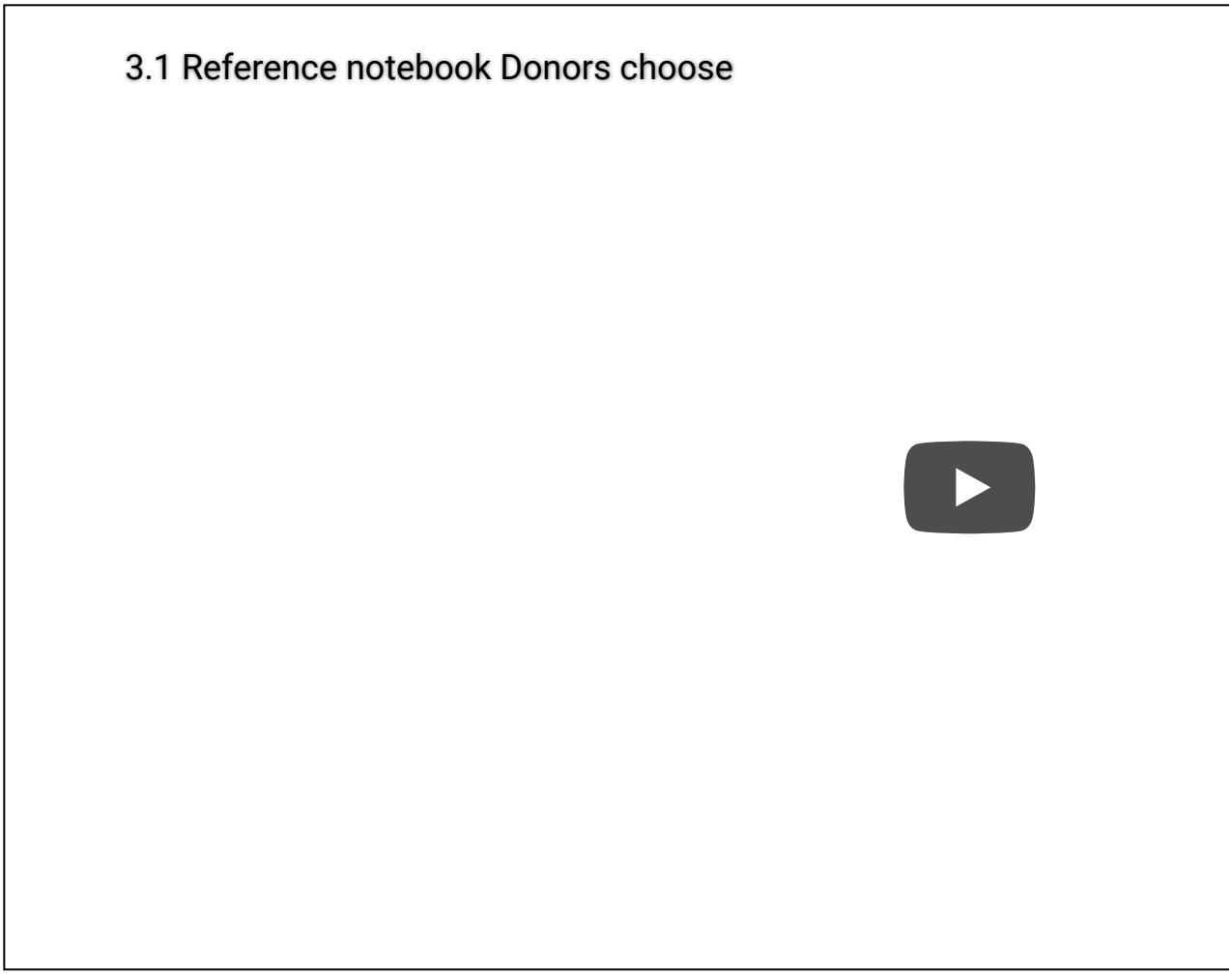
Assignment : DT

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 for more details.

Glove vectors

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

Download glove vectors from this link

In [1]:
```
#please use below code to load glove vectors
```

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

or else , you can use below code

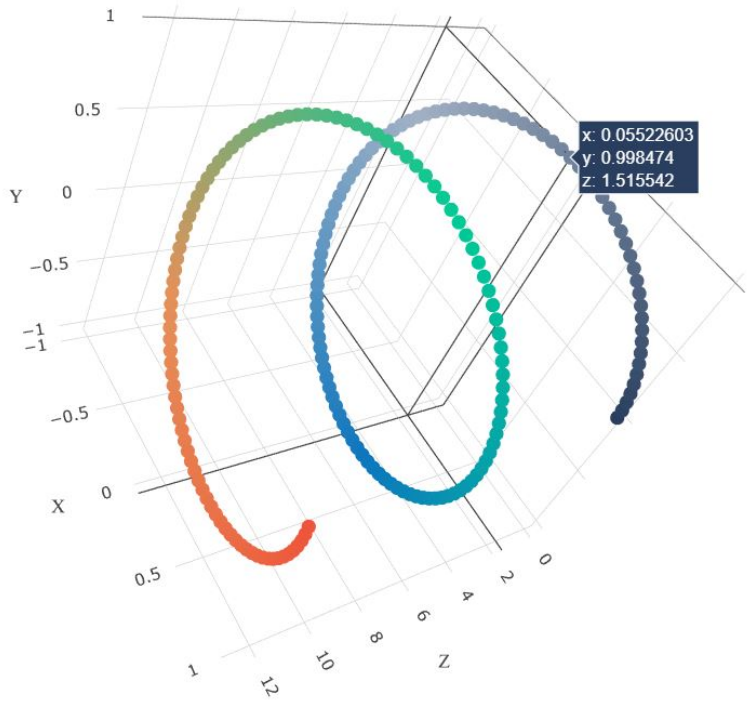
In [3]:

```
'''
# 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-p

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
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFil
e,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine =
line.split()\n        word = splitLine[0]\n        embedding = np.array([float(val) for
val in splitLine[1:]])\n        model[word] = embedding\n    print ("Done.",len(model),"
words loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# =
==========================\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.
69it/s]\nDone. 1917495  words loaded!\n\n# ============================\n\nwords = []\nf
or i in preproced_texts:\n    words.extend(i.split(\' \'))\n\nfor i in preproced_title
s:\n    words.extend(i.split(\' \'))\nprint("all the words in the coupus", len(words))\n
words = 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 bo
th glove vectors and our coupus",       len(inter_words),"(",np.round(len(inter_words)/l
en(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: htt
p://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimpor
t pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)
\n\n\n'
```

Task - 1

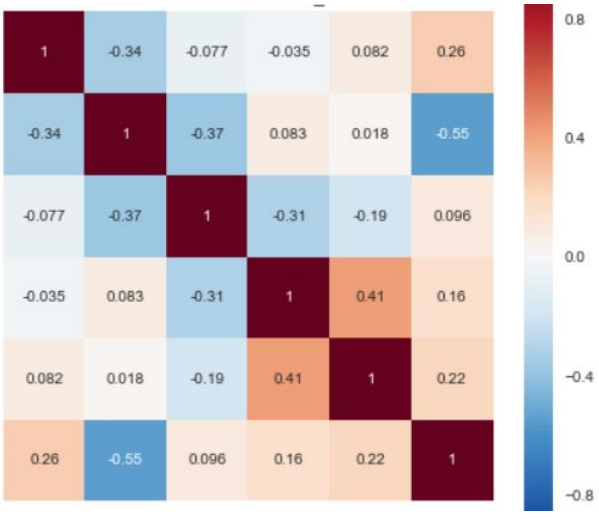
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - **Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)**
    - **Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)**
      ** **
    - **The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_samples_split` in range [5, 10, 100, 500])**
        - **Find the best hyper parameter which will give the maximum AUC value**
        - **find the best hyper paramter using k-fold cross validation(use gridsearch cv or randomsearch cv)/simple cross validation data(you can write your own for loops refer sample solution)**
          ** **
        - **Representation of results**
            - **You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure**



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

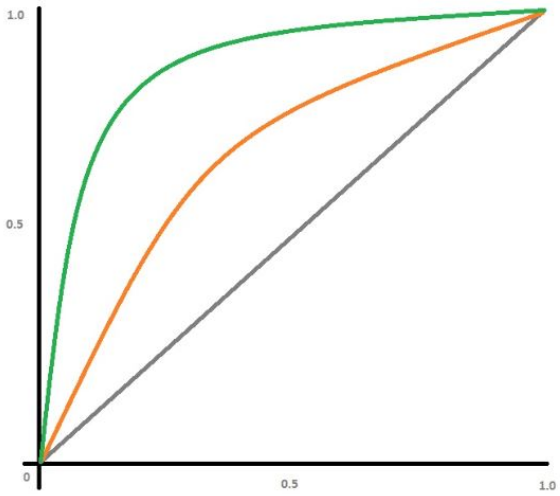
or

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



seaborn heat maps with rows as min_sample_split, columns as max_depth, and values inside the cell representing AUC Score

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



train and test.

- **Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points**

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

- **Once after you plot the confusion matrix with the test data, get all the `false positive data points`**
  - **Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these `false positive data points`**
  - **Plot the box plot with the `price` of these `false positive data points`**
  - **Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`**

** **

Task - 2

For this task consider set-1 features.

- **Select all the features which are having non-zero feature importance.You can get the feature importance using 'feature*importances*` (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html), discard the all other remaining features and then apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM).**
- **You need to do hyperparameter tuning corresponding to the model you selected and procedure in step 2 and step 3**
  Note: when you want to find the feature importance make sure you don't use max_depth

**parameter keep it None. **

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

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

** **

Hint for calculating Sentiment scores

In [2]:
```
import nltk
nltk.download('vader_lexicon')
```

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

Out[2]: True

In [3]:
```
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
for learning my students learn in many different ways using all of our senses and multi
of techniques to help all my students succeed students in my class come from a variety
for wonderful sharing of experiences and cultures including native americans our school
learners which can be seen through collaborative student project based learning in and
in my class love to work with hands on materials and have many different opportunities
mastered having the social skills to work cooperatively with friends is a crucial aspec
montana is the perfect place to learn about agriculture and nutrition my students love
in the early childhood classroom i have had several kids ask me can we try cooking with
and create common core cooking lessons where we learn important math and writing concep
food for snack time my students will have a grounded appreciation for the work that wen
of where the ingredients came from as well as how it is healthy for their bodies this p
nutrition and agricultural cooking recipes by having us peel our own apples to make hom
and mix up healthy plants from our classroom garden in the spring we will also create o
shared with families students will gain math and literature skills as well as a life lo
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,
```

Task - 1

1. Decision Tree

In [4]:
```
#Libraries
%matplotlib inline
```

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

from tqdm import tqdm
import os
```

1.1 Loading Data

In [5]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
```

Splitting Data into Train and Test

In [6]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'],axis =1)
```

In [7]:
```
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 , stratif
```

Encoding

In [8]:
```
#tfidf essay train and test
vectorizer = TfidfVectorizer(min_df = 10)
vectorizer.fit( X_train['essay'].values )
X_train_essay_tfidf = vectorizer.transform( X_train['essay'].values )
X_test_essay_tfidf = vectorizer.transform( X_test['essay'].values )
```

In [8]:
```
# train tfidf w2v using pretrained model

tfidf_model = TfidfVectorizer()
tfidf_model.fit( X_train['essay'].values )
dictionary = dict ( zip (tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

X_train_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in thi
for sentence in tqdm(X_train['essay'].values): # 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((sen
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            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_vectors.append(vector)
```

```
print(len(X_train_tfidf_w2v_vectors))
print(len(X_train_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 73196/
73196 [02:13<00:00, 549.66it/s]
73196
300
```

In [9]:
```
# test tfidf w2v using pretrained model

X_test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(X_test['essay'].values): # 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((sen
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            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_vectors.append(vector)

print(len(X_test_tfidf_w2v_vectors))
print(len(X_test_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 36052/
36052 [01:06<00:00, 543.82it/s]
36052
300
```

In [9]:
```
#sentiment analyser feature for train
sid = SentimentIntensityAnalyzer()

Train_sent_lst = []
for sentence in tqdm(X_train['essay'].values):
    ss = sid.polarity_scores(sentence)
    temp = []
    for k in ss:
        temp.append(ss[k])
    Train_sent_lst.append(temp)
Train_sent_arr = np.array(Train_sent_lst)
print(Train_sent_arr.shape)
```

```
100%|████████████████████████████████████████████████████████████████| 73196/
73196 [01:56<00:00, 629.33it/s]
(73196, 4)
```

In [10]:
```
#sentiment analyser feature for test
sid = SentimentIntensityAnalyzer()

Test_sent_lst = []
for sentence in tqdm(X_test['essay'].values):
    ss = sid.polarity_scores(sentence)
    temp = []
    for k in ss:
        temp.append(ss[k])
```

```
    Test_sent_lst.append(temp)
Test_sent_arr = np.array(Test_sent_lst)
print (Test_sent_arr.shape)
```

```
100%|████████████████████████████████████████████████████████████████████████| 36052/
36052 [00:56<00:00, 641.05it/s]
(36052, 4)
```

In [11]:

```
print ("   One Hot Encoding of feature Teacher Prefix")
vectorizer = CountVectorizer()      #count vectorizer for one hot encoding
vectorizer.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_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].values)   #
X_test_teacher_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_prefix_ohe.shape, y_train.shape)  #checking to ensure both test a
print(X_test_teacher_prefix_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())       #checking feature names
print("="*100)



print ("   One Hot Encoding of Project Grade Category  ")
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on tr

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category_ohe = vectorizer.transform(X_train['project_grade_catego
X_test_project_grade_category_ohe = vectorizer.transform(X_test['project_grade_category



print ("   One Hot Encoding of School State  ")

vectorizer = CountVectorizer()
vectorizer.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.transform(X_train['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)


print ("   One Hot Encoding of cleaned categories  ")

vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train da

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_test_clean_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)


print ("   One Hot Encoding of cleaned sub categories  ")

vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train

# we use the fitted CountVectorizer to convert the text to vector
```

```
X_train_clean_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].v
X_test_clean_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].val
```

```
   One Hot Encoding of feature Teacher Prefix
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
==========================================================================================
============
   One Hot Encoding of Project Grade Category
   One Hot Encoding of School State
   One Hot Encoding of cleaned categories
   One Hot Encoding of cleaned sub categories
```

In [12]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

print(" Normalizing Price feature")

normalizer.fit(X_train['price'].values.reshape(1,-1)) #reshaping to one column from one

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))  #resh
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))

X_train_price_norm = X_train_price_norm.reshape(-1,1) #reshaping back to
X_test_price_norm = X_test_price_norm.reshape(-1,1)

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)  #checking to ensure ouput dimensions ar
print(X_test_price_norm.shape, y_test.shape)
print("="*100)

print(" Normalizing Teacher number of previously posted projects feature")

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1

X_train_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_trai
X_test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_test[

X_train_teacher_number_of_previously_posted_projects_norm =X_train_teacher_number_of_pr
X_test_teacher_number_of_previously_posted_projects_norm = X_test_teacher_number_of_pre
```

```
 Normalizing Price feature
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
==========================================================================================
============
 Normalizing Teacher number of previously posted projects feature
```

In [13]:
```
from scipy.sparse import hstack

X_tr_tfidf = hstack((X_train_essay_tfidf,Train_sent_arr,X_train_teacher_prefix_ohe,X_tr
X_te_tfidf = hstack((X_test_essay_tfidf,Test_sent_arr,X_test_teacher_prefix_ohe,X_test_

print("Final Data matrix tfidf")
print(X_tr_tfidf.shape, y_train.shape)   #final train matrix after horizontally stackin
print(X_te_tfidf.shape, y_test.shape)    #final test matrix after horizontally stacking
print("="*100)
```

```
Final Data matrix tfidf
(73196, 14352) (73196,)
```

```
(36052, 14352) (36052,)
==========================================================================================
============
```

In [16]:
```
X_tr_tfidfw2v = hstack((X_train_tfidf_w2v_vectors,Train_sent_arr,X_train_teacher_prefix
X_te_tfidfw2v = hstack((X_test_tfidf_w2v_vectors,Test_sent_arr,X_test_teacher_prefix_oh

print("Final Data matrix tfidfw2v")
print(X_tr_tfidfw2v.shape, y_train.shape)   #final train matrix after horizontally stac
print(X_te_tfidfw2v.shape, y_test.shape)    #final test matrix after horizontally stack
print("="*100)
```

```
Final Data matrix tfidf
(73196, 14338) (73196,)
(36052, 14338) (36052,)
==========================================================================================
============
Final Data matrix tfidfw2v
(73196, 405) (73196,)
(36052, 405) (36052,)
==========================================================================================
============
```

All the steps for SET - 1 TFIDF

In [18]:
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import math
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

#hyperparameter tuning
dtc = DecisionTreeClassifier(random_state=0)
param_grid = {'max_depth':[1, 5, 10, 50] , 'min_samples_split':[5, 10, 100, 500]}
clf = GridSearchCV(dtc, param_grid, cv=4, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr_tfidf, y_train)
```

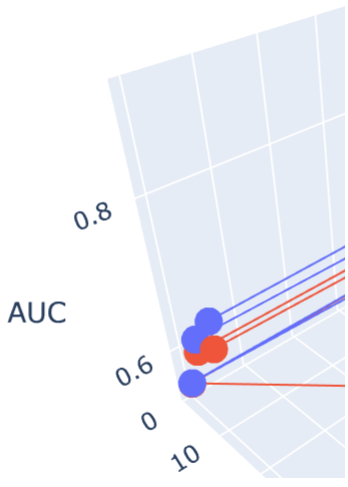
Out[18]:
```
GridSearchCV(cv=4, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc')
```

In [19]:
```
print("Best AUC score : ",clf.best_score_)
print("Best params : ",clf.best_params_)
```

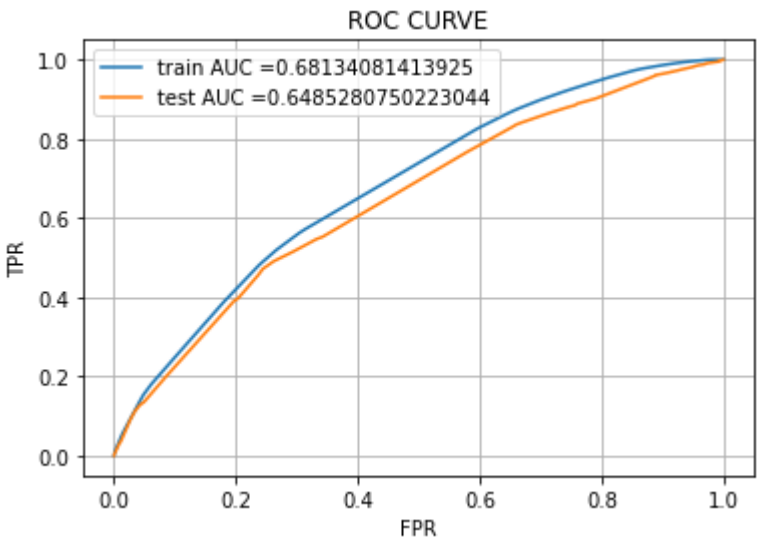
```
Best AUC score :  0.6466249778863526
Best params :  {'max_depth': 10, 'min_samples_split': 500}
```

In [20]:
```
results = pd.DataFrame.from_dict(clf.cv_results_)  #storing results of gridsearch in pa

results = results.sort_values(['rank_test_score'])

train_auc= results['mean_train_score']

cv_auc = results['mean_test_score']
```

```
K =  results['params']

max_d = []
min_sam_splt = []

for i in K:
    max_d.append(i.get('max_depth'))        #max depth values
    min_sam_splt.append(i.get('min_samples_split'))     #min sample split values

#plotting 3D plot as per given ipynb

import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np


x1 = min_sam_splt
y1 = max_d
z1 = train_auc

x2 = min_sam_splt
y2 = max_d
z2 = cv_auc

trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'Train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]

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

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



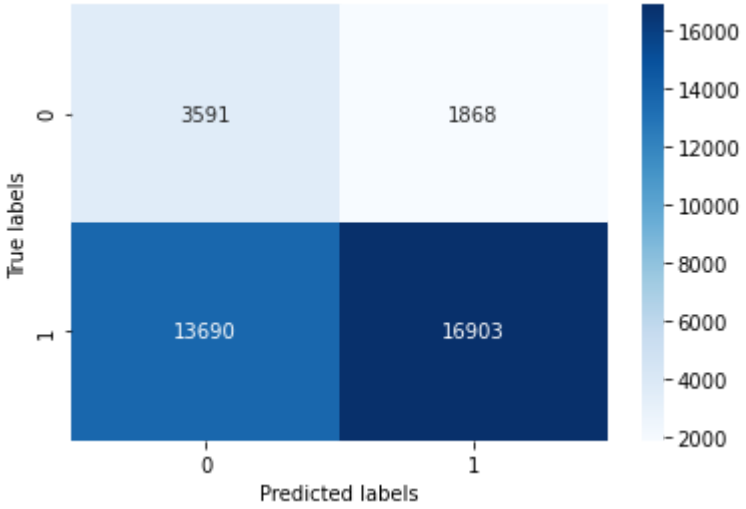
In [21]:
```
from sklearn.metrics import roc_curve, auc

#training on best hyperparameters
best_dt = DecisionTreeClassifier( max_depth = 10, min_samples_split=500 , random_state=
best_dt.fit(X_tr_tfidf, y_train)

#predicted proabilities
y_train_pred = best_dt.predict_proba(X_tr_tfidf)[:,1]  # train predicted probabilities
y_test_pred = best_dt.predict_proba(X_te_tfidf)[:,1]   # test predicted probabilities

#plotting ROC Curve
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)    #train fpr,tr
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)        #test fpr,tes

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))   #p
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 [22]:
```
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.rou
    return t

def predict_with_best_t(proba, threshould):
```

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

    return predictions

print("="*100)

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

#plotting confusion matrix
print("Test confusion matrix")
test_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

ax= plt.subplot()
sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues' , ax =ax)
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
plt.show()
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.39249149946066336 for threshold 0.847
Test confusion matrix
```

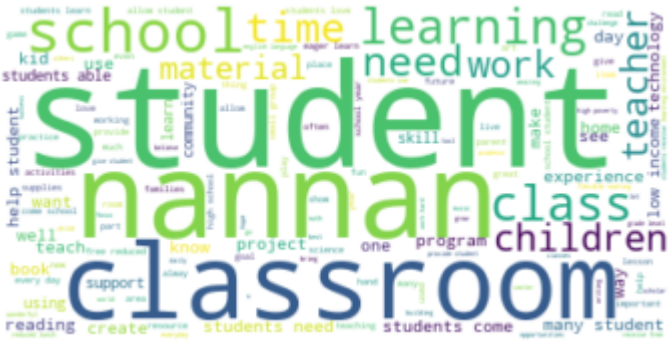


In [23]:

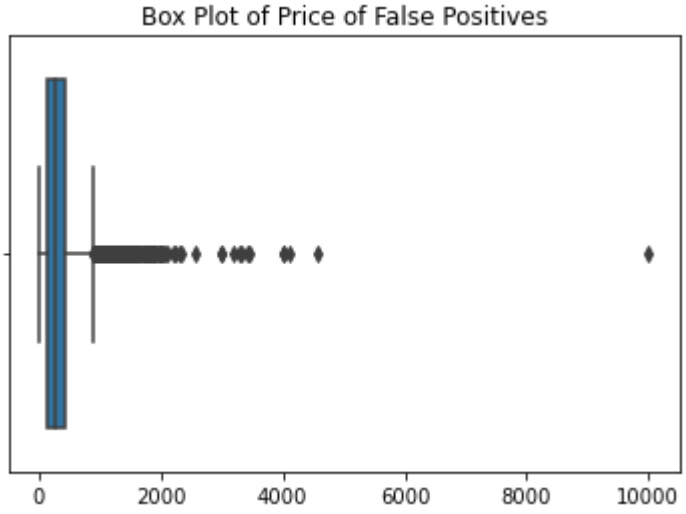
```
y_test_predicted_for_FP = best_dt.predict(X_te_tfidf)


loc_fp = []  #loc of all False positive points
for i in range(len(y_test)):
    if y_test_predicted_for_FP[i]==1 and y_test[i]==0:
        loc_fp.append(i)
#corresponding essay , price ,  teacher_number_of_previously_posted_projects for false

sent = []
for k in loc_fp:
        sent.append(X_test['essay'].values[k])

price = []
for j in loc_fp:
        price.append(X_test['price'].values[j])
```

```
teacher_number_of_previously_posted_projects = []
for l in loc_fp:
        teacher_number_of_previously_posted_projects.append(X_test['teacher_number_of_p
```

In [24]:
```
#https://www.datacamp.com/community/tutorials/wordcloud-python

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

text = " ".join(review for review in sent)
stopwords = set(STOPWORDS)

wordcloud = WordCloud(stopwords=stopwords, background_color="white").generate(text)

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off") #boundary off
plt.show()
```



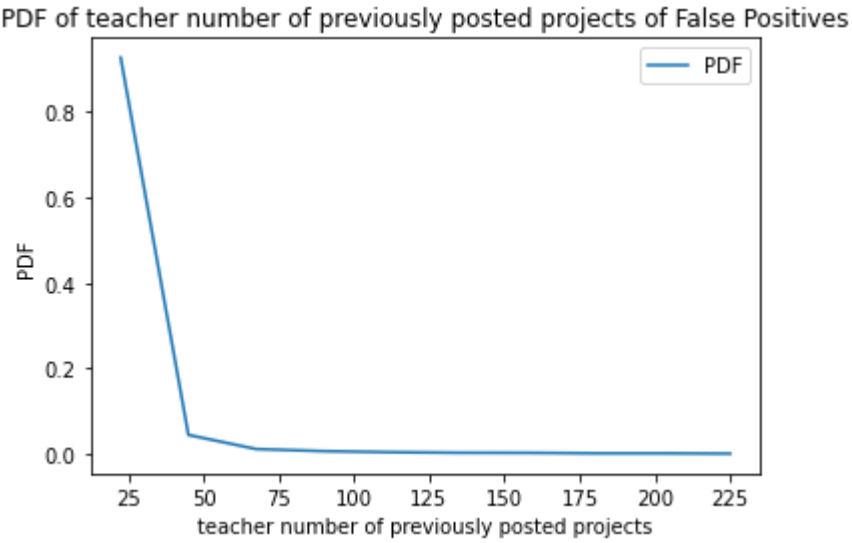
In [25]:
```
# https://seaborn.pydata.org/generated/seaborn.boxplot.html
sns.boxplot(x=price)
plt.title("Box Plot of Price of False Positives")
plt.show()
```



In [26]:
```
counts , bin_edges = np.histogram (teacher_number_of_previously_posted_projects  ,  bin
pdf = counts/(sum(counts))
plt.plot(bin_edges[1:], pdf, label = "PDF")

plt.xlabel("teacher number of previously posted projects")
plt.ylabel("PDF")
```

```
plt.title("PDF of teacher number of previously posted projects of False Positives")
plt.legend()
```

Out[26]: <matplotlib.legend.Legend at 0x187fb7cf910>



All the steps for SET - 2 TFIDF W2V

In [27]:
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import math
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(random_state=0)
param_grid = {'max_depth':[1, 5, 10, 50] , 'min_samples_split':[5, 10, 100, 500]}
clf = GridSearchCV(dtc, param_grid, cv=4, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr_tfidfw2v, y_train)
```

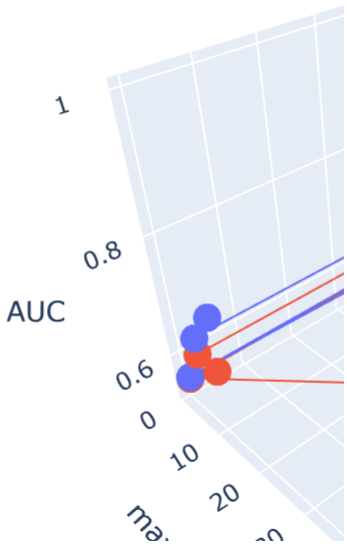
Out[27]:
```
GridSearchCV(cv=4, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc')
```

In [28]:
```
print("Best AUC score : ",clf.best_score_)
print("Best params : ",clf.best_params_)
```

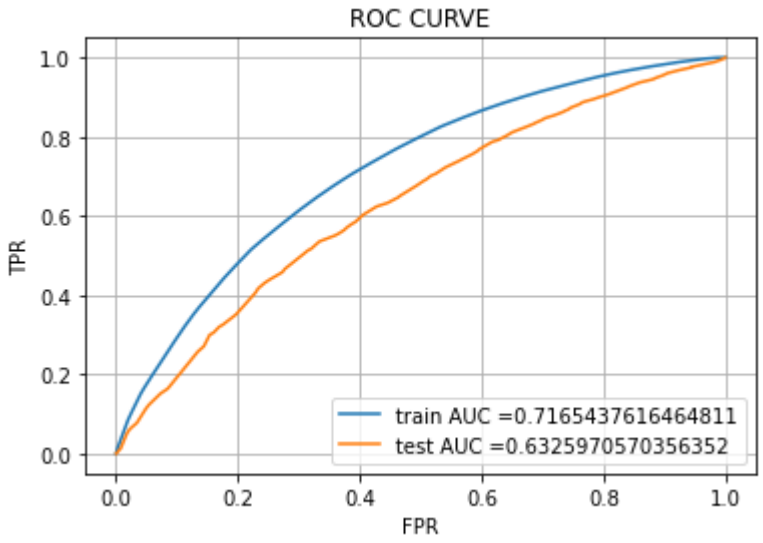
```
Best AUC score :  0.626527495157804
Best params :  {'max_depth': 10, 'min_samples_split': 500}
```

In [29]:
```
results = pd.DataFrame.from_dict(clf.cv_results_)  #storing results of gridsearch in pa

results = results.sort_values(['rank_test_score'])

train_auc= results['mean_train_score']

cv_auc = results['mean_test_score']
```

```
K =  results['params']

max_d = []
min_sam_splt = []

for i in K:
    max_d.append(i.get('max_depth'))
    min_sam_splt.append(i.get('min_samples_split'))

import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np


x1 = min_sam_splt
y1 = max_d
z1 = train_auc

x2 = min_sam_splt
y2 = max_d
z2 = cv_auc

trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'Train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]

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

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



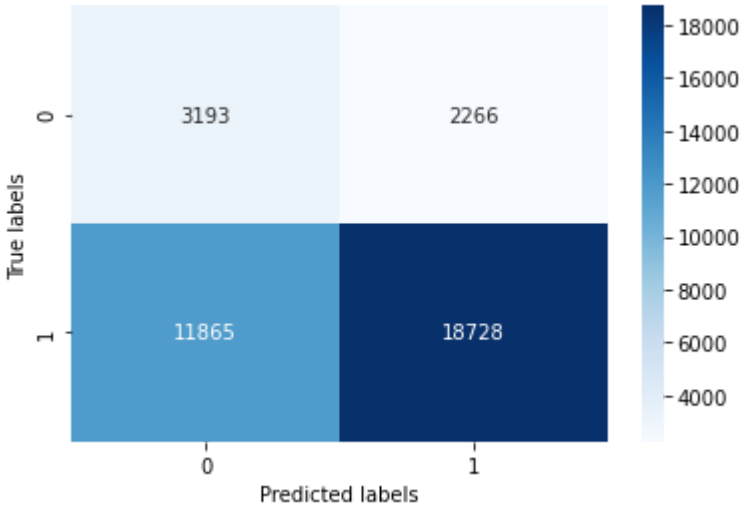
In [30]:
```
from sklearn.metrics import roc_curve, auc


best_dt = DecisionTreeClassifier( max_depth = 10, min_samples_split=500 , random_state=
best_dt.fit(X_tr_tfidfw2v, y_train)

y_train_pred = best_dt.predict_proba(X_tr_tfidfw2v)[:,1]  # train predicted probabiliti
y_test_pred = best_dt.predict_proba(X_te_tfidfw2v)[:,1]   # test predicted probabilitie

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)    #train fpr,tr
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)        #test fpr,tes

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))   #p
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 [31]:
```
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.rou
    return t

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

```
        else:
            predictions.append(0)

    return predictions

print("="*100)

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

print("Test confusion matrix")
test_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

ax= plt.subplot()
sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues' , ax =ax)
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
plt.show()
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.4349609312487936 for threshold 0.854
Test confusion matrix
```



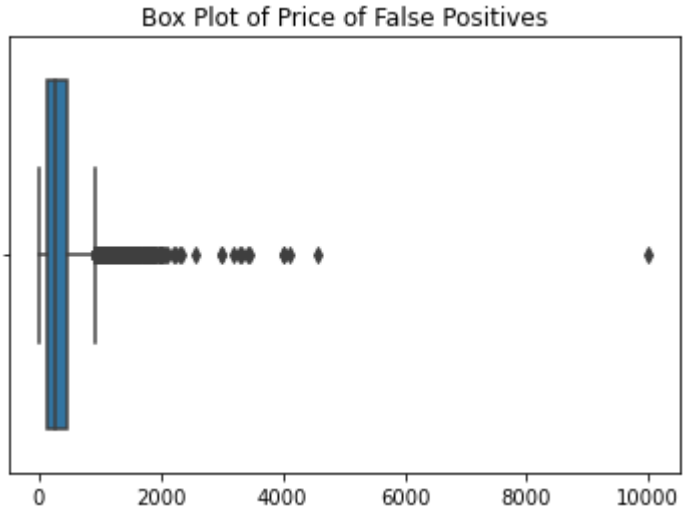
In [32]:

```
# print(X_test,y_test)

y_test_predicted_for_FP = best_dt.predict(X_te_tfidfw2v)


loc_fp = []  #loc of all False positive points
for i in range(len(y_test)):
    if y_test_predicted_for_FP[i]==1 and y_test[i]==0:
        loc_fp.append(i)


sent = []
for k in loc_fp:
        sent.append(X_test['essay'].values[k])

price = []
for j in loc_fp:
        price.append(X_test['price'].values[j])


teacher_number_of_previously_posted_projects = []
```

```
for l in loc_fp:
        teacher_number_of_previously_posted_projects.append(X_test['teacher_number_of_p
```

In [33]:
```
#https://www.datacamp.com/community/tutorials/wordcloud-python

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

text = " ".join(review for review in sent)
stopwords = set(STOPWORDS)

wordcloud = WordCloud(stopwords=stopwords, background_color="white").generate(text)

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off") #boundary off
plt.show()
```

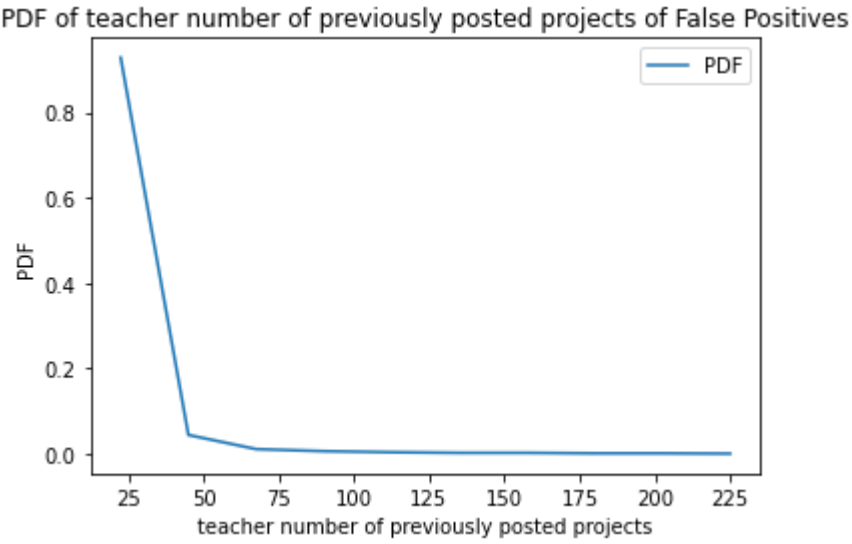
In [34]:
```
# https://seaborn.pydata.org/generated/seaborn.boxplot.html
sns.boxplot(x=price)
plt.title("Box Plot of Price of False Positives")
plt.show()
```



In [35]:
```
counts , bin_edges = np.histogram (teacher_number_of_previously_posted_projects  ,  bin
pdf = counts/(sum(counts))
plt.plot(bin_edges[1:], pdf, label = "PDF")

plt.xlabel("teacher number of previously posted projects")
plt.ylabel("PDF")
plt.title("PDF of teacher number of previously posted projects of False Positives")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x187fb6c1e20>
```

Out[35]:



Task - 2

In [52]:
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import math
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

#training as per set 1
best_dt = DecisionTreeClassifier( max_depth = 10, min_samples_split=500 , random_state=
best_dt.fit(X_tr_tfidf, y_train)
```

Out[52]:
```
DecisionTreeClassifier(max_depth=10, min_samples_split=500, random_state=0)
```

In [53]:
```
#getting index of all non zero feature importances
idx_nonzero = np.nonzero(best_dt.feature_importances_)
print(len(idx_nonzero[0]))
```

```
122
```

In [54]:
```
all_features_train = 0
all_features_train = X_tr_tfidf.toarray()
all_features_train.shape
```

Out[54]:
```
(73196, 14352)
```

In [55]:
```
#keeping all non zero feature importances in train set
nonzero_features_train = []

for i in idx_nonzero[0]:
    nonzero_features_train.append(all_features_train[:,i])
```

```
nonzero_features_train = np.array(nonzero_features_train)
nonzero_features_train = nonzero_features_train.T

all_features_train = 0 #setting to 0 to free up space
```

In [57]:
```
all_features_test = 0
all_features_test = X_te_tfidf.toarray()
all_features_test.shape
```

Out[57]:
```
(36052, 14352)
```

In [58]:
```
#keeping all non zero feature importances in test set
nonzero_features_test = []

for i in idx_nonzero[0]:
    nonzero_features_test.append(all_features_test[:,i])


nonzero_features_test= np.array(nonzero_features_test)
nonzero_features_test = nonzero_features_test.T
all_features_test = 0
```

In [59]:
```
print(nonzero_features_train.shape)
print(nonzero_features_test.shape)
```

```
(73196, 122)
(36052, 122)
```

In [60]:
```
#hyper parameter tuning only keeping non zero features

dtc = DecisionTreeClassifier(random_state=0)
param_grid = {'min_samples_split':[5, 10, 100, 500]}
clf = GridSearchCV(dtc, param_grid, cv=4, scoring='roc_auc',return_train_score=True)
clf.fit(nonzero_features_train, y_train)
```

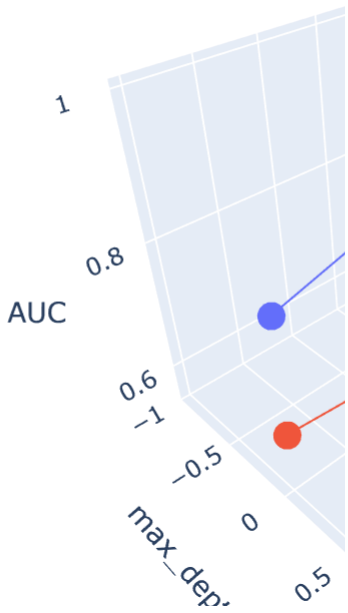
Out[60]:
```
GridSearchCV(cv=4, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc')
```

In [61]:
```
print("Best AUC score : ",clf.best_score_)
print("Best params : ",clf.best_params_)
```

```
Best AUC score :  0.6353070798203522
Best params :  {'min_samples_split': 500}
```

In [62]:
```
#2d plot between min_sample_splits and AUC Score , taken max_depth as 0

results = pd.DataFrame.from_dict(clf.cv_results_)  #storing results of gridsearch in pa

results = results.sort_values(['rank_test_score'])

train_auc= results['mean_train_score']

cv_auc = results['mean_test_score']

K =  results['params']

max_d = []
min_sam_splt = []
```

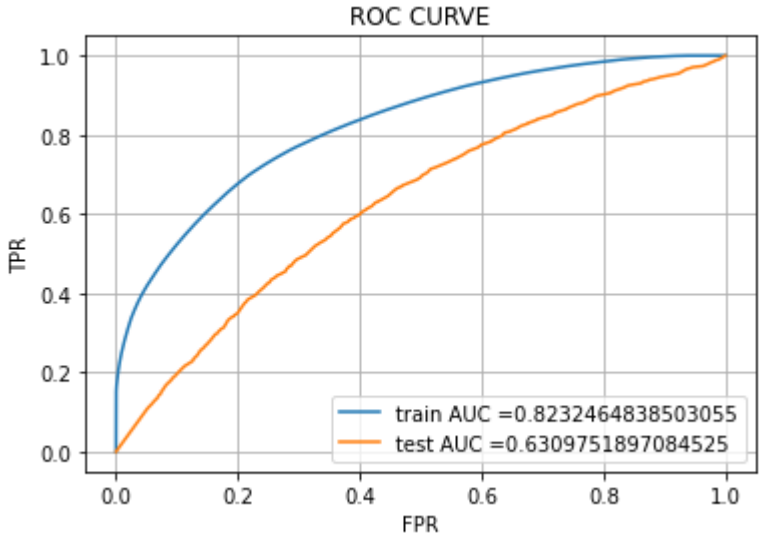
```
for i in K:
    max_d.append(0)
    min_sam_splt.append(i.get('min_samples_split'))

import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np


x1 = min_sam_splt
y1 = max_d
z1 = train_auc

x2 = min_sam_splt
y2 = max_d
z2 = cv_auc

trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'Train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
data = [trace1, trace2]

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

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



In [63]:
```
#training on best min sample split as per grid search
best_dt = DecisionTreeClassifier(  min_samples_split=500 , random_state=0)
best_dt.fit(nonzero_features_train, y_train)
```

Out[63]:
```
DecisionTreeClassifier(min_samples_split=500, random_state=0)
```

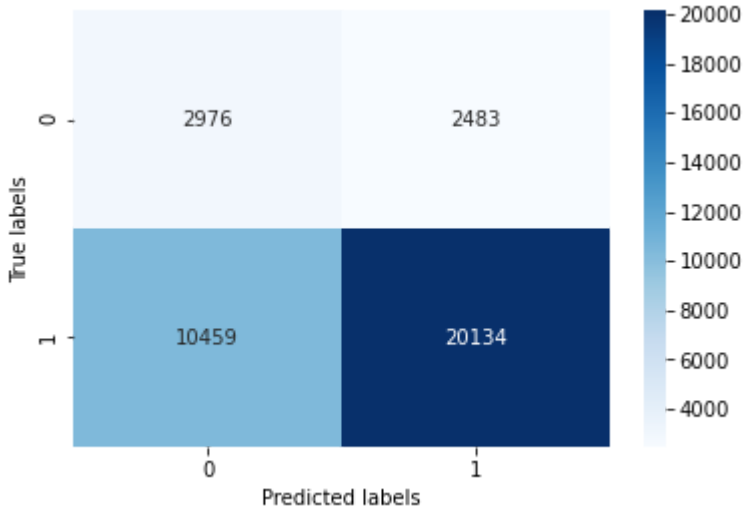
In [64]:
```
from sklearn.metrics import roc_curve, auc

y_train_pred = best_dt.predict_proba(nonzero_features_train)[:,1]  # train predicted pr
y_test_pred = best_dt.predict_proba(nonzero_features_test)[:,1]   # test predicted prob

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)    #train fpr,tr
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)        #test fpr,tes

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))   #p
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 [65]:
```
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.rou
    return t

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

```
    return predictions

print("="*100)

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

print("Test confusion matrix")
test_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

ax= plt.subplot()
sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues' , ax =ax)
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
plt.show()
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.5479968903296909 for threshold 0.843
Test confusion matrix
```



In [66]:

```
# print(X_test,y_test)

y_test_predicted_for_FP = best_dt.predict(nonzero_features_test)


loc_fp = []  #loc of all False positive points
for i in range(len(y_test)):
    if y_test_predicted_for_FP[i]==1 and y_test[i]==0:
        loc_fp.append(i)


sent = []
for k in loc_fp:
        sent.append(X_test['essay'].values[k])

price = []
for j in loc_fp:
        price.append(X_test['price'].values[j])


teacher_number_of_previously_posted_projects = []
```

```
for l in loc_fp:
        teacher_number_of_previously_posted_projects.append(X_test['teacher_number_of_p
```

In [67]:

```
#https://www.datacamp.com/community/tutorials/wordcloud-python

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

text = " ".join(review for review in sent)
stopwords = set(STOPWORDS)

wordcloud = WordCloud(stopwords=stopwords, background_color="white").generate(text)

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off") #boundary off
plt.show()
```