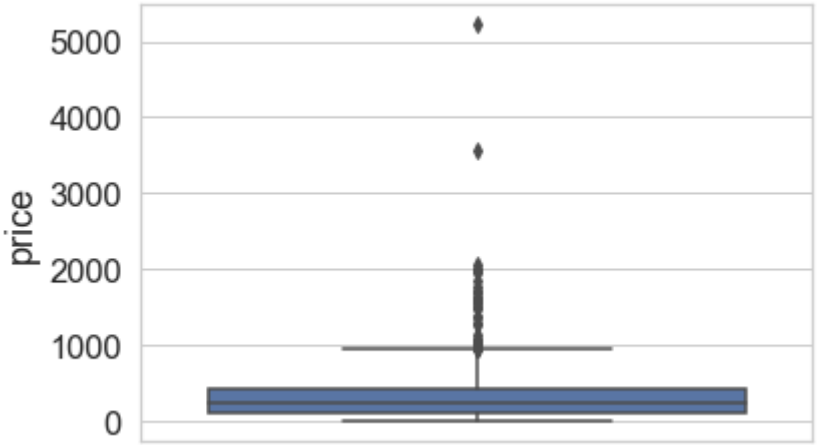
In [53]:
```
!pip install wordcloud
```

```
Requirement already satisfied: wordcloud in c:\programdata\anaconda3\lib\site-packages (1.8.0)
Requirement already satisfied: pillow in c:\programdata\anaconda3\lib\site-packages (from wordcloud) (5.3.0)
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-packages (from wordcloud) (3.0.2)
Requirement already satisfied: numpy>=1.6.1 in c:\users\disha d\appdata\roaming\python\python37\site-packages (from wor
dcloud) (1.18.1)
Requirement already satisfied: cycler>=0.10 in c:\users\disha d\appdata\roaming\python\python37\site-packages (from mat
plotlib->wordcloud) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\disha d\appdata\roaming\python\python37\site-packages (fro
m matplotlib->wordcloud) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\users\disha d\appdata\roaming\python\pyth
on37\site-packages (from matplotlib->wordcloud) (2.4.6)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\disha d\appdata\roaming\python\python37\site-packages
(from matplotlib->wordcloud) (2.8.1)
Requirement already satisfied: six in c:\users\disha d\appdata\roaming\python\python37\site-packages (from cycler>=0.10
->matplotlib->wordcloud) (1.14.0)
Requirement already satisfied: setuptools in c:\programdata\anaconda3\lib\site-packages (from kiwisolver>=1.0.1->matplo
tlib->wordcloud) (40.6.3)
```

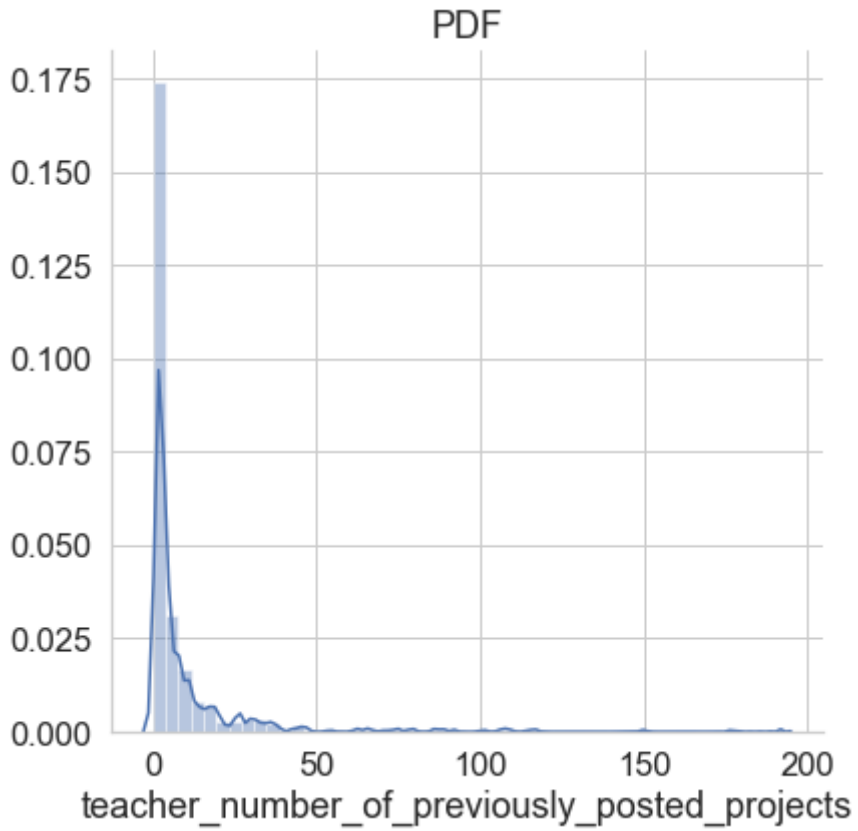
In [70]:
```
from wordcloud import WordCloud, STOPWORDS
comment_words = ''
stopwords = set(STOPWORDS)
for word in x_te_false_pos:
    word = str(word)
    tokens = word.split()

for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
for words in tokens:
    comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stopwords,min_font_size = 10).ge

plt.figure(figsize = (5, 5), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

In [78]:
```
sns.set_style('whitegrid')
sns.boxplot(y='price', data = x_test_false_positive)
```

Out[78]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd2ad0af60>
```



In [85]:
```
sns.set_style("whitegrid")
fig = sns.FacetGrid(x_test_false_positive, height=6)
fig.map(sns.distplot,'teacher_number_of_previously_posted_projects')
fig.add_legend()
plt.title('PDF')
plt.show()
```



Task2

In [260]:
```
important_features = clf.feature_importances_
non_zero_indices = np.nonzero(important_features)
```

In [261]:
```
non_zero_indices = [i for i in non_zero_indices[0]]
```

In [264]:
```
x_te_imp = x_te[:,non_zero_indices]
x_tr_imp = x_tr[:,non_zero_indices]
```

In [268]:
```
clf = DecisionTreeClassifier(class_weight='balanced')
clf.fit(x_tr_imp,y_train)
```

Out[268]:
```
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')
```

In [270]:
```
y_train_pred = clf.predict(x_tr_imp)
```

In [271]:
```
roc_auc_score(y_train,y_train_pred)
```

Out[271]:
```
1.0
```

In [272]:
```
hyperparameters = {'max_depth':[1,5,10,50],'min_samples_split':[5,10,100,500]}
grid_search = GridSearchCV(clf,hyperparameters, scoring = 'roc_auc', cv = 5, verbose = 1,n_jobs = -1,return_train_score=
```

In [273]:
```
grid_search.fit(x_tr_imp,y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:  4.2min finished
```

Out[273]:
```
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'),
       fit_params=None, 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=1)
```

In [275]:
```
grid_search.best_params_['max_depth']
```

Out[275]:
```
10
```

In [276]:
```
grid_search.best_params_['min_samples_split']
```

Out[276]:
```
500
```

In [277]:
```
grid_search.score(x_te_imp,y_test)
```

Out[277]:
```
0.6388448848922821
```

TF-IDF W2V

In [42]:
```
import pickle
from tqdm import tqdm
import os
```

In [43]:
```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [47]:
```
preprocessed_essays_train = x_train['essay'].values
preprocessed_essays_test = x_test['essay'].values
```

In [48]:
```
#S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_train)
# 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 [49]:
```
# average Word2Vec
# compute average word2vec for each review.
x_train_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_train): # 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.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_train_tfidf_w2v.append(vector)

print(len(x_train_tfidf_w2v))
print(len(x_train_tfidf_w2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 33500/33500 [02:23<00:00, 234.12it/s]

33500
300
```

In [50]:
```
# average Word2Vec
# compute average word2vec for each review.
x_test_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_test): # 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.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_test_tfidf_w2v.append(vector)

print(len(x_test_tfidf_w2v))
print(len(x_test_tfidf_w2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 16500/16500 [00:57<00:00, 288.50it/s]

16500
300
```

In [51]:
```
x_tr_w2v = hstack((x_train_tfidf_w2v, x_train_state_ohe, x_train_teacher_ohe,x_train_project_ohe,x_train_clean_categorie
x_te_w2v = hstack((x_test_tfidf_w2v, x_test_state_ohe, x_test_teacher_ohe, x_test_project_ohe, x_test_clean_categories_o

print("Final Data matrix")
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
Final Data matrix
(33500, 8) (33500,)
(16500, 8) (16500,)
```

In [52]:
```
clf2 = DecisionTreeClassifier(class_weight='balanced')
clf2.fit(x_tr_w2v,y_train)
```

Out[52]:
```
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')
```

In [53]:
```
y_train_pred2 = clf2.predict(x_tr_w2v)
```

In [54]:
```
train_auc2 = roc_auc_score(y_train,y_train_pred2)
```

In [55]:
```
train_auc2
```

Out[55]:
```
0.9999822285409632
```

In [56]:
```
hyperparameters = {'max_depth':[1,5,10,50],'min_samples_split':[5,10,100,500]}
grid_search2 = GridSearchCV(clf2,hyperparameters, scoring = 'roc_auc', cv = 5, verbose = 1,n_jobs = -1,return_train_scor
```

In [57]:
```
grid_search2.fit(x_tr_w2v,y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed: 12.3min finished
```

Out[57]:
```
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'),
       fit_params=None, 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=1)
```

In [58]:
```
best_depth2 = grid_search2.best_params_['max_depth']
best_min_samples_split2 = grid_search2.best_params_['min_samples_split']
```

In [59]:
```
clf2 = DecisionTreeClassifier(class_weight = 'balanced',max_depth = best_depth2, min_samples_split = best_min_samples_sp
clf2.fit(x_tr_w2v,y_train)
```

Out[59]:
```
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 [60]:
```
y_test_pred2 = clf2.predict(x_te_w2v)
```

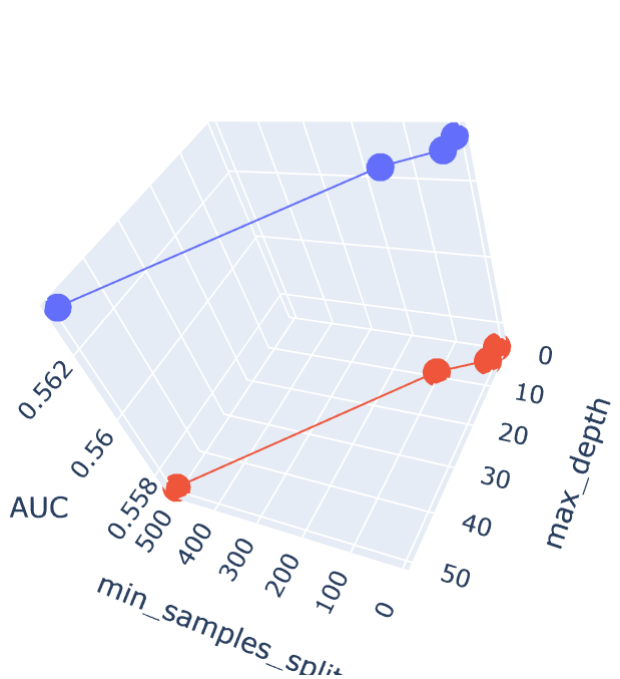
In [61]:
```
test_auc = roc_auc_score(y_test,y_test_pred2)
```

In [62]:
```
test_auc
```

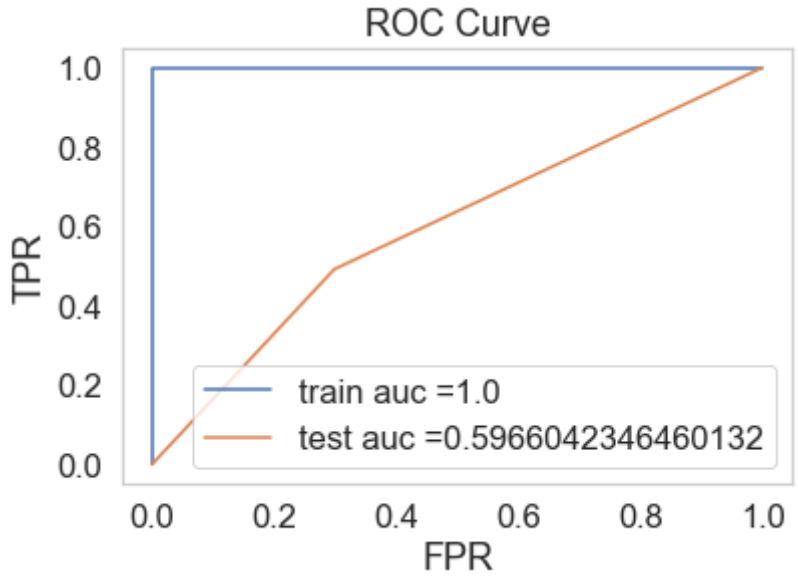
Out[62]:
```
0.588326345437977
```

In [63]:
```
x1 = [5,10,100,500]
y1 = [1,5,10,50]
z1 = grid_search2.cv_results_['mean_train_score']
x2 = [5,10,100,500]
y2 = [1,5,10,50]
z2 = grid_search2.cv_results_['mean_test_score']
```

In [64]:
```
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'test')
data = [trace1, trace2]

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

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

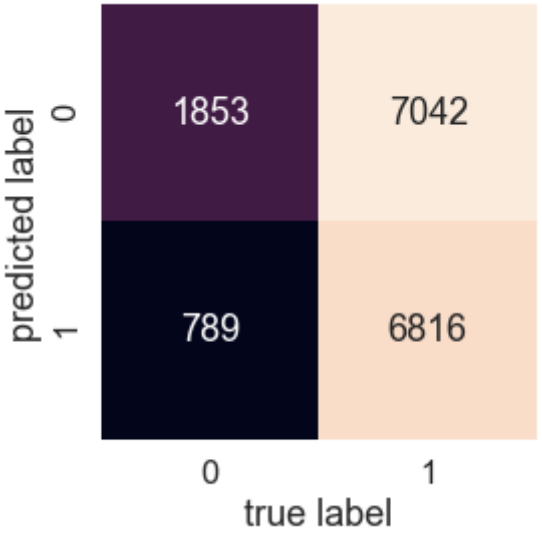


In [183]:
```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred2)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred2)


auc_train=auc(train_fpr, train_tpr)
auc_test=auc(test_fpr, test_tpr)


plt.plot(train_fpr, train_tpr, label="train auc ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test auc ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



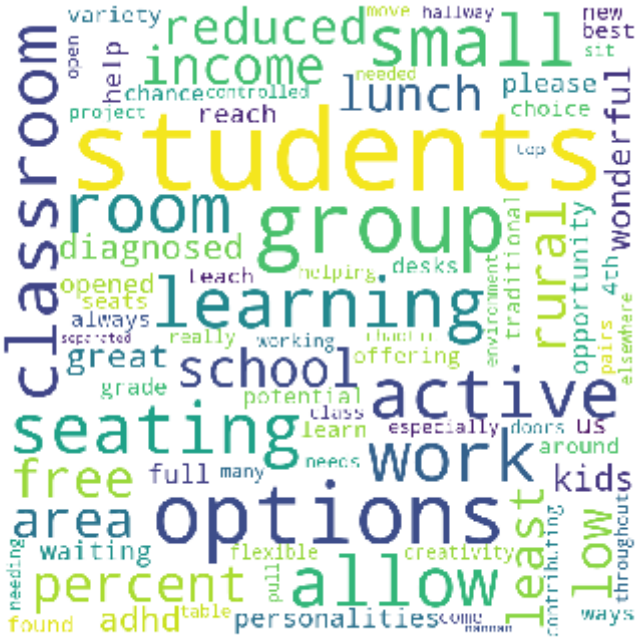
In [184]:
```
con_mat = confusion_matrix(y_test, y_test_pred2)
sns.heatmap(con_mat.T, square=True, annot=True, fmt='d', cbar=False)
sns.set(font_scale=1.5)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[184]:
```
Text(84.18, 0.5, 'predicted label')
```



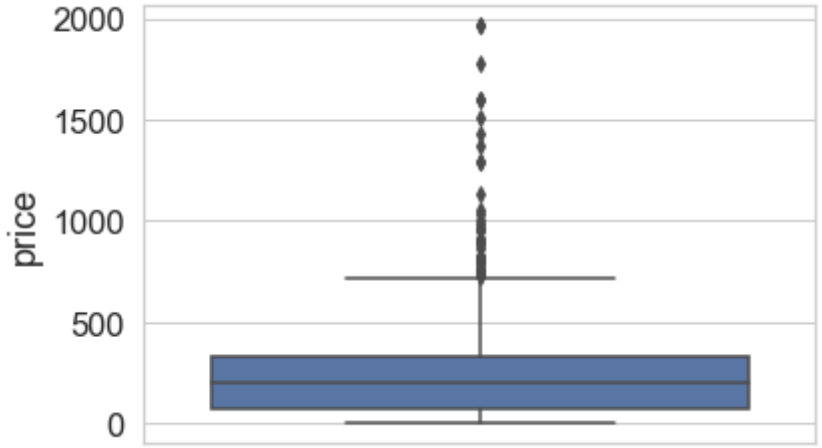
In [185]:
```
false_positives = np.logical_and(y_test == 0, y_test_pred2 == 1)

x_te_false_pos2 = x_test[false_positives]['essay']
x_test_false_positive2 = x_test[false_positives]
```

In [186]:
```
comment_words = ''
stopwords = set(STOPWORDS)
for word in x_te_false_pos2:
    word = str(word)
    tokens = word.split()

for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
for words in tokens:
    comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stopwords,min_font_size = 10).ge

plt.figure(figsize = (5, 5), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



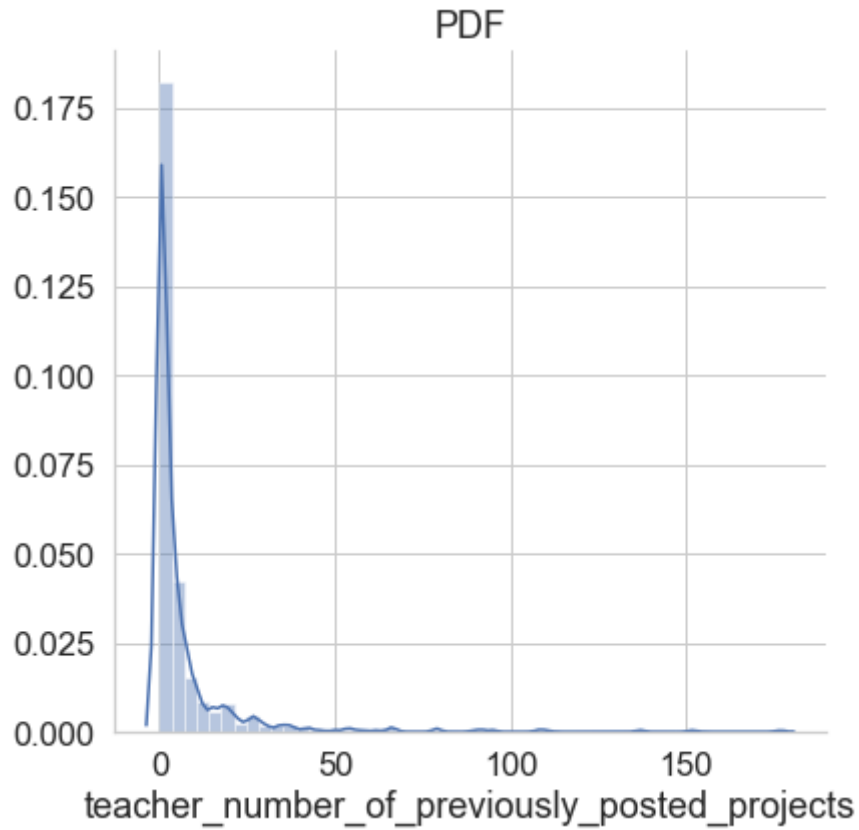
In [187]:
```
sns.set_style('whitegrid')
sns.boxplot(y='price', data = x_test_false_positive2)
```

Out[187]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd2079ebe0>



In [188]:
```
sns.set_style("whitegrid")
fig = sns.FacetGrid(x_test_false_positive2, height=6)
fig.map(sns.distplot,'teacher_number_of_previously_posted_projects')
fig.add_legend()
plt.title('PDF')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning:

Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
```



In [278]:
```
!pip install prettytable
```

```
Collecting prettytable
  Downloading https://files.pythonhosted.org/packages/39/da/8336296a830caa495a25304e12ffb32a8c3a9d2d08ba995f066fe16152e1/prettytable-1.0.1-py2.py3-none-any.whl (https://files.pythonhosted.org/packages/39/da/8336296a830caa495a25304e12ffb32a8c3a9d2d08ba995f066fe16152e1/prettytable-1.0.1-py2.py3-none-any.whl)
Requirement already satisfied: wcwidth in c:\programdata\anaconda3\lib\site-packages (from prettytable) (0.1.7)
Requirement already satisfied: setuptools in c:\programdata\anaconda3\lib\site-packages (from prettytable) (40.6.3)
Installing collected packages: prettytable
Successfully installed prettytable-1.0.1
```

In [280]:
```
from prettytable import PrettyTable


results = PrettyTable(["Vectorizer", "Model", "Maximum Depth", "Minimum sample splits","AUC"])

results.add_row(["TF-IDF", "DT", "10", "500", "0.601"])
results.add_row(["TF-IDF", "DT", "5", "500", "0.596"])
results.add_row(["IMP FEATURES","DT", "10", "500", "0.638"])

print(results)
```

```
+--------------+-------+---------------+-----------------------+-------+
|  Vectorizer  | Model | Maximum Depth | Minimum sample splits |  AUC  |
+--------------+-------+---------------+-----------------------+-------+
|    TF-IDF    |   DT  |       10      |          500          | 0.601 |
|    TF-IDF    |   DT  |       5       |          500          | 0.596 |
| IMP FEATURES |   DT  |       10      |          500          | 0.638 |
+--------------+-------+---------------+-----------------------+-------+
```

References

https://www.geeksforgeeks.org/python-sentiment-analysis-using-vader/ (https://www.geeksforgeeks.org/python-sentiment-analysis-using-vader/)

https://plot.ly/python/3d-axes/ (https://plot.ly/python/3d-axes/)

https://stackoverflow.com/questions/8386675/extracting-specific-columns-in-numpy-array (https://stackoverflow.com/questions/8386675/extracting-specific-columns-in-numpy-array)

https://machinelearningmastery.com/calculate-feature-importance-with-python/ (https://machinelearningmastery.com/calculate-feature-importance-with-python/)

https://www.geeksforgeeks.org/numpy-nonzero-in-python/#:~:text=nonzero()%20function%20is%20used,arr%5Bnonzero(arr)%5D%20 (https://www.geeksforgeeks.org/numpy-nonzero-in-python/#:~:text=nonzero()%20function%20is%20used,arr%5Bnonzero(arr)%5D%20).

https://www.geeksforgeeks.org/generating-word-cloud-python/ (https://www.geeksforgeeks.org/generating-word-cloud-python/)

https://www.geeksforgeeks.org/creating-tables-with-prettytable-library-python/ (https://www.geeksforgeeks.org/creating-tables-with-prettytable-library-python/)

reference notebooks that was provided