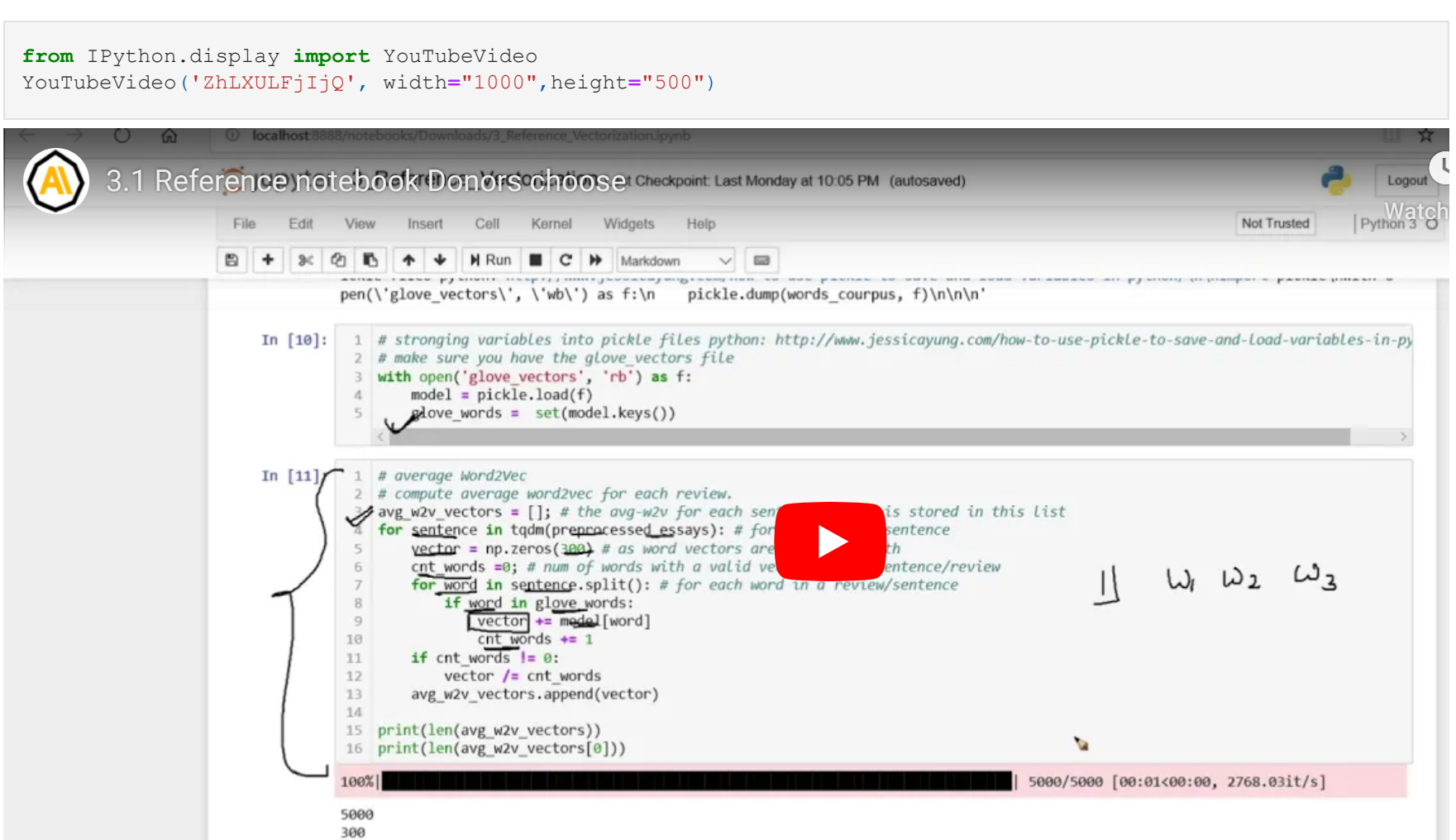
Assignment : DT

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

or else , you can use below code

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

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

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

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

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

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

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

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


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

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


'''
```

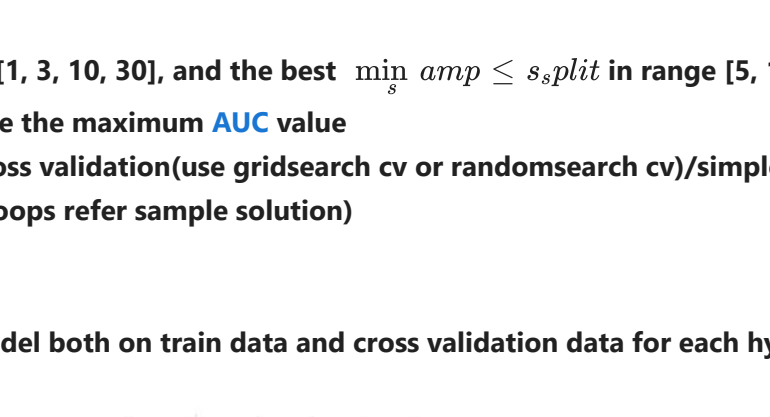
Out[3]:
```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n        word =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

Task - 1

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

   - The hyper paramter tuning (best $depth$ in range [1, 3, 10, 30], and the best $\min_s amp \leq s_s plit$ in range [5, 10, 100, 500])
     - Find the best hyper parameter which will give the maximum AUC value
     - find the best hyper paramter using k-fold cross validation(use gridsearch cv or randomsearch cv)/simple cross validation data(you can write your own for loops refer sample solution)

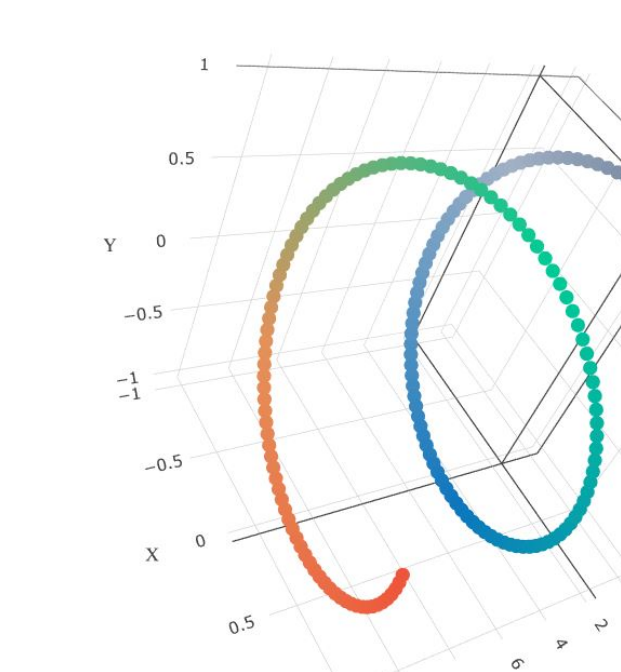
     - Representation of results
       - You need to plot the performance of model both on train data and cross validation data for each hyper



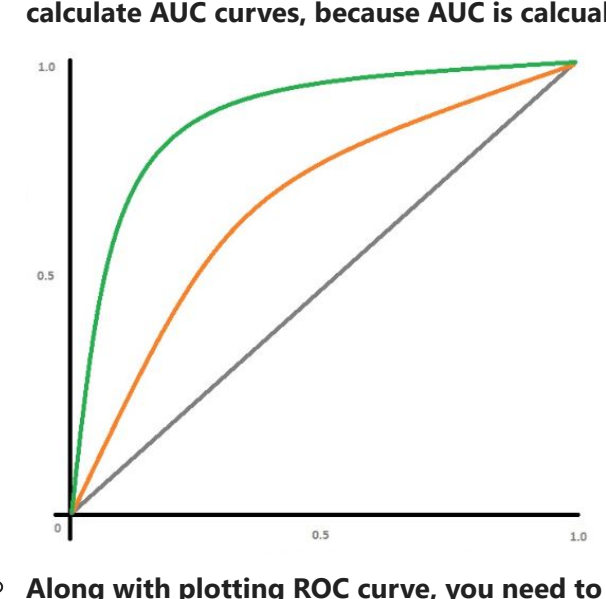
parameter, like shown in the figure
with X-axis as min_sample_split, Y-axis as max_depth, and Z-axis as AUC Score , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

or

       - You need to plot the performance of model both on train data and cross validation data for each hyper



parameter, like shown in the figure seaborn heat maps with rows as min_sample_split, columns as max_depth, and values inside the cell representing AUC Score
       - You choose either of the plotting techniques out of 3d plot or heat map
       - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test. Make sure that you are using predict_proba method to calculate AUC curves, because AUC is calcualted on class probabilities and not on class labels.



       - 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 $false positive data points$

In [ ]:

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

In [5]:
```
import nltk
#nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# import nltk


sid = SentimentIntensityAnalyzer()

sample_sentence_1='I am happy.'
ss_1 = sid.polarity_scores(sample_sentence_1)
print('sentiment score for sentence 1',ss_1)

sample_sentence_2='I am sad.'
ss_2 = sid.polarity_scores(sample_sentence_2)
print('sentiment score for sentence 2',ss_2)

sample_sentence_3='I am going to New Delhi tommorow.'
ss_3 = sid.polarity_scores(sample_sentence_3)
print('sentiment score for sentence 3',ss_3)
```

```
sentiment score for sentence 1 {'neg': 0.0, 'neu': 0.213, 'pos': 0.787, 'compound': 0.5719}
sentiment score for sentence 2 {'neg': 0.756, 'neu': 0.244, 'pos': 0.0, 'compound': -0.4767}
sentiment score for sentence 3 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
```

Decision Tree

Task - 1

1.1 Loading Data

In [6]:
```
#make sure you are loading atleast 50k datapoints
#you can work with features of preprocessed_data.csv for the assignment.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from tqdm import tqdm
from sklearn.metrics import roc_curve, auc, confusion_matrix
from scipy.sparse import hstack
from wordcloud import WordCloud, STOPWORDS

data = pd.read_csv('preprocessed_data.csv')
```

In [7]:
```
# write your code in following steps for task 1
# 1. calculate sentiment scores for the essay feature
# 2. Split your data.
# 3. perform tfidf vectorization of text data.
# 4. perform tfidf w2v vectorization of text data.
# 5. perform encoding of categorical features.
# 6. perform encoding of numerical features
# 7. For task 1 set 1 stack up all the features
# 8. For task 1 set 2 stack up all the features (for stacking dense features you can use np.stack)
# 9. Perform hyperparameter tuning and plot either heatmap or 3d plot.
# 10. Find the best parameters and fit the model. Plot ROC-AUC curve(using predict proba method)
# 11. Plot confusion matrix based on best threshold value
# 12. Find all the false positive data points and plot wordcloud of essay text and pdf of teacher_number_of_pre
# 13. Write your observations about the wordcloud and pdf.
```

In [8]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# 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 code
# 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
```

In [9]:
```
def sentiment_values(dict_values):
    return dict_values["neg"], dict_values["neu"], dict_values["pos"], dict_values["compound"]
```

In [10]:
```
negative = []
neutral = []
positive = []
compound = []
# Computing sentiment values and storing them in list
for essay in tqdm(data["essay"]):
    ss = sid.polarity_scores(essay)
    neg, neu , pos, comp = ss["neg"], ss["neu"], ss["pos"], ss["compound"]
    negative.append(neg)
    neutral.append(neu)
    positive.append(pos)
    compound.append(comp)
```

```
100%|██████████| 109248/109248 [02:29<00:00, 73
1.12it/s]
```

In [11]:
```
#Creating new columns in our dataset with sentiment values
data["negative"] = negative
data["neutral"] = neutral
data["positive"] = positive
data["compound"] = compound
```

In [12]:
```
data.head(2)
```

Out[12]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clea
0	ca	mrs	grades_prek_2	53	1	math_science	h
1	ut	ms	grades_3_5	4	1	specialneeds	

In [13]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
```

1.2 Splitting data into Train and cross validation(or test): Stratified Sampling

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y, random_state=0)
```

1.3 Make Data Model Ready: encoding essay

1.3.1 Essay - TFIDF Representation:

In [15]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)

essay_vectorizer_tfidf = TfidfVectorizer(min_df=10, max_features=5000)
essay_vectorizer_tfidf.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 = essay_vectorizer_tfidf.transform(X_train['essay'].values)
X_test_essay_tfidf = essay_vectorizer_tfidf.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)
```

```
(73196, 12) (73196,)
(36052, 12) (36052,)
====================================================================================================
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
====================================================================================================
```

1.3.2 Essay - TFIDF Word2Vec Representation:

In [16]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["essay"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [17]:
```
#Train data
X_train_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train["essay"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(se
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value fo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_tfidf_w2v_vectors.append(vector)

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

```
100%|██████████| 73196/73196 [01:56<00:00, 62
8.89it/s]
73196
300
```

In [18]:
```
#Test Data
X_test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test["essay"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(se
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value fo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_tfidf_w2v_vectors.append(vector)

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

```
100%|██████████| 36052/36052 [00:57<00:00, 63
1.10it/s]
36052
300
```

1.4 Make Data Model Ready: encoding numerical, categorical features

1.4.1 encoding categorical features: School State

In [19]:
```
school_state_vectorizer = CountVectorizer()
school_state_vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = school_state_vectorizer.transform(X_train['school_state'].values)
X_test_state_ohe = school_state_vectorizer.transform(X_test['school_state'].values)

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

```
After vectorizations
(73196, 51) (73196,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'l
a', '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']
====================================================================================================
```

1.4.2 encoding categorical features: teacher_prefix

In [20]:
```
teacher_prefix_vectorizer = CountVectorizer()
teacher_prefix_vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = teacher_prefix_vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher_ohe = teacher_prefix_vectorizer.transform(X_test['teacher_prefix'].values)

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

```
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

1.4.3 encoding categorical features: project_grade_category

In [21]:
```
project_grade_category_vectorizer = CountVectorizer()
project_grade_category_vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on tra

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

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(project_grade_category_vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 4) (73196,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

1.4.4 encoding categorical features: clean_categories

In [22]:
```
clean_categories_vectorizer = CountVectorizer()
clean_categories_vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_categories_ohe = clean_categories_vectorizer.transform(X_train['clean_categories'].values)
X_test_categories_ohe = clean_categories_vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_categories_ohe.shape, y_train.shape)
print(X_test_categories_ohe.shape, y_test.shape)
print(clean_categories_vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 9) (73196,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'mus
ic_arts', 'specialneeds', 'warmth']
====================================================================================================
```

1.4.5 encoding categorical features: clean_subcategories

In [23]:
```
clean_subcategories_vectorizer = CountVectorizer()
clean_subcategories_vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train dat

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategories_ohe = clean_subcategories_vectorizer.transform(X_train['clean_subcategories'].values)
X_test_subcategories_ohe = clean_subcategories_vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcategories_ohe.shape, y_train.shape)
print(X_test_subcategories_ohe.shape, y_test.shape)
print(clean_subcategories_vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 30) (73196,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communitys
ervice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliterac
y', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literac
y', 'literature_writing', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performi
ngarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

1.4.6 encoding numerical features: Price

In [24]:
```
# Decision Trees do not need feature Standardization or normalization


#Reshaping so that the number of rows of the features is same for all data
X_train_price=X_train["price"].values.reshape(-1,1)
X_test_price=X_test["price"].values.reshape(-1,1)

print("After vectorizations")
print(X_train_price.shape, y_train.shape)
print(X_test_price.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [25]:
```
# Decision Trees do not need feature Standardization or normalization

#Reshaping so that the number of rows of the features is same for all data
X_train_teacher_num_previous_projects=X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1

X_test_teacher_num_previous_projects=X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1

print("After vectorizations")
print(X_train_teacher_num_previous_projects.shape, y_train.shape)
print(X_test_teacher_num_previous_projects.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

1.4.8 encoding numerical features: negative

In [26]:
```
# Decision Trees do not need feature Standardization or normalization
X_train_negative=X_train['negative'].values.reshape(-1,1)
X_test_negative=X_test['negative'].values.reshape(-1,1)

print("After vectorizations")
print(X_train_negative.shape, y_train.shape)
print(X_test_negative.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

1.4.9 encoding numerical features: neutral

In [27]:
```
# Decision Trees do not need feature Standardization or normalization
X_train_neutral=X_train['neutral'].values.reshape(-1,1)
X_test_neutral=X_test['neutral'].values.reshape(-1,1)

print("After vectorizations")
print(X_train_neutral.shape, y_train.shape)
print(X_test_neutral.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

1.4.10 encoding numerical features: positive

In [28]:
```
# Decision Trees do not need feature Standardization or normalization
X_train_positive=X_train['positive'].values.reshape(-1,1)
X_test_positive=X_test['positive'].values.reshape(-1,1)

print("After vectorizations")
print(X_train_positive.shape, y_train.shape)
print(X_test_positive.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

1.4.11 encoding numerical features: compound

In [29]:
```
# Decision Trees do not need feature Standardization or normalization
X_train_compound=X_train['compound'].values.reshape(-1,1)
X_test_compound=X_test['compound'].values.reshape(-1,1)

print("After vectorizations")
print(X_train_compound.shape, y_train.shape)
print(X_test_compound.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

1.4.12 Concatinating all the features

Set - 1

In [30]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_train_set_one = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_tra
X_test_set_one = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_cat

print("Final Data matrix - TFIDF:")
print(X_train_set_one.shape, y_train.shape)
print(X_test_set_one.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix - TFIDF:
(73196, 5105) (73196,)
(36052, 5105) (36052,)
====================================================================================================
```

Set - 2

In [31]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_train_set_two = hstack((X_train_tfidf_w2v_vectors, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_test_set_two = hstack((X_test_tfidf_w2v_vectors, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_te

print("Final Data matrix - w2v-TFIDF:")
print(X_train_set_two.shape, y_train.shape)
print(X_test_set_two.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix - w2v-TFIDF:
(73196, 405) (73196,)
(36052, 405) (36052,)
====================================================================================================
```

1.5 Set 1:

1.5.1 Apply Decision Tree classifier on Dataset

In [32]:
```
decision_tree_parameters = {'max_depth':[1,3,10,30],'min_samples_split':[5,10,100,500]}
decision_tree = DecisionTreeClassifier(random_state = 0)
clf = GridSearchCV(decision_tree, decision_tree_parameters, scoring = 'roc_auc',  cv = 5, return_train_score =
clf.fit(X_train_set_one, y_train)
```

Out[32]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=0), n_jobs=-1,
             param_grid={'max_depth': [1, 3, 10, 30],
                         'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc')
```

In [33]:
```
train_auc = clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

In [34]:
```
print("Best score= ",clf.best_score_)
print("Best Hyper parameters = ",clf.best_params_)
```

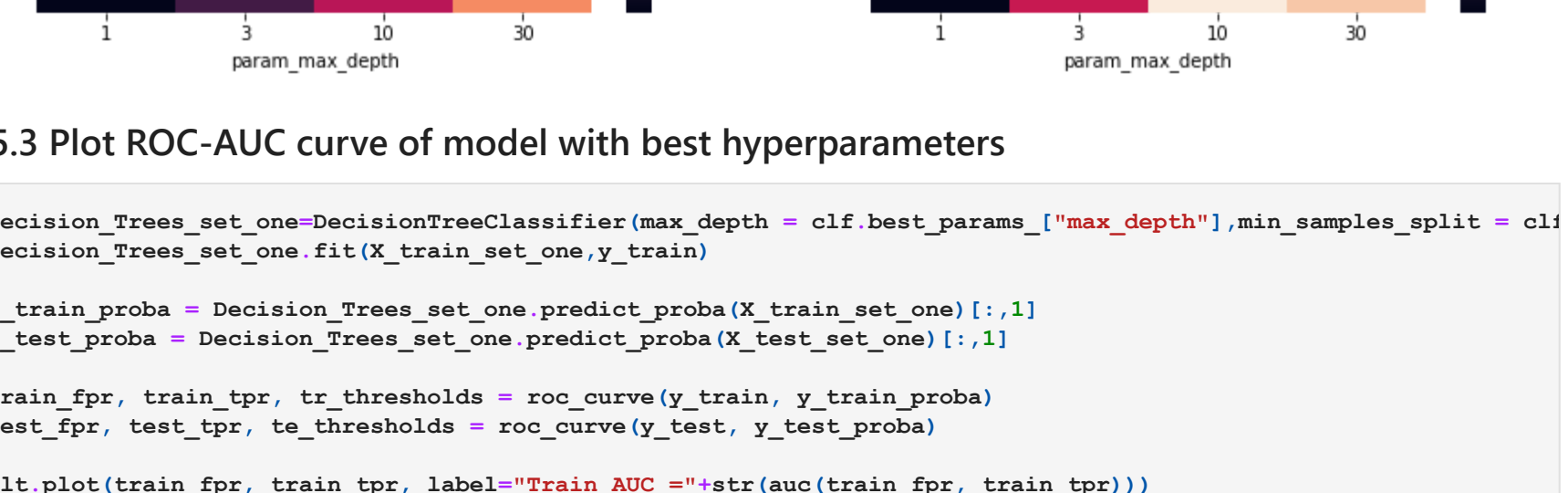
```
Best score=  0.6485339334756481
Best Hyper parameters =  {'max_depth': 10, 'min_samples_split': 500}
```

1.5.2 Hyperparameter vs AUC Plot

In [35]:
```
max_depth = [1,3,10,30]
min_samples_split= [5,10,100,500]
```

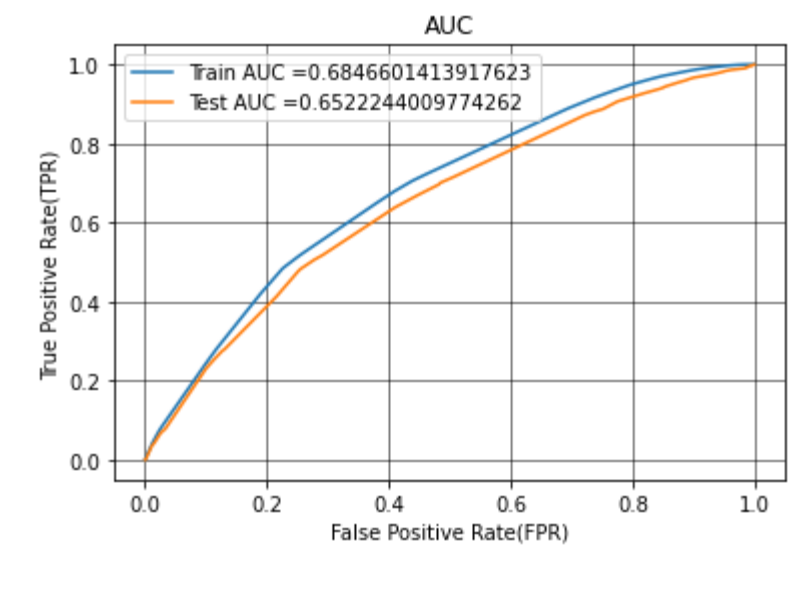
In [36]:
```
#Below code is obtained from: https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-o
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_min_samples_split', 'param_max_depth']).max()
max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
plt.figure(figsize=(15, 5))
plt.subplot(1,2,1)
plt.title("Train Dataset")
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g')
plt.subplot(1,2,2)
plt.title("CV Dataset")
sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g')
```

Out[36]:
```
<AxesSubplot:title={'center':'CV Dataset'}, xlabel='param_max_depth', ylabel='param_min_samples_split'>
```



1.5.3 Plot ROC-AUC curve of model with best hyperparameters

In [37]:
```
Decision_Trees_set_one=DecisionTreeClassifier(max_depth = clf.best_params_["max_depth"],min_samples_split = cl
Decision_Trees_set_one.fit(X_train_set_one,y_train)

y_train_proba = Decision_Trees_set_one.predict_proba(X_train_set_one)[:,1]
y_test_proba = Decision_Trees_set_one.predict_proba(X_test_set_one)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_proba)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_proba)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



1.5.4 Confusion Matrix for Train and Test data

```
In [38]:
def find_best_threshold(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of fpr*(1-tpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t
```

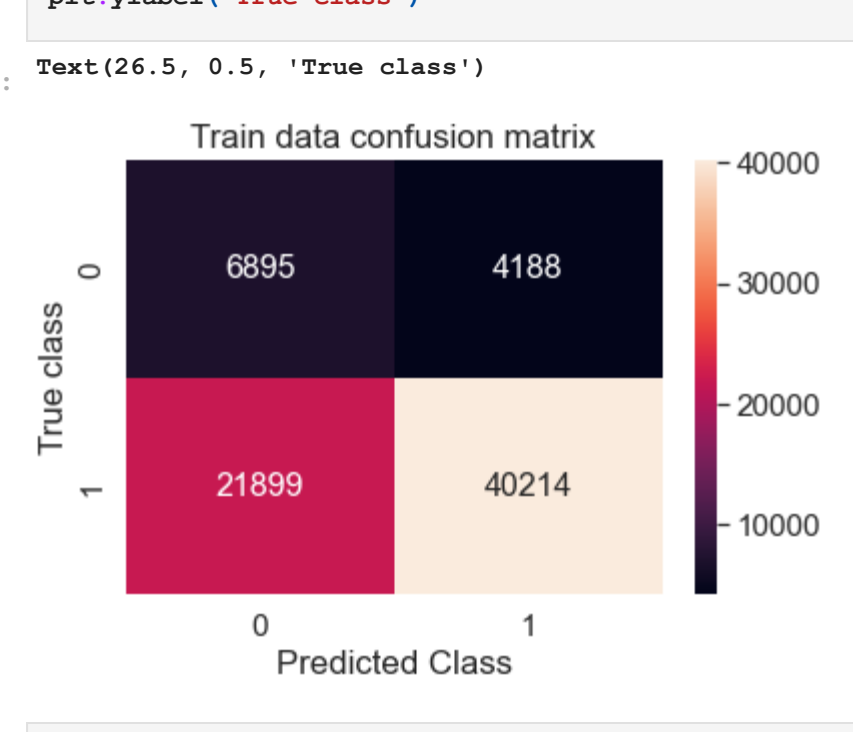
```
In [39]:
def predict_with_best_threshold(proba, t):
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

```
In [40]:
print("="*100)

best_threshold = find_best_threshold(y_train_proba, tr_thresholds, train_fpr, train_tpr)
conf_matr_train_set_one = confusion_matrix(y_train, predict_with_best_threshold(y_train_proba, best_threshold))
print("Train confusion matrix")
print(conf_matr_train_set_one)
```

```
====================================================================================================
the maximum value of fpr*(1-tpr) 0.40278353124942495 for threshold 0.855
Train confusion matrix
[[ 6895  4188]
 [21899 40214]]
```

```
In [41]:
sns.set(font_scale=1.4)#for label size

sns.heatmap(conf_matr_train_set_one, annot=True,annot_kws={"size": 16}, fmt='g')
plt.title("Train data confusion matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True class")
```

```
Out[41]: Text(26.5, 0.5, 'True class')
```

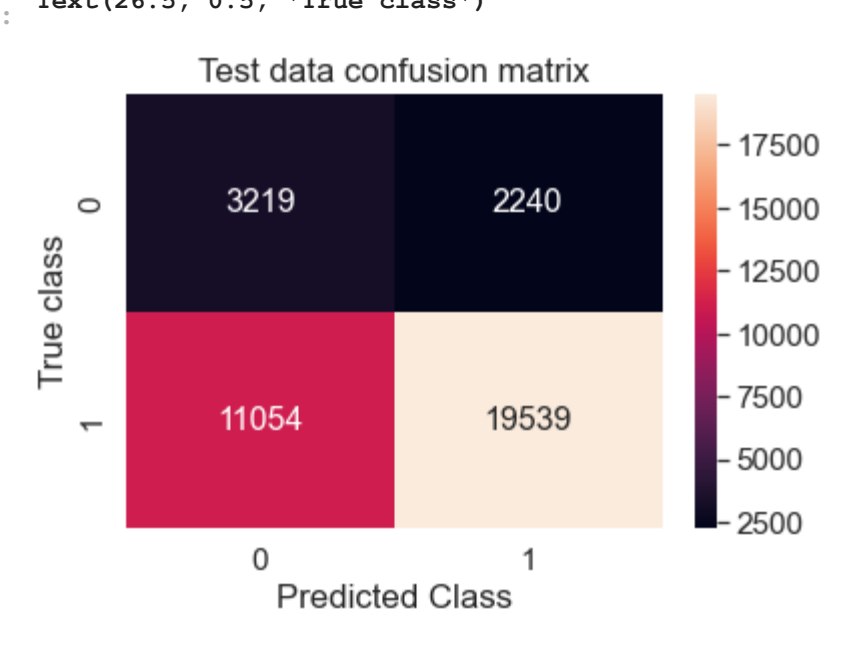



```
In [42]:
print("="*100)
print("Test confusion matrix")
conf_matr_test_set_one = confusion_matrix(y_test, predict_with_best_threshold(y_test_proba, best_threshold))
print(conf_matr_test_set_one)
```

```
====================================================================================================
Test confusion matrix
[[ 3219  2240]
 [11054 19539]]
```

```
In [43]:
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_test_set_one, annot=True,annot_kws={"size": 16}, fmt='g')
plt.title("Test data confusion matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True class")
```

```
Out[43]: Text(26.5, 0.5, 'True class')
```



1.5.5 Getting all false positive data points

```
In [44]:
predictions=predict_with_best_threshold(y_test_proba,best_threshold)
```

```
In [45]:
fp = []
for i in range(len(y_test)):
    if (y_test[i]==0) & (predictions[i] == 1):
        fp.append(i)    #Getting indices for all false positive points
```

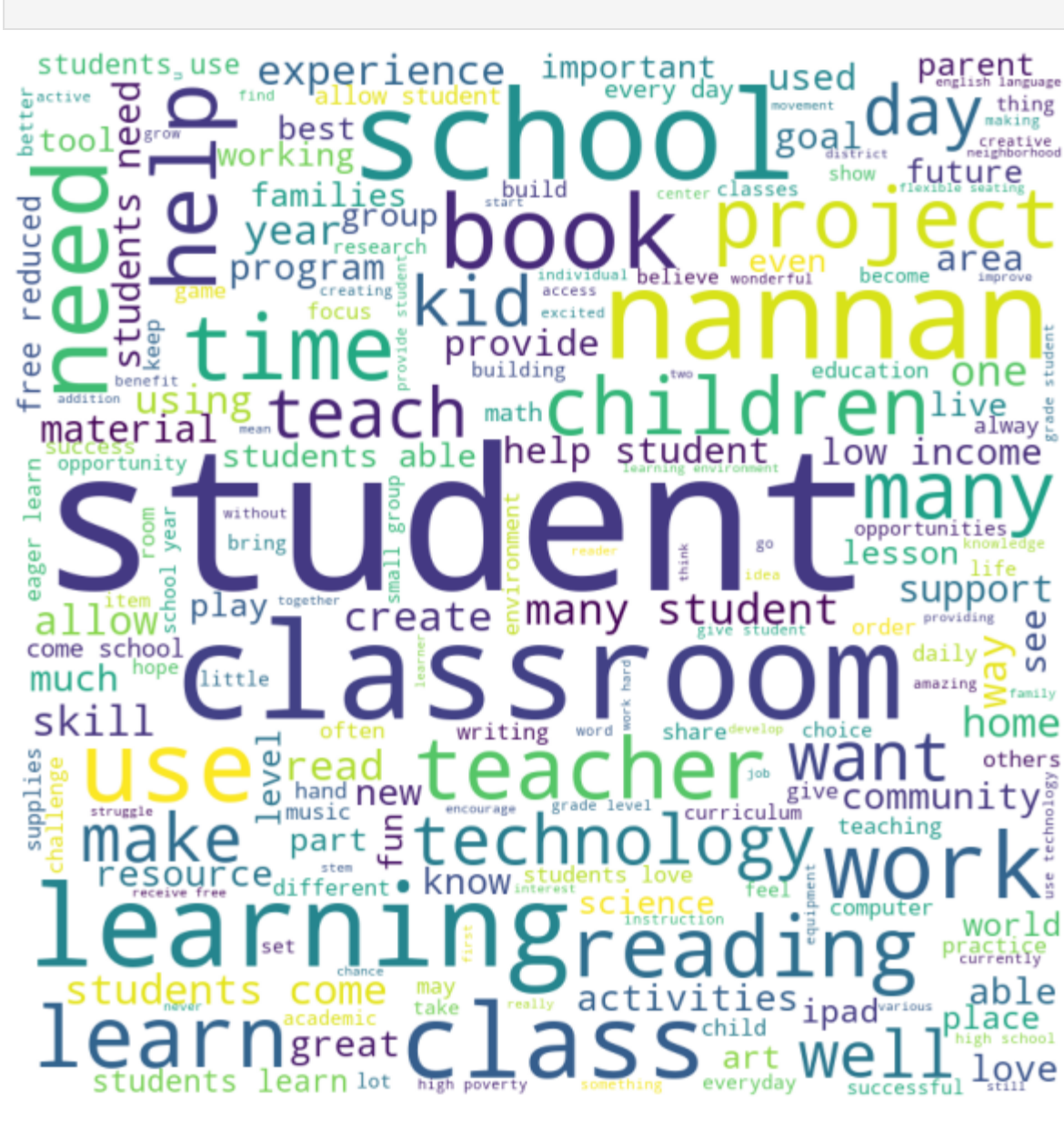
```
In [46]:
column_names = X_test.columns
X_test_false_positive = pd.DataFrame(columns = column_names)
X_test_false_positive = X_test.iloc[fp]
```

```
In [47]:
print(X_test_false_positive.shape)
```

```
(2240, 12)
```

1.5.6 wordcloud Of Essay Text For False Positive Dataset

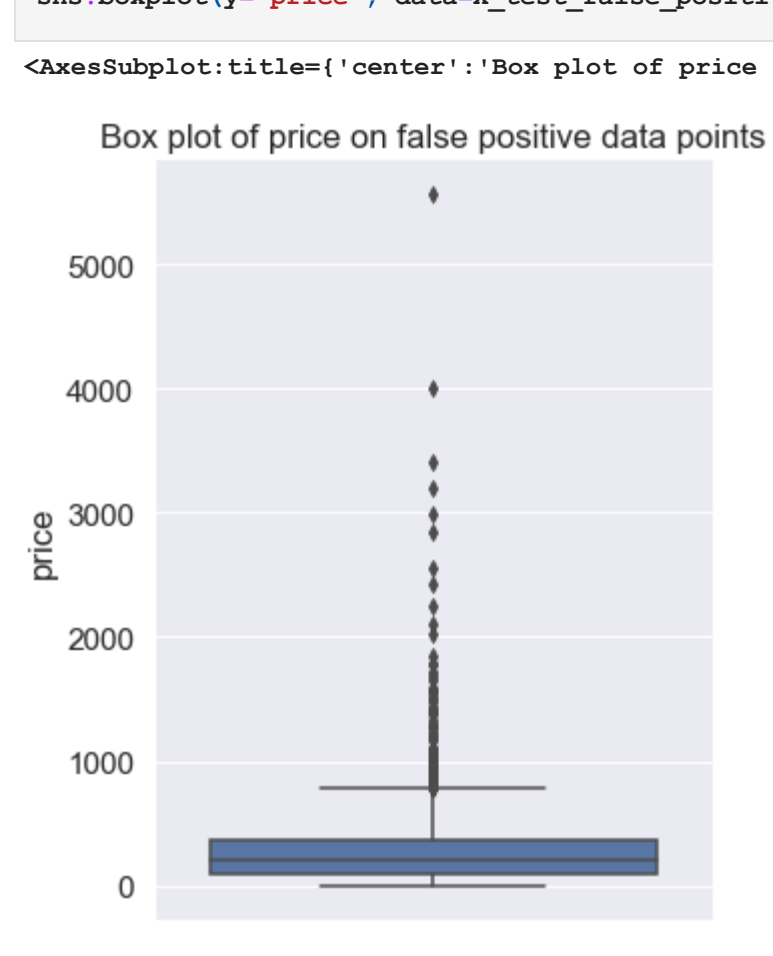
```
In [48]:
# Below code is copied from: https://www.geeksforgeeks.org/generating-word-cloud-python/
comment_words = ''
stopwords = set(STOPWORDS)
for val in X_test_false_positive['essay']:
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

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

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

plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



1.5.7 Box plot of price on false positive data points

```
In [49]:
plt.figure(figsize=(5,7))
plt.title("Box plot of price on false positive data points")
sns.boxplot(y='price', data=X_test_false_positive)
```

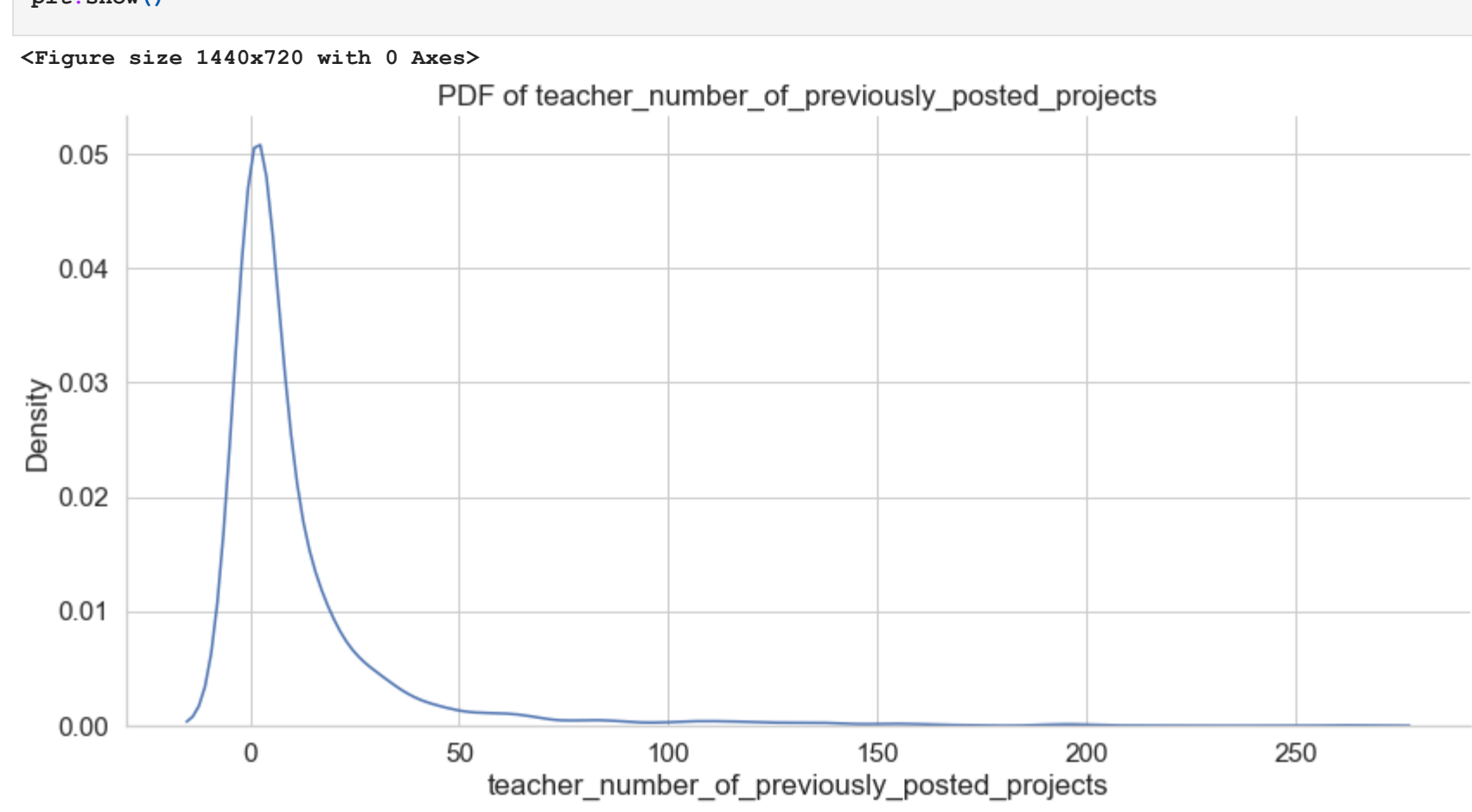
```
Out[49]: <AxesSubplot:title={'center':'Box plot of price on false positive data points'}, ylabel='price'>
```



1.5.8 PDF of teacher_number_of_previously_posted_projects on false positive data points

```
In [50]:
plt.figure(figsize=(20,10))
sns.set_style('whitegrid')
sns.FacetGrid(X_test_false_positive, height=6, aspect=2)\
    .map(sns.kdeplot,"teacher_number_of_previously_posted_projects")\
    .add_legend()
plt.title('PDF of teacher_number_of_previously_posted_projects')
plt.show()
```

```
<Figure size 1440x720 with 0 Axes>
```



1.6 Set - 2

1.6.1 Apply Decision Tree classifier on Dataset

```
In [51]:
decision_tree_parameters = {'max_depth':[1,3,10,30],'min_samples_split':[5,10,100,500]}
decision_tree = DecisionTreeClassifier(random_state = 0)
clf = GridSearchCV(decision_tree, decision_tree_parameters, scoring = 'roc_auc',  cv = 5, return_train_score =
clf.fit(X_train_set_two, y_train)
```

```
Out[51]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=0), n_jobs=-1,
             param_grid={'max_depth': [1, 3, 10, 30],
                         'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc')
```

```
In [52]:
train_auc = clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

```
In [53]:
print("Best score= ",clf.best_score_)
print("Best Hyper parameters = ",clf.best_params_)
```

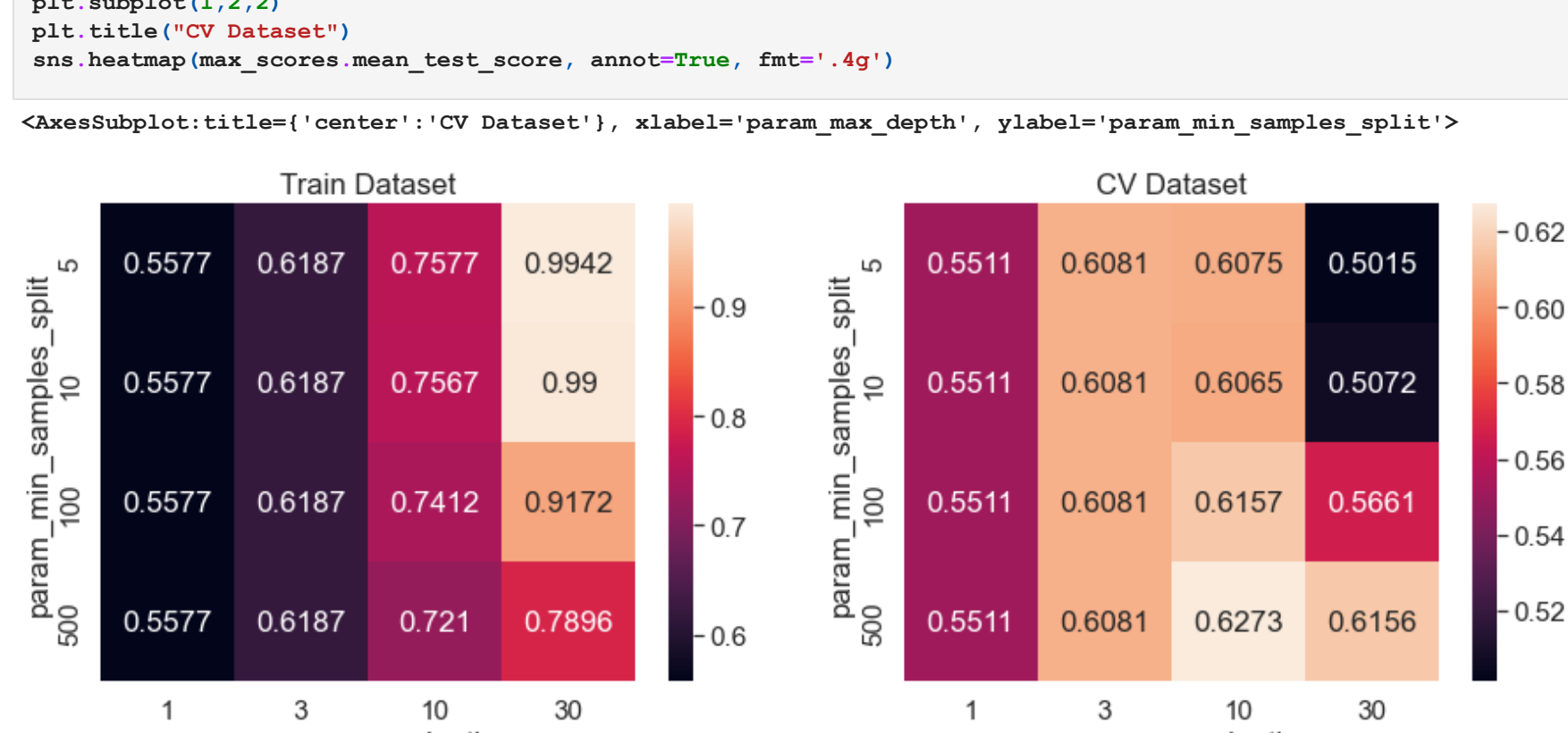
```
Best score=  0.6272631372826134
Best Hyper parameters =  {'max_depth': 10, 'min_samples_split': 500}
```

1.6.2 Hyperparameter vs AUC Plot

```
In [54]:
max_depth = [1,3,10,30]
min_samples_split= [5,10,100,500]
```

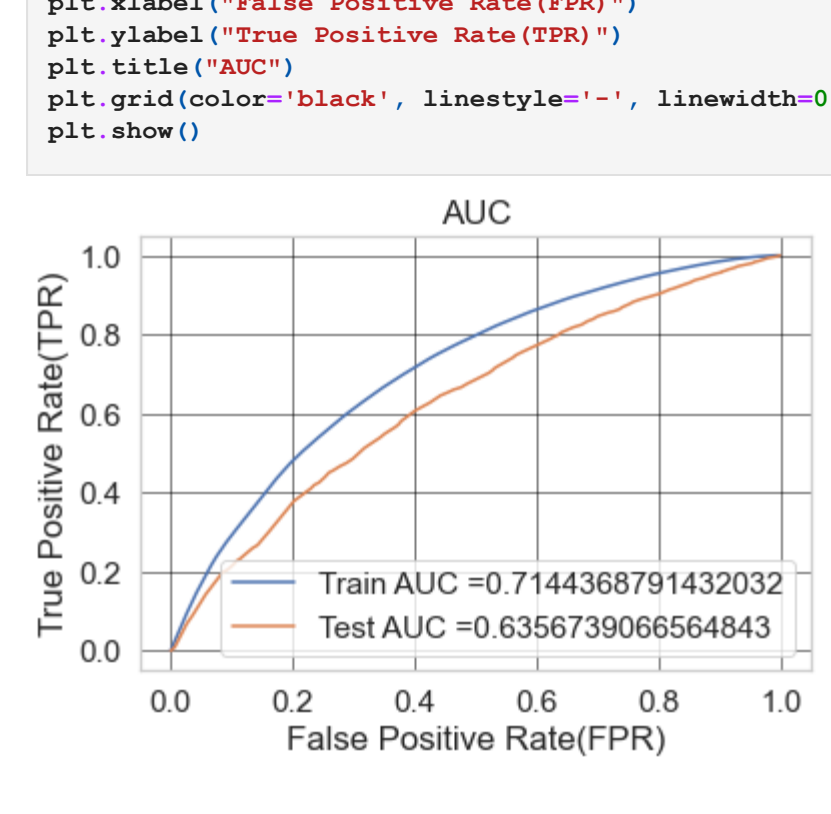
```
In [55]:
#Below code is obtained from: https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-o
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_min_samples_split', 'param_max_depth']).max()
max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
plt.figure(figsize=(15, 5))
plt.subplot(1,2,1)
plt.title("Train Dataset")
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g')
plt.subplot(1,2,2)
plt.title("CV Dataset")
sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g')
```

```
Out[55]: <AxesSubplot:title={'center':'CV Dataset'}, xlabel='param_max_depth', ylabel='param_min_samples_split'>
```



1.6.3 Plot ROC-AUC curve of model with best hyperparameters

```
In [56]:
Decision_Trees_set_two=DecisionTreeClassifier(max_depth = clf.best_params_["max_depth"],min_samples_split = cl
Decision_Trees_set_two.fit(X_train_set_two,y_train)

y_train_proba = Decision_Trees_set_two.predict_proba(X_train_set_two)[:,1]
y_test_proba = Decision_Trees_set_two.predict_proba(X_test_set_two)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_proba)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_proba)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



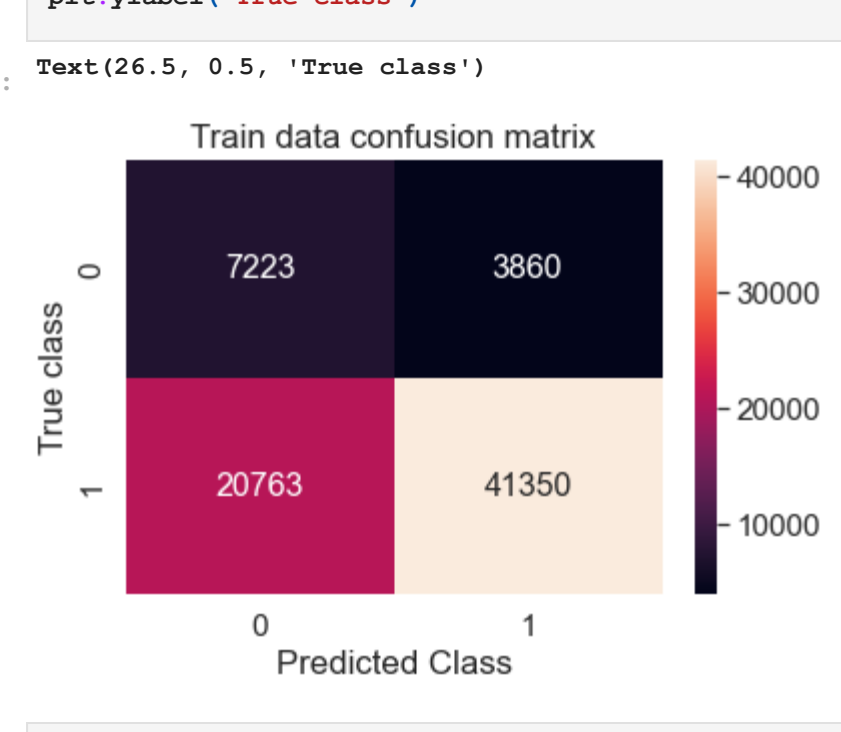
1.6.4 Confusion Matrix for Train and Test data

```
In [57]:
print("="*100)

best_threshold = find_best_threshold(y_train_proba, tr_thresholds, train_fpr, train_tpr)
conf_matr_train_set_two = confusion_matrix(y_train, predict_with_best_threshold(y_train_proba, best_threshold))
print("Train confusion matrix")
print(conf_matr_train_set_two)
```

```
====================================================================================================
the maximum value of fpr*(1-tpr) 0.43386367416184746 for threshold 0.859
Train confusion matrix
[[ 7223  3860]
 [20763 41350]]
```

```
In [58]:
sns.set(font_scale=1.4)#for label size

sns.heatmap(conf_matr_train_set_two, annot=True,annot_kws={"size": 16}, fmt='g')
plt.title("Train data confusion matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True class")
```

```
Out[58]: Text(26.5, 0.5, 'True class')
```

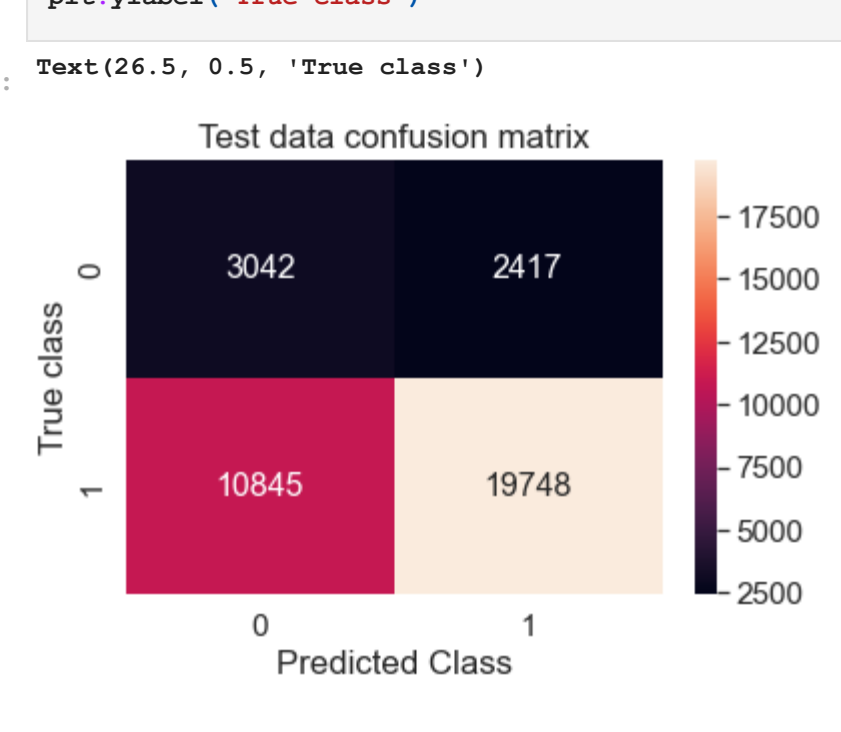



```
In [59]:
print("="*100)
print("Test confusion matrix")
conf_matr_test_set_two = confusion_matrix(y_test, predict_with_best_threshold(y_test_proba, best_threshold))
print(conf_matr_test_set_two)
```

```
====================================================================================================
Test confusion matrix
[[ 3042  2417]
 [10845 19748]]
```

```
In [60]:
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_test_set_two, annot=True,annot_kws={"size": 16}, fmt='g')
plt.title("Test data confusion matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True class")
```

```
Out[60]: Text(26.5, 0.5, 'True class')
```



1.6.5 Getting all false positive data points

```
In [61]:
predictions=predict_with_best_threshold(y_test_proba,best_threshold)
```

```
In [62]:
fp = []
for i in range(len(y_test)):
    if (y_test[i]==0) & (predictions[i] == 1):
        fp.append(i)    #Getting indices for all false positive points
```

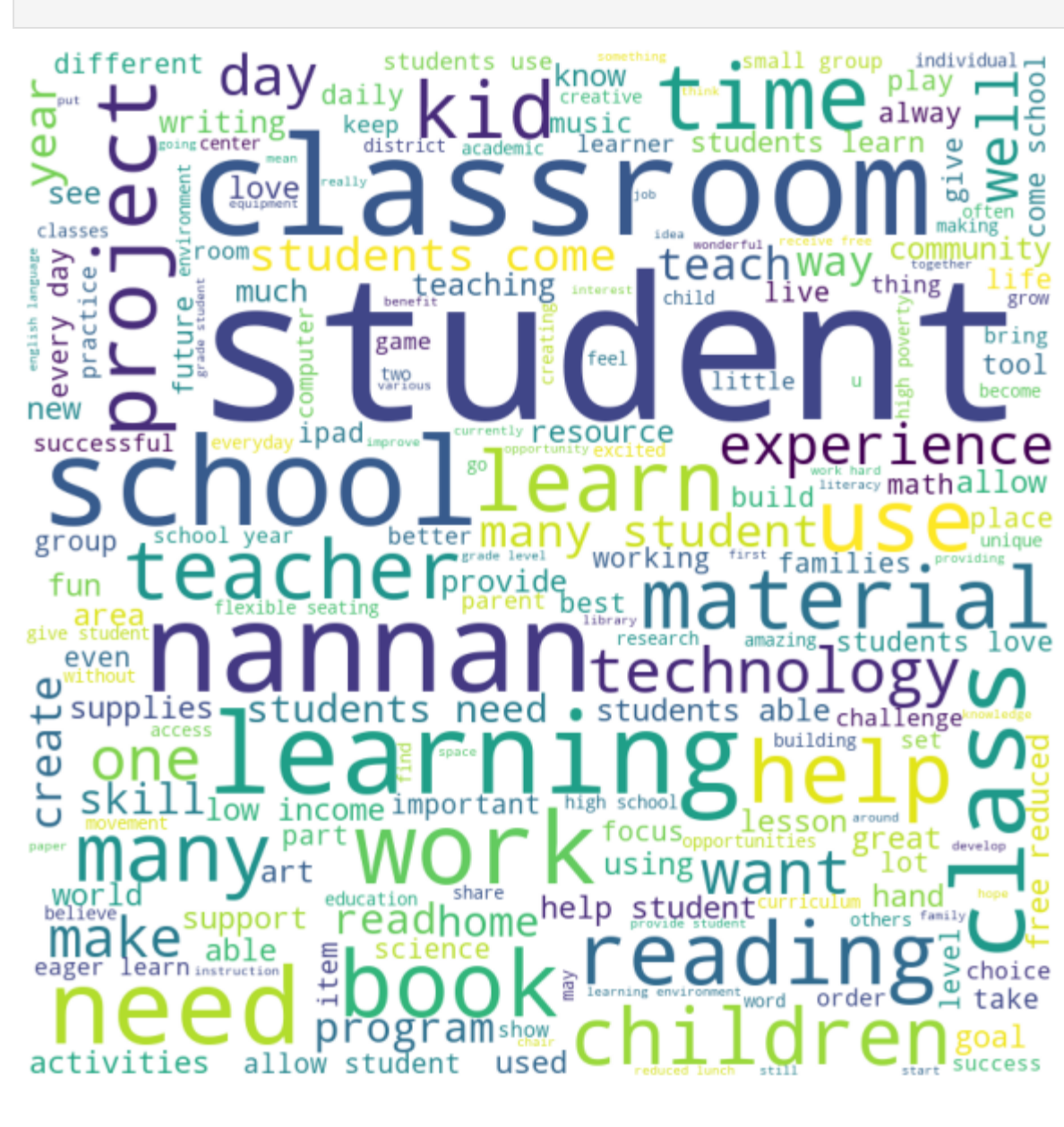
```
In [63]:
column_names = X_test.columns
X_test_false_positive = pd.DataFrame(columns = column_names)
X_test_false_positive = X_test.iloc[fp]
```

```
In [64]:
print(X_test_false_positive.shape)
```

```
(2417, 12)
```

1.6.6 wordcloud Of Essay Text For False Positive Dataset

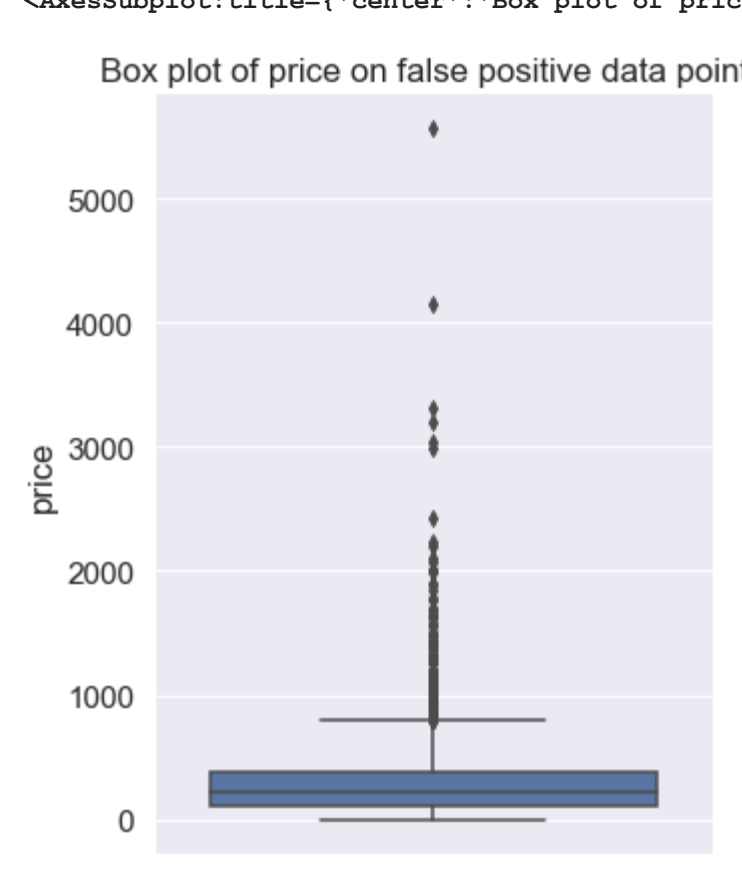
```
In [65]:
# Below code is copied from: https://www.geeksforgeeks.org/generating-word-cloud-python/
comment_words = ''
stopwords = set(STOPWORDS)
for val in X_test_false_positive['essay']:
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

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

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

plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



1.6.7 Box plot of price on false positive data points

```
In [66]:
plt.figure(figsize=(5,7))
plt.title("Box plot of price on false positive data points")
sns.boxplot(y='price', data=X_test_false_positive)
```

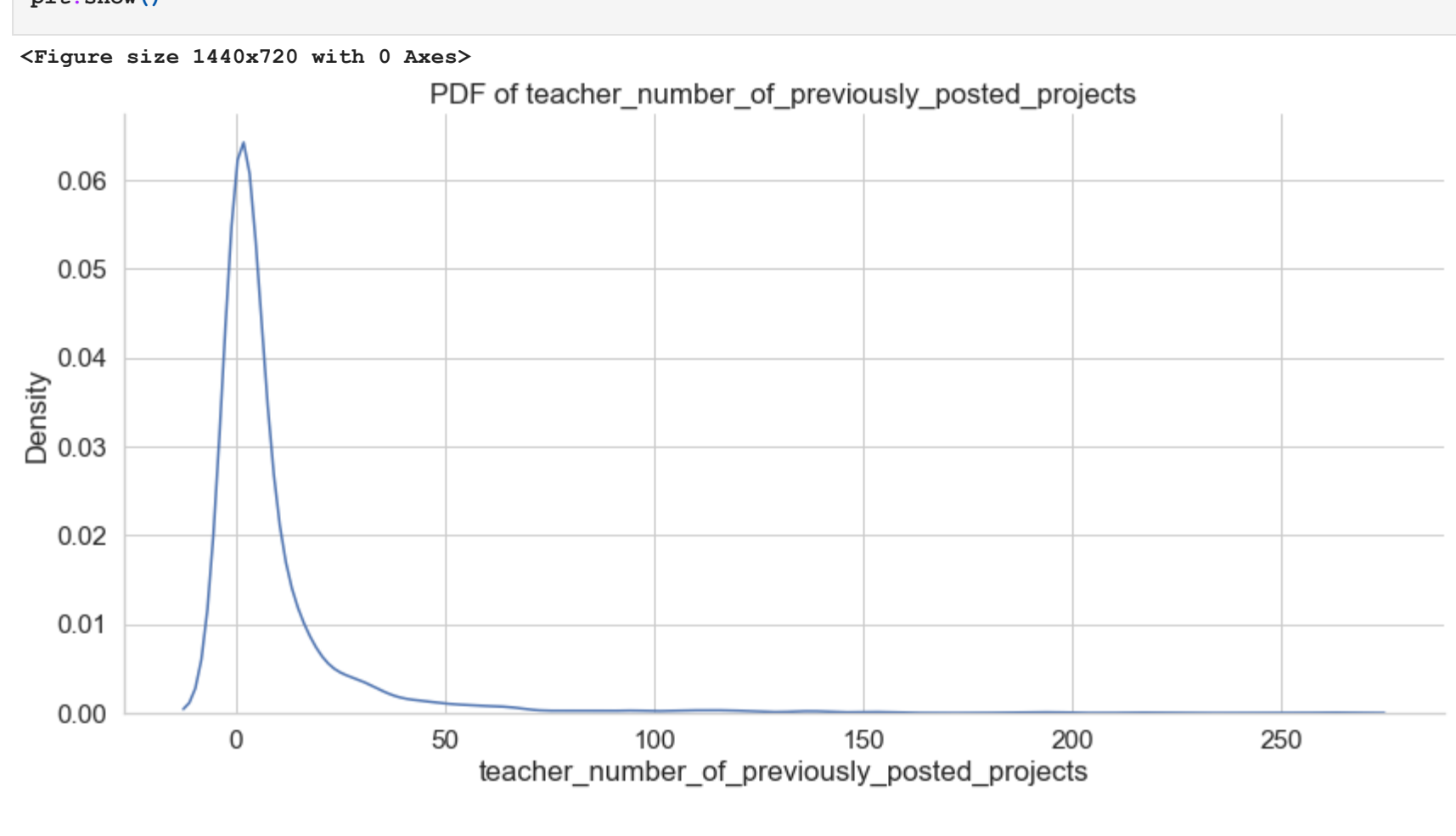
```
Out[66]: <AxesSubplot:title={'center':'Box plot of price on false positive data points'}, ylabel='price'>
```



1.6.8 PDF of teacher_number_of_previously_posted_projects on false positive data points

```
In [67]:
plt.figure(figsize=(20,10))
sns.set_style('whitegrid')
sns.FacetGrid(X_test_false_positive, height=6, aspect=2)\
    .map(sns.kdeplot,"teacher_number_of_previously_posted_projects")\
    .add_legend()

plt.title('PDF of teacher_number_of_previously_posted_projects')
plt.show()
```

```
<Figure size 1440x720 with 0 Axes>
```



Task - 2

```
In [68]:
# 1. write your code in following steps for task 2
# 2. select all non zero features
# 3. Update your dataset i.e. X_train,X_test and X_cv so that it contains all rows and only non zero features
# 4. perform hyperparameter tuning and plot either heatmap or 3d plot.
# 5. Fit the best model. Plot ROC AUC curve and confusion matrix similar to model 1.
```

2.1 Selecting all the features which are having non-zero feature importance

```
In [69]:
tree_clf = DecisionTreeClassifier(max_depth=None,min_samples_split = 500, random_state = 0)
```

```
In [70]:
tree_clf.fit(X_train_set_one, y_train)
```

```
Out[70]: DecisionTreeClassifier(min_samples_split=500, random_state=0)
```

```
In [71]:
feature_importances = tree_clf.feature_importances_
```

```
In [72]:
non_zero_feature_indices = []
for i in range(len(feature_importances)):
    if feature_importances[i]>0:
        non_zero_feature_indices.append(i)
```

```
In [73]:
print("Number of non-zero features:",len(non_zero_feature_indices))
```

```
Number of non-zero features: 2237
```

```
In [74]:
X_train_non_zero_feature = X_train_set_one[:,non_zero_feature_indices]
X_test_non_zero_feature = X_test_set_one[:,non_zero_feature_indices]
```

2.2 Apply Decision Tree classifier on Dataset containing only important features

```
In [75]:
decision_tree_parameters = {'max_depth':[1,3,10,30],'min_samples_split':[5,10,100,500]}
decision_tree = DecisionTreeClassifier(random_state = 0)
clf = GridSearchCV(decision_tree, decision_tree_parameters, scoring = 'roc_auc',  cv = 5, return_train_score =
clf.fit(X_train_non_zero_feature, y_train)
```

```
Out[75]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=0), n_jobs=-1,
             param_grid={'max_depth': [1, 3, 10, 30],
                         'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc')
```

```
In [76]:
train_auc = clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

```
In [77]:
print("Best score= ",clf.best_score_)
print("Best Hyper parameters = ",clf.best_params_)
```

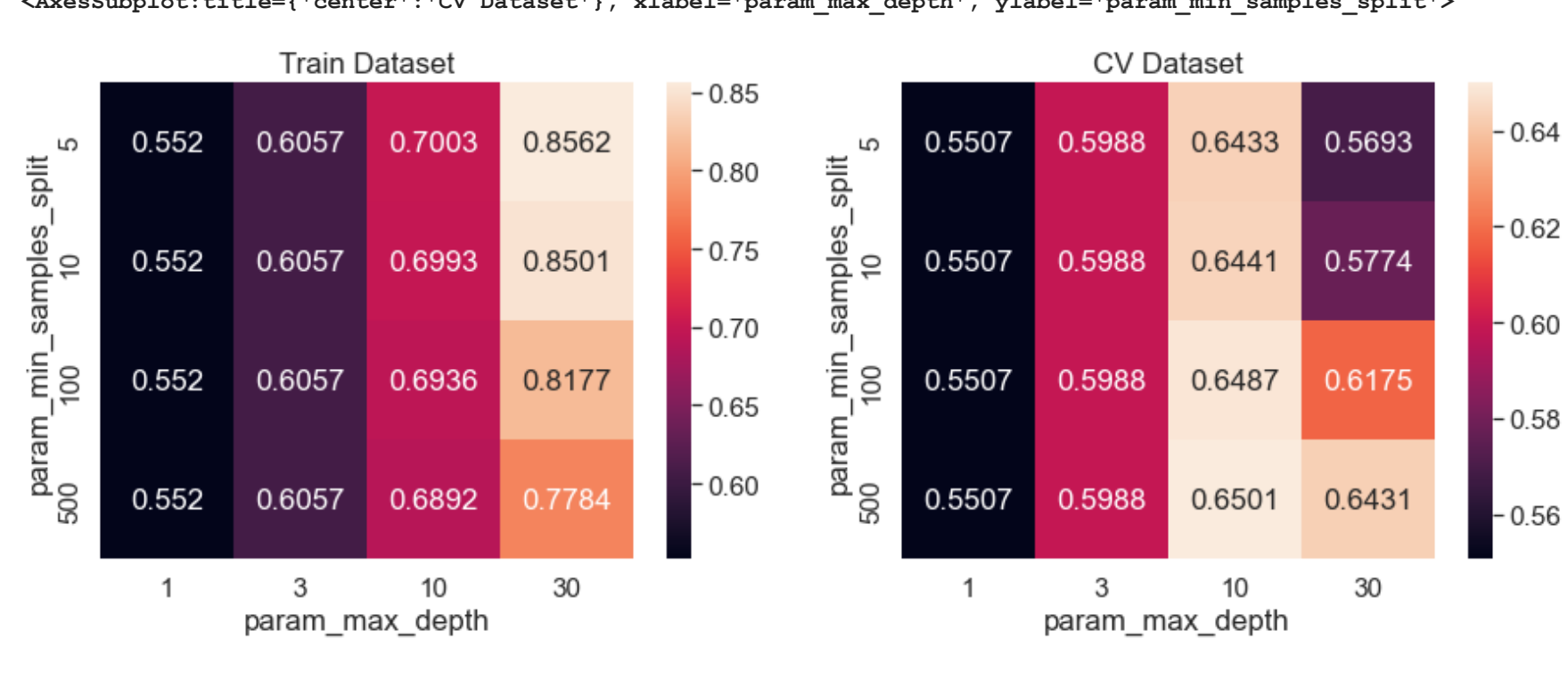
```
Best score=  0.6500548549667108
Best Hyper parameters =  {'max_depth': 10, 'min_samples_split': 500}
```

2.3 Hyperparameter vs AUC Plot

```
In [78]:
max_depth = [1,3,10,30]
min_samples_split= [5,10,100,500]
```

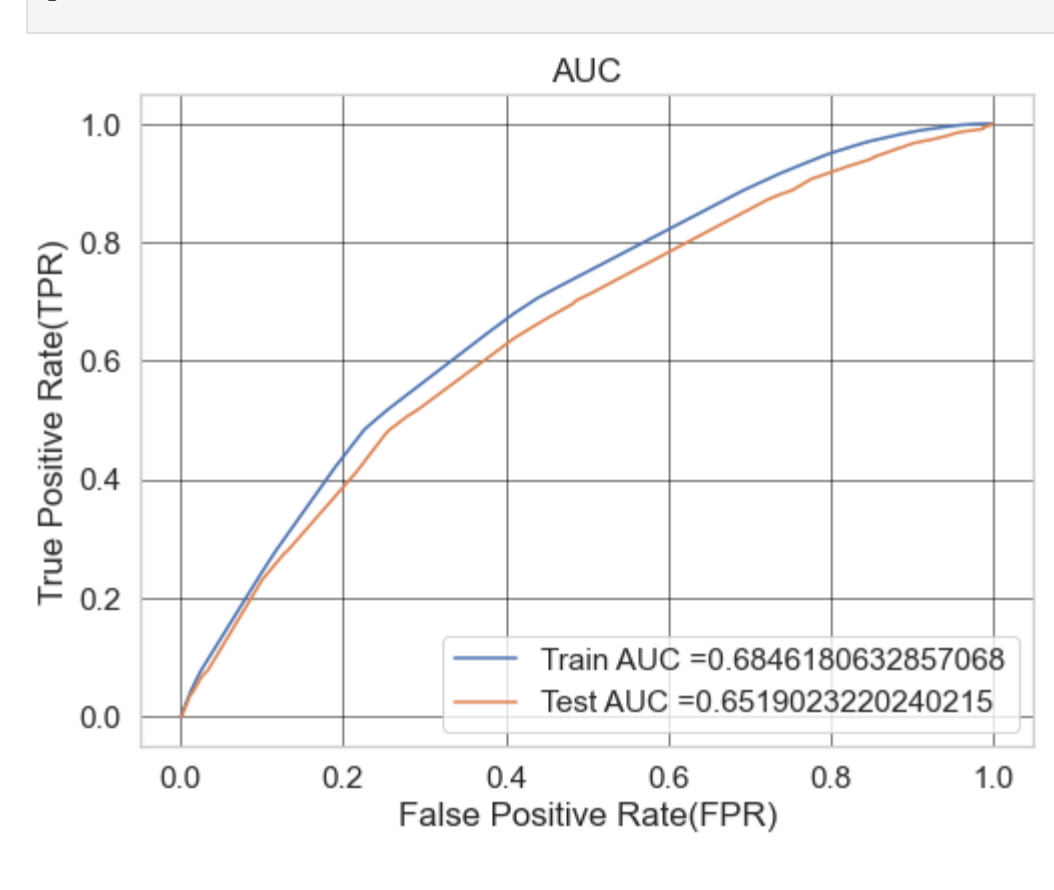
```
In [79]:
#Below code is obtained from: https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-o
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_min_samples_split', 'param_max_depth']).max()
max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
plt.figure(figsize=(15, 5))
plt.subplot(1,2,1)
plt.title("Train Dataset")
sns.heatmap(max_scores.mean_train_score, annot=True, fmt='.4g')
plt.subplot(1,2,2)
plt.title("CV Dataset")
sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g')
```

```
Out[79]: <AxesSubplot:title={'center':'CV Dataset'}, xlabel='param_max_depth', ylabel='param_min_samples_split'>
```



2.4 Plot ROC-AUC curve of model with best hyperparameters

```
In [80]:
Decision_Trees_non_zero_feature_importances=DecisionTreeClassifier(max_depth = clf.best_params_["max_depth"],mi
Decision_Trees_non_zero_feature_importances.fit(X_train_non_zero_feature,y_train)

y_train_proba = Decision_Trees_non_zero_feature_importances.predict_proba(X_train_non_zero_feature)[:,1]
y_test_proba = Decision_Trees_non_zero_feature_importances.predict_proba(X_test_non_zero_feature)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_proba)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_proba)

plt.figure(figsize=(8,6))
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



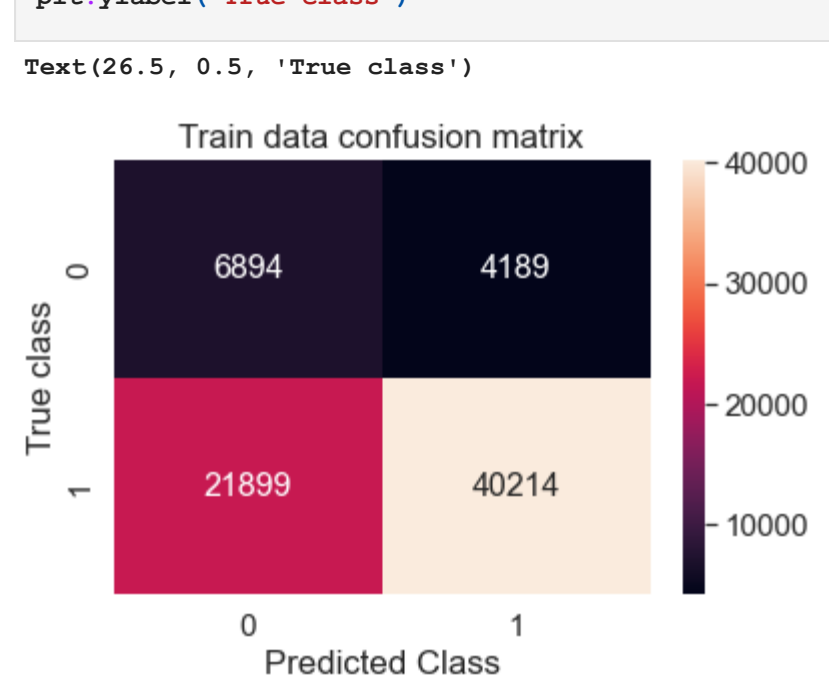
2.5 Confusion Matrix for Train and Test data

```
In [81]:
print("="*100)

best_threshold = find_best_threshold(y_train_proba, tr_thresholds, train_fpr, train_tpr)
conf_matr_train_non_zero_features = confusion_matrix(y_train, predict_with_best_threshold(y_train_proba, best_t
print("Train confusion matrix")
print(conf_matr_train_non_zero_features)
```

```
====================================================================================================
the maximum value of fpr*(1-tpr) 0.4027251144936237 for threshold 0.855
Train confusion matrix
[[ 6894  4189]
 [21899 40214]]
```

```
In [82]:
sns.set(font_scale=1.4)#for label size

sns.heatmap(conf_matr_train_non_zero_features, annot=True,annot_kws={"size": 16}, fmt='g')
plt.title("Train data confusion matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True class")
```

```
Out[82]: Text(26.5, 0.5, 'True class')
```

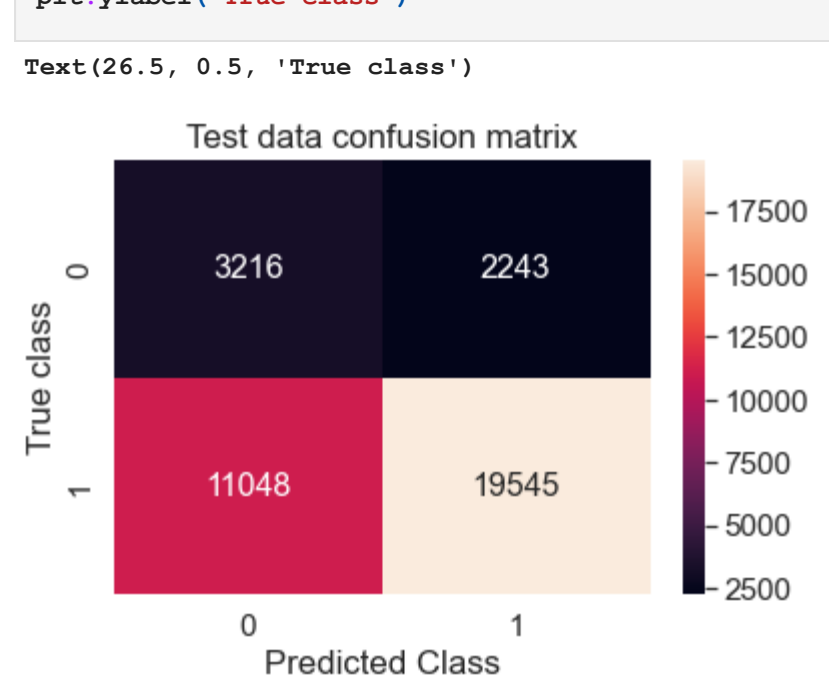



```
In [83]:
print("="*100)
print("Test confusion matrix")
conf_matr_test_non_zero_features = confusion_matrix(y_test, predict_with_best_threshold(y_test_proba, best_thre
print(conf_matr_test_non_zero_features)
```

```
====================================================================================================
Test confusion matrix
[[ 3216  2243]
 [11048 19545]]
```

```
In [84]:
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_test_non_zero_features, annot=True,annot_kws={"size": 16}, fmt='g')
plt.title("Test data confusion matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True class")
```

```
Out[84]: Text(26.5, 0.5, 'True class')
```



2.6 Getting all false positive data points

```
In [85]:
predictions=predict_with_best_threshold(y_test_proba,best_threshold)
```

In [86]:

```python
fp = []
for i in range(len(y_test)):
    if (y_test[i]==0) & (predictions[i] == 1):
        fp.append(i)     #Getting indices for all false positive points
```

In [87]:

```python
column_names = X_test.columns
X_test_false_positive = pd.DataFrame(columns = column_names)
X_test_false_positive = X_test.iloc[fp]
```

In [88]:

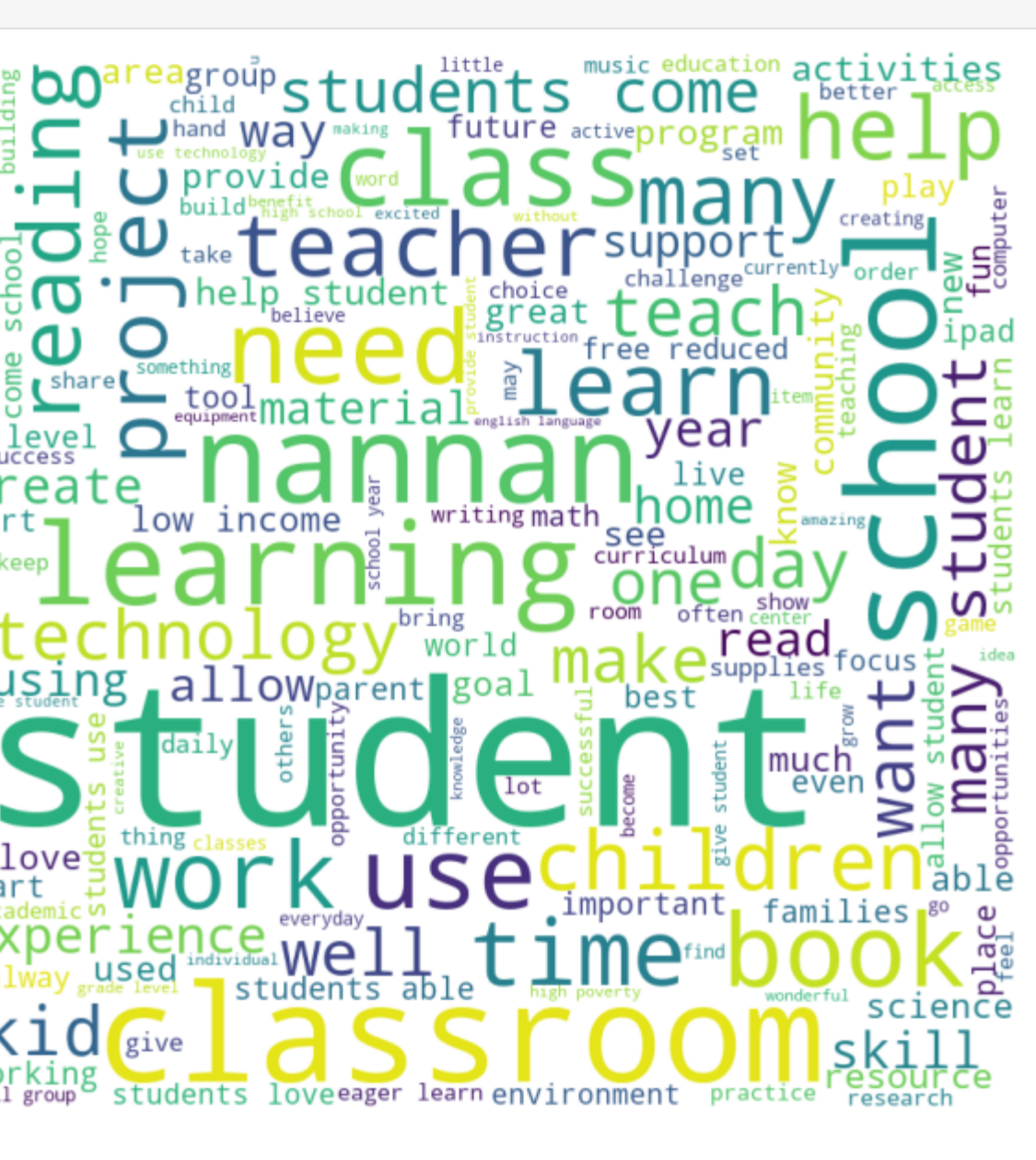
```python
print(X_test_false_positive.shape)
```

```
(2243, 12)
```

2.7 wordcloud Of Essay Text For False Positive Dataset

In [89]:

```python
# Below code is copied from: https://www.geeksforgeeks.org/generating-word-cloud-python/
comment_words = ''
stopwords = set(STOPWORDS)
for val in X_test_false_positive['essay']:
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

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

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

plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



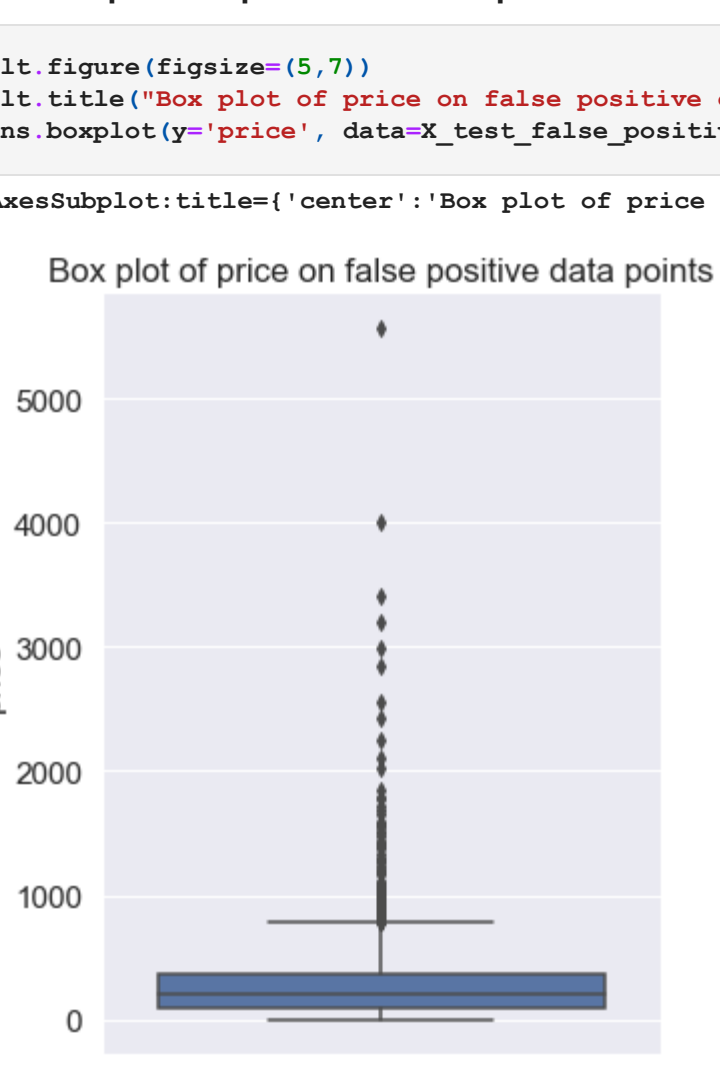
2.8 Box plot of price on false positive data points

In [90]:

```python
plt.figure(figsize=(5,7))
plt.title("Box plot of price on false positive data points")
sns.boxplot(y='price', data=X_test_false_positive)
```

Out[90]:

```
&lt;AxesSubplot:title={'center':'Box plot of price on false positive data points'}, ylabel='price'&gt;
```

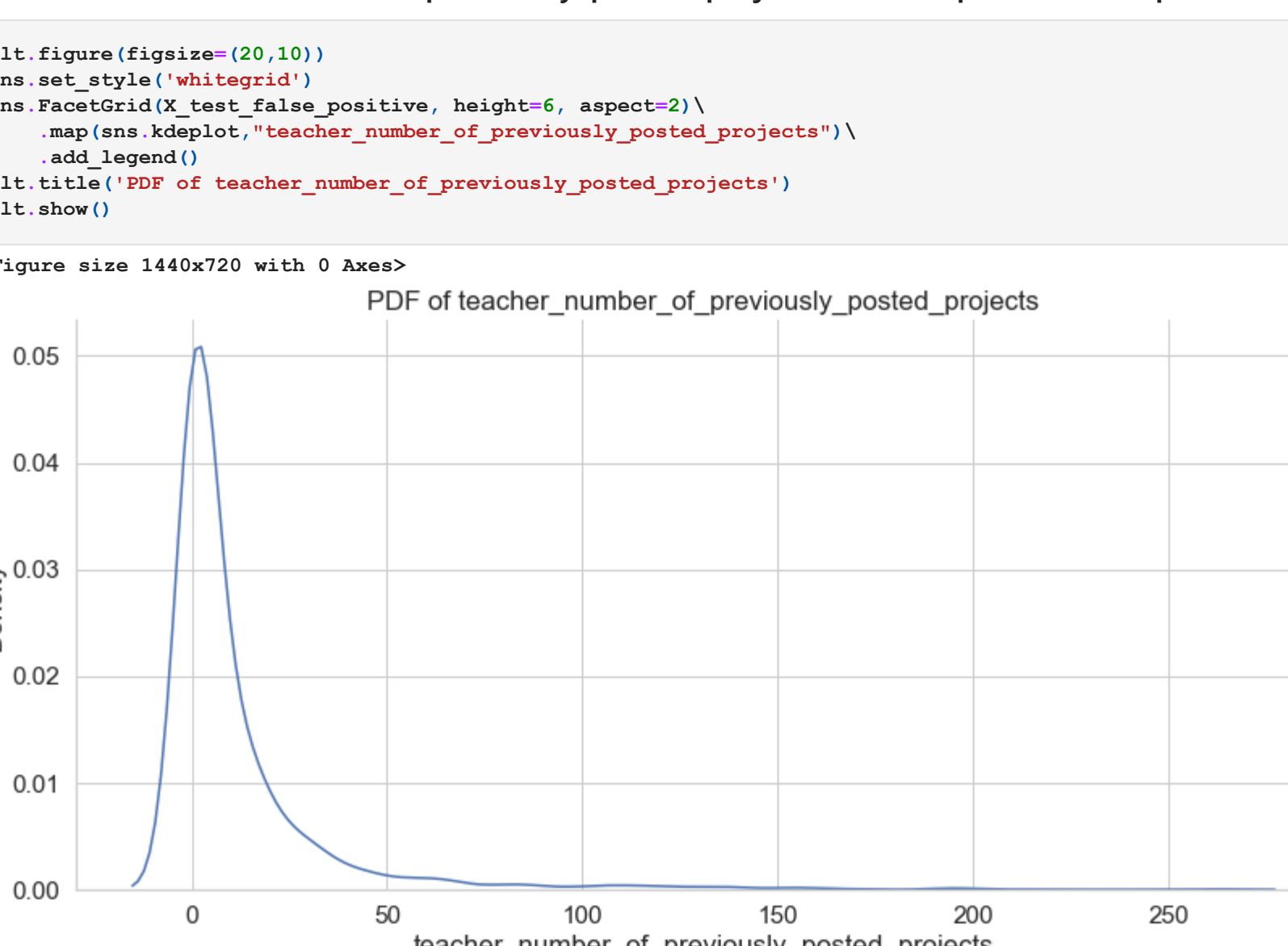


2.9 PDF of teacher_number_of_previously_posted_projects on false positive data points

In [91]:

```python
plt.figure(figsize=(20,10))
sns.set_style('whitegrid')
sns.FacetGrid(X_test_false_positive, height=6, aspect=2)\
    .map(sns.kdeplot,"teacher_number_of_previously_posted_projects")\
    .add_legend()
plt.title('PDF of teacher_number_of_previously_posted_projects')
plt.show()
```

```
&lt;Figure size 1440x720 with 0 Axes&gt;
```



Summary

In [92]:

```python
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]

x.add_row(["TFIDF", "Decision Tree","max_depth = 10, min_samples_split = 500", "0.6523"])
x.add_row(["TFIDF W2V", "Decision Tree", "max_depth = 10, min_samples_split = 500", "0.6354"])
x.add_row(["TFIDF NON ZERO FEATURE IMPORTANCE", "Decision Tree", "max_depth = 10, min_samples_split = 500", "0.

print(x)
```

Vectorizer	Model	Hyper Parameter	AUC
TFIDF	Decision Tree	max_depth = 10, min_samples_split = 500	0.6523
TFIDF W2V	Decision Tree	max_depth = 10, min_samples_split = 500	0.6354
TFIDF NON ZERO FEATURE IMPORTANCE	Decision Tree	max_depth = 10, min_samples_split = 500	0.6519

In [ ]: