Assignment : DT

Please check below video before attempting this assignment

In [1]:

```
from IPython.display import YouTubeVideo
YouTubeVideo('ZhLXULFjIjQ', width="1000",height="500")
```

Out[1]:

TF-IDFW2V

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

(Optional) Please check course video on [AVgw2V and TF-IDFW2V](#) for more details.

Glove vectors

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

Download glove vectors from this [link](#)

In [2]:

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

or else , you can use below code

In [3]:

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

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

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

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

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

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

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

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


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

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


'''
```

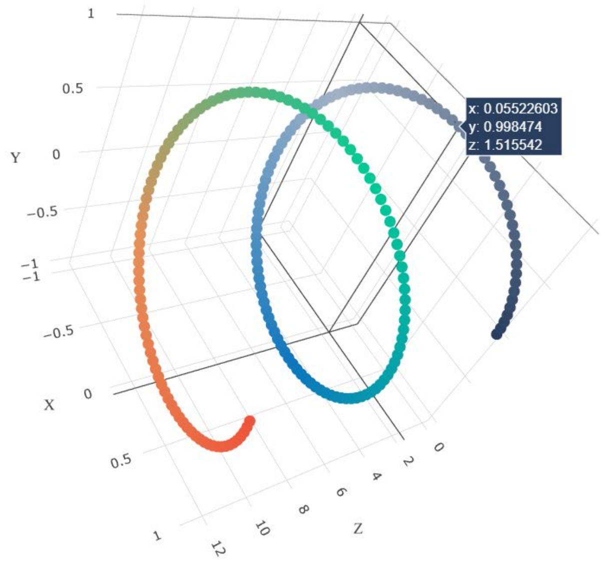
Out[3]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef loadGloveModel(
gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    mod
el = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n        word = splitLine[0]\n
embedding = np.array([float(val) for val in splitLine[1:]])\n        model[word] = embedding\n    print
("Done.",len(model)," words loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')
\n\n# ============================\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\
nDone. 1917495  words loaded!\n\n# ============================\n\nwords = []\nfor i in preproced_texts
:\n    words.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\npr
int("all the words in the coupus", len(words))\nwords = set(words)\nprint("the unique words in the coup
us", len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"(",np.round(len(inter_word
s)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n
if i in words_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length", len(words_courpus
```

```
))\n\n\n# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to
-save-and-load-variables-in-python/\n\nimport pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n    p
ickle.dump(words_courpus, f)\n\n\n'
```

Task - 1

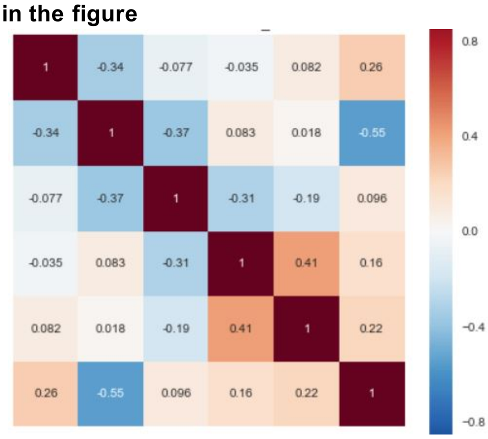
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - **Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)**
    - **Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)**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



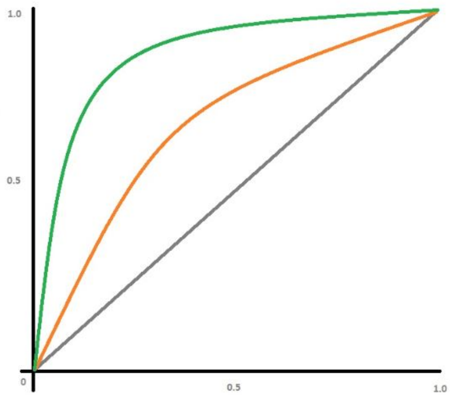
with X-axis as min_sample_split, Y-axis as max_depth, and Z-axis as AUC Score , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

or

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



seaborn heat maps with rows as min_sample_split, columns as max_depth, and values inside the cell representing AUC Score
  - You choose either of the plotting techniques out of 3d plot or heat map
  - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



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

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

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

1.1 Import necessary Libraries

In [4]:

```
!pip install chart_studio
```

```
Requirement already satisfied: chart_studio in /usr/local/lib/python3.6/dist-packages (1.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from chart_studio) (
2.23.0)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from chart_st
udio) (1.3.3)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from chart_studio) (1.15.
0)
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from chart_studio) (4.
4.1)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from reques
ts->chart_studio) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from reque
sts->chart_studio) (2020.12.5)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->c
hart_studio) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist
-packages (from requests->chart_studio) (1.24.3)
```

In [5]:

```
# importing necessary libraries
%matplotlib inline
import warnings
```

```
warnings.filterwarnings("ignore")

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.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

import pickle
from tqdm import tqdm,tnrange,tqdm_notebook
import os
import plotly as ply
import plotly.graph_objs as go
from plotly import offline
offline.init_notebook_mode()
from collections import Counter
```

1.2 Loading Data:

In [6]:

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

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

In [7]:

```
# Reading Preprocessed data
data = pd.read_csv("/content/drive/My Drive/11_Applying Decision Trees on Donors choose Dataset/preproc
essed_data.csv",nrows = 50000)
# shape of the data
print(data.shape)
```

```
(50000, 9)
```

In [8]:

```
data.head(2)
```

Out[8]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_cat
0	ca	mrs	grades_prek_2	53	1	math_s
1	ut	ms	grades_3_5	4	1	specia

school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_cat

1.3 Split X and Y into Train and Cross Validation data(Test data) based on Stratify Sampling

In [9]:

```
# make data into as X and Y
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[9]:

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

In [10]:

```
# train test split
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 )
```

1.4 Vectorization

Set -1

1.4.1 Encoding categorical features:(using One Hot Encoding)

1. school_state
2. teacher_prefix
3. project_grade_category
4. clean_categories
5. clean_subcategories

In [11]:

```
# 1. school_state encoding
vectorizer = OneHotEncoder(sparse=False,handle_unknown='ignore')              # initialize One Hot En
coder.
vectorizer.fit(X_train['school_state'].values.reshape(-1,1))                  # fit has to happen onl
y on train data.

# use the vectorizer to convert string categories of school_state to numerical vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values.reshape(-1,1))
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values.reshape(-1,1))

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

```
After vectorizations
(33500, 51) (33500,)
(16500, 51) (16500,)
[array(['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', 'ne', 'nh', 'nj', 'nm',
       'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx',
       'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy'], dtype=object)]
====================================================================================================
```

In [12]:

```
# 2. teacher_prefix
vectorizer = OneHotEncoder(sparse=False,handle_unknown='ignore')               # initialize One Hot En
coder.
vectorizer.fit(X_train['teacher_prefix'].values.reshape(-1,1))                  # fit has to happen on
ly on train data.

# use the vectorizer to convert categories of teacher_prefix to vector
X_train_teacher_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].values.reshape(-1,1))
X_test_teacher_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_teacher_prefix_ohe.shape, y_train.shape)
print(X_test_teacher_prefix_ohe.shape, y_test.shape)
print(vectorizer.categories_)
print("="*100)
```

```
After vectorizations
(33500, 5) (33500,)
(16500, 5) (16500,)
[array(['dr', 'mr', 'mrs', 'ms', 'teacher'], dtype=object)]
====================================================================================================
```

In [13]:

```
# 3. project_grade_category
vectorizer = OneHotEncoder(sparse=False,handle_unknown='ignore')               # initialize One Hot En
coder.
vectorizer.fit(X_train['project_grade_category'].values.reshape(-1,1))                  # fit has to h
appen only on train data.

# use the vectorizer to convert categories of project_grade_category to vector
X_train_project_grade_category_ohe = vectorizer.transform(X_train['project_grade_category'].values.resh
ape(-1,1))
X_test_project_grade_category_ohe = vectorizer.transform(X_test['project_grade_category'].values.reshap
e(-1,1))

print("After vectorizations")
print(X_train_project_grade_category_ohe.shape, y_train.shape)
print(X_test_project_grade_category_ohe.shape, y_test.shape)
print(vectorizer.categories_)
print("="*100)
```

```
After vectorizations
(33500, 4) (33500,)
(16500, 4) (16500,)
[array(['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2'],
      dtype=object)]
====================================================================================================
```

In [14]:

```
# 4. clean_categories
vectorizer = OneHotEncoder(sparse=False,handle_unknown='ignore')               # initialize One Hot En
coder.
vectorizer.fit(X_train['clean_categories'].values.reshape(-1,1))                  # fit has to happen
only on train data.

# use the vectorizer to convert categories of clean_categories to vector
X_train_clean_categories_ohe = vectorizer.transform(X_train['clean_categories'].values.reshape(-1,1))
X_test_clean_categories_ohe = vectorizer.transform(X_test['clean_categories'].values.reshape(-1,1))

print("After vectorizations")
```

```
print(X_train_clean_categories_ohe.shape, y_train.shape)
print(X_test_clean_categories_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 43) (33500,)
(16500, 43) (16500,)
====================================================================================================
```

In [15]:

```
# 5.clean_subcategories
vectorizer = OneHotEncoder(sparse=False,handle_unknown='ignore')                # initialize One Hot En
coder.
vectorizer.fit(X_train['clean_subcategories'].values.reshape(-1,1))                  # fit has to happ
en only on train data.

# use the vectorizer to convert categories of clean_subcategories to vector
X_train_clean_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].values.reshape(-1
,1))
X_test_clean_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].values.reshape(-1,1
))

print("After vectorizations")
print(X_train_clean_subcategories_ohe.shape, y_train.shape)
print(X_test_clean_subcategories_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 340) (33500,)
(16500, 340) (16500,)
====================================================================================================
```

1.4.2 Encoding Numerical Featues: (using Normalizer)

1. price

In [16]:

```
# https://imgur.com/ldZA1zg
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))
X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1)).reshape(-1,1)
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1)).reshape(-1,1)

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,)
====================================================================================================
```

1.4.3 Encoding Text Features: (using TFIDF)

1. essay

In [17]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
vectorizer.fit(X_train['essay'].values)                                      # fit has to happen
only on train data

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

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
====================================================================================================
```

1.4.4 Encoding Text Features: (using Sentiment Score)

In [18]:

```
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

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

In [19]:

```
# Polarity function to get the sentiment polarity for give sentence.
def Polarity(preprocessed_essays):
  polarity= []                                  # list to store polarity for all sentences.
  sid = SentimentIntensityAnalyzer()
  for sentence in tqdm_notebook(preprocessed_essays):
    scores = sid.polarity_scores(sentence)      # having polarity scores
    l = []                                      # list to store polarity scores for each sente
nce.
    for pol in scores:
      l.append(scores[pol])
    polarity.append(l)
  polarity = np.array(polarity)
  return polarity
```

In [20]:

```
# train and test polarity
train_polarity = Polarity(X_train['essay'])
test_polarity = Polarity(X_test['essay'])
```

In [21]:

```
print(train_polarity.shape,test_polarity.shape)
```

```
(33500, 4) (16500, 4)
```

Set-2

Set 2

1.4.5 Encoding Text Features : (using TFIDF W2V)

1. **essay**

In [22]:

```
# glove vector file
with open('/content/drive/My Drive/11_Applying Decision Trees on Donors choose Dataset/glove_vectors','
rb') as f:
  model =  pickle.load(f)
  glove_words = set(model.keys())
```

In [23]:

```
tfidf_model = TfidfVectorizer()
# fit using trian essay
tfidf_model.fit(X_train['essay'])
# converting word as key and idf values as value.
idf_dict = dict(zip(tfidf_model.get_feature_names(),list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [24]:

```
# function to compute tfidf_w2v
def tfidfW2v(essays):
  tfidf_w2v_vectors = []
  for sentence in tqdm(essays):
    vector = np.zeros(300)
    tf_idf_weight = 0
    for word in sentence.split():
      # check word is present in both glove_words and tfidf_words
      if (word in glove_words) and (word in tfidf_words):
        w2v = model[word]
        tf_idf = idf_dict[word]*(sentence.count(word)/len(sentence.split()))
        vector += (w2v * tf_idf)
        tf_idf_weight += tf_idf
    if tf_idf_weight!=0:
      vector/=tf_idf_weight
    tfidf_w2v_vectors.append(vector)
  return tfidf_w2v_vectors
```

In [25]:

```
# computing tfidf_w2v for train and test
tfidfW2v_train = tfidfW2v(X_train['essay'])
tfidfW2v_test = tfidfW2v(X_test['essay'])
```

```
100%|██████████| 33500/33500 [01:12<00:00, 459.62it/s]
100%|██████████| 16500/16500 [00:35<00:00, 463.34it/s]
```

In [26]:

```
print(len(tfidfW2v_train),len(X_train))
print(len(tfidfW2v_test),len(X_test))
```

```
33500 33500
16500 16500
```

In [27]:

```
# converting list into arrays
tfidfW2v_train = np.array(tfidfW2v_train)
tfidfW2v_test = np.array(tfidfW2v_test)
```

In [28]:

```
print(tfidfW2v_train.shape)
print(tfidfW2v_test.shape)
```

```
(33500, 300)
(16500, 300)
```

In [29]:

```
from scipy.sparse import  csr_matrix
```

In [30]:

```
# converting into compressed sparse matrix
tfidfW2v_train = csr_matrix(tfidfW2v_train)
tfidfW2v_test  = csr_matrix(tfidfW2v_test)
```

1.4.6 : Concatinating all features:

1. **set 1**
2. **set 2**

In [31]:

```
# set-1
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_state_ohe,X_train_teacher_prefix_ohe,X_train_project_grade_category_ohe,
               X_train_clean_categories_ohe,X_train_clean_subcategories_ohe,X_train_price_norm,X_train_
essay_tfidf,
               train_polarity)).tocsr()
X_te_tfidf = hstack((X_test_state_ohe,X_test_teacher_prefix_ohe,X_test_project_grade_category_ohe,
               X_test_clean_categories_ohe,X_test_clean_subcategories_ohe,X_test_price_norm,X_test_essa
y_tfidf,
               test_polarity)).tocsr()

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, 5448) (33500,)
(16500, 5448) (16500,)
====================================================================================================
```

In [32]:

```
# set-2
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf_w2v = hstack((X_train_state_ohe,X_train_teacher_prefix_ohe,X_train_project_grade_category_oh
e,
               X_train_clean_categories_ohe,X_train_clean_subcategories_ohe,X_train_price_norm,tfidfW2v
_train,
               train_polarity)).tocsr()
X_te_tfidf_w2v = hstack((X_test_state_ohe,X_test_teacher_prefix_ohe,X_test_project_grade_category_ohe,
               X_test_clean_categories_ohe,X_test_clean_subcategories_ohe,X_test_price_norm,tfidfW2v_te
st,
               test_polarity)).tocsr()

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, 748) (33500,)
(16500, 748) (16500,)
====================================================================================================
```

Model on set-1

In [33]:

```python
parameters = {'max_depth':[1,5,10,15],'min_samples_split':[5,10,100,500]}
```

Grid Search (decision tree on set-1)

In [34]:

```python
# initializing decision tree classifier
dtree = DecisionTreeClassifier(random_state=42)
# using GridSearch with given parameters and "roc_auc" as a metric - 3 fold cross validation.
clf = GridSearchCV(dtree,parameters,scoring='roc_auc',n_jobs=-1,cv=3,return_train_score=True)
```

In [35]:

```python
clf.fit(X_tr_tfidf,y_train)
results = pd.DataFrame.from_dict(clf.cv_results_)
results.head(5)
```

Out[35]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_split	params
0	0.754487	0.046261	0.018494	0.001107	1	5	{'max_depth': 1, 'min_samples_split': 5}
1	0.701310	0.007480	0.018553	0.000958	1	10	{'max_depth': 1, 'min_samples_split': 10}
2	0.692218	0.035558	0.018056	0.001505	1	100	{'max_depth': 1, 'min_samples_split': 100}
3	0.698570	0.037080	0.018893	0.001338	1	500	{'max_depth': 1, 'min_samples_split': 500}
4	3.393957	0.072439	0.016626	0.002794	5	5	{'max_depth': 5, 'min_samples_split': 5}

In [36]:

```python
results = results.sort_values(['param_min_samples_split','param_max_depth'])
```

In [37]:

```python
x_min_samples = results['param_min_samples_split']
y_max_depth   = results['param_max_depth']
z_train_auc   = results['mean_train_score']
z_cv_auc      = results['mean_test_score']
```

3-D scatter plot

In [38]:

```
# setting render for google colab
import plotly.io as pio
pio.renderers.default ='colab'
```

In [39]:

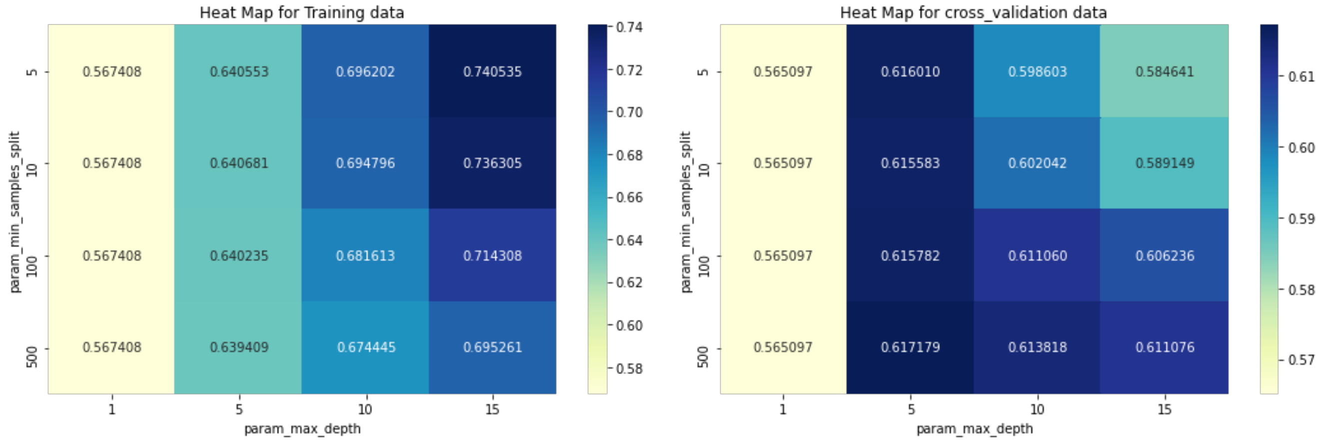
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x_min_samples,y=y_max_depth,z=z_train_auc, name = 'train')
trace2 = go.Scatter3d(x=x_min_samples,y=y_max_depth,z=z_cv_auc, name = 'Cross-validation')
data = [trace1, trace2]

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

fig = go.Figure(data=data, layout=layout)
ply.offline.iplot(fig)
```

Heat Maps:

In [40]:

```
# resource: https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
# pivot_table is used to create a spreadsheet-style table as a DataFrame.

df1 = pd.concat([x_min_samples,y_max_depth,z_train_auc],axis=1).reset_index()
df1.drop(columns=['index'],inplace=True)
df2 = pd.pivot_table(df1,values='mean_train_score',index='param_min_samples_split',columns='param_max_d
epth')

df3 = pd.concat([x_min_samples,y_max_depth,z_cv_auc],axis=1).reset_index()
df3.drop(columns=['index'],inplace=True)
df4 = pd.pivot_table(df3,values='mean_test_score',index='param_min_samples_split',columns='param_max_de
pth')


fig,ax =plt.subplots(1,2,figsize=(15,5))
sns.heatmap(df2,cmap='YlGnBu',annot=True,fmt='f',ax=ax[0]).set_title('Heat Map for Training data')
sns.heatmap(df4,cmap='YlGnBu',annot=True,fmt='f',ax=ax[1]).set_title('Heat Map for cross_validation dat
a')
```

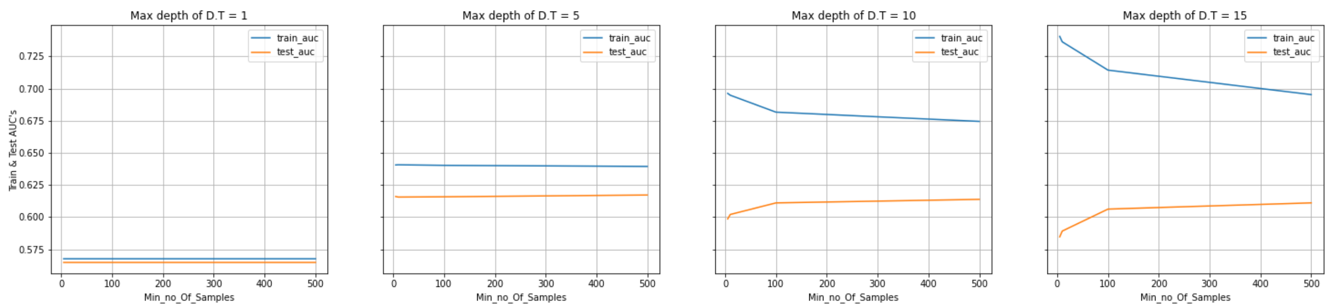
```
plt.tight_layout()
```



Train model with best parameters:

In [41]:

```
# finding best hyper-parameters:
# results:
x_min_samples = results['param_min_samples_split']
y_max_depth   = results['param_max_depth']
z_train_auc   = results['mean_train_score']
z_cv_auc      = results['mean_test_score']

# sorted unique depths and samples
depths = np.sort(y_max_depth.unique())
samples = np.sort(x_min_samples.unique())

# ploting subplots to get an clear idea to select best hpyer-parameters.
fig,ax = plt.subplots(1,4,figsize=(25,5),sharey=True,sharex=True)
for i,depth in enumerate(depths):
  train_auc = []
  test_auc  = []
  for sample in samples:
    train_auc.append(results[(results['param_min_samples_split']==sample) & (results['param_max_depth']
==depth)]['mean_train_score'])
    test_auc.append(results[(results['param_min_samples_split']==sample) & (results['param_max_depth']=
=depth)]['mean_test_score'])

  ax[i].plot(samples,train_auc,label="train_auc")
  ax[i].plot(samples,test_auc,label="test_auc")
  ax[i].set_xlabel('Min_no_Of_Samples')
  #ax[i].set_ylabel("Train & Test AUC's ")
  ax[i].set_title("Max depth of D.T = "+ str(depth))
  ax[i].grid()
  ax[i].legend()
fig.text(0.1,0.5,"Train & Test AUC's",va='center',rotation='vertical')
plt.show()
```



Observations:

```
from above subplots, we can see that for the best Hyperparameters are
     max_depth         = 10
     min_no_of_samples = 500
   both train Auc and test Auc are closer which overcomes the overfit
   (at max_depth = 15) and underfit (at max_depth = 1)
```

In [42]:

```python
# training decison Tree classifier with best hyperparameters:
# https://stackoverflow.com/questions/37522191/how-to-balance-classification-using-decisiontreeclassifier/37522252#37522252

clf = DecisionTreeClassifier(max_depth=10,min_samples_split=500,random_state=42,class_weight ='balanced')
clf.fit(X_tr_tfidf,y_train)
```

Out[42]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

In [43]:

```python
# classes present order
print(clf.classes_)
```

```
[0 1]
```

In [44]:

```python
# predicted probability scores of test data
proba     = clf.predict_proba(X_tr_tfidf)
# proba contains both classes probabilities, hence we need to pick class-1 proba scores
prob_train = proba[:,1]

# predicted probability scores of train data
proba     = clf.predict_proba(X_te_tfidf)
prob_test = proba[:,1]
```

In [45]:

```python
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score
from sklearn.metrics import roc_auc_score
auc_train = roc_auc_score(y_train,prob_train)
auc_test = roc_auc_score(y_test,prob_test)
auc_test_model1 = auc_test
print(" Train auc = " + str(auc_train),'\n',"Test auc = "+ str(auc_test))
```

```
 Train auc = 0.716802584927451
 Test auc = 0.6310570833682482
```

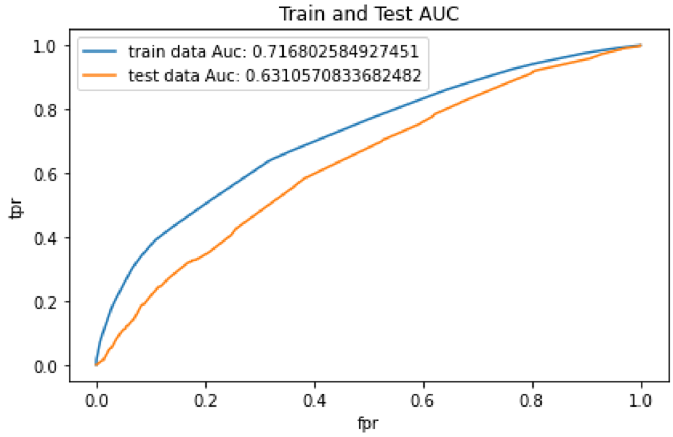
In [46]:

```python
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn import metrics
fpr_tr, tpr_tr, thresholds_tr = metrics.roc_curve(y_train,prob_train, pos_label=1)
fpr_te, tpr_te, thresholds_te = metrics.roc_curve(y_test,prob_test, pos_label=1)
```

AUC -Model-1

In [47]:

```
plt.plot(fpr_tr,tpr_tr,label="train data Auc: "+str(auc_train))
plt.plot(fpr_te,tpr_te,label="test data Auc: "+str(auc_test))
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("Train and Test AUC")
plt.legend()
plt.tight_layout()
plt.show()
```



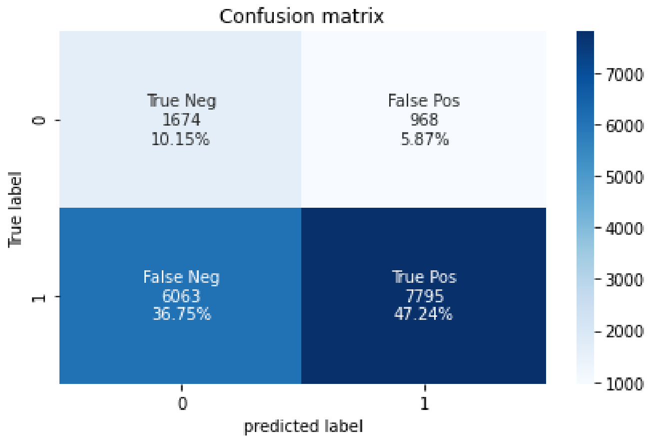
Confusion matrix:

In [48]:

```
# predicted
y_predicted = clf.predict(X_te_tfidf)
mat = confusion_matrix(y_test,y_predicted)
```

In [49]:

```
#  https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                mat.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     mat.flatten()/np.sum(mat)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(mat, annot=labels, fmt='', cmap='Blues').set_title("Confusion matrix")
plt.ylabel("True label")
plt.xlabel("predicted label")
plt.tight_layout()
```



Word cloud

In [50]:

```
pip install wordcloud
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.6/dist-packages (1.5.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from wordcloud) (7.0.0
)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.6/dist-packages (from wordcloud)
(1.19.4)
```

In [51]:

```
from wordcloud import WordCloud, STOPWORDS
```

In [52]:

```
# function to get false positive indexes :
def returnIndexOfFP(y_test,y_predicted):
  indexOfFP = []
  for index in range(len(y_test)):
    if y_test[index]==0 and y_predicted[index]==1:
      indexOfFP.append(index)
  return indexOfFP
```

In [53]:

```
indexOfFP = returnIndexOfFP(y_test,y_predicted)
len(indexOfFP)
```

Out[53]:

```
968
```

In [54]:

```
# test data false positive essay words:
fp = X_test.iloc[indexOfFP,[3,6,7]]
fp.head(2)
```

Out[54]:

	teacher_number_of_previously_posted_projects	essay	price
46461	4	play often talked relief serious learning but ...	224.98
33198	0	this first year teaching i excited special edu...	88.73

In [55]:

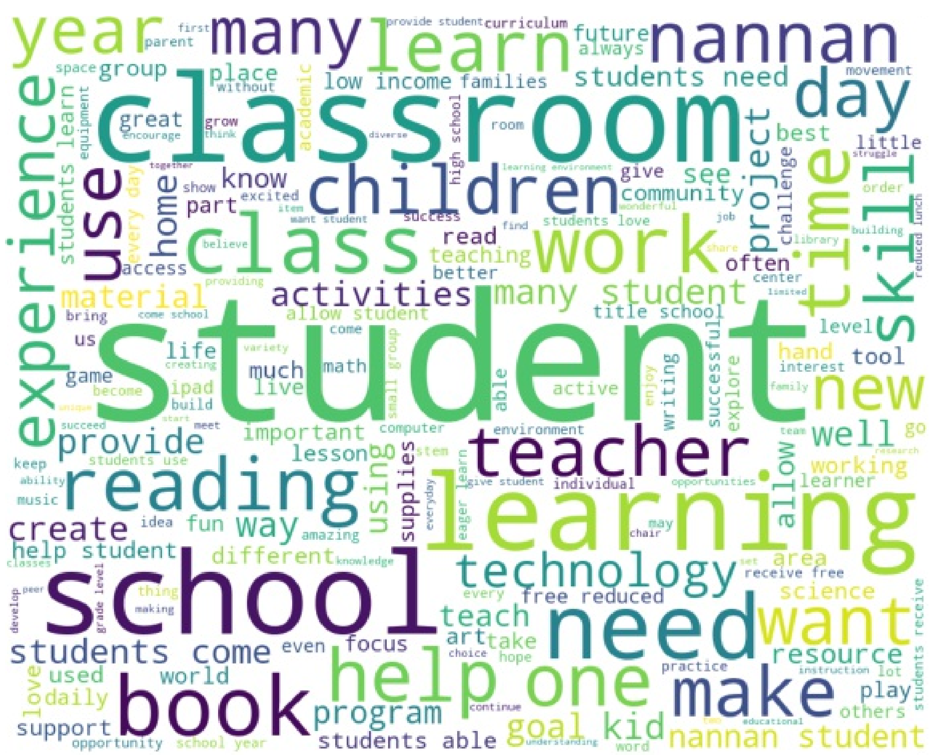
```
# https://www.geeksforgeeks.org/generating-word-cloud-python/
"""
Word Cloud is a data visualization technique used for representing text data in which the
size of each word indicates its frequency or importance. Significant textual data points can
be highlighted using a word cloud.
Word clouds are widely used for analyzing data from social network websites.
"""
# generating word cloud
fp_essays = fp['essay']
comment_words = ''
for sentence in fp_essays:
  val = str(sentence)
  tokens = val.split()
```

```
  for i in range(len(tokens)):
    tokens[i]=tokens[i].lower()

  comment_words+= " ".join(tokens)+ " "

# stopwords from wordcloud
stopwords = set(STOPWORDS)
```

In [56]:

```
wordcloud = WordCloud(width = 1000, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.title("word Cloud\n")
plt.tight_layout(pad = 0)
plt.show()
```

word Cloud



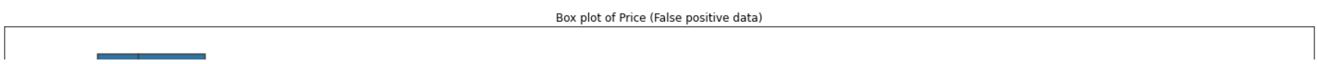
In [57]:

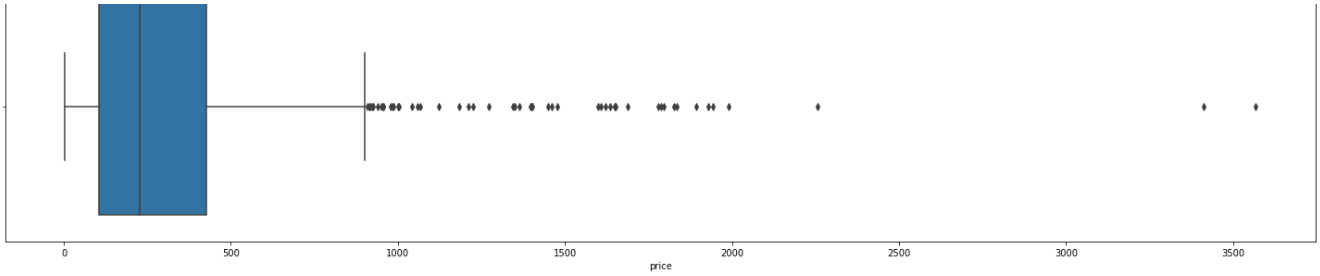
```
# Plot the box plot with the `price` of these `false positive data points`
# Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data po
ints`
```

Box Plot of `price` in FP data

In [58]:

```
plt.figure(figsize=(20,5))
sns.boxplot(fp['price'],).set_title("Box plot of Price (False positive data)")
plt.tight_layout()
```





In [59]:

```
q1 = np.quantile(fp['price'],.25)
q2 = np.quantile(fp['price'],.50)
q3 = np.quantile(fp['price'],.75)
print("Q1 quantile (25th percentile) : " + str(q1))
print("Q2 quantile (50th percentile) : " + str(q2))
print("Q3 quantile (75th percentile) : " + str(q3))
```

```
Q1 quantile (25th percentile) : 106.36250000000001
Q2 quantile (50th percentile) : 226.71500000000003
Q3 quantile (75th percentile) : 425.5325
```

Observations:

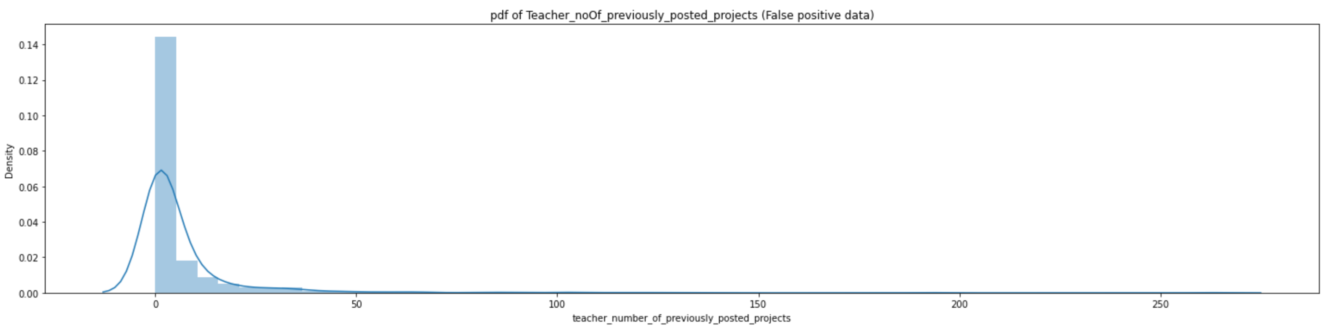
1. **firstly false positive means, in our Donors choose dataset, projects which are actually rejected are accepted by our model.**
2. **So, to find what percentage of our prices falls in which range.**

```
here from boxplot,we can see Quartile1,Quartile2,Quartile3
```

Pdf of `teacher_number_of_previously_posted_projects` in FP data

In [60]:

```
fig ,ax = plt.subplots(1,1,figsize=(20,5))
sns.distplot(fp['teacher_number_of_previously_posted_projects']).set_title("pdf of Teacher_noOf_previou
sly_posted_projects (False positive data)")
plt.tight_layout()
```



In [61]:

```
import statistics
from scipy import stats
print("median = " +str(statistics.median(fp['teacher_number_of_previously_posted_projects'])))
print("Medain Absolute Deviation = " +str(stats.median_absolute_deviation(fp['teacher_number_of_previou
sly_posted_projects'])))
```

```
median = 1.0
Medain Absolute Deviation = 1.4826
```

Model on set-2

Grid Search (decision tree on set-2)

In [62]:

```
# initializing decision tree classifier
dtree = DecisionTreeClassifier(random_state=42)
# using GridSearch with given parameters and "roc_auc" as a metric - 3 fold cross validation.
clf = GridSearchCV(dtree,parameters,scoring='roc_auc',n_jobs=-1,cv=3,return_train_score=True)
```

In [63]:

```
print(X_tr_tfidf_w2v.shape,y_train.shape)
```

```
(33500, 748) (33500,)
```

In [64]:

```
clf.fit(X_tr_tfidf_w2v,y_train)
results = pd.DataFrame.from_dict(clf.cv_results_)
results.head(5)
```

Out[64]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_split	params
0	1.667749	0.122172	0.024653	0.002510	1	5	{'max_depth': 1, 'min_samples_split': 5}
1	1.512677	0.028097	0.024654	0.002113	1	10	{'max_depth': 1, 'min_samples_split': 10}
2	1.616800	0.057169	0.025938	0.000364	1	100	{'max_depth': 1, 'min_samples_split': 100}
3	1.615494	0.043024	0.025993	0.001767	1	500	{'max_depth': 1, 'min_samples_split': 500}
4	8.371070	0.032533	0.029619	0.001346	5	5	{'max_depth': 5, 'min_samples_split': 5}

In [65]:

```
results = results.sort_values(['param_min_samples_split','param_max_depth'])
```

In [66]:

```
x_min_samples = results['param_min_samples_split']
y_max_depth   = results['param_max_depth']
z_train_auc   = results['mean_train_score']
z_cv_auc      = results['mean_test_score']
```

3-D Scatter plot

In [67]:

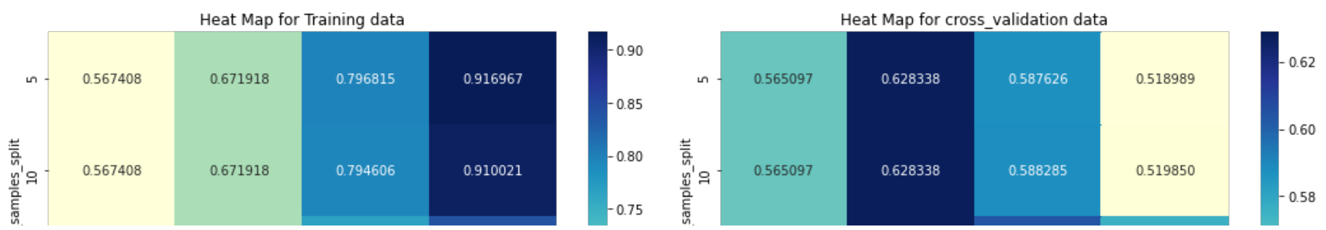
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x_min_samples,y=y_max_depth,z=z_train_auc, name = 'train')
trace2 = go.Scatter3d(x=x_min_samples,y=y_max_depth,z=z_cv_auc, name = 'Cross-validation')
data = [trace1, trace2]
```

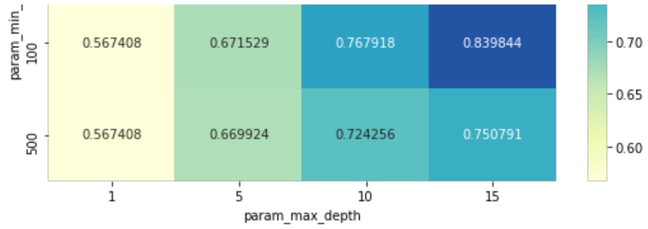
```
data = [trace1, trace2]

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

fig = go.Figure(data=data, layout=layout)
ply.offline.iplot(fig)
```

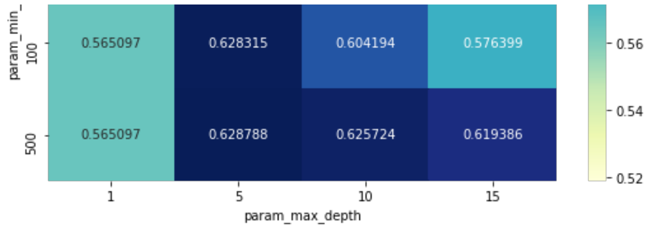
Heat Maps:

In [68]:

```
# resource: https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
# pivot_table is used to create a spreadsheet-style table as a DataFrame.

df1 = pd.concat([x_min_samples,y_max_depth,z_train_auc],axis=1).reset_index()
df1.drop(columns=['index'],inplace=True)
df2 = pd.pivot_table(df1,values='mean_train_score',index='param_min_samples_split',columns='param_max_d
epth')

df3 = pd.concat([x_min_samples,y_max_depth,z_cv_auc],axis=1).reset_index()
df3.drop(columns=['index'],inplace=True)
df4 = pd.pivot_table(df3,values='mean_test_score',index='param_min_samples_split',columns='param_max_de
pth')


fig,ax =plt.subplots(1,2,figsize=(15,5))
sns.heatmap(df2,cmap='YlGnBu',annot=True,fmt='f',ax=ax[0]).set_title('Heat Map for Training data')
sns.heatmap(df4,cmap='YlGnBu',annot=True,fmt='f',ax=ax[1]).set_title('Heat Map for cross_validation dat
a')
plt.tight_layout()
```



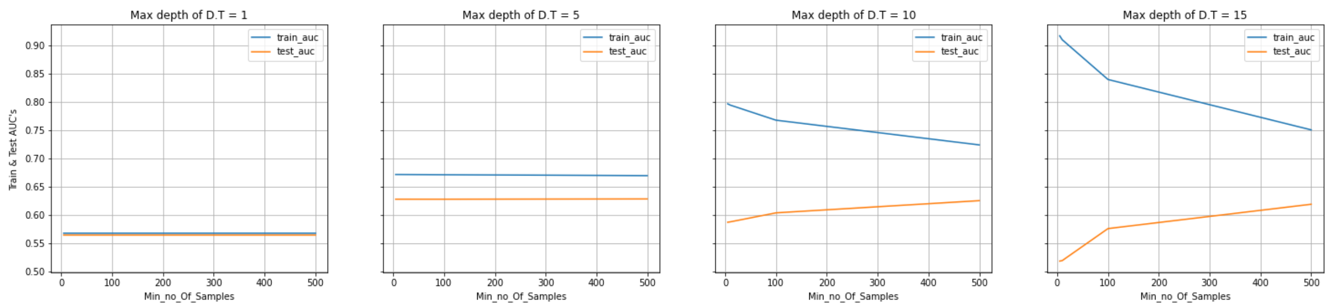




Train model with best parameters:

In [69]:

```
# finding best hyper-parameters:
# results:
x_min_samples = results['param_min_samples_split']
y_max_depth   = results['param_max_depth']
z_train_auc   = results['mean_train_score']
z_cv_auc      = results['mean_test_score']

# sorted unique depths and samples
depths = np.sort(y_max_depth.unique())
samples = np.sort(x_min_samples.unique())

# ploting subplots to get an clear idea to select best hpyer-parameters.
fig,ax = plt.subplots(1,4,figsize=(25,5),sharey=True,sharex=True)
for i,depth in enumerate(depths):
  train_auc = []
  test_auc  = []
  for sample in samples:
    train_auc.append(results[(results['param_min_samples_split']==sample) & (results['param_max_depth']
==depth)]['mean_train_score'])
    test_auc.append(results[(results['param_min_samples_split']==sample) & (results['param_max_depth']=
=depth)]['mean_test_score'])

  ax[i].plot(samples,train_auc,label="train_auc")
  ax[i].plot(samples,test_auc,label="test_auc")
  ax[i].set_xlabel('Min_no_Of_Samples')
  #ax[i].set_ylabel("Train & Test AUC's ")
  ax[i].set_title("Max depth of D.T = "+ str(depth))
  ax[i].grid()
  ax[i].legend()
fig.text(0.1,0.5,"Train & Test AUC's",va='center',rotation='vertical')
plt.show()
```



Observations:

```
from above subplots, we can see that for the best Hyperparameters are
     max_depth         = 10
     min_no_of_samples = 500
   both train Auc and test Auc are closer which overcomes the overfit
   (at max_depth = 15) and underfit (at max_depth = 1)
```

In [70]:

```
# training decison Tree classifier with best hyperparameters:
```

```
# https://stackoverflow.com/questions/37522191/how-to-balance-classification-using-decisiontreeclassifi
er/37522252#37522252
clf = DecisionTreeClassifier(max_depth=10,min_samples_split=500,random_state=42,class_weight ='balanced
')
clf.fit(X_tr_tfidf_w2v,y_train)
```

Out[70]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

In [71]:

```
# predicted probability scores of test data
proba     = clf.predict_proba(X_tr_tfidf_w2v)
# proba contains both classes probabilities, hence we need to pick class-1 proba scores
prob_train = proba[:,1]

# predicted probability scores of train data
proba     = clf.predict_proba(X_te_tfidf_w2v)
prob_test = proba[:,1]
```

In [72]:

```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics
.roc_auc_score
from sklearn.metrics import roc_auc_score
auc_train = roc_auc_score(y_train,prob_train)
auc_test = roc_auc_score(y_test,prob_test)
auc_test_model2 = auc_test
print(" Train auc = " + str(auc_train),'\n',"Test auc = "+ str(auc_test))
```

```
 Train auc = 0.7365176287739299
 Test auc = 0.6201153606347238
```

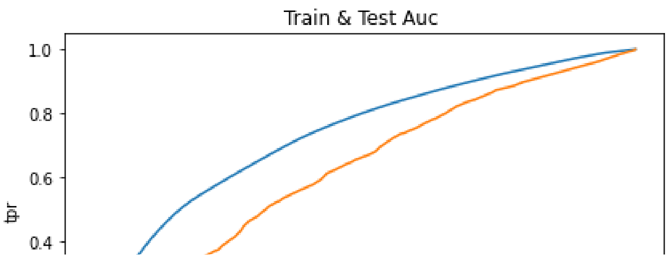
In [73]:

```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn import metrics
fpr_tr, tpr_tr, thresholds_tr = metrics.roc_curve(y_train,prob_train, pos_label=1)
fpr_te, tpr_te, thresholds_te = metrics.roc_curve(y_test,prob_test, pos_label=1)
```

AUC-Model-2

In [74]:

```
plt.plot(fpr_tr,tpr_tr,label="train data Auc: "+str(auc_train))
plt.plot(fpr_te,tpr_te,label="test data Auc: "+str(auc_test))
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title("Train & Test Auc ")
plt.legend()
plt.tight_layout()
```



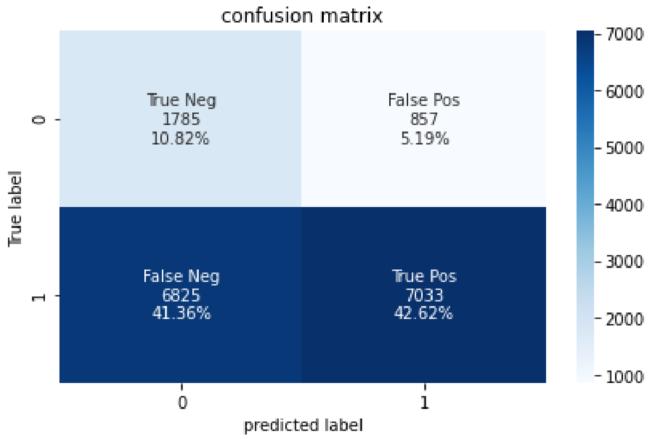
Confusion matrix:

In [75]:

```
# predicted
y_predicted = clf.predict(X_te_tfidf_w2v)
mat = confusion_matrix(y_test,y_predicted)
```

In [76]:

```
#  https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                mat.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     mat.flatten()/np.sum(mat)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(mat, annot=labels, fmt='', cmap='Blues').set_title("confusion matrix")
plt.ylabel("True label")
plt.xlabel("predicted label")
plt.tight_layout()
```



Word cloud

In [77]:

```
indexOfFP = returnIndexOfFP(y_test,y_predicted)
len(indexOfFP)
```

Out[77]:

```
857
```

In [78]:

```
# test data false positive essay words:
fp = X_test.iloc[indexOfFP,[3,6,7]]
fp.head(2)
```

Out[78]:

	teacher_number_of_previously_posted_projects	essay	price
44244	17	my students low income areas need exposure sev...	383.99
33198	0	this first year teaching i excited special edu...	88.73

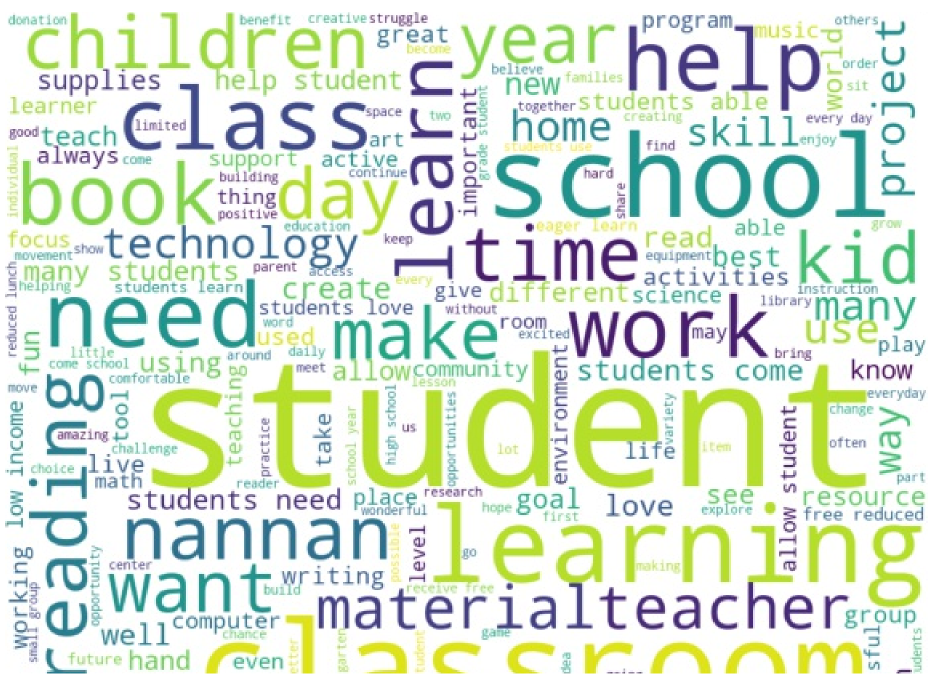
In [79]:

```
# https://www.geeksforgeeks.org/generating-word-cloud-python/
"""
Word Cloud is a data visualization technique used for representing text data in which the
size of each word indicates its frequency or importance. Significant textual data points can
be highlighted using a word cloud.
Word clouds are widely used for analyzing data from social network websites.
"""
# generating word cloud
fp_essays = fp['essay']
comment_words = ''
for sentence in fp_essays:
  val = str(sentence)
  tokens = val.split()
  for i in range(len(tokens)):
    tokens[i]=tokens[i].lower()

  comment_words+= " ".join(tokens)+ " "

# stopwords from wordcloud
stopwords = set(STOPWORDS)
```

In [80]:

```
wordcloud = WordCloud(width = 1000, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.title("Word Cloud\n")
plt.tight_layout(pad = 0)
plt.show()
```

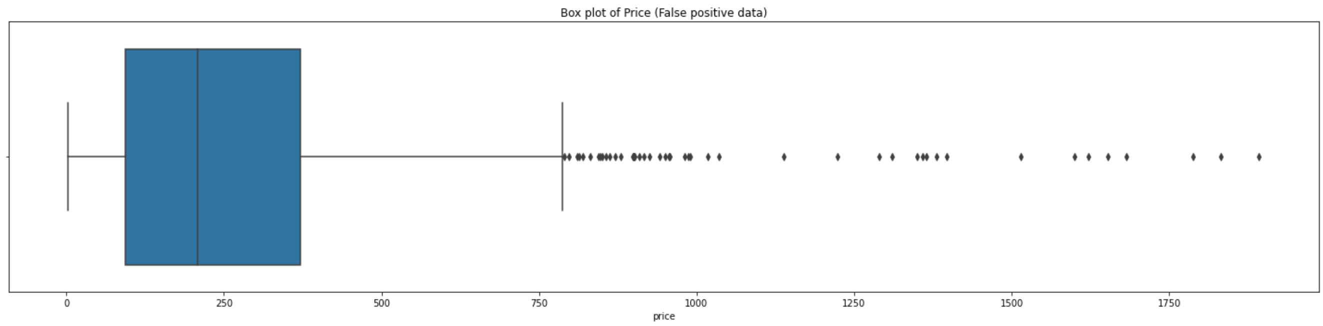
Word Cloud



Box Plot of `price` in FP data

In [81]:

```
plt.figure(figsize=(20,5))
sns.boxplot(fp['price'],).set_title("Box plot of Price (False positive data)")
plt.tight_layout()
```



In [82]:

```
q1 = np.quantile(fp['price'],.25)
q2 = np.quantile(fp['price'],.50)
q3 = np.quantile(fp['price'],.75)
print("Q1 quantile (25th percentile) : " + str(q1))
print("Q2 quantile (50th percentile) : " + str(q2))
print("Q3 quantile (75th percentile) : " + str(q3))
```

```
Q1 quantile (25th percentile) : 94.25
Q2 quantile (50th percentile) : 208.68
Q3 quantile (75th percentile) : 371.89
```

Observations:

1. firstly false positive means, in our Donors choose dataset, projects which are actually rejected are accepted by our model.
2. So, to find what percentage of our prices falls in which range.

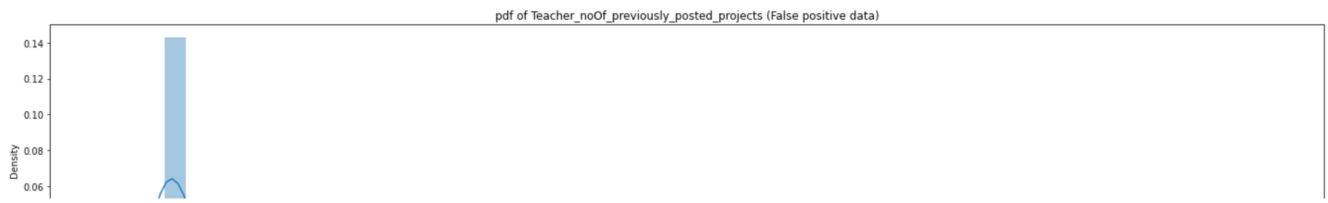
here from boxplot,

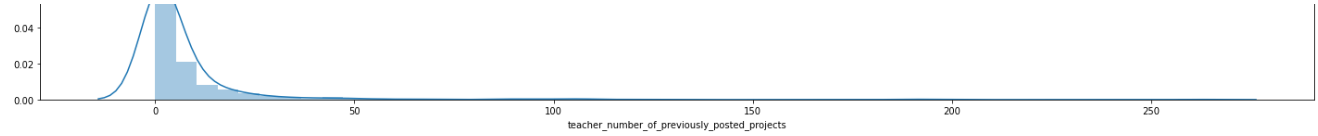
```
we can see Quartile1,Quartile2,Quartile3
```

Pdf of `teacher_number_of_previously_posted_projects` in FP data

In [83]:

```
fig ,ax = plt.subplots(1,1,figsize=(20,5))
sns.distplot(fp['teacher_number_of_previously_posted_projects']).set_title("pdf of Teacher_noOf_previou
sly_posted_projects (False positive data)")
plt.tight_layout()
```





In [84]:

```
import statistics
from scipy import stats
print("median = "+str(statistics.median(fp['teacher_number_of_previously_posted_projects'])))
print("median Absolute deviation = "+str(stats.median_absolute_deviation(fp['teacher_number_of_previous
ly_posted_projects'])))
```

```
median = 2
median Absolute deviation = 2.9652
```

Observations:

1. **set-1 vectorization based model gives more test AUC than set-2 vectorization based model.**

In [84]:

```

```

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 [85]:

```
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[85]:

```
True
```

In [86]:

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

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students with t
he biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelligence
s i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of different bac
kgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a caring com
munity of successful \
learners which can be seen through collaborative student project based learning in and out of the class
room kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice a sk
ill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kinderg
arten 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 real food i wil
l 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 th
e food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project would exp
and our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade 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)

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

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

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

1. Decision Tree

In [87]:

```
parameters = {'max_depth':[1,5,10,15],'min_samples_split':[5,10,100,500]}
```

In [88]:

```
# set-1 is already prepared as part of task-1, we use it here.

# initializing decision tree classifier
dtree = DecisionTreeClassifier(random_state=42)
# using GridSearch with given parameters and "roc_auc" as a metric - 3 fold cross validation.
clf = GridSearchCV(dtree,parameters,scoring='roc_auc',n_jobs=-1,cv=3,return_train_score=True)
```

In [89]:

```
clf.fit(X_tr_tfidf,y_train)
```

Out[89]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
```

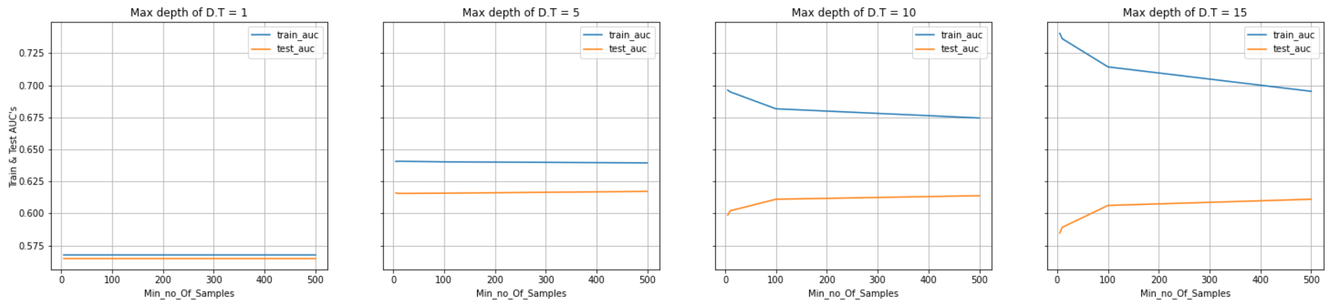
```
                                                    max_features=None,
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    presort='deprecated',
                                                    random_state=42,
                                                    splitter='best'),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 15],
                         'min_samples_split': [5, 10, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [90]:

```
results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_min_samples_split','param_max_depth'])
```

In [91]:

```
# finding best hyper-parameters:
# results:
x_min_samples = results['param_min_samples_split']
y_max_depth   = results['param_max_depth']
z_train_auc   = results['mean_train_score']
z_cv_auc      = results['mean_test_score']

# sorted unique depths and samples
depths = np.sort(y_max_depth.unique())
samples = np.sort(x_min_samples.unique())

# ploting subplots to get an clear idea to select best hpyer-parameters.
fig,ax = plt.subplots(1,4,figsize=(25,5),sharey=True,sharex=True)
for i,depth in enumerate(depths):
  train_auc = []
  test_auc  = []
  for sample in samples:
    train_auc.append(results[(results['param_min_samples_split']==sample) & (results['param_max_depth']
==depth)]['mean_train_score'])
    test_auc.append(results[(results['param_min_samples_split']==sample) & (results['param_max_depth']=
=depth)]['mean_test_score'])

  ax[i].plot(samples,train_auc,label="train_auc")
  ax[i].plot(samples,test_auc,label="test_auc")
  ax[i].set_xlabel('Min_no_Of_Samples')
  #ax[i].set_ylabel("Train & Test AUC's ")
  ax[i].set_title("Max depth of D.T = "+ str(depth))
  ax[i].grid()
  ax[i].legend()
fig.text(0.1,0.5,"Train & Test AUC's",va='center',rotation='vertical')
plt.show()
```



Observations:

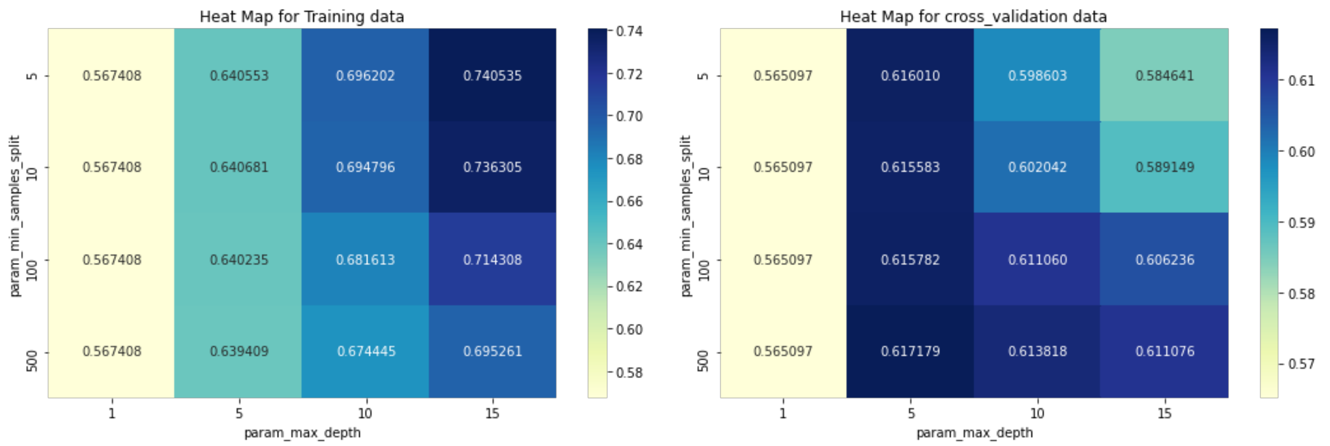
from above subplots, we can see that for the best Hyperparameters are

```
      max_depth         = 10
      min_no_of_samples = 500
    both train Auc and test Auc are closer which overcomes the overfit
    (at max_depth = 15) and underfit (at max_depth = 1)
```

Heat Maps:

In [92]:

```python
# resource: https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
# pivot_table is used to create a spreadsheet-style table as a DataFrame.

df1 = pd.concat([x_min_samples,y_max_depth,z_train_auc],axis=1).reset_index()
df1.drop(columns=['index'],inplace=True)
df2 = pd.pivot_table(df1,values='mean_train_score',index='param_min_samples_split',columns='param_max_d
epth')

df3 = pd.concat([x_min_samples,y_max_depth,z_cv_auc],axis=1).reset_index()
df3.drop(columns=['index'],inplace=True)
df4 = pd.pivot_table(df3,values='mean_test_score',index='param_min_samples_split',columns='param_max_de
pth')


fig,ax =plt.subplots(1,2,figsize=(15,5))
sns.heatmap(df2,cmap='YlGnBu',annot=True,fmt='f',ax=ax[0]).set_title('Heat Map for Training data')
sns.heatmap(df4,cmap='YlGnBu',annot=True,fmt='f',ax=ax[1]).set_title('Heat Map for cross_validation dat
a')
plt.tight_layout()
```



Feature Importance

In [93]:

```python
## Feature importance:
clf = DecisionTreeClassifier(class_weight='balanced')
clf.fit(X_tr_tfidf,y_train)
```

Out[93]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [94]:

```python
# converting compressed sparse matrix into numpy matrix
X_tr_tfidf_numpyMatrix = X_tr_tfidf.toarray()
X_te_tfidf_numpyMatrix = X_te_tfidf.toarray()
```

In [95]:

```
# change check
print(X_tr_tfidf.shape,X_tr_tfidf_numpyMatrix.shape)
print(X_te_tfidf.shape,X_te_tfidf_numpyMatrix.shape)
```

```
(33500, 5448) (33500, 5448)
(16500, 5448) (16500, 5448)
```

In [96]:

```
# selecting important features:
impFeatures = []
for feature,feauture_imp in  enumerate(clf.feature_importances_):
  if (feauture_imp!=0):
    impFeatures.append(feature)
print("Non-zero Important features are = "+str(len(impFeatures)))
```

```
Non-zero Important features are = 1702
```

In [97]:

```
# extracting important features into consideration
X_tr_tfidf_ImpFeat = X_tr_tfidf_numpyMatrix[:,impFeatures]
X_te_tfidf_ImpFeat = X_te_tfidf_numpyMatrix[:,impFeatures]
print(X_tr_tfidf_ImpFeat.shape)
print(X_te_tfidf_ImpFeat.shape)
```

```
(33500, 1702)
(16500, 1702)
```

In [98]:

```
# training decison Tree classifier with best hyperparameters and important features:
# https://stackoverflow.com/questions/37522191/how-to-balance-classification-using-decisiontreeclassifi
er/37522252#37522252

clf = DecisionTreeClassifier(max_depth=10,min_samples_split=500,random_state=42,class_weight ='balanced
')
clf.fit(X_tr_tfidf_ImpFeat,y_train)
```

Out[98]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

In [99]:

```
# predicted probability scores of test data
proba      = clf.predict_proba(X_te_tfidf_ImpFeat)
# proba contains both classes probabilities, hence we need to pick class-1 proba scores
prob_test = proba[:,1]

# predicted probability scores of train data
proba      = clf.predict_proba(X_tr_tfidf_ImpFeat)
prob_train = proba[:,1]
```

In [100]:

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

```
.roc_auc_score
from sklearn.metrics import roc_auc_score
auc_train = roc_auc_score(y_train,prob_train)
auc_test = roc_auc_score(y_test,prob_test)
print(" Train auc = " + str(auc_train),'\n',"Test auc = "+ str(auc_test))
```

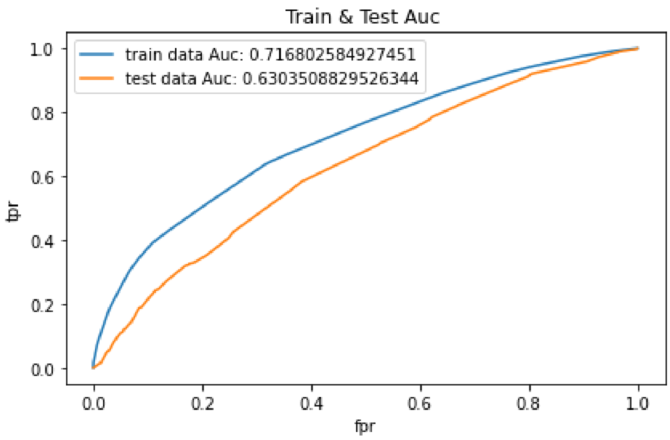
```
 Train auc = 0.716802584927451
 Test auc = 0.6303508829526344
```

In [101]:

```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn import metrics
fpr_tr, tpr_tr, thresholds_tr = metrics.roc_curve(y_train,prob_train, pos_label=1)
fpr_te, tpr_te, thresholds_te = metrics.roc_curve(y_test,prob_test, pos_label=1)
```

In [102]:

```
plt.plot(fpr_tr,tpr_tr,label="train data Auc: "+str(auc_train))
plt.plot(fpr_te,tpr_te,label="test data Auc: "+str(auc_test))
plt.legend()
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title("Train & Test Auc ")
plt.tight_layout()
```



Confusion matrix:

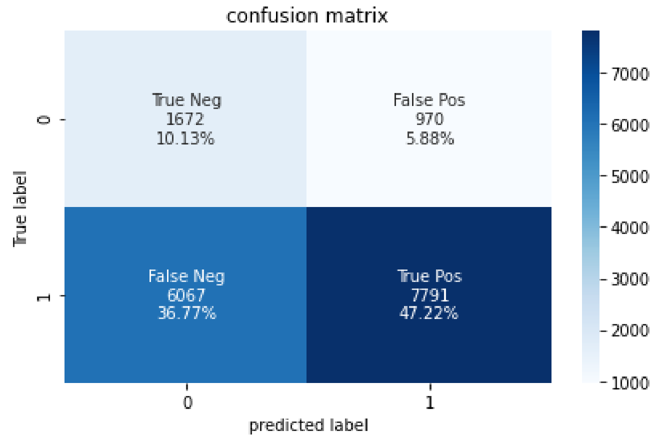
In [103]:

```
# predicted
from sklearn.metrics import confusion_matrix

y_predicted = clf.predict(X_te_tfidf_ImpFeat)
mat = confusion_matrix(y_test,y_predicted)
```

In [104]:

```
#  https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                mat.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     mat.flatten()/np.sum(mat)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(mat, annot=labels, fmt='', cmap='Blues').set_title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("True label")
plt.tight_layout()
```

```
plt.tight_layout()
```



In [105]:

```
# getting indexes of false postive data points
indexOfFP = returnIndexOfFP(y_test,y_predicted)
len(indexOfFP)
```

Out[105]:

970

In [106]:

```
# test data false positive essay words:
fp = X_test.iloc[indexOfFP,[3,6,7]]
fp.head(2)
```

Out[106]:

	teacher_number_of_previously_posted_projects	essay	price
46461	4	play often talked relief serious learning but ...	224.98
33198	0	this first year teaching i excited special edu...	88.73

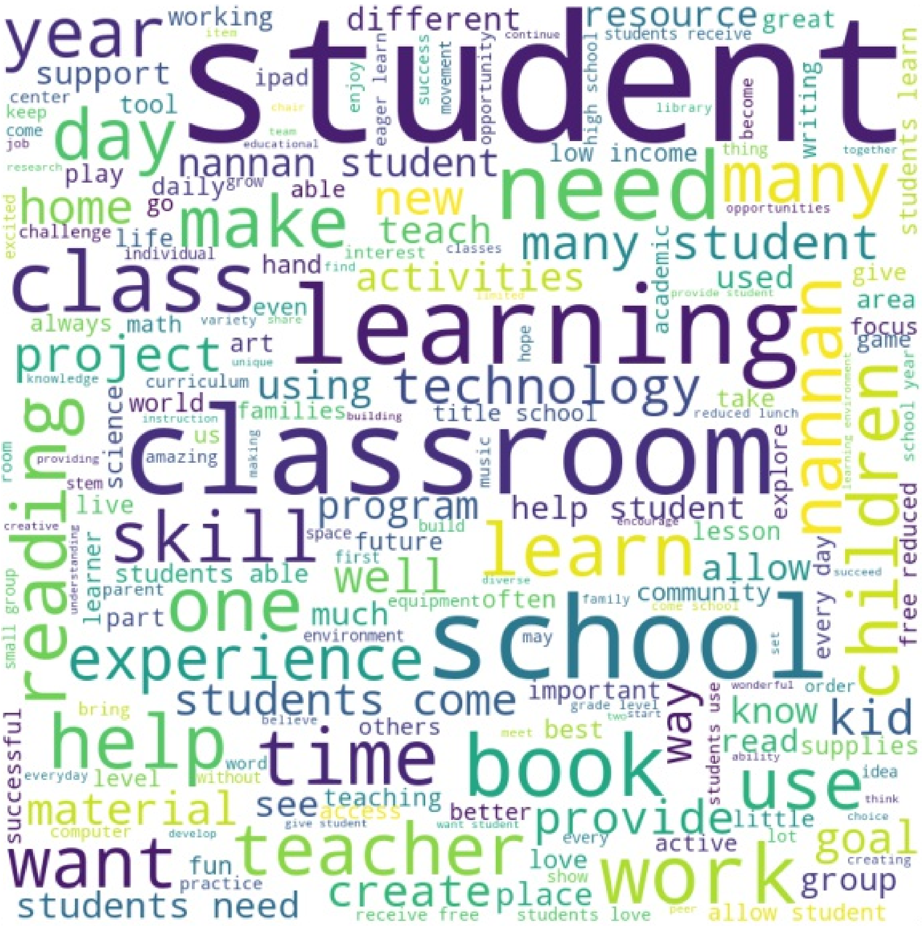
In [107]:

```
# https://www.geeksforgeeks.org/generating-word-cloud-python/
"""
Word Cloud is a data visualization technique used for representing text data in which the
size of each word indicates its frequency or importance. Significant textual data points can
be highlighted using a word cloud.
Word clouds are widely used for analyzing data from social network websites.
"""
# generating word cloud
fp_essays = fp['essay']
comment_words = ''
for sentence in fp_essays:
  val = str(sentence)
  tokens = val.split()
  for i in range(len(tokens)):
    tokens[i]=tokens[i].lower()

  comment_words+= " ".join(tokens)+ " "

# stopwords from wordcloud
stopwords = set(STOPWORDS)
```

In [108]:

```
wordcloud = WordCloud(width = 800, height = 800,
```

```
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# 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 of `price` in FP data

In [109]:

```
plt.figure(figsize=(20,5))
sns.boxplot(fp['price'],).set_title("Box plot of Price (False positive data)")
plt.tight_layout()
```

In [110]:

```
q1 = np.quantile(fp['price'],.25)
q2 = np.quantile(fp['price'],.50)
q3 = np.quantile(fp['price'],.75)
print("Q1 quantile (25th percentile) : " + str(q1))
print("Q2 quantile (50th percentile) : " + str(q2))
print("Q3 quantile (75th percentile) : " + str(q3))
```

```
Q1 quantile (25th percentile) : 106.4825
Q2 quantile (50th percentile) : 225.86
Q3 quantile (75th percentile) : 425.3075
```

Observations:

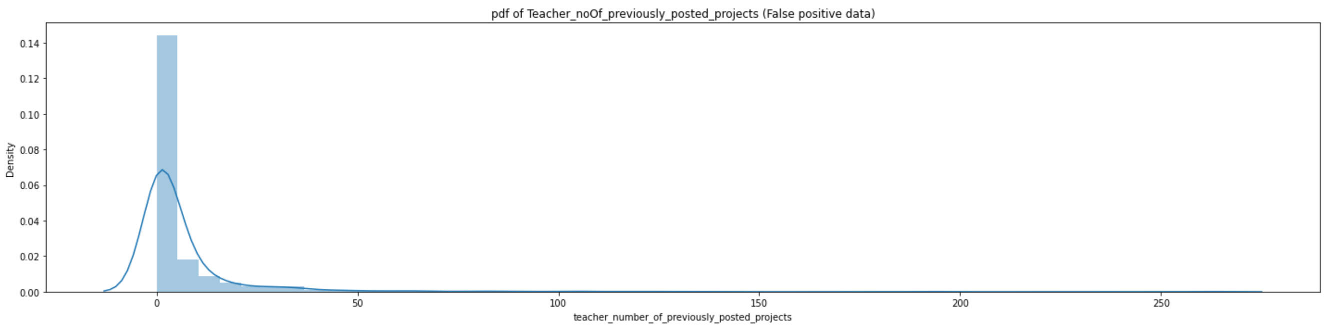
here from boxplot,

we can see Quartile1,Quartile2,Quartile3

Pdf of `teacher_number_of_previously_posted_projects` in FP data

In [111]:

```
fig ,ax = plt.subplots(1,1,figsize=(20,5))
sns.distplot(fp['teacher_number_of_previously_posted_projects']).set_title("pdf of Teacher_noOf_previou
sly_posted_projects (False positive data)")
plt.tight_layout()
```



In [112]:

```
import statistics
from scipy import stats
print("Median = "+ str(statistics.median(fp['teacher_number_of_previously_posted_projects'])))
print("Median Absoulte Deviation = "+str(stats.median_absolute_deviation(fp['teacher_number_of_previous
ly_posted_projects'])))
```

```
Median = 1.0
Median Absoulte Deviation = 1.4826
```

Summary:

In [113]:

```
from tabulate import tabulate
table = [["TFIDF","D.T",500,10,auc_test_model1],["TFIDF_W2v","D.T",500,10,auc_test_model2]]
headers = ["Vectorizer","Model","Min_samples (Hyper parameter)","Depth (Hyper parameter)","AUC"]
print(tabulate(table,headers,tablefmt="grid"))
```

```
+-------------+---------+--------------------------------+---------------------------+----------+
| Vectorizer  | Model   |  Min_samples (Hyper parameter) |   Depth (Hyper parameter) |      AUC |
+=============+=========+================================+===========================+==========+
| TFIDF       | D.T     |                            500 |                        10 | 0.631057 |
+-------------+---------+--------------------------------+---------------------------+----------+
```

```
| TFIDF_W2v    | D.T      |                                 500 |                          10 | 0.620115 |
+--------------+----------+-------------------------------------+-----------------------------+----------+
```