Assignment : DT

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 [11]:

```
from google.colab import drive
drive.mount('/gdrive')
```

```
Drive already mounted at /gdrive; to attempt to forcibly remount, call drive.mount("/gdri
ve", force_remount=True).
```

In [12]:

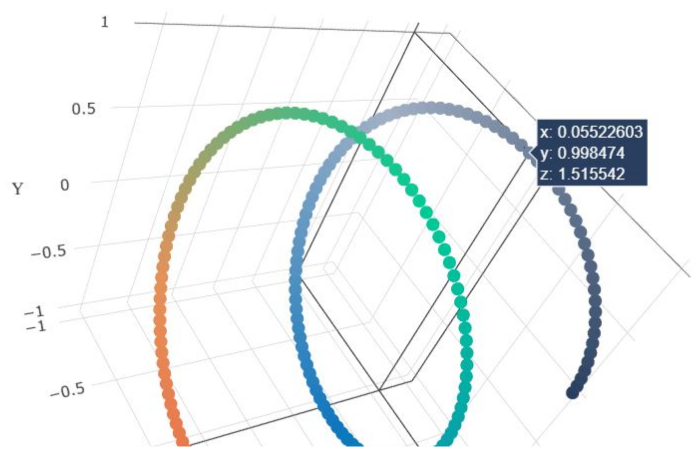
```
#please use below code to load glove vectors
import pickle
with open('/gdrive/MyDrive/9_Donors_choose_DT/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

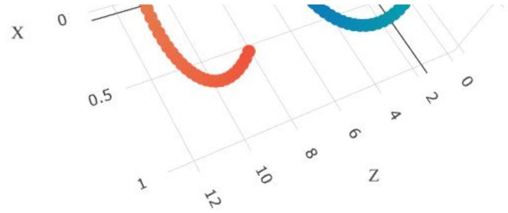
or else , you can use below code

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](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**

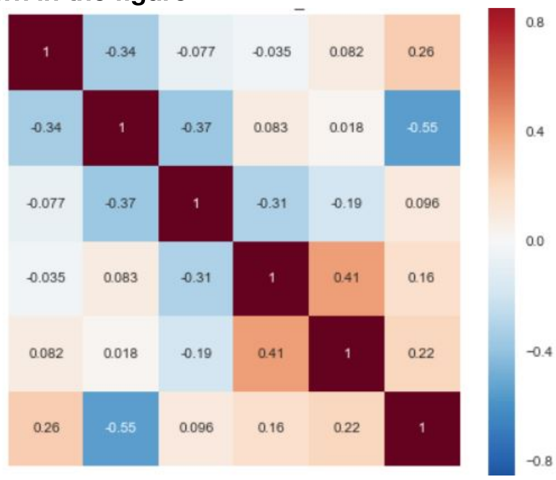




with X-axis as **min_sample_split**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

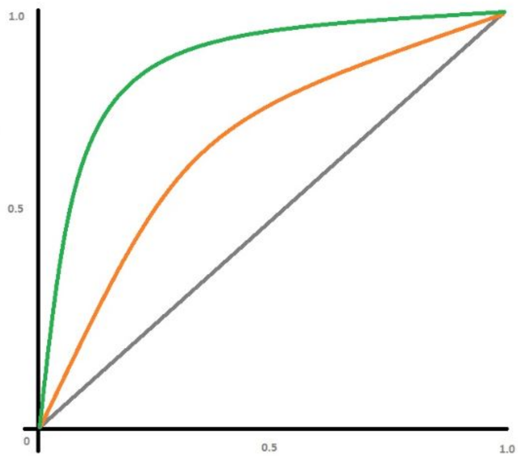
or

- 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 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 [13]:

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

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[13]:

```
True
```

In [14]:

```
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 s
tudents with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multipl
e intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of
different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school i
s a caring community of successful \
learners which can be seen through collaborative student project based learning in and ou
t of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to
practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect
of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to
role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with r
eal food i will take their idea \
```

```
and create common core cooking lessons where we learn important math and writing concepts
while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went
into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this pro
ject would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homem
ade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our
own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a life long
enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

print (ss['neg'])
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
```

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

1. Decision Tree

1.1 Loading Data

In [15]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

## To read data
import pandas as pd
import numpy as np
import pickle
from tqdm import tqdm
import os

## Data preprocessing
import nltk
import re

## EDA
import matplotlib.pyplot as plt
import seaborn as sns

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

## Feature Vectorization
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

## Model Performance
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

data = pd.read_csv('/gdrive/MyDrive/9_Donors_choose_DT/preprocessed_data.csv',nrows = 700
00)
```

In [16]:

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

1.2 Splitting data into Train and test: Stratified Sampling

In [17]:

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

```
(70000, 9)
  school_state  ...   price
0           ca  ...  725.05

[1 rows x 8 columns]
[1 1 1 ... 1 1 1]
```

In [18]:

```
data.columns.values
```

Out[18]:

```
array(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects',
       'project_is_approved', 'clean_categories', 'clean_subcategories',
       'essay', 'price'], dtype=object)
```

In [19]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X, Y, test_size=0.33, stratify=Y,random
_state = 100)
```

In [20]:

```
print("Train : ",X_train.shape, Y_train.shape)
print("Test  : ",X_test.shape, Y_test.shape)

print("="*100)
```

```
Train :  (46900, 8) (46900,)
Test  :  (23100, 8) (23100,)
=========================================================================================
===========
```

1.3 Make Data Model Ready: Encoding text features

1) Essay (TFIDF)

In [21]:

```
import time
```

```
vectorizer = TfidfVectorizer(min_df = 20,ngram_range=(1,4), max_features=5000)
start = time.time()
vectorizer.fit(X_train['essay'].values)
end = time.time()
print (end-start,'s')
```

107.91121053695679 s

In [22]:

```
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After TFIDF vectorizations\n")
print("Train Essay : ",X_train_essay_tfidf.shape, Y_train.shape)
print("Test Essay : ",X_test_essay_tfidf.shape, Y_test.shape)
print("="*100)
```

```
After TFIDF vectorizations

Train Essay :  (46900, 5000) (46900,)
Test Essay :  (23100, 5000) (23100,)
====================================================================================================
```

1.4 Make Data Model Ready: Encoding Categorical features

2) school_state

In [23]:

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

feature_list = []

for i in vectorizer.get_feature_names():
    feature_list.append(i)
print (len(feature_list))

# 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\n")
print("Train School State : ",X_train_state_ohe.shape, Y_train.shape)
print("Test School State : ",X_test_state_ohe.shape, Y_test.shape)
print("\n",vectorizer.get_feature_names())
print("="*100)
```

```
51
After vectorizations

Train School State :  (46900, 51) (46900,)
Test School State :  (23100, 51) (23100,)

 ['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il
', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'n
e', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', '
ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
====================================================================================================
```

3) teacher_prefix

In [24]:

```
vectorizer = CountVectorizer()
```

```
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data
feature_list = []

for i in vectorizer.get_feature_names():
    feature_list.append(i)
print (len(feature_list))
# 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\n")
print("Train Teacher Prefix : ",X_train_teacher_ohe.shape, Y_train.shape)
print("Test Teacher Prefix : ",X_test_teacher_ohe.shape, Y_test.shape)
print("\n",vectorizer.get_feature_names())
print("="*100)
```

```
5
After vectorizations

Train Teacher Prefix :  (46900, 5) (46900,)
Test Teacher Prefix :  (23100, 5) (23100,)

 ['dr', 'mr', 'mrs', 'ms', 'teacher']
============================================================================================
===========
```

4) project_grade_category

In [25]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on trai
n data
feature_list = []

for i in vectorizer.get_feature_names():
    feature_list.append(i)
print (len(feature_list))

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

print("After vectorizations\n")
print("Train Project Grade : ",X_train_grade_ohe.shape, Y_train.shape)
print("Test Project Grade : ",X_test_grade_ohe.shape, Y_test.shape)
print("\n",vectorizer.get_feature_names())
print("="*100)
```

```
4
After vectorizations

Train Project Grade :  (46900, 4) (46900,)
Test Project Grade :  (23100, 4) (23100,)

 ['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
============================================================================================
===========
```

5) clean_categories

In [26]:

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

for i in vectorizer.get_feature_names():
```

```
    feature_list.append(i)
print (len(feature_list))

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

print("After vectorizations\n")
print("Train Clean Categories : ",X_train_clean_category_ohe.shape, Y_train.shape)
print("Test Clean Categories : ",X_test_clean_category_ohe.shape, Y_test.shape)
print("\n",vectorizer.get_feature_names()[0:10])
print("="*100)
```

```
9
After vectorizations

Train Clean Categories :  (46900, 9) (46900,)
Test Clean Categories :  (23100, 9) (23100,)

 ['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language
', 'math_science', 'music_arts', 'specialneeds', 'warmth']
==========================================================================================
==========
```

6) clean_subcategories

In [27]:

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

for i in vectorizer.get_feature_names():
    feature_list.append(i)
print (len(feature_list))

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].value
s)
X_test_clean_subcategory_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations\n")
print("Train Clean Subcategory : ",X_train_clean_subcategory_ohe.shape, Y_train.shape)
print("Test Clean Subcategory  : ",X_test_clean_subcategory_ohe.shape, Y_test.shape)
print("\n",vectorizer.get_feature_names()[0:10])
print("="*100)
```

```
30
After vectorizations

Train Clean Subcategory :  (46900, 30) (46900,)
Test Clean Subcategory  :  (23100, 30) (23100,)

 ['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_c
areerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl']
==========================================================================================
==========
```

1.5 Make Data Model Ready: Encoding Numerical features

7) price

In [28]:

```
from sklearn.preprocessing import Normalizer
```

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


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


print("After vectorizations\n")
print("Train Price : ",X_train_price_norm.shape, Y_train.shape)
print("Test Price : ",X_test_price_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations

Train Price :  (46900, 1) (46900,)
Test Price :  (23100, 1) (23100,)
==========================================================================================
===========
```

8) teacher_number_of_previously_posted_projects

In [29]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1
,1))

X_train_previous_norm = normalizer.transform(X_train['teacher_number_of_previously_posted
_projects'].values.reshape(-1,1))
X_test_previous_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_p
rojects'].values.reshape(-1,1))


print("After vectorizations\n")
print("Train Previous Norm : ",X_train_previous_norm.shape, Y_train.shape)
print("Test Previous Norm : ",X_test_previous_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations

Train Previous Norm :  (46900, 1) (46900,)
Test Previous Norm :  (23100, 1) (23100,)
==========================================================================================
===========
```

9) Sentiment scores(preprocessed_essay)

In [30]:

```
sid = SentimentIntensityAnalyzer()

train_essays = X_train['essay'].values
X_train_negative = []
X_train_neutral = []
X_train_positive = []
X_train_compound = []

for i in tqdm(train_essays):
    ss = sid.polarity_scores(i)
    X_train_negative.append(ss['neg'])
    X_train_neutral.append(ss['neu'])
    X_train_positive.append(ss['pos'])
    X_train_compound.append(ss['compound'])

test_essays = X_test['essay'].values
X_test_negative = []
X_test_neutral = []
```

```
X_test_positive = []
X_test_compound = []

for i in tqdm(test_essays):
    ss = sid.polarity_scores(i)
    X_test_negative.append(ss['neg'])
    X_test_neutral.append(ss['neu'])
    X_test_positive.append(ss['pos'])
    X_test_compound.append(ss['compound'])

X_train_negative = np.array(X_train_negative).reshape(-1,1)
X_train_neutral = np.array(X_train_neutral).reshape(-1,1)
X_train_positive = np.array(X_train_positive).reshape(-1,1)
X_train_compound = np.array(X_train_compound).reshape(-1,1)

X_test_negative = np.array(X_test_negative).reshape(-1,1)
X_test_neutral = np.array(X_test_neutral).reshape(-1,1)
X_test_positive = np.array(X_test_positive).reshape(-1,1)
X_test_compound = np.array(X_test_compound).reshape(-1,1)
```

```
100%|██████████| 46900/46900 [01:30<00:00, 520.35it/s]
100%|██████████| 23100/23100 [00:44<00:00, 523.33it/s]
```

Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)

In [31]:

```
from scipy.sparse import hstack

X_tr_1 = hstack((X_train_essay_tfidf,X_train_state_ohe, X_train_teacher_ohe,X_train_grade
_ohe, X_train_clean_category_ohe,X_train_clean_subcategory_ohe,X_train_price_norm,X_train
_previous_norm,X_train_negative,X_train_neutral,X_train_positive,X_train_compound)).tocsr
()
X_te_1 = hstack((X_test_essay_tfidf,X_test_state_ohe, X_test_teacher_ohe,X_test_grade_ohe
, X_test_clean_category_ohe,X_test_clean_subcategory_ohe,X_test_price_norm,X_test_previou
s_norm,X_test_negative,X_test_neutral,X_test_positive,X_test_compound)).tocsr()

print("Final Data matrix")
print(X_tr_1.shape, Y_train.shape)
print(X_te_1.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(46900, 5105) (46900,)
(23100, 5105) (23100,)
==========================================================================================
==========
```

1.6 Applying Decision Tree on Set 1 using RandomizedSearch CV

In [32]:

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

dtc1 = DecisionTreeClassifier(random_state=0)
parameters = {'max_depth':[1, 5, 10, 50],'min_samples_split' : [5, 10, 100, 500]}
clf_dtc1 = RandomizedSearchCV(dtc1, parameters,n_jobs = -1,cv=10, scoring='roc_auc',retu
rn_train_score = True)

start = time.time()
```

```
clf_dtc1.fit(X_tr_1, Y_train)
end = time.time()
print (end-start,'s')
```

485.3024673461914 s

In [33]:

```
results1 = pd.DataFrame.from_dict(clf_dtc1.cv_results_)[['param_min_samples_split', 'para
m_max_depth',
                                                          'params','mean_test_score',
                                                          'std_test_score','rank_test_score',
                                                          'mean_train_score','std_train_score'
]].sort_values(['rank_test_score'])
results1.head()
```

Out[33]:

	param_min_samples_split	param_max_depth	params	mean_test_score	std_test_score	rank_test_score	mean_
5	500	10	{'min_samples_split': 500, 'max_depth': 10}	0.617632	0.004043	1	
8	100	10	{'min_samples_split': 100, 'max_depth': 10}	0.614379	0.004068	2	
0	5	10	{'min_samples_split': 5, 'max_depth': 10}	0.609365	0.006648	3	
4	100	5	{'min_samples_split': 100, 'max_depth': 5}	0.604433	0.008295	4	
3	10	5	{'min_samples_split': 10, 'max_depth': 5}	0.603909	0.008559	5	

In [34]:

```
print (results1.shape)
```

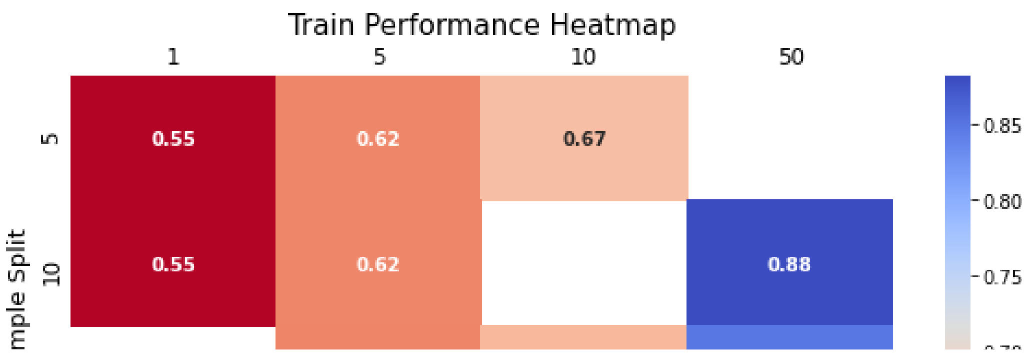
(10, 8)

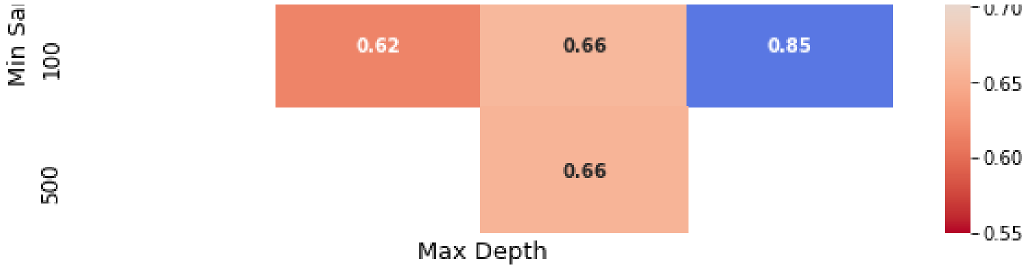
In [35]:

```
plt.figure(figsize=(10, 5))
heat_data = results1.pivot(index='param_min_samples_split', columns='param_max_depth', v
alues='mean_train_score')
vmin = results1['mean_train_score'].min()
vmax = results1['mean_train_score'].max()
heatmap = sns.heatmap(heat_data, vmin, vmax, annot=True,annot_kws={"fontsize":10,"weight
": "bold"}, cmap='coolwarm_r')
heatmap.tick_params(axis='both', which='major', labelsize=12, labelbottom = False, botto
m=False, top = False, labeltop=True,length = 0)
heatmap.set_title('Train Performance Heatmap : Set 1',fontsize = 15)
plt.xlabel('Max Depth',fontsize = 13)
plt.ylabel('Min Sample Split',fontsize = 13)
```

Out[35]:

Text(69.0, 0.5, 'Min Sample Split')



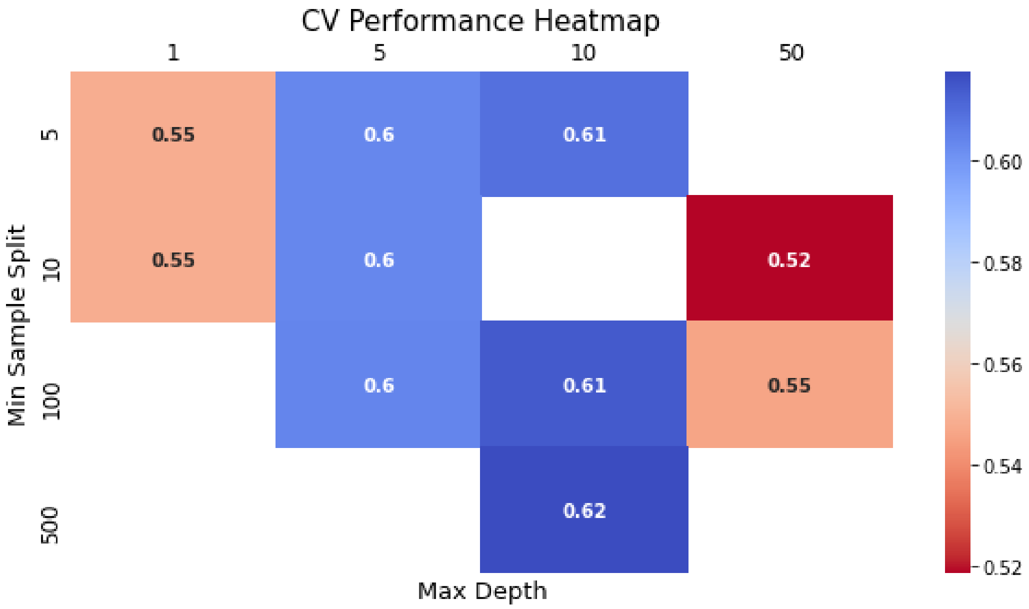


In [36]:

```
plt.figure(figsize=(10, 5))
heat_data = results1.pivot(index='param_min_samples_split', columns='param_max_depth', v
alues='mean_test_score')
vmin = results1['mean_test_score'].min()
vmax = results1['mean_test_score'].max()
heatmap = sns.heatmap(heat_data, vmin, vmax, annot=True,annot_kws={"fontsize":10,"weight
": "bold"}, cmap='coolwarm_r')
heatmap.tick_params(axis='both', which='major', labelsize=12, labelbottom = False, botto
m=False, top = False, labeltop=True,length = 0)
heatmap.set_title('CV Performance Heatmap : Set 1',fontsize = 15)
plt.xlabel('Max Depth',fontsize = 13)
plt.ylabel('Min Sample Split',fontsize = 13)
```

Out[36]:

```
Text(69.0, 0.5, 'Min Sample Split')
```



In [37]:

```
best_depth1 =  results1[results1['rank_test_score'] == 1]['param_max_depth'].values[0]
best_min_sample_split1 = results1[results1['rank_test_score'] == 1]['param_min_samples_s
plit'].values[0]
```

In [38]:

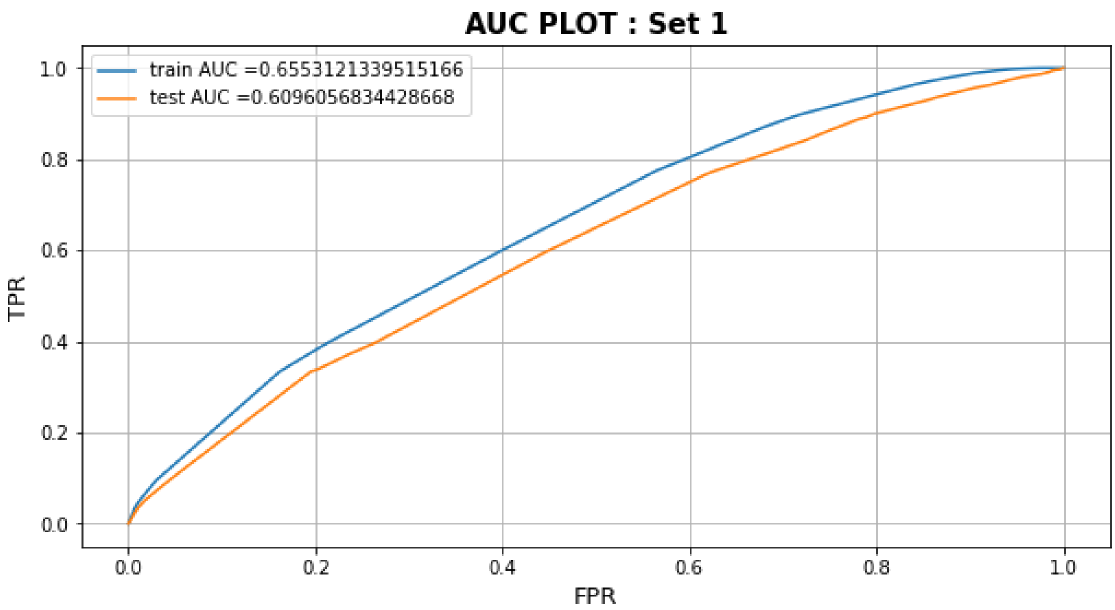
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
the positive class not the predicted outputs
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [39]:

```
from sklearn.metrics import roc_curve, auc

dt_optimal1 = DecisionTreeClassifier(max_depth = best_depth1 ,min_samples_split = best_m
in_sample_split1,random_state=0)
dt_optimal1.fit(X_tr_1, Y_train)

Y_train_pred1 = batch_predict(dt_optimal1, X_tr_1)
Y_test_pred1 = batch_predict(dt_optimal1, X_te_1)
```

In [40]:

```
plt.figure(figsize=(10, 5))
train_fpr1, train_tpr1,tr_thresholds1 = roc_curve(Y_train, Y_train_pred1)
test_fpr1, test_tpr1,te_thresholds1 = roc_curve(Y_test, Y_test_pred1)

AUC1 = auc(test_fpr1, test_tpr1)
plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
plt.legend()
plt.xlabel("FPR",fontsize = 13)
plt.ylabel("TPR",fontsize = 13)
plt.title("AUC PLOT : Set 1",fontsize = 15,weight = "bold")
plt.grid()
plt.show()
```

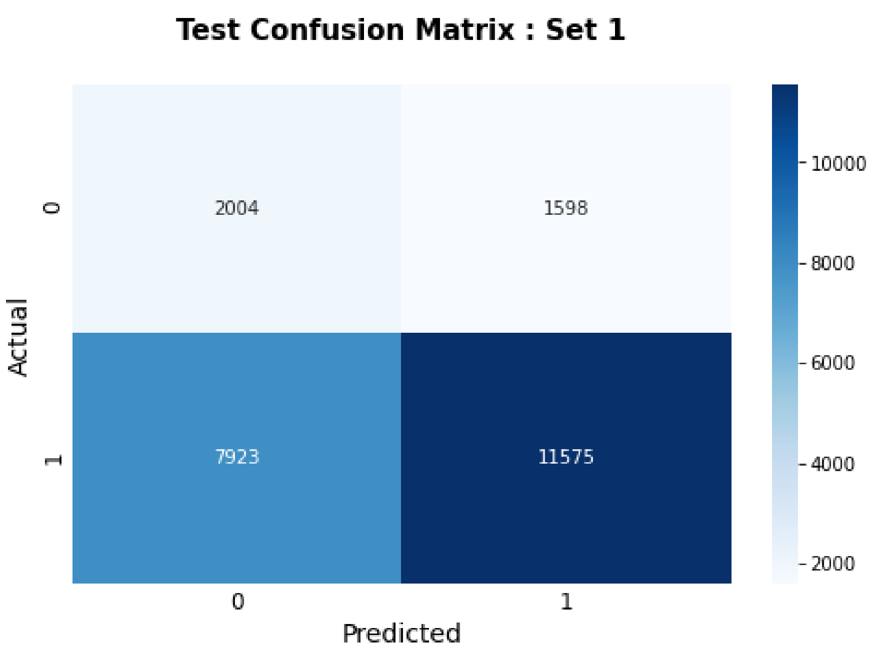


In [41]:

```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("The maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round
(t,3))
    print ('\n')
    return t

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

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
best_t1 = find_best_threshold(tr_thresholds1, train_fpr1, train_tpr1)

print("Test confusion matrix : Set 1\n")
print (confusion_matrix(Y_test, predict_with_best_t(Y_test_pred1, best_t1)))
tn1, fp1, fn1, tp1 = confusion_matrix(Y_test, predict_with_best_t(Y_test_pred1, best_t1)
).ravel()
print ("\nTrue Negative  : ",tn1)
print ("False Positive : ",fp1)
print ("False Negative : ",fn1)
print ("True Positive  : ",tp1)
print ("\n")
test_cm = np.array([[tn1,fp1 ],[fn1, tp1 ]])

plt.figure(figsize=(8, 5))
cm = sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues')
cm.tick_params(axis='both', which='major', labelsize=12, labelbottom = True, bottom=Fals
e, top = False, labeltop=False,length = 0)
cm.set_title('Test Confusion Matrix : Set 1\n',fontsize = 15,weight = 'bold')
plt.xlabel('Predicted',fontsize = 14)
plt.ylabel('Actual',fontsize = 14)
plt.show()
```

```
The maximum value of tpr*(1-fpr) 0.35998211687457493 for threshold 0.855


Test confusion matrix : Set 1

[[ 2004  1598]
 [ 7923 11575]]

True Negative  :  2004
False Positive :  1598
False Negative :  7923
True Positive  :  11575
```



1.7 EDA on False Positive Data - Set 1

a. Word Cloud : Essay

In [42]:

```
!pip install wordcloud
```

```
!pip install wordcloud
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.7/dist-packages (1.5.0
)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.7/dist-packages (fr
om wordcloud) (1.19.5)
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-packages (from wor
dcloud) (7.1.2)
```

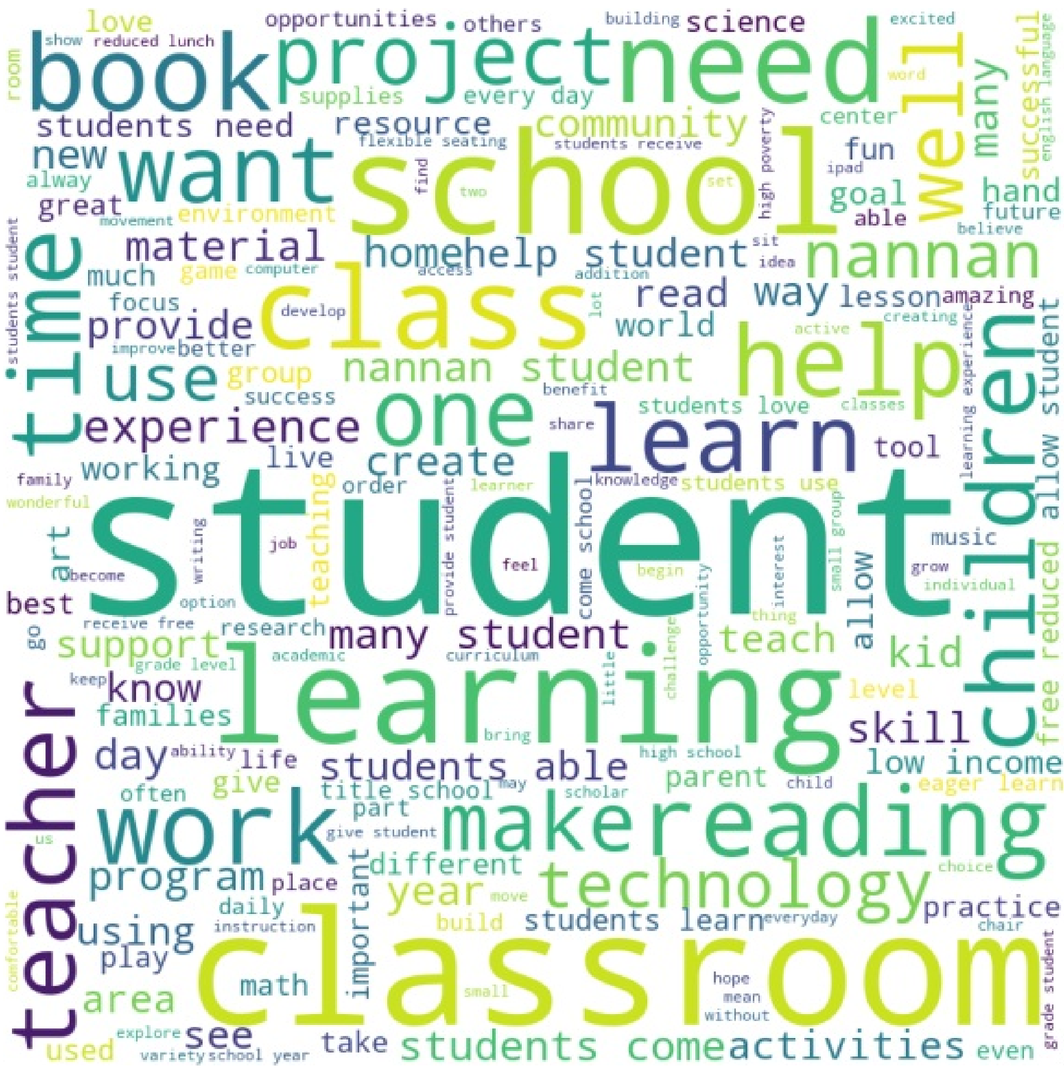
In [43]:

```
predicted_data1 = X_test
predicted_data1['Y_test'] = Y_test
predicted_data1['Y_predicted'] =predict_with_best_t(Y_test_pred1, best_t1)
false_positive_data1 = predicted_data1.loc[(predicted_data1['Y_test'] == 0) & (predicted
_data1['Y_predicted'] == 1 )]
print (false_positive_data1.shape)
```

```
(1598, 10)
```

In [44]:

```
from wordcloud import WordCloud,STOPWORDS
essay_words = ''
stopwords = set(STOPWORDS)
for sent in false_positive_data1['essay']:
  tokens = sent.split()
  essay_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,stopwords = stopwords,background_color =
'white',min_font_size = 10).generate(essay_words)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



b. Box Plot : Price

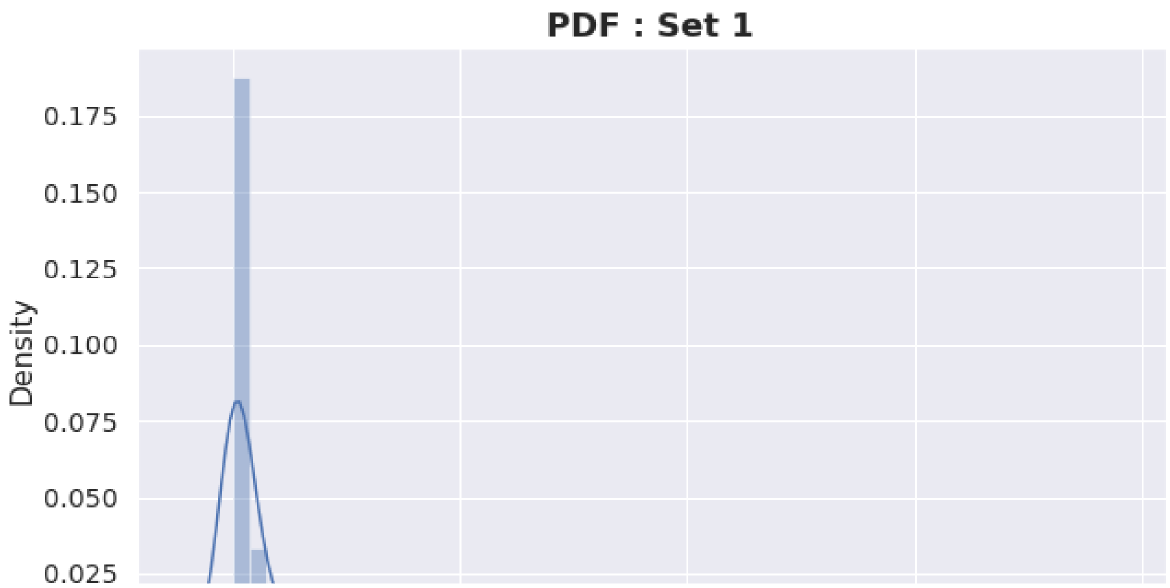
In [45]:

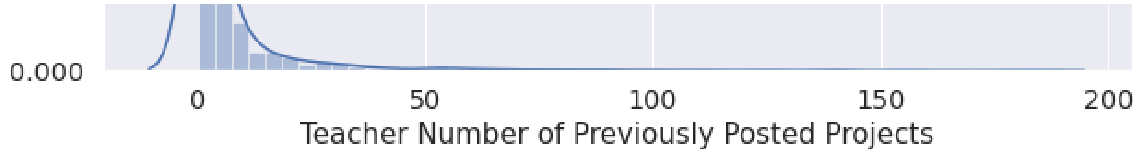
```
sns.set_theme(style="whitegrid")
sns.set(font_scale = 1.3)
plt.figure(figsize=(10, 6))
bp = sns.boxplot(x = false_positive_data1['price'],palette="Set3",linewidth=2.5)
bp.set_title('Box-Plot with "Price" : Set 1',fontsize = 15,weight = 'bold')
plt.xlabel('Price',fontsize = 15)
plt.show()
```

c. PDF Plot : Teacher Number of Previously Posted Projects

In [46]:

```
sns.set_theme(style="whitegrid")
sns.set(font_scale = 1.3)
plt.figure(figsize=(10, 6))
pdf = sns.distplot(false_positive_data1['teacher_number_of_previously_posted_projects'])
pdf.set_title('PDF : Set 1',fontsize = 18,weight = 'bold')
plt.xlabel('Teacher Number of Previously Posted Projects',fontsize = 15)
plt.show()
```





Set 2: categorical, numerical features + preprocessed_essay (TFIDFW2V) + Sentiment scores(preprocessed_essay)

a. Essay(TFIDFW2V)

In [47]:

```
vectorizer = TfidfVectorizer(min_df = 20,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values)

dictionary = dict(zip(vectorizer.get_feature_names(), list(vectorizer.idf_)))
tfidf_words = set(vectorizer.get_feature_names())
```

In [48]:

```
#compute average word2vec for each essay.
X_train_tfidf_w2v_vectors = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each essay
    vector = np.zeros(300)
    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

            # 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
    X_train_tfidf_w2v_vectors.append(vector)

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

```
100%|██████████| 46900/46900 [01:21<00:00, 573.32it/s]
```

```
46900
300
```

In [49]:

```
# calculating tfidf weighted w2v of X_test['essay']
X_test_tfidf_w2v_vectors = [];
for sentence in tqdm(X_test['essay'].values):
    vector = np.zeros(300)
    tf_idf_weight =0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            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%|██████████| 23100/23100 [00:40<00:00, 574.69it/s]
```

```
23100
300
```

In [50]:

```
from scipy.sparse import hstack

X_tr_2 = hstack((X_train_tfidf_w2v_vectors,X_train_state_ohe, X_train_teacher_ohe,X_train_grade_ohe, X_train_clean_category_ohe,X_train_clean_subcategory_ohe,X_train_price_norm,X_train_previous_norm,X_train_negative,X_train_neutral,X_train_positive,X_train_compound)).tocsr()

X_te_2 = hstack((X_test_tfidf_w2v_vectors,X_test_state_ohe, X_test_teacher_ohe,X_test_grade_ohe, X_test_clean_category_ohe,X_test_clean_subcategory_ohe,X_test_price_norm,X_test_previous_norm,X_test_negative,X_test_neutral,X_test_positive,X_test_compound)).tocsr()

print("Final Data matrix")
print(X_tr_2.shape, Y_train.shape)
print(X_te_2.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(46900, 405) (46900,)
(23100, 405) (23100,)
====================================================================================================
```

In [51]:

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

dtc2 = DecisionTreeClassifier(random_state=0)
parameters = {'max_depth':[1, 5, 10, 50],'min_samples_split' : [5, 10, 100, 500]}
clf_dtc2 = RandomizedSearchCV(dtc2, parameters,cv=5, scoring='roc_auc',return_train_score= True)

start = time.time()
clf_dtc2.fit(X_tr_2, Y_train)
end = time.time()
print (end-start,'s')
```

```
1944.5656170845032 s
```

In [53]:

```
results2 = pd.DataFrame.from_dict(clf_dtc2.cv_results_)[['param_min_samples_split', 'param_max_depth',
                                                          'params','mean_test_score',
                                                          'std_test_score','rank_test_score',
                                                          'mean_train_score','std_train_score'
]].sort_values(['rank_test_score'])
results2.head()
```

Out[53]:

	param_min_samples_split	param_max_depth	params	mean_test_score	std_test_score	rank_test_score	mean_
7	100	5	{'min_samples_split': 100, 'max_depth': 5}	0.603291	0.005999	1	
4	500	10	{'min_samples_split': 500, 'max_depth': 10}	0.597847	0.006017	2	

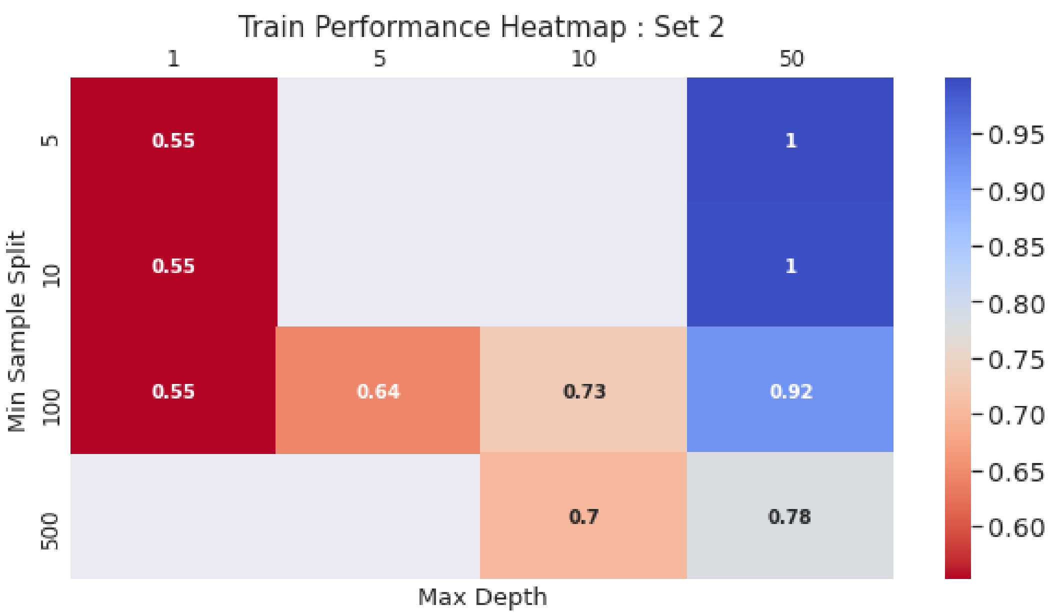
	param_min_samples_split	param_max_depth	params	mean_test_score	std_test_score	rank_test_score	mean_
3	500	50	{'min_samples_split': 500, 'max_depth': 50}	0.589025	0.006381	3	
9	100	10	{'min_samples_split': 100, 'max_depth': 10}	0.585383	0.005094	4	
2	100	50	{'min_samples_split': 100, 'max_depth': 50}	0.559088	0.006950	5	

In [54]:

```
plt.figure(figsize=(10, 5))
heat_data = results2.pivot(index='param_min_samples_split', columns='param_max_depth', v
alues='mean_train_score')
vmin = results2['mean_train_score'].min()
vmax = results2['mean_train_score'].max()
heatmap = sns.heatmap(heat_data, vmin, vmax, annot=True,annot_kws={"fontsize":10,"weight
": "bold"}, cmap='coolwarm_r')
heatmap.tick_params(axis='both', which='major', labelsize=12, labelbottom = False, botto
m=False, top = False, labeltop=True,length = 0)
heatmap.set_title('Train Performance Heatmap : Set 2',fontsize = 15)
plt.xlabel('Max Depth',fontsize = 13)
plt.ylabel('Min Sample Split',fontsize = 13)
```

Out[54]:

```
Text(62.5, 0.5, 'Min Sample Split')
```

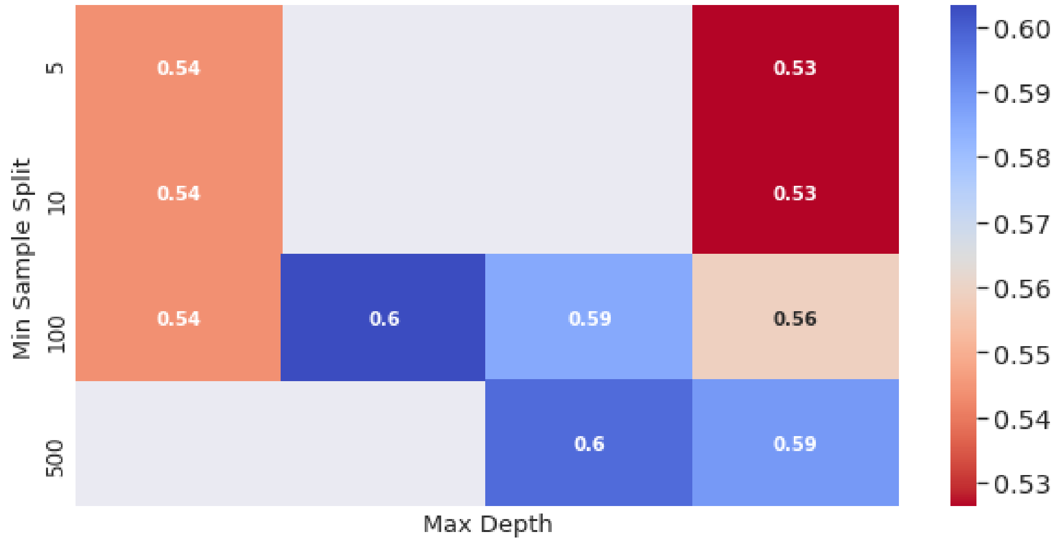


In [55]:

```
plt.figure(figsize=(10, 5))
heat_data = results2.pivot(index='param_min_samples_split', columns='param_max_depth', v
alues='mean_test_score')
vmin = results2['mean_test_score'].min()
vmax = results2['mean_test_score'].max()
heatmap = sns.heatmap(heat_data, vmin, vmax, annot=True,annot_kws={"fontsize":10,"weight
": "bold"}, cmap='coolwarm_r')
heatmap.tick_params(axis='both', which='major', labelsize=12, labelbottom = False, botto
m=False, top = False, labeltop=True,length = 0)
heatmap.set_title('CV Performance Heatmap : Set 1',fontsize = 15)
plt.xlabel('Max Depth',fontsize = 13)
plt.ylabel('Min Sample Split',fontsize = 13)
```

Out[55]:

```
Text(62.5, 0.5, 'Min Sample Split')
```



In [56]:

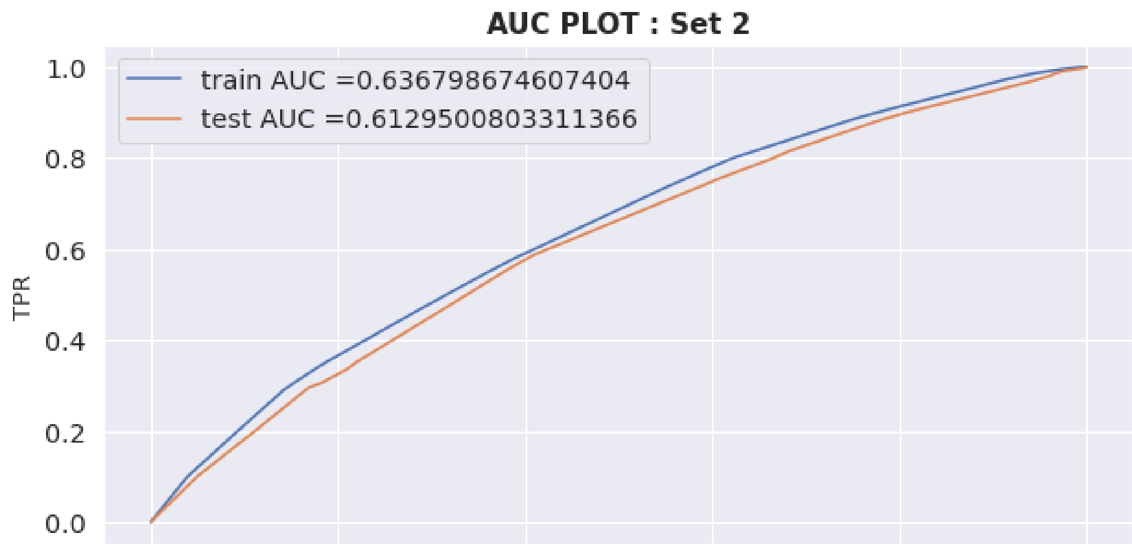
```
best_depth2 =  results2[results2['rank_test_score'] == 1]['param_max_depth'].values[0]
best_min_sample_split2 = results2[results2['rank_test_score'] == 1]['param_min_samples_s
plit'].values[0]
```

In [57]:

```
dt_optimal2 = DecisionTreeClassifier(max_depth = best_depth2 ,min_samples_split = best_m
in_sample_split2,random_state=0)
dt_optimal2.fit(X_tr_2, Y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
positive class
# not the predicted outputs

Y_train_pred2 = batch_predict(dt_optimal2, X_tr_2)
Y_test_pred2 = batch_predict(dt_optimal2, X_te_2)
```

In [86]:

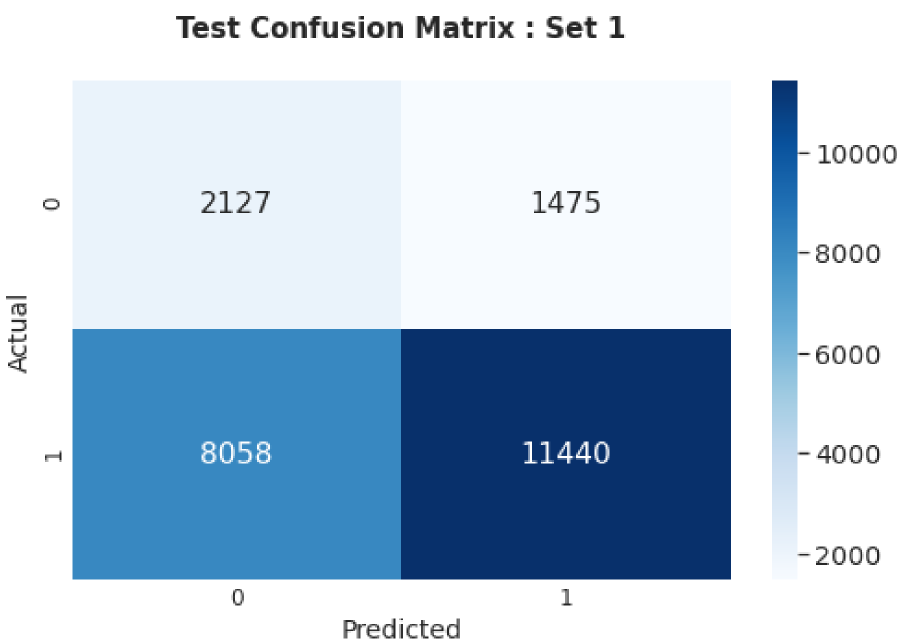
```
plt.figure(figsize=(10, 5))
plt.grid()
train_fpr2, train_tpr2,tr_thresholds2 = roc_curve(Y_train, Y_train_pred2)
test_fpr2, test_tpr2,te_thresholds2 = roc_curve(Y_test, Y_test_pred2)

AUC2 = auc(test_fpr2, test_tpr2)
plt.plot(train_fpr2, train_tpr2, label="train AUC ="+str(auc(train_fpr2, train_tpr2)))
plt.plot(test_fpr2, test_tpr2, label="test AUC ="+str(auc(test_fpr2, test_tpr2)))
plt.legend()
plt.xlabel("FPR",fontsize = 13)
plt.ylabel("TPR",fontsize = 13)
plt.title("AUC PLOT : Set 2",fontsize = 15,weight = "bold")
plt.grid()
plt.show()
```



In [67]:

```
best_t2 = find_best_threshold(tr_thresholds2, train_fpr2, train_tpr2)

print("Test confusion matrix : Set 2\n")
print (confusion_matrix(Y_test, predict_with_best_t(Y_test_pred2, best_t2)))
tn2, fp2, fn2, tp2 = confusion_matrix(Y_test, predict_with_best_t(Y_test_pred2, best_t2)
).ravel()
print ("\nTrue Negative  : ",tn2)
print ("False Positive : ",fp2)
print ("False Negative : ",fn2)
print ("True Positive  : ",tp2)
test_cm = np.array([[tn2,fp2 ],[fn2, tp2 ]])

plt.figure(figsize=(8, 5))
cm = sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues')
cm.tick_params(axis='both', which='major', labelsize=12, labelbottom = True, bottom=Fals
e, top = False, labeltop=False,length = 0)
cm.set_title('\nTest Confusion Matrix : Set 1\n',fontsize = 15,weight = 'bold')
plt.xlabel('Predicted',fontsize = 14)
plt.ylabel('Actual',fontsize = 14)
plt.show()
```

```
The maximum value of tpr*(1-fpr) 0.353957081915216 for threshold 0.851


Test confusion matrix : Set 2

[[ 2127  1475]
 [ 8058 11440]]

True Negative  :  2127
False Positive :  1475
False Negative :  8058
True Positive  :  11440
```



1.8 EDA on False Positive Data - Set 2

In [68]:

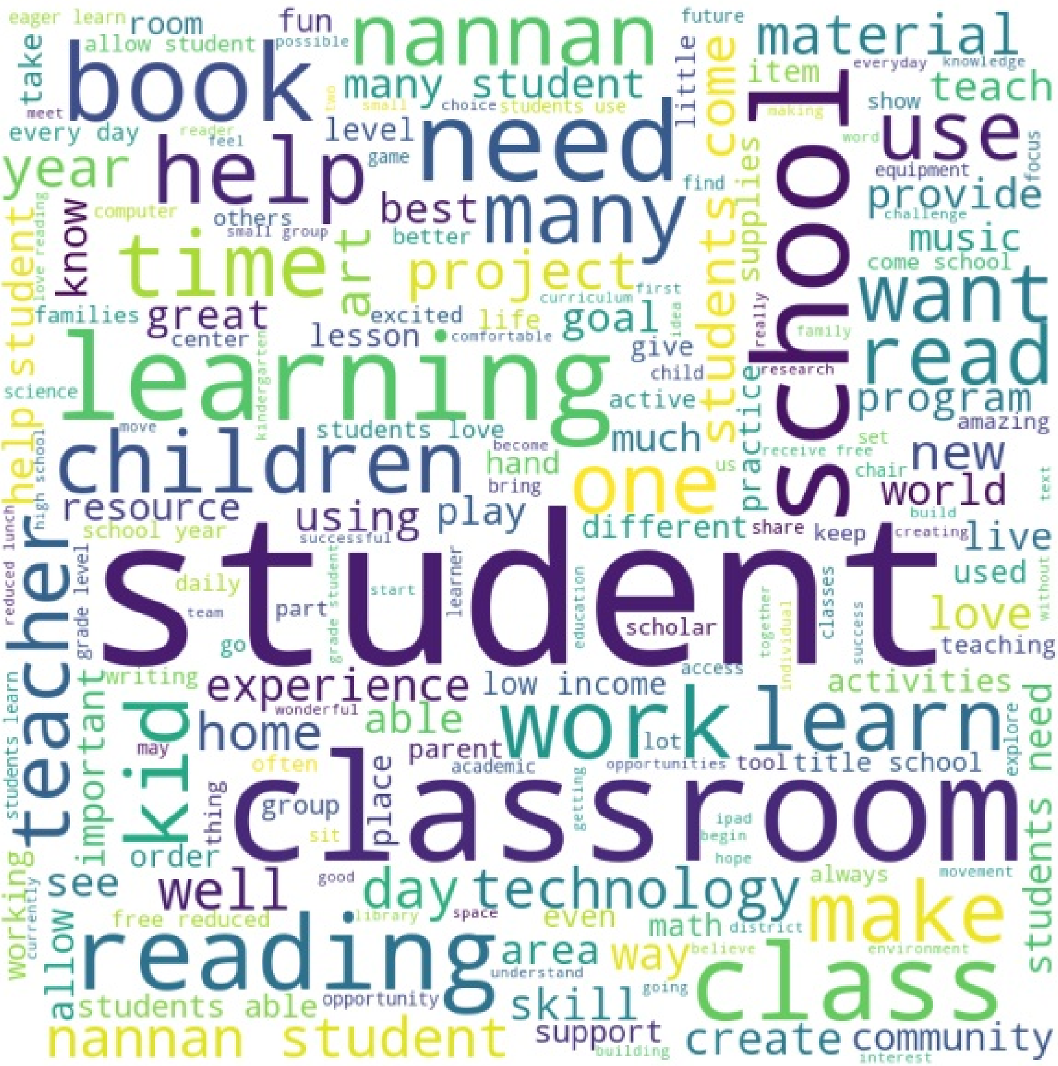
```
predicted_data2 = X_test
```

```
predicted_data2['Y_test'] = Y_test
predicted_data2['Y_predicted'] =predict_with_best_t(Y_test_pred2, best_t2)

false_positive_data2 = predicted_data2.loc[(predicted_data2['Y_test'] == 0) & (predicted
_data2['Y_predicted'] == 1 )]
```

a. Word Cloud : Essay

In [69]:

```
essay_words2 = ''
for sent in false_positive_data2['essay']:
  tokens = sent.split()
  essay_words2 += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,stopwords = stopwords,background_color =
'white',min_font_size = 10).generate(essay_words2)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

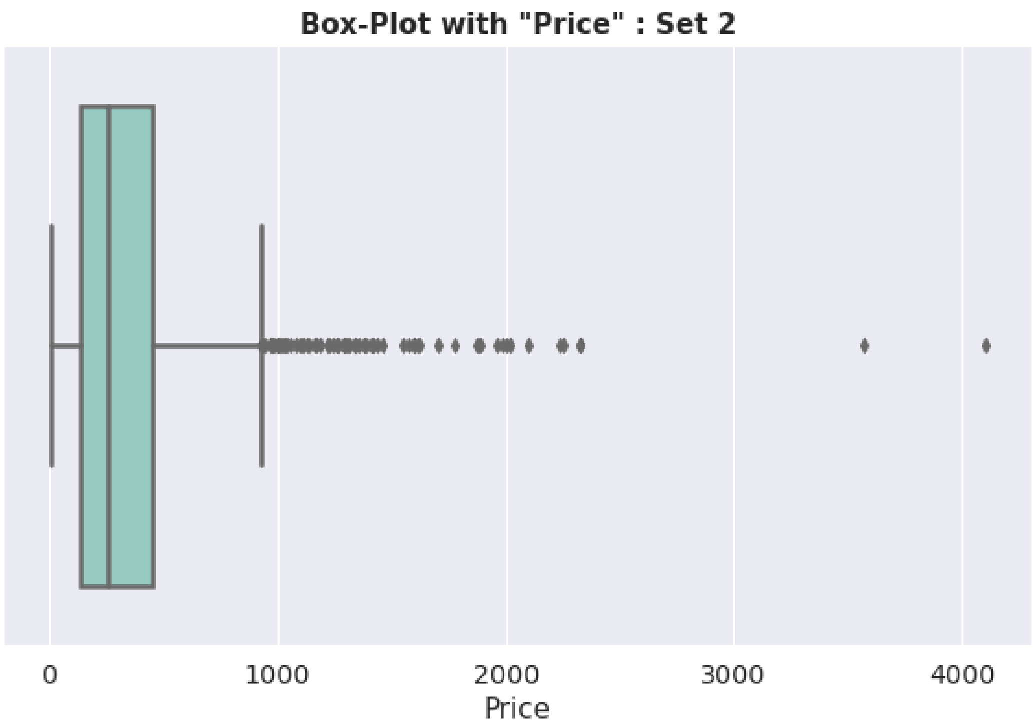


b. Box Plot : Price

In [71]:

```
sns.set_theme(style="whitegrid")
sns.set(font_scale = 1.3)
plt.figure(figsize=(10, 6))
bp = sns.boxplot(x = false_positive_data2['price'],palette="Set3",linewidth=2.5)
```

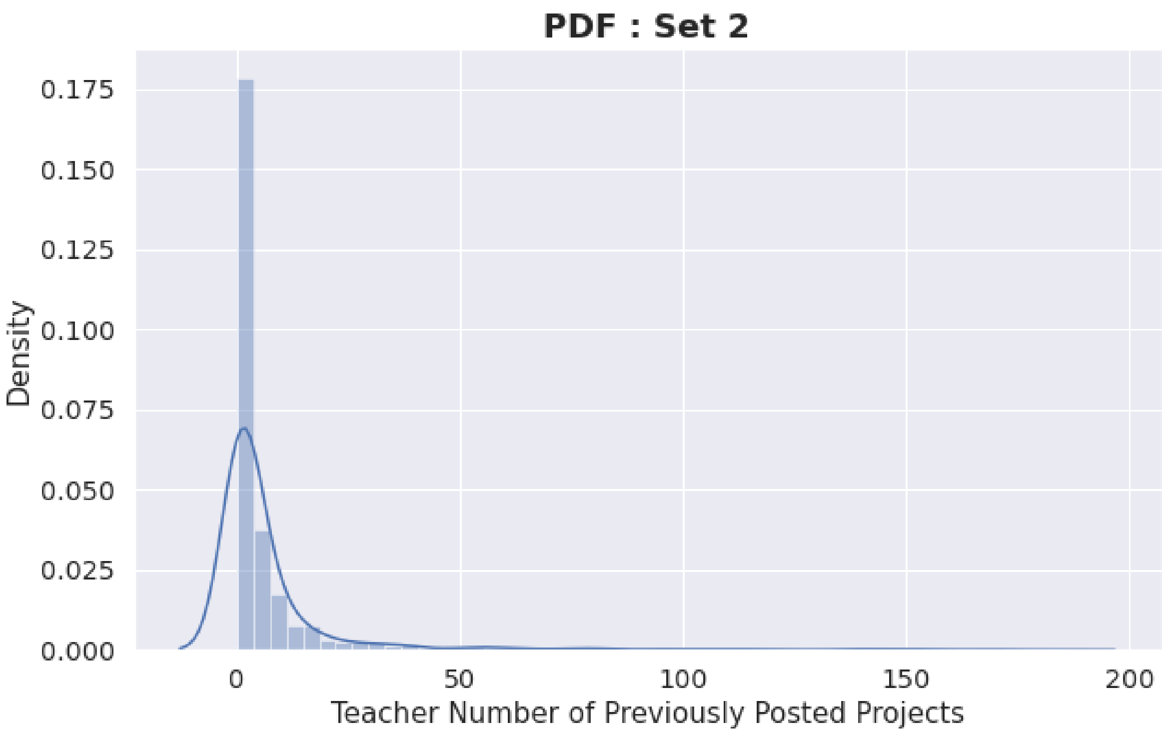
```
bp.set_title('Box-Plot with "Price" : Set 2',fontsize = 15,weight = 'bold')
plt.xlabel('Price',fontsize = 15)
plt.show()
```



c. PDF Plot : Teacher Number of Previously Posted Projects

In [72]:

```
sns.set_theme(style="whitegrid")
sns.set(font_scale = 1.3)
plt.figure(figsize=(10, 6))
pdf = sns.distplot(false_positive_data2['teacher_number_of_previously_posted_projects'])
pdf.set_title('PDF : Set 2',fontsize = 18,weight = 'bold')
plt.xlabel('Teacher Number of Previously Posted Projects',fontsize = 15)
plt.show()
```



TASK-2 - Using SET 1

In [73]:

```
dt_optimal1 = DecisionTreeClassifier(max_depth = None ,min_samples_split = best_min_samp
le_split1,random_state=0)
dt_optimal1.fit(X_tr_1, Y_train)
dt_optimal1.feature_importances_
```

Out[73]:

```
array([0.00042982, 0.        , 0.        , ..., 0.00574753, 0.00115005,
       0.00164773])
```

Selecting Features with Non Zero Feature Importance based on Decision Tree : Set 1

In [74]:

```
X_train_new = X_tr_1[:, dt_optimal1.feature_importances_ != 0]
X_test_new =  X_te_1[:, dt_optimal1.feature_importances_ != 0]
```

Selecting Best Model and Hyperparamter Using GridSearch

In [75]:

```
from sklearn.linear_model import SGDClassifier
alpha = [0.000001,0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
param_grid = dict(alpha=alpha,loss = ['log','hinge','squared_loss'])
model  = SGDClassifier(eta0=0.001, learning_rate='constant',random_state=0)
clf = GridSearchCV(estimator = model,param_grid = param_grid,cv = 10,scoring='roc_auc',r
eturn_train_score = True)
clf.fit(X_train_new, Y_train)
```

Out[75]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight=None, early_stopping=False,
                                     epsilon=0.1, eta0=0.001,
                                     fit_intercept=True, l1_ratio=0.15,
                                     learning_rate='constant', loss='hinge',
                                     max_iter=1000, n_iter_no_change=5,
                                     n_jobs=None, penalty='l2', power_t=0.5,
                                     random_state=0, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1,
                                   10, 100, 1000],
                         'loss': ['log', 'hinge', 'squared_loss']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [76]:

```
results_new = pd.DataFrame.from_dict(clf.cv_results_)[['param_alpha','param_loss',
                                                   'params','mean_test_score',
                                                   'std_test_score','rank_test_score',
                                                   'mean_train_score','std_train_score'
]].sort_values(['rank_test_score'])
results_new.head()
```

Out[76]:

	param_alpha	param_loss	params	mean_test_score	std_test_score	rank_test_score	mean_train_score	std_train
2	1e-06	squared_loss	{'alpha': 1e-06, 'loss': 'squared_loss'}	0.673705	0.009923	1	0.690736	0.(
5	1e-05	squared_loss	{'alpha': 1e-05, 'loss': 'squared_loss'}	0.673668	0.009922	2	0.690690	0.(

	param_alpha	param_loss	params	mean_test_score	std_test_score	rank_test_score	mean_train_score	std_train
8	0.0001	squared_loss	{'alpha': 0.0001, 'loss': 'squared_loss'}	0.672983	0.010222	3	0.689854	0.(
11	0.001	squared_loss	{'alpha': 0.001, 'loss': 'squared_loss'}	0.667770	0.010477	4	0.682990	0.(
14	0.01	squared_loss	{'alpha': 0.01, 'loss': 'squared_loss'}	0.621674	0.013153	5	0.629194	0.(

In [78]:

```
print('Best Score: ', clf.best_score_)
print('Best Params: ', clf.best_params_)
```

```
Best Score:  0.673705149198544
Best Params:  {'alpha': 1e-06, 'loss': 'squared_loss'}
```

The best model is Linear Regression with alpha = 1e-06

In [79]:

```
best_model_svm = SGDClassifier(loss = 'squared_loss',alpha = 1e-06,eta0 = 0.001,learning_
rate='constant',random_state = 0)
best_model_svm.fit(X_train_new,Y_train)
```

Out[79]:

```
SGDClassifier(alpha=1e-06, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.001, fit_intercept=True,
              l1_ratio=0.15, learning_rate='constant', loss='squared_loss',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=0, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

Callibrating the model to get probability

In [80]:

```
from sklearn.calibration import CalibratedClassifierCV
calibrated_clf = CalibratedClassifierCV(base_estimator=best_model_svm,cv="prefit",method
= 'isotonic')
calibrated_clf.fit(X_train_new,Y_train)
```

Out[80]:

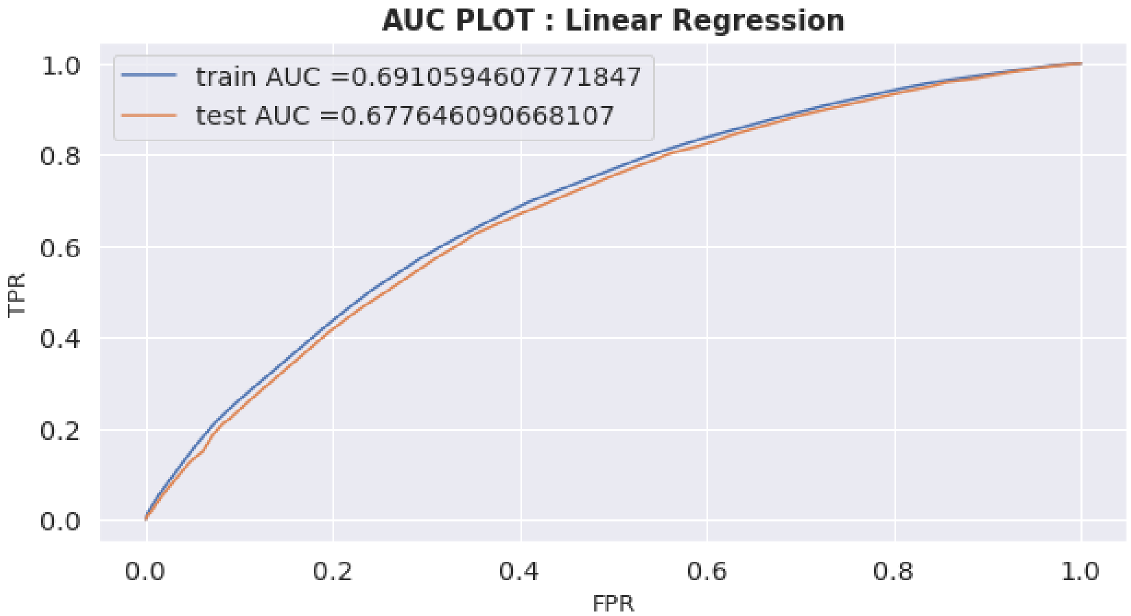
```
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=1e-06, average=False,
                                                    class_weight=None,
                                                    early_stopping=False,
                                                    epsilon=0.1, eta0=0.001,
                                                    fit_intercept=True,
                                                    l1_ratio=0.15,
                                                    learning_rate='constant',
                                                    loss='squared_loss',
                                                    max_iter=1000,
                                                    n_iter_no_change=5,
                                                    n_jobs=None, penalty='l2',
                                                    power_t=0.5, random_state=0,
                                                    shuffle=True, tol=0.001,
                                                    validation_fraction=0.1,
                                                    verbose=0,
                                                    warm_start=False),
                       cv='prefit', method='isotonic')
```

In [81]:

```
Y_train_pred_new = batch_predict(calibrated_clf, X_train_new)
Y_test_pred_new = batch_predict(calibrated_clf, X_test_new)
```

In [87]:

```
plt.figure(figsize=(10, 5))
plt.grid()
train_fpr_new, train_tpr_new,tr_thresholds_new = roc_curve(Y_train, Y_train_pred_new)
test_fpr_new, test_tpr_new,te_thresholds_new = roc_curve(Y_test, Y_test_pred_new)

AUC_new = auc(test_fpr_new, test_tpr_new)
plt.plot(train_fpr_new, train_tpr_new, label="train AUC ="+str(auc(train_fpr_new, train_t
pr_new)))
plt.plot(test_fpr_new, test_tpr_new, label="test AUC ="+str(auc(test_fpr_new, test_tpr_ne
w)))
plt.legend()
plt.xlabel("FPR",fontsize = 13)
plt.ylabel("TPR",fontsize = 13)
plt.title("AUC PLOT : Linear Regression",fontsize = 15,weight = "bold")
plt.grid()
plt.show()
```



In [89]:

```
Vectorizer = ['TFIDF','TFIDFW2V','TFIDF']
Model = ['Decision Tree','Decision Tree','Linear Regression']
Hyper_Parameter = [(results1.loc[results1['rank_test_score'] == 1,'params']).to_string(i
ndex = False),(results2.loc[results2['rank_test_score']==1,'params']).to_string(index =
False),(results_new.loc[results_new['rank_test_score'] ==1,'params']).to_string(index =
False)]
AUC = [AUC1,AUC2,AUC_new]

summary_df = pd.DataFrame(list(zip(Vectorizer,Model,Hyper_Parameter,AUC)),columns= ['Vect
orizer','Model','Hyper_Parameter','AUC'])
```

In [90]:

```
from tabulate import tabulate
print(tabulate(summary_df, headers='keys', tablefmt='psql',showindex=False))
```

```
+--------------+-------------------+---------------------------------------------+-----
-----+
| Vectorizer   | Model             | Hyper_Parameter                             |
AUC |
|--------------+-------------------+---------------------------------------------+-----
-----|
| TFIDF        | Decision Tree     | {'min_samples_split': 500, 'max_depth': 10} | 0.6096
06 |
| TFIDFW2V     | Decision Tree     | {'min_samples_split': 100, 'max_depth': 5}  | 0.6129
5  |
| TFIDF        | Linear Regression | {'alpha': 1e-06, 'loss': 'squared_loss'}    | 0.6776
46 |
+--------------+-------------------+---------------------------------------------+-----
-----+
```