Problem statement

- The goal of this Kaggle competition(our project) is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school.

Decision Tree

In [42]:

```
import pandas as pd
import pickle
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import math
import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import pickle
from tqdm import tqdm
import os
from collections import Counter
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
import numpy as np
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.preprocessing import MinMaxScaler
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
from tqdm import tqdm
from wordcloud import WordCloud
from sklearn.metrics import roc_curve, auc
from  sklearn.linear_model import LogisticRegression
```

Task-1

Featurising data

In [44]:

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

Loading Data

In [45]:

```
#make sure you are loading atleast 50k datapoints
#you can work with features of preprocessed_data.csv for the assignment.
import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows=50000)
```

In [47]:

```
df_data=data.copy()      #copyting original dataframe
```

In [48]:

```
df_data.head(1)    #copy of original dataframe
```

Out[48]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_subcat
0	ca	mrs	grades_prek_2	53	1	math_science	applieds health_life

In [49]:

```
# find the sentiment score for easy using sentiment intensity analyzer

df_polarity=pd.DataFrame(columns=["negative","neutral","positive","compound"])    #how to enter values rowwise in dataframe: #https://stac
sid = SentimentIntensityAnalyzer()  #initialising sentiment intensity analyzer

for idx,row in tqdm(enumerate(data["essay"])):# for  essay in each project
    ss_1 = sid.polarity_scores(row)     #finding polarity score
    polarity_features=list(ss_1.values())
    df_polarity.loc[idx] = (polarity_features[0] , polarity_features[1] , polarity_features[2],polarity_features[3]) #making new datafram
```

```
50000it [02:42, 307.74it/s]
```

In [50]:

```
df_polarity.head(5)    #dataframe with polarity score
```

Out[50]:

	negative	neutral	positive	compound
0	0.013	0.783	0.205	0.9867
1	0.072	0.680	0.248	0.9897
2	0.017	0.721	0.262	0.9860
3	0.030	0.783	0.187	0.9524
4	0.029	0.683	0.288	0.9873

In [51]:

```
#merging polarity df with data_df
data = pd.merge(df_data,df_polarity,left_index=True, right_index=True, how='left')
```

In [52]:

```
data.head(2)     #after merging
```

Out[52]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_subcat
0	ca	mrs	grades_prek_2	53	1	math_science	applieds health_life
1	ut	ms	grades_3_5	4	1	specialneeds	speci

In [53]:

```
len(data.columns)   #after two dataframe get merged we get 13 columns
```

Out[53]:

```
13
```

Splitting data

In [54]:

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

Out[54]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	essay	p
0	ca	mrs	grades_prek_2	53	math_science	appliedsciences health_lifescience	i fortunate enough use fairy tale stem kits cl...	72

In [55]:

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

In [56]:

```
'''with open('X_train.pickle', 'wb') as f:
    pickle.dump(X_train, f)

with open('X_test.pickle', 'wb') as f:
    pickle.dump(X_test, f)

with open('y_train.pickle', 'wb') as f:
    pickle.dump(y_train, f)

with open('y_test.pickle', 'wb') as f:
    pickle.dump(y_test, f)'''
```

Vectorizing categorical featues

In [57]:

```
#featurising school_state categorical feature

vectorizer = CountVectorizer(binary=True)
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("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 51) (33500,)
(16500, 51) (16500,)
====================================================================================================
```

In [58]:

```
#featurising teacher_prefix categorical feature

vectorizer = CountVectorizer(binary=True)
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_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 5) (33500,)
(16500, 5) (16500,)
====================================================================================================
```

In [59]:

```
#featurising project_grade_category categorical features

vectorizer = CountVectorizer(binary=True)
X_train_grade_ohe=vectorizer.fit_transform(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_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)


print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
#print(vectorizer.get_feature_names())             #feature names of 4 dim vector
print("="*100)
```

```
After vectorizations
(33500, 4) (33500,)
(16500, 4) (16500,)
====================================================================================================
```

In [60]:

```
#featurising clean_categories categorical features

vectorizer = CountVectorizer(binary=True)
X_train_category_ohe=vectorizer.fit_transform(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_test_category_ohe = vectorizer.transform(X_test['clean_categories'].values)


print("After vectorizations")
print(X_train_category_ohe.shape, y_train.shape)
print(X_test_category_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 9) (33500,)
(16500, 9) (16500,)
====================================================================================================
```

In [61]:

```
#featurising clean_subcategories categorical features

vectorizer = CountVectorizer(binary=True)
X_train_subcategory_ohe=vectorizer.fit_transform(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_test_subcategory_ohe = vectorizer.transform(X_test['clean_categories'].values)


print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 30) (33500,)
(16500, 30) (16500,)
====================================================================================================
```

Vectorizing Numberical featues

In [62]:

```
#featurising price - numerical features


scaler = MinMaxScaler()       #doing minmaxscaling to numerical feature(price)
#reshape(-1, 1)--so that minmaxscaling applied on price column(feature column)
X_train_price_norm=scaler.fit_transform(X_train['price'].values.reshape(-1, 1)) #fitting train data
X_test_price_norm = scaler.transform(X_test['price'].values.reshape(-1, 1))     #converting test data using fitted minmaxscaler


print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

In [63]:

```
#featurising teacher_number_of_previously_posted_projects - numerical features

scaler = MinMaxScaler()
#reshape(-1, 1)--so that minmaxscaling applied on teacher_number_of_previously_posted_projects column(feature column)
X_train_previous_projects_norm = scaler.fit_transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
X_test_previous_projects_norm = scaler.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))


print("After vectorizations")
print(X_train_previous_projects_norm.shape, y_train.shape)
print(X_test_previous_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

Tfidf-vectorization

In [64]:

```
#tfidf vectorization

vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=8000)                #TFIDF vectorizer

X_train_essay_tfidf = vectorizer.fit_transform(X_train['essay'].values) #fitted only the train data
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values) # we use the fitted TfidfVectorizer to convert the test text to vector


print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)  #size of train and test vector
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 8000) (33500,)
(16500, 8000) (16500,)
====================================================================================================
```

Combining all the featuers of set1

In [65]:

```
from scipy.sparse import hstack
#all the necessary feature for set1 is stacked together horizontally
#stacked train features
X_tr_tfidf = hstack((X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_category_ohe, X_train_subcategory_ohe,X_train_pric
#stacked test features
X_te_tfidf = hstack((X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,X_test_price_norm


print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 8105) (33500,)
(16500, 8105) (16500,)
====================================================================================================
```

Tfidf weighted w2v

calculate idf value with only train data

In [66]:

```
preprocessed_essays_train = X_train['essay'].values    #using X_train essay to find idf_value of words

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())
```

creating tfidf wighted w2v vectors for train data

In [67]:

```
# average Word2Vec
# compute average word2vec for each essay in train data.

tfidf_w2v_vectors_train = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for  each essay
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the each essay
    for word in sentence.split(): # for each word in a essay
        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 tfidf 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%|██████████████████████████████████████████████████████████████████████████| 33500/33500 [01:00<00:00, 556.46it/s]

33500
300
```

Creating tfidf wighted w2v vectors for test data

In [68]:

```
# average Word2Vec
# compute average word2vec for each essay in test data.
preprocessed_essays_test = X_test['essay'].values

tfidf_w2v_vectors_test = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(preprocessed_essays_test): # for each essay
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        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 tfidf 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%|██████████████████████████████████████████████████████████████████████████| 16500/16500 [00:29<00:00, 558.68it/s]

16500
300
```

Combining all the features of set2

In [69]:

```
from scipy.sparse import hstack
#all the necessary feature for set2 is stacked together horizontally
#stacked train features
X_tr_tfidf_w2v = hstack((X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_category_ohe, X_train_subcategory_ohe,X_train_
#stacked test features
X_te_tfidf_w2v = hstack((X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,X_test_price_

print("Final Data matrix")
print(X_tr_tfidf_w2v.shape, y_train.shape)
print(X_te_tfidf_w2v.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 405) (33500,)
(16500, 405) (16500,)
====================================================================================================
```

In [70]:

```
'''with open('X_tr_tfidf_w2v.pickle', 'wb') as f:
    pickle.dump(X_tr_tfidf_w2v, f)

with open('X_te_tfidf_w2v.pickle', 'wb') as f:
    pickle.dump(X_te_tfidf_w2v, f)

with open('X_tr_tfidf.pickle', 'wb') as f:
    pickle.dump(X_tr_tfidf, f)

with open('X_te_tfidf.pickle', 'wb') as f:
    pickle.dump(X_te_tfidf, f)'''
```

In [71]:

```
'''with open('X_tr_tfidf_w2v.pickle', 'rb') as f:
    X_tr_tfidf_w2v=pickle.load(f)

with open('X_te_tfidf_w2v.pickle', 'rb') as f:
    X_te_tfidf_w2v=pickle.load(f)

with open('X_tr_tfidf.pickle', 'rb') as f:
    X_tr_tfidf=pickle.load(f)

with open('X_te_tfidf.pickle', 'rb') as f:
    X_te_tfidf=pickle.load(f)'''
```

Out[71]:

```
"with open('X_tr_tfidf_w2v.pickle', 'rb') as f:\n    X_tr_tfidf_w2v=pickle.load(f)\n    \nwith open('X_te_tfidf_w2v.pickl
e', 'rb') as f:\n    X_te_tfidf_w2v=pickle.load(f)\n\nwith open('X_tr_tfidf.pickle', 'rb') as f:\n    X_tr_tfidf=pickle.loa
d(f)\n\nwith open('X_te_tfidf.pickle', 'rb') as f:\n    X_te_tfidf=pickle.load(f)"
```

Set1-Hyperparameter tuning

In [72]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier


DT1 = DecisionTreeClassifier(random_state=42)  #decision tree classifier

parameters = {'max_depth': [1, 3, 10, 30],'min_samples_split':[5, 10, 100, 500]}      #hyper parameter list for gridsearch
clf = GridSearchCV(DT1, parameters, cv=3, scoring='roc_auc',return_train_score=True)            #applying gridsearch to find best hyperpa
clf.fit(X_tr_tfidf, y_train)                                #fitting the DT model with train data

results = pd.DataFrame.from_dict(clf.cv_results_)           #storing Gridsearch results
print(results)


train_auc= results['mean_train_score']#storing required Gridsearch results in required variable
cv_auc = results['mean_test_score']
param_max_depth  =  results['param_max_depth'].tolist()
param_min_samples_split  = results['param_min_samples_split'].tolist()
```

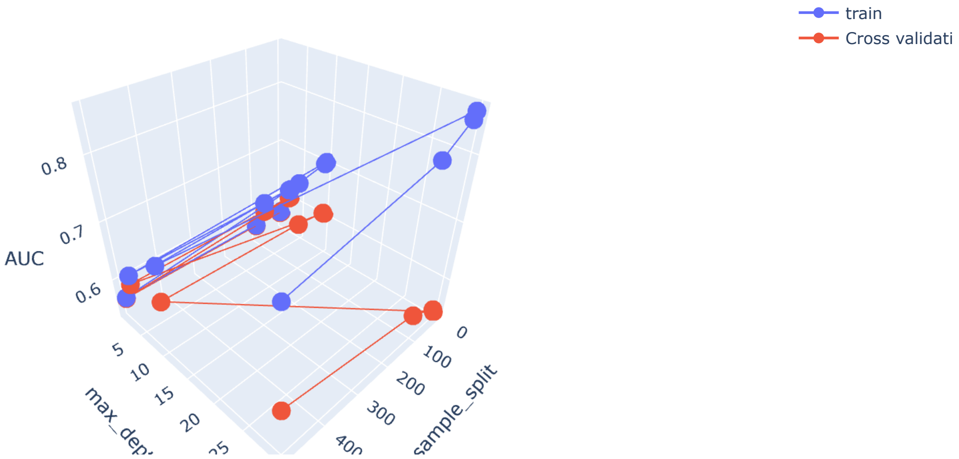
```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0        0.435998      0.005889         0.014344        0.000473   
1        0.431657      0.004922         0.014674        0.000952   
2        0.431356      0.004127         0.014669        0.000473   
3        0.429326      0.002352         0.014341        0.000480   
4        1.146088      0.005861         0.014254        0.001274   
5        1.199376      0.063374         0.017000        0.003543   
6        1.158660      0.018069         0.014014        0.000818   
7        1.147327      0.008743         0.014001        0.000816   
8        4.972050      0.033745         0.015007        0.000009   
9        4.867980      0.050178         0.014337        0.000477   
10       4.233403      0.035882         0.014041        0.000056   
11       3.810146      0.133725         0.014363        0.000452   
12      16.380135      0.465631         0.014639        0.000531   
13      19.343961      1.134951         0.017717        0.000511   
14      14.813069      1.049074         0.015330        0.000474   
15      11.219415      0.613333         0.016111        0.000697   

   param_max_depth param_min_samples_split  \
```

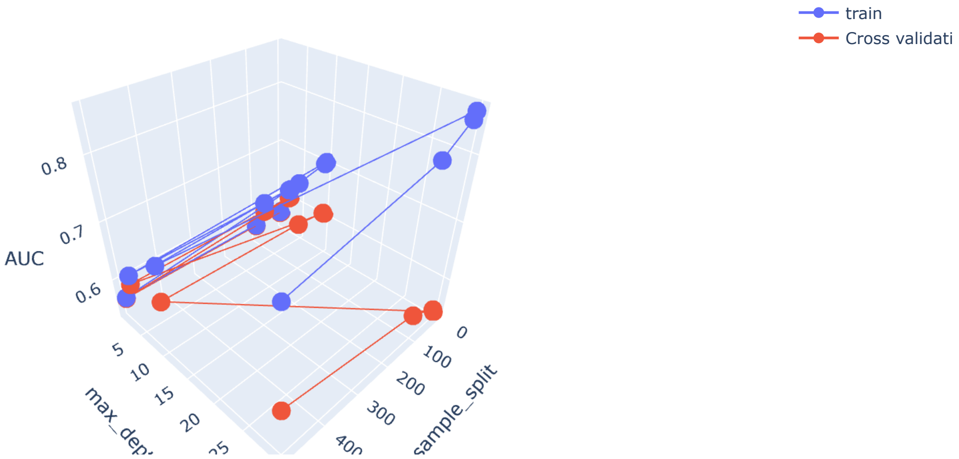
Set1-Representation of results

Set1 - plotting auc score vs hyperparameters

In [73]:

```
#plotting auc score vs hyperparameter using plotly library

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=param_min_samples_split,y=param_max_depth,z=train_auc, name = 'train')          #train data
trace2 = go.Scatter3d(x=param_min_samples_split,y=param_max_depth,z=cv_auc, name = 'Cross validation')  #cv data
data = [trace1, trace2]

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

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





Set1 - Heatmap

For train data

In [74]:

```
#how to draw three variable heatmap: https://stackoverflow.com/a/39042065/17345549
df_train_hyperparameter = pd.DataFrame({'param_min_samples_split': param_min_samples_split, 'param_max_depth': param_max_depth, 'train_auc
df_train_hyper_pivoted = df_train_hyperparameter.pivot("param_min_samples_split", "param_max_depth", "train_auc") #pivoting based on our r
```

In [75]:

```
df_train_hyperparameter.head(5)
```

Out[75]:

	param_min_samples_split	param_max_depth	train_auc
0	5	1	0.56651
1	10	1	0.56651
2	100	1	0.56651
3	500	1	0.56651
4	5	3	0.62148

In [76]:

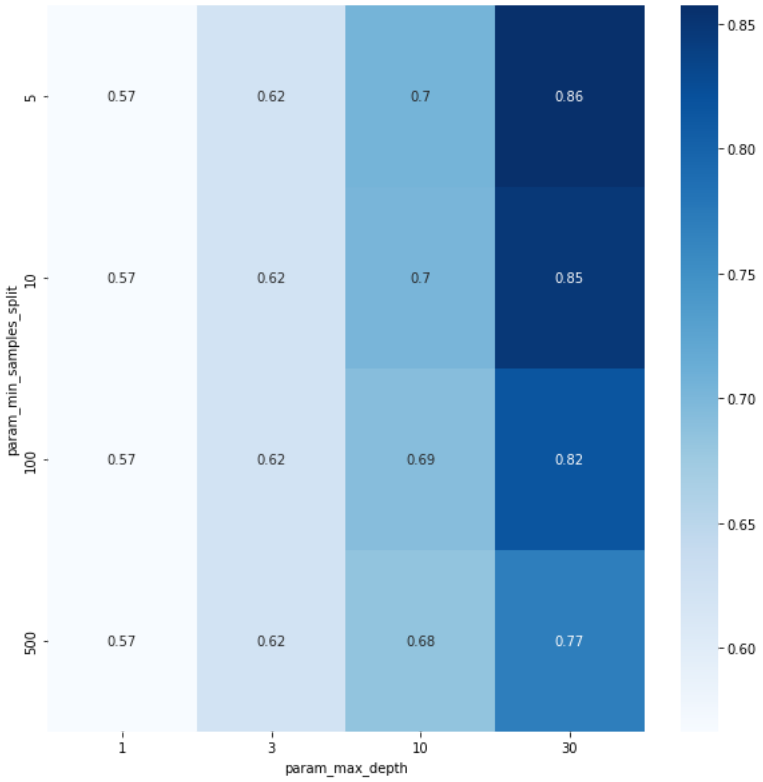
```
df_train_hyper_pivoted
```

Out[76]:

param_max_depth	1	3	10	30
param_min_samples_split				
5	0.56651	0.62148	0.704276	0.857315
10	0.56651	0.62148	0.702509	0.847442
100	0.56651	0.62148	0.692305	0.816417
500	0.56651	0.62148	0.683837	0.770594

In [77]:

```
#plotting heatmap with x axis as parameter max depth , y axis as param min number of smaples splits and has auc value in inside cell

plt.figure(figsize = (10,10))  #how to draw correlation heatmap(3 values): https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlati
dataplot = sns.heatmap(df_train_hyper_pivoted, cmap="Blues", annot=True)
```



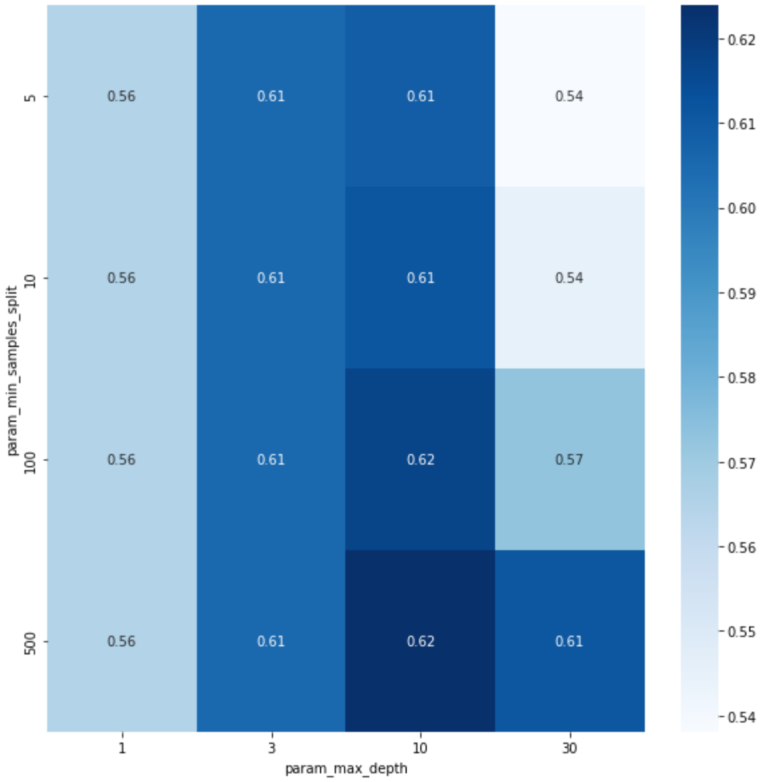
For test data

In [78]:

```
#how to draw three variable heatmap: https://stackoverflow.com/a/39042065/17345549
df_cv_hyper = pd.DataFrame({'param_min_samples_split': param_min_samples_split, 'param_max_depth': param_max_depth, 'cv_auc': cv_auc})
df_cv_hyper_pivoted = df_cv_hyper.pivot("param_min_samples_split", "param_max_depth", "cv_auc")
```

In [79]:

```
plt.figure(figsize = (10,10))  #how to draw correlation heatmap: https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlation-heatmap
dataplot = sns.heatmap(df_cv_hyper_pivoted, cmap="Blues", annot=True)
```

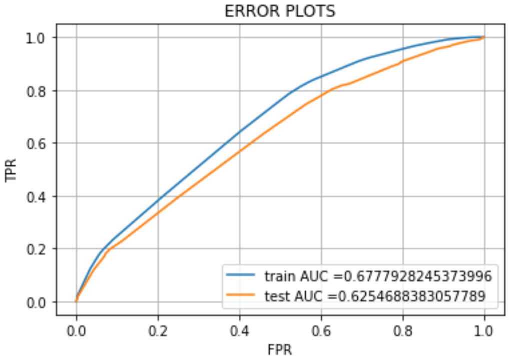


set 1 - Fitting data with Best model

From the above 3d plot and heatmap we can see that the mean test auc is maximum and the gap between mean train auc and mean test auc is minimum when min sample split=500 and max_depth=10

In [80]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

DT1 = DecisionTreeClassifier(random_state=42,min_samples_split=500,max_depth=10)
DT1.fit(X_tr_tfidf, y_train)     #fitting the DT model with best alpha
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class,not the predicted value

y_train_pred_proba = DT1.predict_proba(X_tr_tfidf)
y_test_pred_proba = DT1.predict_proba(X_te_tfidf)

#how to get a perticular column in nd array:https://stackoverflow.com/a/8386737/17345549
#[:,[1]].reshape(1,-1)[0]---to take probability values for the positive class

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_proba[:,[1]].reshape(1,-1)[0]) #find FPR and TPR for plotting roc cu
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_proba[:,[1]].reshape(1,-1)[0])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))) #finding area under the ROC curve using FPR and 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("ERROR PLOTS")
plt.grid()
plt.show()
```



In [81]:

```
with open('DT1.pickle', 'wb') as f:
    pickle.dump(DT1, f)
```

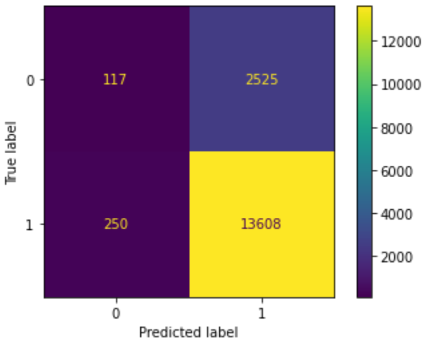
Set1-plotting confusion matrix

In [82]:

```
plot_confusion_matrix(DT1, X_te_tfidf, y_test)
```

Out[82]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x206b0dade50>
```



set1- Getting false positive datapoint

We got 4898 False positive datapoints. So, now we are going to get those points into a new dataframe.

In [83]:

```
#to calculate row index of false positive datapoints
y_pred=DT1.predict(X_te_tfidf)     #predict for test data

fp_row = []    #store FP datapoint's index
for i in range(len(y_test)):                    #finding false positive datapoints: https://datascience.stackexchange.com/a/97501
    if y_pred[i] == 1 and y_test[i] == 0:
        fp_row.append(i)
```

In [85]:

```
len(fp_row)   #same as the number of FP datapoints
```

Out[85]:

```
2525
```

In [88]:

```
df_tem=X_test.copy()   #copying dataframe and reseting its index
df_tem.reset_index(inplace = True)
```

In [91]:

```
df_fp=df_tem.iloc[fp_row]  #creating dataframe with only FP datapoints using FP datapoints index list
```

In [92]:

```
df_fp
```

Out[92]:

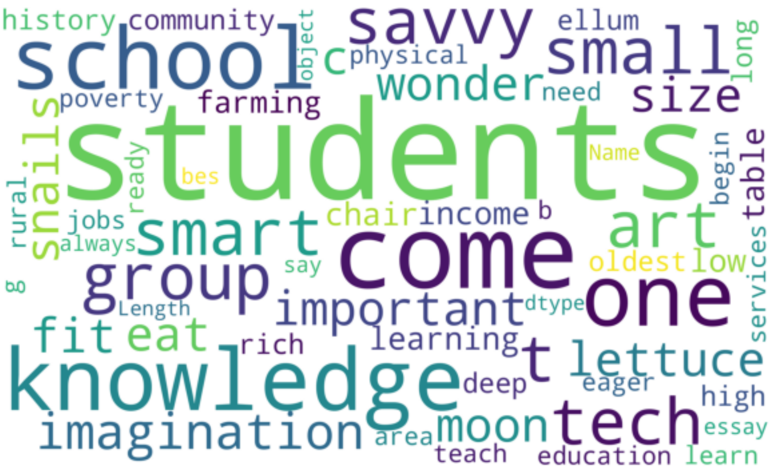
	index	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories
4	9916	wi	mrs	grades_3_5	1	history_civics	economics
13	16179	tx	mrs	grades_prek_2	0	literacy_language	esl literacy
17	41496	ut	ms	grades_prek_2	8	math_science	environmentalscience health_lifescience
18	1629	tx	mrs	grades_3_5	0	literacy_language	literature_writing
27	45511	mo	ms	grades_9_12	10	health_sports	gym_fitness teamsports
...	...	...	...	...	...	...	...
16485	15354	tx	mr	grades_6_8	0	music_arts	visualarts
16490	11782	ms	ms	grades_9_12	1	math_science	mathematics
16492	19137	tx	mrs	grades_prek_2	0	literacy_language	literacy
16495	8200	ny	ms	grades_9_12	8	health_sports music_arts	health_wellness performingarts
16496	5772	in	mrs	grades_prek_2	5	literacy_language	literature_writing

2525 rows × 13 columns

Word cloud for FP datapoints's essay

In [93]:

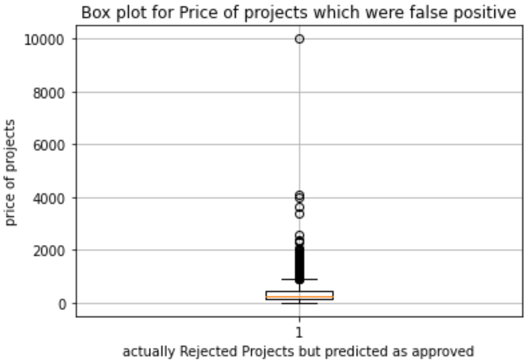
```
# Python program to generate WordCloud: #https://www.geeksforgeeks.org/generating-word-cloud-python/

wordcloud = WordCloud(width = 2500, height = 1500,
                background_color ='white',
                min_font_size = 5).generate(str(df_fp["essay"])) #creating word cloud for FP datapoins's essay

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Box plot for Price of projects which were false positive

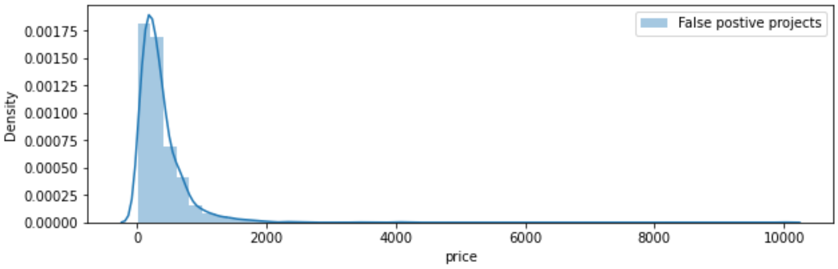
In [94]:

```
plt.boxplot(df_fp["price"])    #boxplot for price
plt.title('Box plot for Price of projects which were false positive')
plt.xlabel('actually Rejected Projects but predicted as approved')
plt.ylabel('price of projects')
plt.grid()
plt.show()
```



In [95]:

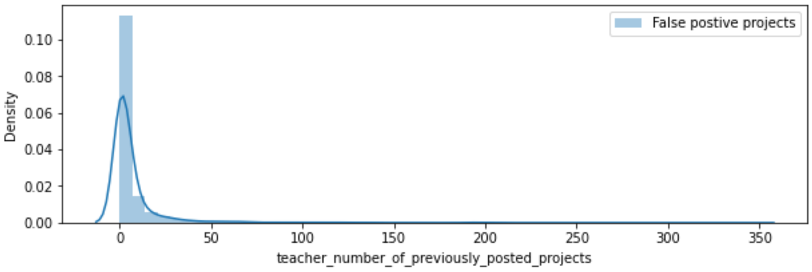
```
plt.figure(figsize=(10,3))      #PDF for price
sns.distplot(df_fp["price"], hist=True, label="False postive projects")
plt.legend()
plt.show()
```



PDF numbre of previously_posted_projects by teacher

In [96]:

```
plt.figure(figsize=(10,3))      #PDF for number of previously posted projects by teacher
sns.distplot(df_fp["teacher_number_of_previously_posted_projects"], hist=True, label="False postive projects")
plt.legend()
plt.show()
```



Eventhough number of previously posted projects by teachers are very less since the project's cost is reasonably low and the essay contains words like students,poverty,farming,rural,knowledge,important,ect are appeared more number of times,so we can interprete that the model predicted those projects as positive even thought they are actually not..

Set2-Hyperparameter tuning

In [97]:

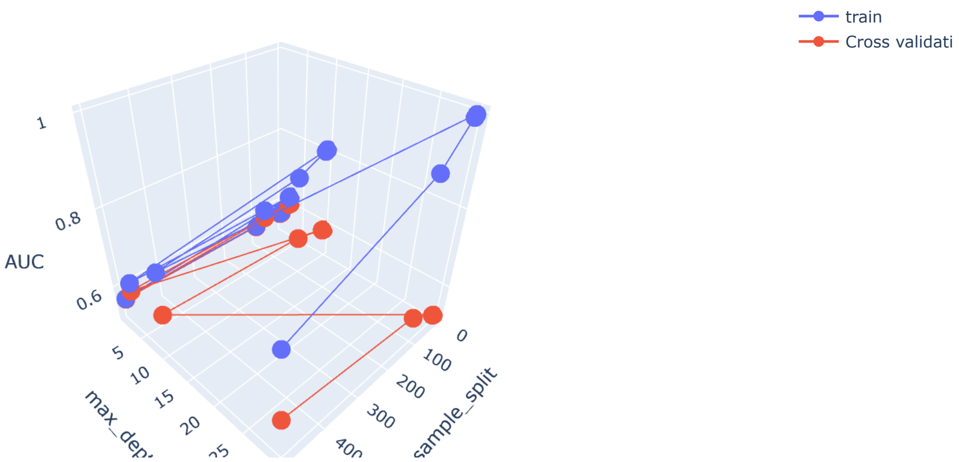
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html


DT2 = DecisionTreeClassifier(random_state=52)      #different random state for different set of features

parameters = {'max_depth': [1, 3, 10, 30],'min_samples_split':[5, 10, 100, 500]}     #hyper parameter list for gridsearch
clf = GridSearchCV(DT2, parameters, cv=3, scoring='roc_auc',return_train_score=True)            #applying gridsearch to find best hyperpa
clf.fit(X_tr_tfidf_w2v, y_train)                                   #fitting the DT model with train data

results = pd.DataFrame.from_dict(clf.cv_results_)          #storing Gridsearch results
print(results)


train_auc= results['mean_train_score']#storing required Gridsearch results in required variable
cv_auc = results['mean_test_score']
param_max_depth  =  results['param_max_depth'].tolist()
param_min_samples_split  = results['param_min_samples_split'].tolist()
```

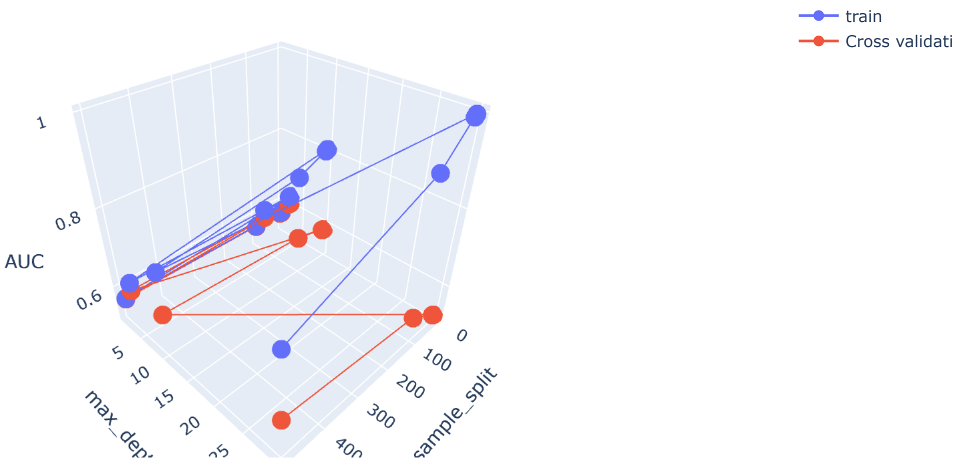
```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0        0.933336      0.001714         0.023010        0.000810   
1        0.918503      0.004300         0.022009        0.000812   
2        0.928995      0.009896         0.022004        0.000806   
3        0.928323      0.006849         0.021679        0.000488   
4        2.505204      0.003818         0.023679        0.002495   
5        2.505912      0.007793         0.021007        0.000814   
6        2.501756      0.005827         0.020654        0.000468   
7        2.498961      0.006084         0.020669        0.000473   
8       12.469136      0.138374         0.022155        0.000763   
9       12.300381      0.113104         0.021675        0.000461   
10      11.332923      0.067916         0.022314        0.001522   
11       8.217366      0.069131         0.020968        0.000228   
12      35.806861      0.188428         0.022943        0.000555   
13      34.982509      0.515647         0.022691        0.000959   
14      30.531073      0.645983         0.022667        0.001695   
15      14.688079      0.070887         0.023339        0.001134   

   param_max_depth param_min_samples_split  \
```

Set2 - plotting auc score vs hyperparameters

In [98]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=param_min_samples_split,y=param_max_depth,z=train_auc, name = 'train')
trace2 = go.Scatter3d(x=param_min_samples_split,y=param_max_depth,z=cv_auc, name = 'Cross validation')
data = [trace1, trace2]

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

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





Set2 - Heatmap

for train data

In [99]:

```
#how to draw three variable heatmap: https://stackoverflow.com/a/39042065/17345549
df_train_hyper = pd.DataFrame({'param_min_samples_split': param_min_samples_split, 'param_max_depth': param_max_depth, 'train_auc': train_
df_train_hyper_pivoted = df_train_hyper.pivot("param_min_samples_split", "param_max_depth", "train_auc")
```

In [100]:

```
df_train_hyper
```

Out[100]:

	param_min_samples_split	param_max_depth	train_auc
0	5	1	0.566510
1	10	1	0.566510
2	100	1	0.566510
3	500	1	0.566510
4	5	3	0.630771
5	10	3	0.630771
6	100	3	0.630771
7	500	3	0.630771
8	5	10	0.808170
9	10	10	0.805441
10	100	10	0.771366
11	500	10	0.728381
12	5	30	0.997286
13	10	30	0.992950
14	100	30	0.916752
15	500	30	0.778341

In [101]:

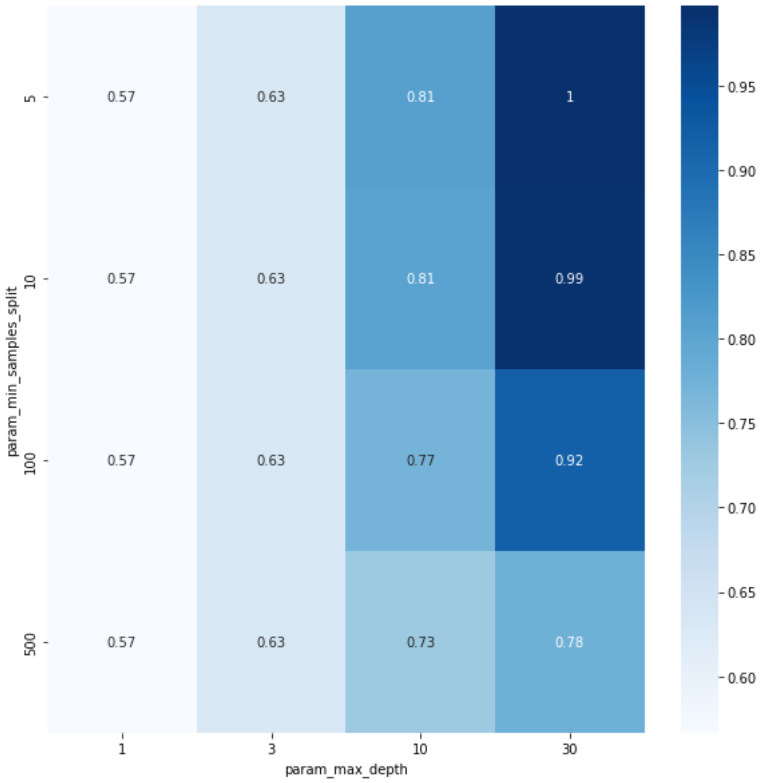
```
df_train_hyper_pivoted
```

Out[101]:

param_max_depth	1	3	10	30
param_min_samples_split				
5	0.56651	0.630771	0.808170	0.997286
10	0.56651	0.630771	0.805441	0.992950
100	0.56651	0.630771	0.771366	0.916752
500	0.56651	0.630771	0.728381	0.778341

In [102]:

```
plt.figure(figsize = (10,10))  #how to draw correlation heatmap: https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlation-heatmap
dataplot = sns.heatmap(df_train_hyper_pivoted, cmap="Blues", annot=True)
```



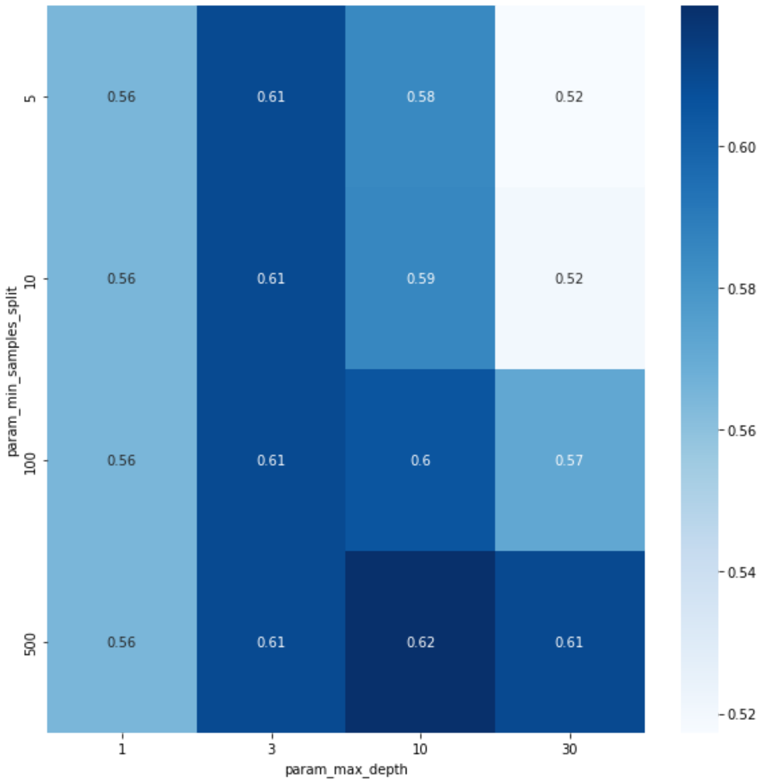
for cross validation data

In [103]:

```
#how to draw three variable heatmap: https://stackoverflow.com/a/39042065/17345549
df_cv_hyper = pd.DataFrame({'param_min_samples_split': param_min_samples_split, 'param_max_depth': param_max_depth, 'cv_auc': cv_auc})
df_cv_hyper_pivoted = df_cv_hyper.pivot("param_min_samples_split", "param_max_depth", "cv_auc")
```

In [104]:

```
plt.figure(figsize = (10,10))  #how to draw correlation heatmap: https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlation-heatmap
dataplot = sns.heatmap(df_cv_hyper_pivoted, cmap="Blues", annot=True)
```

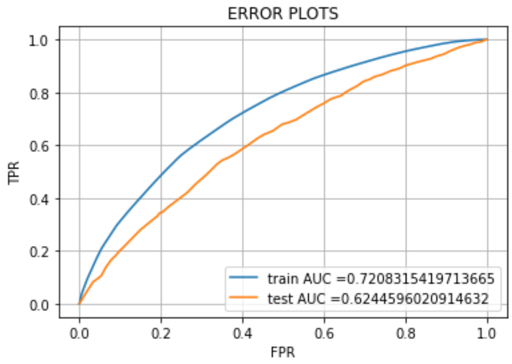


set 2 - Best model

- From the above 3d plot and heatmap we can see that the mean test auc is maximum and the gap between mean train auc and mean test auc is minimum when min sample split=500 and max_depth=10

In [105]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

DT2 = DecisionTreeClassifier(random_state=52,min_samples_split=500,max_depth=10)
DT2.fit(X_tr_tfidf_w2v, y_train)     #fitting the DT model with parameter
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class,not the predicted value

y_train_pred_proba = DT2.predict_proba(X_tr_tfidf_w2v)
y_test_pred_proba = DT2.predict_proba(X_te_tfidf_w2v)

#how to get a perticular column in nd array:https://stackoverflow.com/a/8386737/17345549
#[:,[1]].reshape(1,-1)[0]---to take probability values for the positive class

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_proba[:,[1]].reshape(1,-1)[0]) #find FPR and TPR for plotting roc c
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_proba[:,[1]].reshape(1,-1)[0])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))) #finding area under the ROC curve using FPR and 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("ERROR PLOTS")
plt.grid()
plt.show()
```



In [106]:

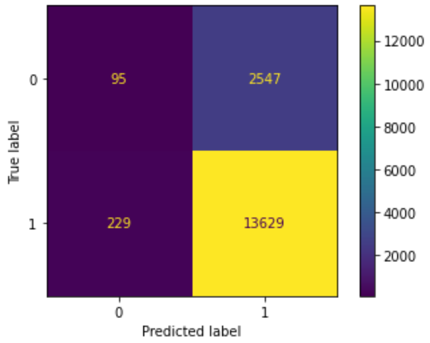
```
'''with open('DT2.pickle', 'wb') as f:
    pickle.dump(DT2, f)'''
```

In [108]:

```
plot_confusion_matrix(DT2, X_te_tfidf_w2v, y_test)
```

Out[108]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2068a5d9160>
```



Getting false positive datapoint

In [109]:

```
#to calculate row index of false positive datapoints
y_pred=DT2.predict(X_te_tfidf_w2v)

fp_row = []    #store FP datapoint's index
for i in range(len(y_test)):                      #finding false positive datapoints: https://datascience.stackexchange.com/a/97501
    if y_pred[i] == 1 and y_test[i] == 0:
        fp_row.append(i)
```

In [110]:

```
len(fp_row) #same as number of false positive datapoint
```

Out[110]:

```
2547
```

In [112]:

```
df_tem=X_test.copy()     #copying dataframe and reset index
df_tem.reset_index(inplace = True)
```

In [114]:

```
df_fp=df_tem.loc[fp_row]        #creatinf dataframe with only FP data points
```

In [115]:

```
df_fp
```

Out[115]:

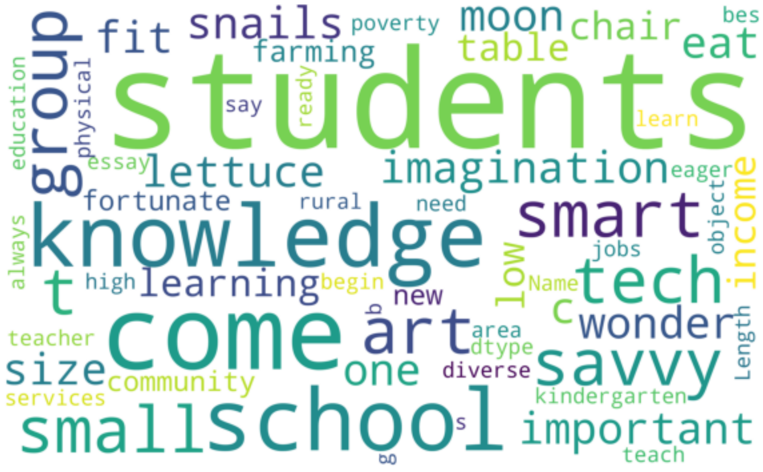
	index	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories
4	9916	wi	mrs	grades_3_5	1	history_civics	economics
13	16179	tx	mrs	grades_prek_2	0	literacy_language	esl literacy
17	41496	ut	ms	grades_prek_2	8	math_science	environmentalscience health_lifescience
18	1629	tx	mrs	grades_3_5	0	literacy_language	literature_writing
27	45511	mo	ms	grades_9_12	10	health_sports	gym_fitness teamsports
...	...	...	...	...	...	...	...
16479	26898	il	ms	grades_prek_2	3	literacy_language	esl literacy
16490	11782	ms	ms	grades_9_12	1	math_science	mathematics
16492	19137	tx	mrs	grades_prek_2	0	literacy_language	literacy
16495	8200	ny	ms	grades_9_12	8	health_sports music_arts	health_wellness performingarts
16496	5772	in	mrs	grades_prek_2	5	literacy_language	literature_writing

2547 rows × 13 columns

Word cloud for FP datapoints

In [116]:

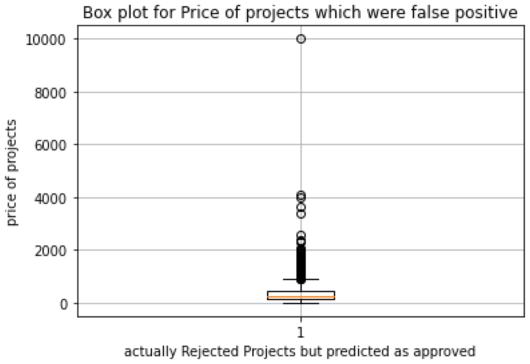
```
# Python program to generate WordCloud         #https://www.geeksforgeeks.org/generating-word-cloud-python/

wordcloud = WordCloud(width = 2500, height = 1500,
                background_color ='white',
                min_font_size = 5).generate(str(df_fp["essay"]))

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Box plot and PDF for " Price of projects" which were false positive

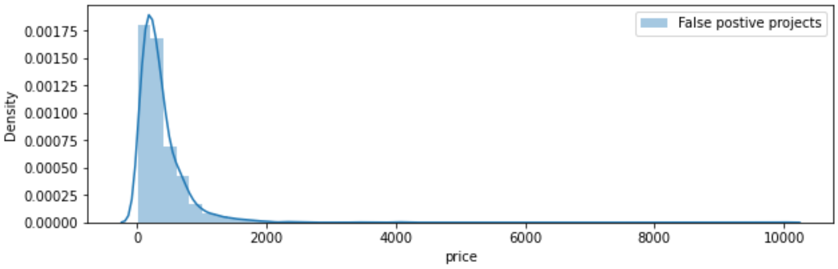
In [117]:

```
plt.boxplot([df_fp["price"]]) #boxplot for price
plt.title('Box plot for Price of projects which were false positive')
plt.xlabel('actually Rejected Projects but predicted as approved')
plt.ylabel('price of projects')
plt.grid()
plt.show()
```



In [118]:

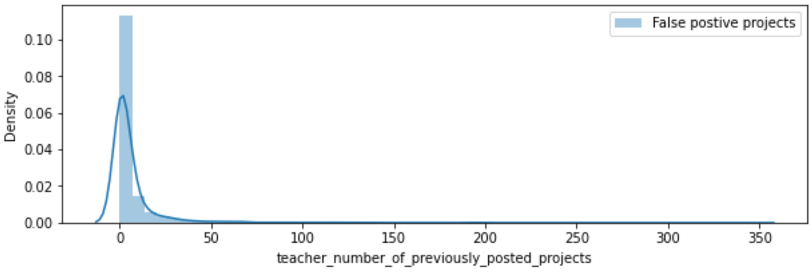
```
plt.figure(figsize=(10,3))     #PDF for price
sns.distplot(df_fp["price"], hist=True, label="False postive projects")
#sns.distplot(rejected_word_count, hist=False, label="Not Approved Projects")
plt.legend()
plt.show()
```



PDF for number of previously_posted_projects by teacher

In [119]:

```
plt.figure(figsize=(10,3))    #PDF for number of previoursly posted projects by teacher
sns.distplot(df_fp["teacher_number_of_previously_posted_projects"], hist=True, label="False postive projects")
plt.legend()
plt.show()
```



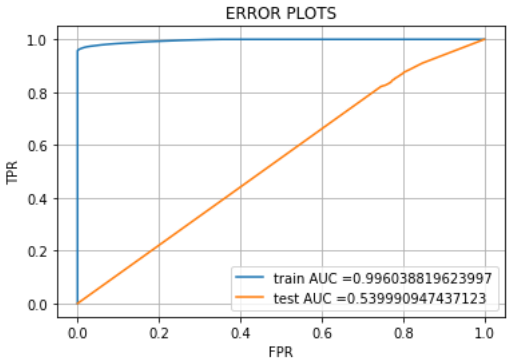
Eventhough number of previously posted projects are very less since the project's cost is reasonably low and the essay contains words like students,knowledge,tech,rural,farming,poverty,ect more number of times,so we can interprete that the model predicted those projects as positive even thought they are actually not..

Task - 2

Training DT with "max_depth=None"

In [123]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

DT3 = DecisionTreeClassifier(random_state=62,min_samples_split=10,max_depth=None)        #no max_depth parameter
DT3.fit(X_tr_tfidf, y_train)     #fitting the DT model with best alpha
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class,not the predicted value

y_train_pred_proba = DT3.predict_proba(X_tr_tfidf)
y_test_pred_proba = DT3.predict_proba(X_te_tfidf)

#how to get a perticular column in nd array:https://stackoverflow.com/a/8386737/17345549
#[:,[1]].reshape(1,-1)[0]---to take probability values for the positive class

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_proba[:,[1]].reshape(1,-1)[0]) #find FPR and TPR for plotting roc cu
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_proba[:,[1]].reshape(1,-1)[0])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))) #finding area under the ROC curve using FPR and 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("ERROR PLOTS")
plt.grid()
plt.show()
```



In [124]:

```
'''with open('DT3.pickle', 'wb') as f:
    pickle.dump(DT3, f)'''
```

In [4]:

```
'''with open('DT3.pickle', 'rb') as f:
    DT3=pickle.load(f)'''
```

In [125]:

```
important_features=DT3.feature_importances_  #get feature importance from model
```

In [126]:

```
np.count_nonzero(important_features)       #number of useful features out of 5105
```

Out[126]:

```
1656
```

In [128]:

```
X_tr_tfidf_dense=X_tr_tfidf.todense()  #converting csr represent of train vector into dense vector
```

In [129]:

```
#to store list of indices of feature with zero feature importance

list_of_zero_indices=[]
for idx in range(len(important_features)):
    if important_features[idx]==0:
        list_of_zero_indices.append(idx)

X_tr_tfidf_important = np.delete(X_tr_tfidf_dense, list_of_zero_indices, axis=1) #remove zero feature important features form x_train vect

X_te_tfidf_dense=X_te_tfidf.todense()
X_te_tfidf_important = np.delete(X_te_tfidf_dense, list_of_zero_indices, axis=1)  #remove zero feature important features form x_test vect
```

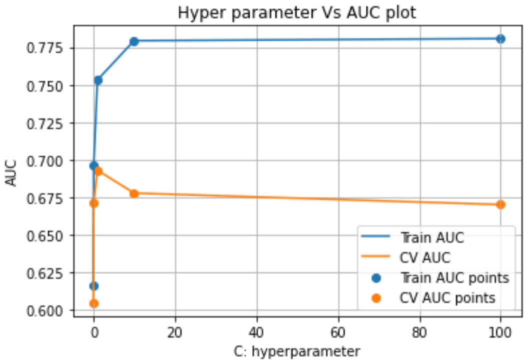
Hyperparameter tuning

Logistic regression

In [133]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

lr=LogisticRegression()
parameters={"C":[100, 10, 1.0, 0.1, 0.01],"penalty":['l2']} #hyperparameter list

clf = GridSearchCV(lr, parameters, cv=3, scoring='roc_auc',return_train_score=True) #applying gridsearch to find best hyperparameter
clf.fit(X_tr_tfidf_important, y_train)    #fitting the NB model with train data

results = pd.DataFrame.from_dict(clf.cv_results_)   #storing Gridsearch results



train_auc= results['mean_train_score']       #storing required Gridsearch results in required variable
cv_auc = results['mean_test_score']
C =  results['param_C'].tolist()
print(C)

plt.plot(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')

plt.scatter(C, train_auc, label='Train AUC points')
plt.scatter(C, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```

```
[100, 10, 1.0, 0.1, 0.01]
```



Out[133]:

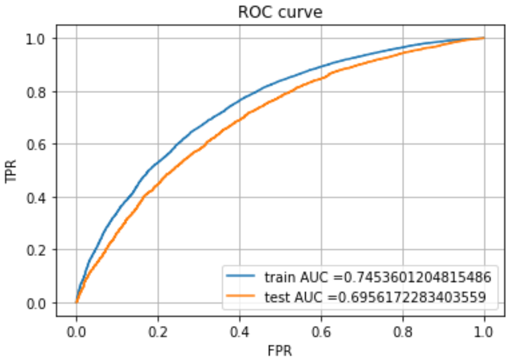
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_penalty	params	split0_test_score	split1_test_score	split2_test_score
0	4.618489	0.319633	0.040665	0.003307	100	l2	{'C': 100, 'penalty': 'l2'}	0.665574	0.661644	0.682782
1	4.368215	0.026454	0.040301	0.004042	10	l2	{'C': 10, 'penalty': 'l2'}	0.672123	0.669328	0.691547
2	4.323001	0.055905	0.039652	0.003783	1.0	l2	{'C': 1.0, 'penalty': 'l2'}	0.681426	0.686419	0.710813
3	3.299255	0.060674	0.036647	0.000462	0.1	l2	{'C': 0.1, 'penalty': 'l2'}	0.654492	0.665448	0.695025
4	1.523195	0.044557	0.037308	0.000479	0.01	l2	{'C': 0.01, 'penalty': 'l2'}	0.587993	0.605743	0.619911

Best model

-We can see that best hyperparameters are C=1 and penality=l2

In [134]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve


lr=LogisticRegression(C=1,penalty='l2')

lr.fit(X_tr_tfidf_important, y_train)              #fitting the NB model with best alpha
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class,not the predicted outputs

y_train_pred_proba = lr.predict_proba(X_tr_tfidf_important)
y_test_pred_proba = lr.predict_proba(X_te_tfidf_important)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_proba[:,[1]])       #find FPR and TPR for plotting roc curve
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_proba[:,[1]])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))   #finding area under the ROC curve using FPR and 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()
```



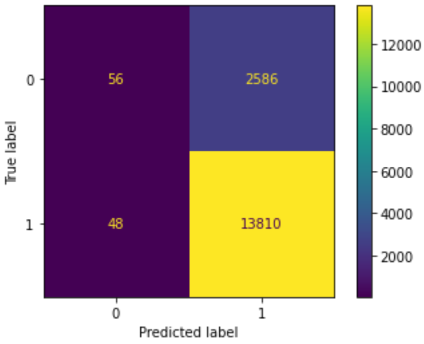
Confustion matrix

In [135]:

```
plot_confusion_matrix(lr, X_te_tfidf_important, y_test)
```

Out[135]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2047d066eb0>
```



Finding false positive datapoints

In [136]:

```
y_pred=lr.predict(X_te_tfidf_important) #predict for test data
```

In [137]:

```
#appending indices of False positive datapoint to list

fp_row = []
for i in range(len(y_test)):                    #https://datascience.stackexchange.com/a/97501
    if y_pred[i] == 1 and y_test[i] == 0:
        fp_row.append(i)
```

In [138]:

```
len(fp_row) #same as number of FP datapoints
```

Out[138]:

```
2586
```

In [141]:

```
df_tem=X_test.copy()        #copying df and resetting index
df_tem.reset_index(inplace = True)
```

In [143]:

```
df_fp=df_tem.loc[fp_row]      #getting dataframe with FP datapoints
```

In [144]:

```
df_fp
```

Out[144]:

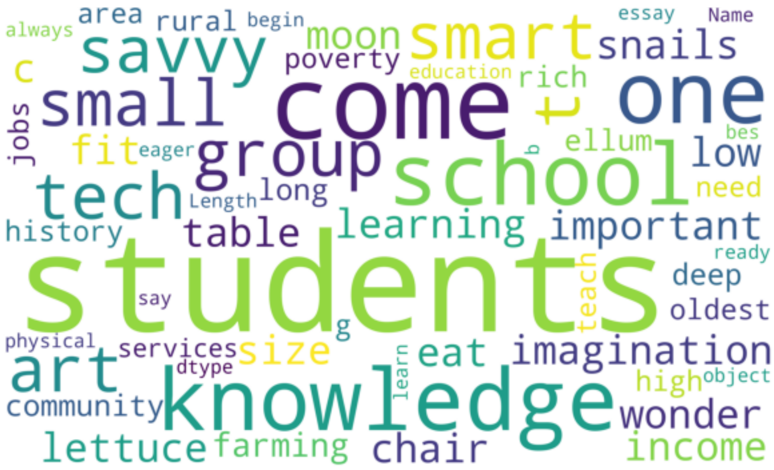
	index	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories
4	9916	wi	mrs	grades_3_5	1	history_civics	economics
13	16179	tx	mrs	grades_prek_2	0	literacy_language	esl literacy
17	41496	ut	ms	grades_prek_2	8	math_science	environmentalscience health_lifescience
18	1629	tx	mrs	grades_3_5	0	literacy_language	literature_writing
27	45511	mo	ms	grades_9_12	10	health_sports	gym_fitness teamsports
...	...	...	...	...	...	...	...
16485	15354	tx	mr	grades_6_8	0	music_arts	visualarts
16490	11782	ms	ms	grades_9_12	1	math_science	mathematics
16492	19137	tx	mrs	grades_prek_2	0	literacy_language	literacy
16495	8200	ny	ms	grades_9_12	8	health_sports music_arts	health_wellness performingarts
16496	5772	in	mrs	grades_prek_2	5	literacy_language	literature_writing

2586 rows × 13 columns

Word cloud

In [145]:

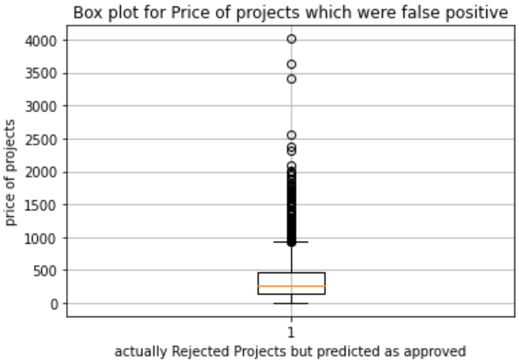
```
# Python program to generate WordCloud         #https://www.geeksforgeeks.org/generating-word-cloud-python/

wordcloud = WordCloud(width = 2500, height = 1500,
                background_color ='white',
                min_font_size = 5).generate(str(df_fp["essay"]))

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



Box plot and PDF plot for price

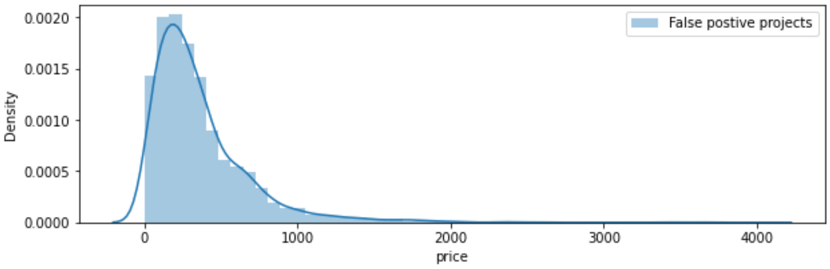
In [146]:

```
#boxplot of price
plt.boxplot([df_fp["price"]])
plt.title('Box plot for Price of projects which were false positive')
plt.xlabel('actually Rejected Projects but predicted as approved')
plt.ylabel('price of projects')
plt.grid()
plt.show()
```



In [147]:

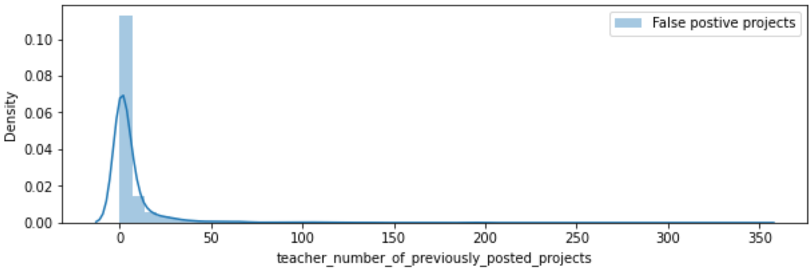
```
#PDF of number of previously posted projects by teacher
plt.figure(figsize=(10,3))
sns.distplot(df_fp["price"], hist=True, label="False postive projects")
plt.legend()
plt.show()
```



PDF of number of previously posted project by given teacher

In [148]:

```
#PDF of number of previously posted projects by teacher
plt.figure(figsize=(10,3))
sns.distplot(df_fp["teacher_number_of_previously_posted_projects"], hist=True, label="False postive projects")
plt.legend()
plt.show()
```



Eventhough number of previously posted projects are very less since the project's cost is reasonably low and the essay contains words like students,knowledge,smart,farming,income,art,important,learning,ect more number of times,so we can interprete that the model predicted those projects as positive even thought they are actually not..

Summary

In [159]:

```
from prettytable import PrettyTable             # Reference Link for Pretty table:  https://pypi.org/project/prettytable/
x = PrettyTable()
```

In [160]:

```
x.field_names = ["Vectorizer","Model", "hyperparamerter", "train_AUC","test_AUC"]
x.add_row(["TFIDF","DT", "MSS=500 and MD=10",0.67779,0.62546])
x.add_row(["TFIDF weighted W2V","DT", "MSS=500 and MD=10", 0.72083,0.62445])
x.add_row(["TFIDF Features with Non zero feature important score ","logistic regression", "C=1", 0.74536,0.69561])
```

In [161]:

```
print(x)
```

```
+--------------------------------------------------------+---------------------+-------------------+-----------+----------+
|                       Vectorizer                       |        Model        |  hyperparamerter  | train_AUC | test_AUC |
+--------------------------------------------------------+---------------------+-------------------+-----------+----------+
|                         TFIDF                          |          DT         | MSS=500 and MD=10 |  0.67779  | 0.62546  |
|                   TFIDF weighted W2V                   |          DT         | MSS=500 and MD=10 |  0.72083  | 0.62445  |
| TFIDF Features with Non zero feature important score  | logistic regression |        C=1        |  0.74536  | 0.69561  |
+--------------------------------------------------------+---------------------+-------------------+-----------+----------+
```

Here in Hyperparameter column - MSS means "min sample split" and MD means "Max depth"

- After removing zero feature important score features from featurised vector and fit the logistic regression model with the new vector, the models perfomance is higher than Decision trees