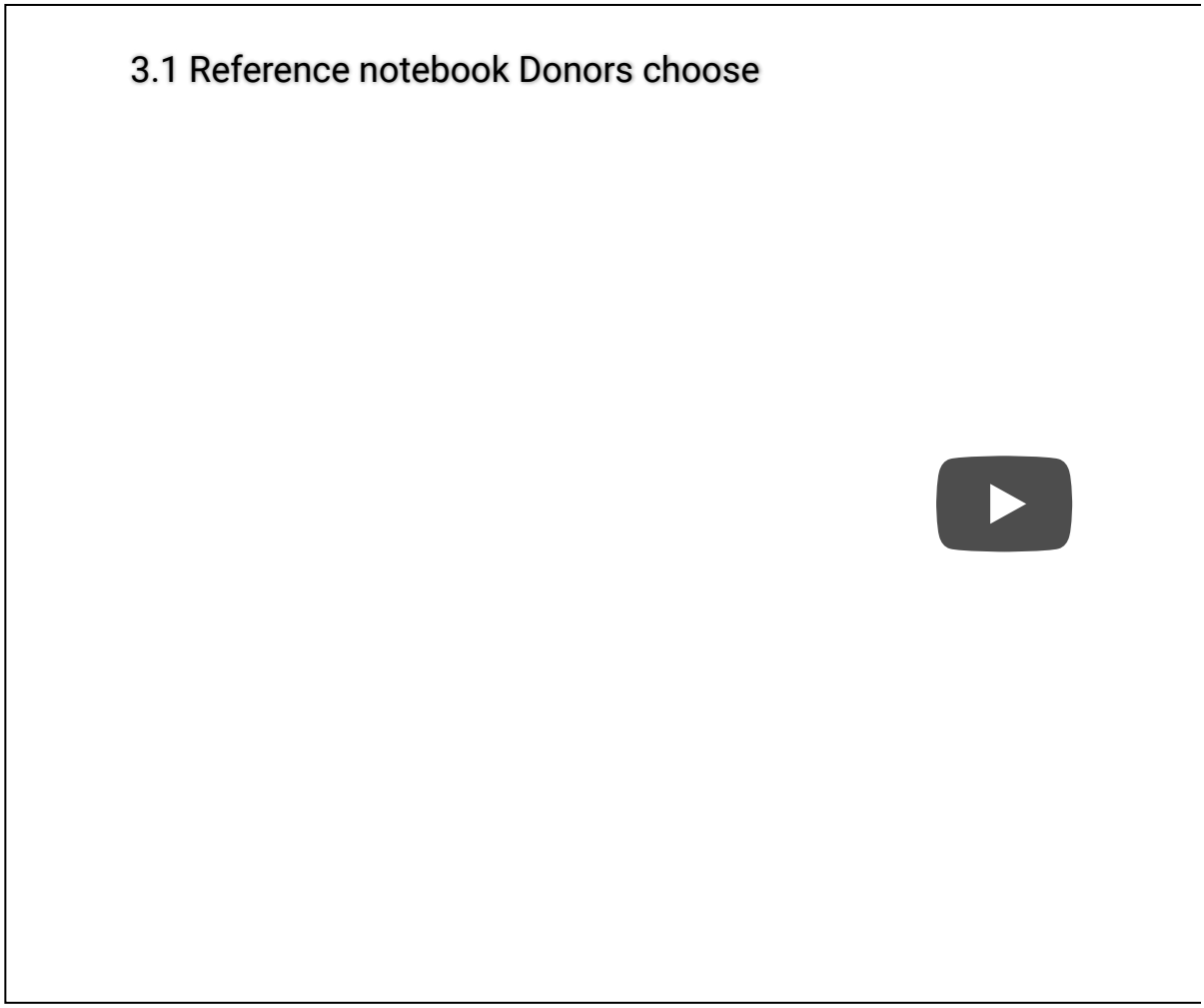
Assignment : DT

Please check below video before attempting this assignment

In [169]:

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

Out[169]:



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 (https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/2916/avg-word2vec-tf-idf-weighted-word2vec/3/module-3-foundations-of-natural-language-processing-and-machine-learning)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 (https://drive.google.com/file/d/1lDca_ge-GYO0iQ6_XDLWePQFMdAA2b8f/view?usp=sharing)

In [170]:

```
#please use below code to load glove vectors
```

or else , you can use below code

In [171]:

```
'''
# 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-variables-in-python/

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


'''
```

Out[171]:

```
'\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# ================================\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(\' \'))\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'
```

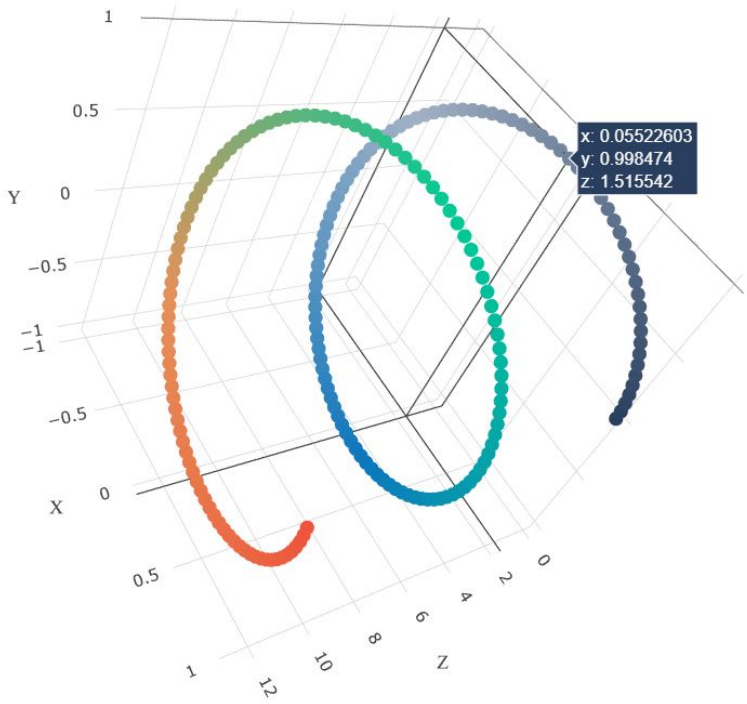
Task - 1 Description

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 $\min_s amp \leq s_s plit$ 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)

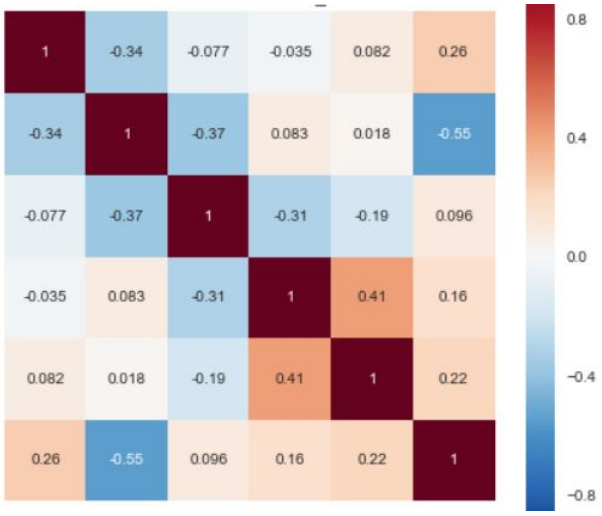
  - Representation of results
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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*

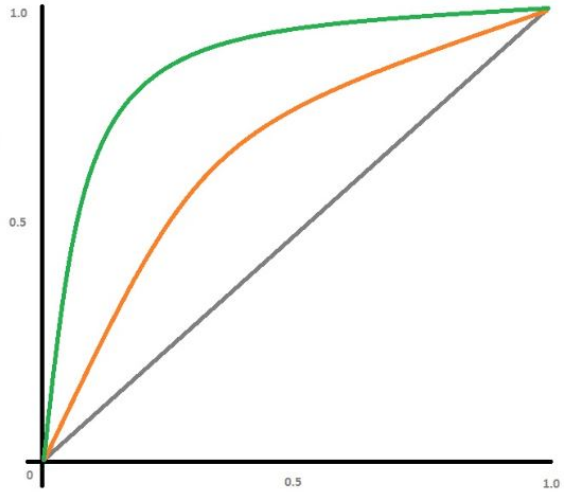
or

- 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) 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 $falsepositivedatap\oint s$

- Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these $falsepositivedatap\oint s$
- Plot the box plot with the $price$ of these $falsepositivedatap\oint s$
- Plot the pdf with the $teacher_{\nu}mber_{o}f_{p}reviously_{p}osted_{p}rojects$ of these $falsepositivedatap\oint s$

Task - 2 Description

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 (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 [172]:

```
import numpy as np
import pandas as pd
from tqdm import tqdm
import nltk
nltk.download('vader_lexicon')
```

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

Out[172]:

```
True
```

In [173]:

```
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 s
mallest students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses an
d multiple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a v
ariety of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our
school is a caring community of successful \
learners which can be seen through collaborative student project based learning
 in and out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportu
nities to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucia
l aspect of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my student
s love to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooki
ng with real food i will take their idea \
and create common core cooking lessons where we learn important math and writing
concepts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work t
hat went into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies
this project would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to m
ake homemade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also c
reate 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. Task 1 - Decision Tree

1.1 Loading Data

In [174]:

```
import pandas
data = pandas.read_csv('../input/donorschoosewithglovevectors/preprocessed_data.csv')
```

In [175]:

```
data.head(1)
```

Out[175]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_post
0	ca	mrs	grades_prek_2	53

In [176]:

```
y = data['project_is_approved']
x = data.drop(['project_is_approved'],axis=1)
```

In [177]:

```
print(x.shape,y.shape)
```

```
(109248, 8) (109248,)
```

Loading glove vectors

In [178]:

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

1.2 Splitting data

In [179]:

```
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)
```

In [180]:

```
print(x_train.shape,y_train.shape)
print(x_test.shape,y_test.shape)
```

```
(73196, 8) (73196,)
(36052, 8) (36052,)
```

In [182]:

```
features = x_train.columns.values
```

In [183]:

```
features
```

Out[183]:

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

1.3 Make Data Model Ready: encoding essay (with TFIDF)

1.3.0 Encoding Essay with TF-IDF

In [184]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

TFIDF = TfidfVectorizer(min_df=10,ngram_range=(1,1),stop_words='english')
TFIDF.fit(x_train['essay'].values)

x_train_tfidf = TFIDF.transform(x_train['essay'].values)
x_test_tfidf = TFIDF.transform(x_test['essay'].values)
```

In [185]:

```
print(x_train_tfidf.shape,y_train.shape)
print(x_test_tfidf.shape,y_test.shape)
```

```
(73196, 13946) (73196,)
(36052, 13946) (36052,)
```

1.3.1 Encoding Essay with TF-IDF W2V

In [187]:

```
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 [188]:

```
#referred from medium
x_train_essays_tfidf_w2v_vectors = []
for essay in tqdm(x_train['essay']):
    vector = np.zeros(300)
    tf_idf_weight =0
    for word in essay.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(essay.count(word)/len(essay.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_train_essays_tfidf_w2v_vectors.append(vector)
```

```
100%|██████████| 73196/73196 [03:27<00:00, 352.44it/s]
```

In [189]:

```
x_test_essays_tfidf_w2v_vectors = []
for essay in tqdm(x_test['essay']):
    vector = np.zeros(300)
    tf_idf_weight =0
    for word in essay.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(essay.count(word)/len(essay.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_test_essays_tfidf_w2v_vectors.append(vector)
```

```
100%|██████████| 36052/36052 [01:43<00:00, 348.72it/s]
```

1.4 Make Data Model Ready: encoding categorical features (using ONE HOT ENCODING)

Using my code from Naive Bayes Assignment

In [190]:

```
from sklearn.feature_extraction.text import CountVectorizer
k=[]
#state
vectorizer = CountVectorizer()
vectorizer.fit(x_train['school_state'].values)

x_train_state_ohe = vectorizer.transform(x_train['school_state'].values)
x_test_state_ohe = vectorizer.transform(x_test['school_state'].values)
k.append(vectorizer.get_feature_names())

print("After Vectorization, School State")
print(x_train_state_ohe.shape,y_train.shape)
print(x_test_state_ohe.shape,y_test.shape)
```

```
After Vectorization, School State
(73196, 51) (73196,)
(36052, 51) (36052,)
```

In [191]:

```
#teacher_prefix
vectorizer.fit(x_train['teacher_prefix'].values)
x_train_teacher_prefix_ohe = vectorizer.transform(x_train['teacher_prefix'].valu
es)
x_test_teacher_prefix_ohe = vectorizer.transform(x_test['teacher_prefix'].values
)
k.append(vectorizer.get_feature_names())

print("After Vectorization, teacher_prefix")
print(x_train_teacher_prefix_ohe.shape,y_train.shape)
print(x_test_teacher_prefix_ohe.shape,y_test.shape)
```

```
After Vectorization, teacher_prefix
(73196, 5) (73196,)
(36052, 5) (36052,)
```

In [192]:

```
#project_grade_category
vectorizer.fit(x_train['project_grade_category'].values)
x_train_project_grade_category_ohe = vectorizer.transform(x_train['project_grade
_category'].values)
x_test_project_grade_category_ohe = vectorizer.transform(x_test['project_grade_c
ategory'].values)
k.append(vectorizer.get_feature_names())

print("After Vectorization, project_grade_category")
print(x_train_project_grade_category_ohe.shape,y_train.shape)
print(x_test_project_grade_category_ohe.shape,y_test.shape)
```

```
After Vectorization, project_grade_category
(73196, 4) (73196,)
(36052, 4) (36052,)
```

In [193]:

```
#clean_categories
vectorizer.fit(x_train['clean_categories'].values)
x_train_clean_categories_ohe = vectorizer.transform(x_train['clean_categories'].
values)
x_test_clean_categories_ohe = vectorizer.transform(x_test['clean_categories'].va
lues)
k.append(vectorizer.get_feature_names())

print("After Vectorization, clean_categories")
print(x_train_clean_categories_ohe.shape,y_train.shape)
print(x_test_clean_categories_ohe.shape,y_test.shape)
```

```
After Vectorization, clean_categories
(73196, 9) (73196,)
(36052, 9) (36052,)
```

In [194]:

```
#clean_sub_categories
vectorizer.fit(x_train['clean_subcategories'].values)
x_train_clean_subcategories_ohe = vectorizer.transform(x_train['clean_subcategor
ies'].values)
x_test_clean_subcategories_ohe = vectorizer.transform(x_test['clean_subcategorie
s'].values)
k.append(vectorizer.get_feature_names())

print("After Vectorization, clean_subcategories")
print(x_train_clean_subcategories_ohe.shape,y_train.shape)
print(x_test_clean_subcategories_ohe.shape,y_test.shape)
```

```
After Vectorization, clean_subcategories
(73196, 30) (73196,)
(36052, 30) (36052,)
```

1.5 Make Data Model Ready: encoding numerical features (using normalization)

In [195]:

```
from sklearn.preprocessing import Normalizer

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

x_train_price_normalized = normalizer.transform(x_train['price'].values.reshape(
1,-1)).reshape(-1,1)
x_test_price_normalized = normalizer.transform(x_test['price'].values.reshape(1,
-1)).reshape(-1,1)

print(x_train_price_normalized.shape)
print(x_test_price_normalized.shape)

#teacher_number_of_previously_posted_projects
normalizer.fit(x_train['teacher_number_of_previously_posted_projects'].values.re
shape(1,-1))

x_train_teacher_number_of_previously_posted_projects_normalized = normalizer.tra
nsform(x_train['teacher_number_of_previously_posted_projects'].values.reshape(1,
-1)).reshape(-1,1)
x_test_teacher_number_of_previously_posted_projects_normalized = normalizer.tran
sform(x_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1
)).reshape(-1,1)

print(x_train_teacher_number_of_previously_posted_projects_normalized.shape)
print(x_test_teacher_number_of_previously_posted_projects_normalized.shape)
```

```
(73196, 1)
(36052, 1)
(73196, 1)
(36052, 1)
```

In [196]:

```
k.append('price')
k.append('teacher_number_of_previously_posted_projects')
```

1.6 Taking sentiment scores of preprocessed essay

In [197]:

```
x_train_essay = x_train['essay'].values
x_test_essay = x_test['essay'].values
```

In [198]:

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

sid = SentimentIntensityAnalyzer()
neg_train=[]
neu_train=[]
pos_train=[]
compound_train=[]

neg_test=[]
neu_test=[]
pos_test=[]
compound_test=[]

for description in tqdm(x_train_essay):
    for_sentiment = description
    ss = sid.polarity_scores(for_sentiment)
    neg_train.append(ss['neg'])
    neu_train.append(ss['neu'])
    pos_train.append(ss['pos'])
    compound_train.append(ss['compound'])

for description in tqdm(x_test_essay):
    for_sentiment = description
    ss = sid.polarity_scores(for_sentiment)
    neg_test.append(ss['neg'])
    neu_test.append(ss['neu'])
    pos_test.append(ss['pos'])
    compound_test.append(ss['compound'])



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

```
  0%|          | 0/73196 [00:00<?, ?it/s]


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


100%|██████████| 73196/73196 [02:58<00:00, 409.77it/s]
100%|██████████| 36052/36052 [01:26<00:00, 416.28it/s]
```

In [199]:

```
import numpy as np
import pandas as pd
#reshaping train sentiment values
neg_train_sentiment = np.array(neg_train).reshape(-1,1).astype('float')
neu_train_sentiment = np.array(neu_train).reshape(-1,1).astype('float')
pos_train_sentiment = np.array(pos_train).reshape(-1,1).astype('float')
compound_train_sentiment = np.array(compound_train).reshape(-1,1).astype('float'
)

#reshaping test sentiment values
neg_test_sentiment = np.array(neg_test).reshape(-1,1).astype('float')
neu_test_sentiment = np.array(neu_test).reshape(-1,1).astype('float')
pos_test_sentiment = np.array(pos_test).reshape(-1,1).astype('float')
compound_test_sentiment = np.array(compound_test).reshape(-1,1).astype('float')
```

1.7 Creating X_Train, X_Test based on encoded values

1.7.1 X_Train, X_Test on TF-IDF

In [200]:

```
from scipy.sparse import hstack

X_Train = hstack((x_train_tfidf,
                  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_normalized,
                  x_train_teacher_number_of_previously_posted_projects_normalize
d,
                  neg_train_sentiment,
                  neu_train_sentiment,
                  pos_train_sentiment,
                  compound_train_sentiment)).tocsr()

X_Test = hstack((x_test_tfidf,
                  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_normalized,
                  x_test_teacher_number_of_previously_posted_projects_normalized
,
                  neg_test_sentiment,
                  neu_test_sentiment,
                  pos_test_sentiment,
                  compound_test_sentiment)).tocsr()
```

In [201]:

```
print("Final Data Matrix")
print(X_Train.shape, y_train.shape)
print(X_Test.shape, y_test.shape)
```

```
Final Data Matrix
(73196, 14051) (73196,)
(36052, 14051) (36052,)
```

1.7.2 X_Train,X_Test on TF-IDF W2V

In [202]:

```
X_Train_tfidf_w2v = hstack((x_train_essays_tfidf_w2v_vectors,
                  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_normalized,
                  x_train_teacher_number_of_previously_posted_projects_normalize
d,
                  neg_train_sentiment,
                  neu_train_sentiment,
                  pos_train_sentiment,
                  compound_train_sentiment)).tocsr()

X_Test_tfidf_w2v = hstack((x_test_essays_tfidf_w2v_vectors,
                  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_normalized,
                  x_test_teacher_number_of_previously_posted_projects_normalized
,
                  neg_test_sentiment,
                  neu_test_sentiment,
                  pos_test_sentiment,
                  compound_test_sentiment)).tocsr()
```

In [203]:

```
print("Final Data Matrix")
print(X_Train_tfidf_w2v.shape, y_train.shape)
print(X_Test_tfidf_w2v.shape, y_test.shape)
```

```
Final Data Matrix
(73196, 405) (73196,)
(36052, 405) (36052,)
```

1.8 Decision Tree Model

1.8.1 Fitting the model with Train data

In [204]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dt = DecisionTreeClassifier(class_weight='balanced')
params = {'max_depth':[1, 5, 10, 50],'min_samples_split':[5, 10, 100, 500]}
clf = GridSearchCV(dt, params, cv= 3, scoring='roc_auc',verbose=1,return_train_s
core=True,n_jobs=-1)
clf.fit(X_Train,y_train)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed:  5.5min finis
hed
```

Out[204]:

```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(class_weight='b
alanced'),
             n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

In [205]:

```
dt_tfidf_w2v = DecisionTreeClassifier(class_weight='balanced')
params = {'max_depth':[1, 5, 10, 50],'min_samples_split':[5, 10, 100, 500]}
clf_tfidf_w2v = GridSearchCV(dt_tfidf_w2v, params, cv= 3, scoring='roc_auc',verb
ose=1,return_train_score=True,n_jobs=-1)
clf_tfidf_w2v.fit(X_Train_tfidf_w2v,y_train)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed: 10.7min finis
hed
```

Out[205]:

```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(class_weight='b
alanced'),
             n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

1.8.2 Finding GridSearchCV Results

In [206]:

```
clf.cv_results_
```

Out[206]:


```
{'mean_fit_time': array([  1.4985683 ,   1.45811677,   1.38435396,
1.42841379,
         6.16861542,   6.15904554,   6.14756274,   5.95943141,
        18.09620563,  17.67427039,  15.93525147,  13.66796573,
       105.48199844, 101.14642866,  75.8042407 ,  47.98834904]),
 'std_fit_time': array([0.0355099 , 0.0668039 , 0.04531246, 0.01170
548, 0.02108441,
       0.00929065, 0.03459466, 0.02107563, 0.29244573, 0.37379803,
       0.24203176, 0.17193645, 1.94864493, 2.02453942, 3.14152222,
       1.13497892]),
 'mean_score_time': array([0.04270895, 0.04580879, 0.04420233, 0.04
63678 , 0.04620949,
       0.04484773, 0.04620481, 0.04506755, 0.04598459, 0.04923232,
       0.04572694, 0.04377635, 0.05539838, 0.05157057, 0.04524668,
       0.0457832 ]),
 'std_score_time': array([0.00075968, 0.00128824, 0.00289893, 0.001
37744, 0.00065247,
       0.00071575, 0.00328359, 0.0021584 , 0.00097466, 0.00543923,
       0.00287756, 0.00092027, 0.00621021, 0.00086453, 0.00582441,
       0.00766907]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10,
10, 10, 10, 50, 50, 50, 50],
             mask=[False, False, False, False, False, False, Fals
e, False,
                   False, False, False, False, False, False, Fals
e, False],
       fill_value='?',
            dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 100, 500, 5,
10, 100, 500, 5, 10, 100, 500, 5,
                   10, 100, 500],
             mask=[False, False, False, False, False, False, Fals
e, False,
                   False, False, False, False, False, False, Fals
e, False],
       fill_value='?',
            dtype=object),
 'params': [{'max_depth': 1, 'min_samples_split': 5},
  {'max_depth': 1, 'min_samples_split': 10},
  {'max_depth': 1, 'min_samples_split': 100},
  {'max_depth': 1, 'min_samples_split': 500},
  {'max_depth': 5, 'min_samples_split': 5},
  {'max_depth': 5, 'min_samples_split': 10},
```

```
  {'max_depth': 5, 'min_samples_split': 100},
  {'max_depth': 5, 'min_samples_split': 500},
  {'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 10, 'min_samples_split': 500},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 500}],
 'split0_test_score': array([0.55014719, 0.55014719, 0.55014719, 0.
55014719, 0.63364725,
        0.63369546, 0.63367218, 0.63409596, 0.65004336, 0.64992128,
        0.65322424, 0.66138941, 0.57332479, 0.57622263, 0.59522706,
        0.62488748]),
 'split1_test_score': array([0.54377348, 0.54377348, 0.54377348, 0.
54377348, 0.62496138,
        0.62496138, 0.62475339, 0.62594997, 0.64185688, 0.64083001,
        0.64100989, 0.65196764, 0.57941547, 0.57772761, 0.59971626,
        0.62228731]),
 'split2_test_score': array([0.5480099 , 0.5480099 , 0.5480099 , 0.
5480099 , 0.6319078 ,
        0.6319078 , 0.63201637, 0.63158765, 0.65024316, 0.64863418,
        0.65183712, 0.65835774, 0.56828087, 0.56930073, 0.59668497,
        0.62342211]),
 'mean_test_score': array([0.54731019, 0.54731019, 0.54731019, 0.54
731019, 0.63017214,
        0.63018821, 0.63014731, 0.63054452, 0.64738113, 0.64646182,
        0.64869042, 0.65723826, 0.57367371, 0.57441699, 0.59720943,
        0.6235323 ]),
 'std_test_score': array([0.00264868, 0.00264868, 0.00264868, 0.002
64868, 0.00375237,
        0.00376729, 0.00387352, 0.0034064 , 0.00390709, 0.00401681,
        0.0054604 , 0.00392703, 0.00455237, 0.00366954, 0.00186985,
        0.00106437]),
 'rank_test_score': array([13, 13, 13, 13,  7,  6,  8,  5,  3,  4,
2,  1, 12, 11, 10,  9],
       dtype=int32),
 'split0_train_score': array([0.5486728 , 0.5486728 , 0.5486728 ,
0.5486728 , 0.64855681,
        0.64855681, 0.64787856, 0.64609151, 0.75212537, 0.75022302,
        0.72936699, 0.71287584, 0.98512006, 0.97816197, 0.92294894,
        0.8406894 ]),
 'split1_train_score': array([0.54836648, 0.54836648, 0.54836648,
0.54836648, 0.65189596,
```

```
        0.65189596, 0.651467  , 0.64988036, 0.75364397, 0.7528533 ,
        0.73721496, 0.71556841, 0.9828366 , 0.97547977, 0.91754269,
        0.84713445]),
 'split2_train_score': array([0.54806949, 0.54806949, 0.54806949,
0.54806949, 0.64640395,
        0.64640395, 0.64581514, 0.64439563, 0.74777075, 0.74616449,
        0.73068967, 0.71211685, 0.98146051, 0.97462263, 0.91590724,
        0.84087543]),
 'mean_train_score': array([0.54836959, 0.54836959, 0.54836959, 0.5
4836959, 0.64895224,
        0.64895224, 0.6483869 , 0.64678917, 0.75118003, 0.74974694,
        0.73242387, 0.71352037, 0.98313905, 0.97608812, 0.91879963,
        0.84289976]),
 'std_train_score': array([0.00024631, 0.00024631, 0.00024631, 0.00
024631, 0.00225947,
        0.00225947, 0.00233519, 0.00229283, 0.00248917, 0.00275137,
        0.00343057, 0.00148096, 0.00150924, 0.0015076 , 0.00300902,
        0.00299534])}
```

In [207]:

```
clf_tfidf_w2v.cv_results_
```

Out[207]:

```
{'mean_fit_time': array([  4.81634855,   4.67399565,   4.63076973,
4.66260084,
        22.49942462,  22.47034089,  22.2868851 ,  21.74832225,
        64.45120263,  64.22504727,  63.25223382,  50.34109942,
       149.44433109, 145.69558899, 114.02461584,  57.27738969]),
 'std_fit_time': array([0.0443123 , 0.10183588, 0.0163685 , 0.03570
209, 0.31572018,
       0.28001073, 0.23330761, 0.16906832, 1.39976369, 1.83842119,
       1.53980863, 1.90325355, 3.22973295, 1.28999334, 5.0810194 ,
       1.2707432 ]),
 'mean_score_time': array([0.08096933, 0.07792171, 0.09115338, 0.09
13678 , 0.08043599,
       0.07702843, 0.09033195, 0.08483879, 0.07801644, 0.0787166 ,
       0.07826082, 0.11581389, 0.08481574, 0.08067218, 0.07533161,
       0.07022786]),
 'std_score_time': array([0.01455819, 0.00443644, 0.01438225, 0.014
6115 , 0.00384881,
       0.00053598, 0.0107953 , 0.01540525, 0.00091252, 0.00181088,
       0.00207531, 0.0221968 , 0.0034179 , 0.00076452, 0.00865075,
       0.00793974]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10,
10, 10, 10, 50, 50, 50, 50],
             mask=[False, False, False, False, False, False, Fals
e, False,
                   False, False, False, False, False, False, Fals
e, False],
       fill_value='?',
            dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 100, 500, 5,
10, 100, 500, 5, 10, 100, 500, 5,
                   10, 100, 500],
             mask=[False, False, False, False, False, False, Fals
e, False,
                   False, False, False, False, False, False, Fals
e, False],
       fill_value='?',
            dtype=object),
 'params': [{'max_depth': 1, 'min_samples_split': 5},
  {'max_depth': 1, 'min_samples_split': 10},
  {'max_depth': 1, 'min_samples_split': 100},
  {'max_depth': 1, 'min_samples_split': 500},
  {'max_depth': 5, 'min_samples_split': 5},
  {'max_depth': 5, 'min_samples_split': 10},
```

```
  {'max_depth': 5, 'min_samples_split': 100},
  {'max_depth': 5, 'min_samples_split': 500},
  {'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 10, 'min_samples_split': 500},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 500}],
 'split0_test_score': array([0.55014719, 0.55014719, 0.55014719, 0.
55014719, 0.63707377,
        0.63707377, 0.63724736, 0.63724736, 0.60871463, 0.61023612,
        0.61067837, 0.62141264, 0.5330442 , 0.54318206, 0.57298677,
        0.61725737]),
 'split1_test_score': array([0.54377348, 0.54377348, 0.54377348, 0.
54377348, 0.62436369,
        0.62436369, 0.62436369, 0.62497682, 0.5974019 , 0.59736743,
        0.60131234, 0.61906055, 0.53290352, 0.5375853 , 0.57008674,
        0.61260081]),
 'split2_test_score': array([0.5480099 , 0.5480099 , 0.5480099 , 0.
5480099 , 0.63346198,
        0.63346198, 0.63346198, 0.63426902, 0.61292315, 0.6119882 ,
        0.61011247, 0.62684132, 0.53174744, 0.53398098, 0.56443784,
        0.62103358]),
 'mean_test_score': array([0.54731019, 0.54731019, 0.54731019, 0.54
731019, 0.63163315,
        0.63163315, 0.63169101, 0.6321644 , 0.60634656, 0.60653058,
        0.60736773, 0.62243817, 0.53256505, 0.53824945, 0.56917045,
        0.61696392]),
 'std_test_score': array([0.00264868, 0.00264868, 0.00264868, 0.002
64868, 0.00534758,
        0.00534758, 0.00540675, 0.00522581, 0.00655404, 0.00651869,
        0.00428803, 0.00325821, 0.00058099, 0.00378557, 0.00354972,
        0.00344891]),
 'rank_test_score': array([11, 11, 11, 11,  3,  3,  2,  1,  9,  8,
7,  5, 16, 15, 10,  6],
       dtype=int32),
 'split0_train_score': array([0.5486728 , 0.5486728 , 0.5486728 ,
0.5486728 , 0.66087683,
        0.66087683, 0.66045565, 0.66045565, 0.82350841, 0.82216528,
        0.79466163, 0.74108384, 0.99983935, 0.99899224, 0.90264977,
        0.75843221]),
 'split1_train_score': array([0.54836648, 0.54836648, 0.54836648,
0.54836648, 0.66531253,
```

```
        0.66531253, 0.66531253, 0.66422542, 0.8225945 , 0.82134091,
        0.79302031, 0.74327052, 0.99990504, 0.99900819, 0.90515013,
        0.76450473]),
 'split2_train_score': array([0.54806949, 0.54806949, 0.54806949,
0.54806949, 0.66004633,
        0.66004633, 0.66004633, 0.65830667, 0.81342866, 0.81175683,
        0.78425324, 0.74027804, 0.99988942, 0.99907328, 0.90751846,
        0.76091993]),
 'mean_train_score': array([0.54836959, 0.54836959, 0.54836959, 0.5
4836959, 0.66207857,
        0.66207857, 0.66193817, 0.66099592, 0.81984386, 0.81842101,
        0.79064506, 0.74154413, 0.99987794, 0.99902457, 0.90510612,
        0.76128562]),
 'std_train_score': array([2.46307238e-04, 2.46307238e-04, 2.463072
38e-04, 2.46307238e-04,
        2.31175980e-03, 2.31175980e-03, 2.39187868e-03, 2.44633363e
-03,
        4.55154309e-03, 4.72428732e-03, 4.56909700e-03, 1.26428655e
-03,
        2.80221168e-05, 3.50544808e-05, 1.98787846e-03, 2.49254575e
-03])}
```

1.8.3 Converting GridSearchCV results to dataframe and then group by hyperparameters to return train_score and CV_score dfs

In [208]:

```
cv_results_df = pd.DataFrame(clf.cv_results_)
```

In [209]:

```
cv_results_df
```

Out[209]:

st_score	split0_train_score	split1_train_score	split2_train_score	mean_train_score	std_train_sc
	0.548673	0.548366	0.548069	0.548370	0.000246
	0.548673	0.548366	0.548069	0.548370	0.000246
	0.548673	0.548366	0.548069	0.548370	0.000246
	0.548673	0.548366	0.548069	0.548370	0.000246
	0.648557	0.651896	0.646404	0.648952	0.002259
	0.648557	0.651896	0.646404	0.648952	0.002259
	0.647879	0.651467	0.645815	0.648387	0.002335
	0.646092	0.649880	0.644396	0.646789	0.002293
	0.752125	0.753644	0.747771	0.751180	0.002489
	0.750223	0.752853	0.746164	0.749747	0.002751
	0.729367	0.737215	0.730690	0.732424	0.003431
	0.712876	0.715568	0.712117	0.713520	0.001481
	0.985120	0.982837	0.981461	0.983139	0.001509
	0.978162	0.975480	0.974623	0.976088	0.001508
	0.922949	0.917543	0.915907	0.918800	0.003009
	0.840689	0.847134	0.840875	0.842900	0.002995

In [210]:

```
scores = pd.DataFrame(clf.cv_results_).groupby(['param_max_depth','param_min_samples_split']).max().unstack()
'''This Groups by the data frame with two columns and finds the max for each column in this combination
   and unstack returns as a dataframe with row index as column 1 and column index as column 2'''
```

Out[210]:

```
'This Groups by the data frame with two columns and finds the max for each column in this combination \n   and unstack returns as a dataframe with row index as column 1 and column index as column 2'
```

In [211]:

```
train_score = scores['mean_train_score']
train_score
```

Out[211]:

param_min_samples_split	5	10	100	500
param_max_depth				
1	**0.548370**	**0.548370**	**0.548370**	**0.548370**
5	**0.648952**	**0.648952**	**0.648387**	**0.646789**
10	**0.751180**	**0.749747**	**0.732424**	**0.713520**
50	**0.983139**	**0.976088**	**0.918800**	**0.842900**

In [212]:

```
test_score = scores['mean_test_score']
test_score
```

Out[212]:

param_min_samples_split	5	10	100	500
param_max_depth				
1	**0.547310**	**0.547310**	**0.547310**	**0.547310**
5	**0.630172**	**0.630188**	**0.630147**	**0.630545**
10	**0.647381**	**0.646462**	**0.648690**	**0.657238**
50	**0.573674**	**0.574417**	**0.597209**	**0.623532**

In [213]:

```
cv_results_df_tfidf_w2v = pd.DataFrame(clf_tfidf_w2v.cv_results_)
```

In [214]:

```
scores_tfidf_w2v = pd.DataFrame(clf_tfidf_w2v.cv_results_).groupby(['param_max_depth','param_min_samples_split']).max().unstack()
```

In [215]:

```
train_score_tfidf_w2v = scores_tfidf_w2v['mean_train_score']
train_score_tfidf_w2v
```

Out[215]:

param_min_samples_split	5	10	100	500
param_max_depth				
1	**0.548370**	**0.548370**	**0.548370**	**0.548370**
5	**0.662079**	**0.662079**	**0.661938**	**0.660996**
10	**0.819844**	**0.818421**	**0.790645**	**0.741544**
50	**0.999878**	**0.999025**	**0.905106**	**0.761286**

In [216]:

```
test_score_tfidf_w2v = scores_tfidf_w2v['mean_test_score']
test_score_tfidf_w2v
```

Out[216]:

param_min_samples_split	5	10	100	500
param_max_depth				
1	**0.547310**	**0.547310**	**0.547310**	**0.547310**
5	**0.631633**	**0.631633**	**0.631691**	**0.632164**
10	**0.606347**	**0.606531**	**0.607368**	**0.622438**
50	**0.532565**	**0.538249**	**0.569170**	**0.616964**

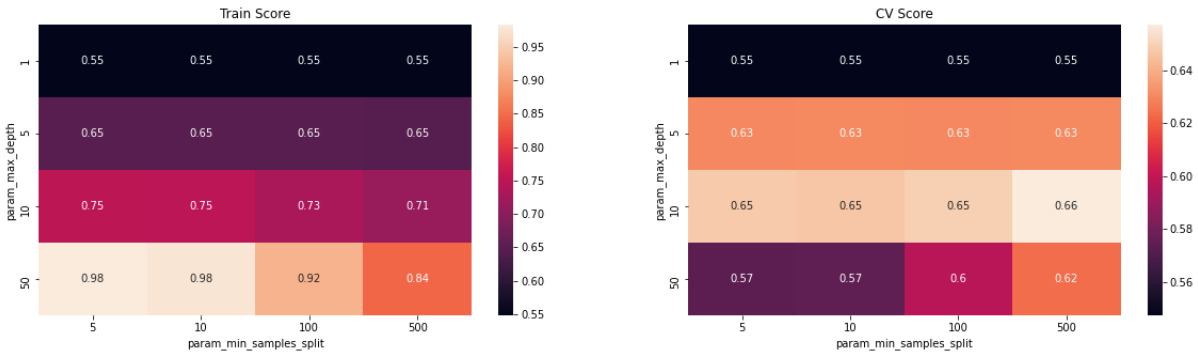
1.8.4 Plotting Train and CV Score heatmaps

In [217]:

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
fig,ax = plt.subplots(ncols=2,figsize=(20,5))
sns.heatmap(train_score,annot=True,ax=ax[0])
sns.heatmap(test_score,annot=True,ax=ax[1])
ax[0].set_title('Train Score')
ax[1].set_title('CV Score')
```

Out[217]:

```
Text(0.5, 1.0, 'CV Score')
```

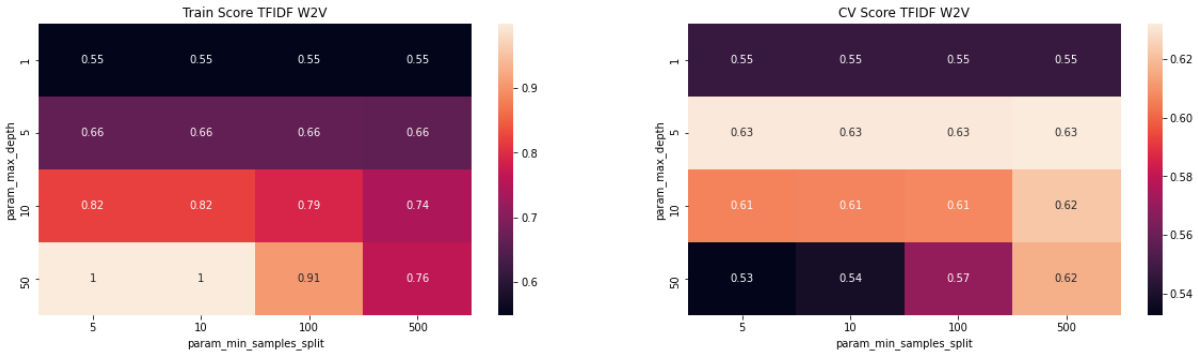


In [218]:

```
fig,ax = plt.subplots(ncols=2,figsize=(20,5))
sns.heatmap(train_score_tfidf_w2v,annot=True,ax=ax[0])
sns.heatmap(test_score_tfidf_w2v,annot=True,ax=ax[1])
ax[0].set_title('Train Score TFIDF W2V')
ax[1].set_title('CV Score TFIDF W2V')
```

Out[218]:

```
Text(0.5, 1.0, 'CV Score TFIDF W2V')
```



1.8.5 Finding the best_hyperparameters

In [219]:

```
best_model_dt = clf.best_estimator_
best_model_dt.fit(X_Train,y_train)
```

Out[219]:

```
DecisionTreeClassifier(class_weight='balanced', max_depth=10,
                       min_samples_split=500)
```

In [220]:

```
clf.best_params_
```

Out[220]:

```
{'max_depth': 10, 'min_samples_split': 500}
```

In [221]:

```
best_model_dt_max_depth = clf.best_params_['max_depth']
best_model_dt_min_sample_split = clf.best_params_['min_samples_split']
```

In [222]:

```
best_model_dt_tfidf_w2v = clf_tfidf_w2v.best_estimator_
best_model_dt_tfidf_w2v.fit(X_Train_tfidf_w2v,y_train)
```

Out[222]:

```
DecisionTreeClassifier(class_weight='balanced', max_depth=5,
                       min_samples_split=500)
```

In [223]:

```
best_model_dt_tfidf_w2v_max_depth = clf_tfidf_w2v.best_params_['max_depth']
best_model_dt_tfidf_w2v_min_sample_split = clf_tfidf_w2v.best_params_['min_sampl
es_split']
```

1.8.6 Plotting ROC

In [224]:

```
X_Train
```

Out[224]:

```
<73196x14051 sparse matrix of type '<class 'numpy.float64'>'
        with 7436963 stored elements in Compressed Sparse Row forma
t>
```

In [225]:

```
y_pred_prob_train = best_model_dt.predict_proba(X_Train)
y_pred_prob_test = best_model_dt.predict_proba(X_Test)
```

In [226]:

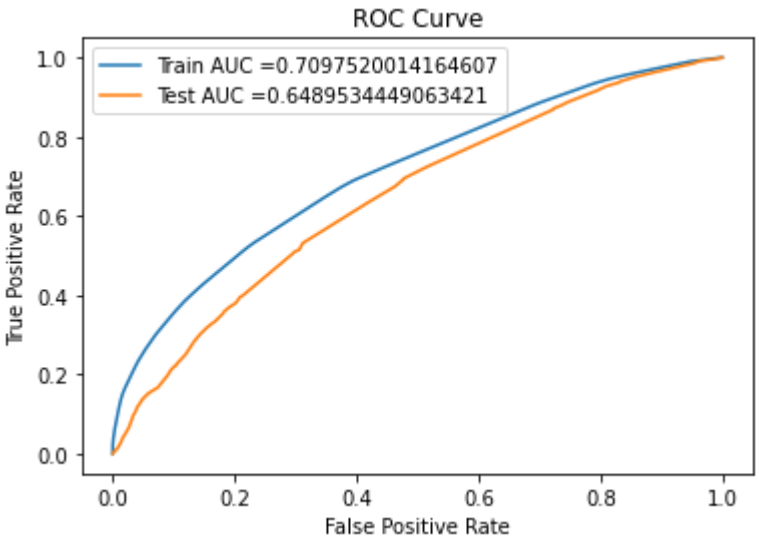
```
y_pred_prob_train
#returns value for two classes
```

Out[226]:

```
array([[0.42835393, 0.57164607],
       [0.        , 1.        ],
       [0.28769908, 0.71230092],
       ...,
       [0.77837587, 0.22162413],
       [0.60997123, 0.39002877],
       [0.64735674, 0.35264326]])
```

In [227]:

```
from sklearn.metrics import roc_curve,auc
import matplotlib.pyplot as plt
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_prob_train[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_prob_test[:,1])

train_auc_dt = auc(train_fpr, train_tpr)
test_auc_dt = auc(test_fpr,test_tpr)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(train_auc_dt))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(test_auc_dt))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

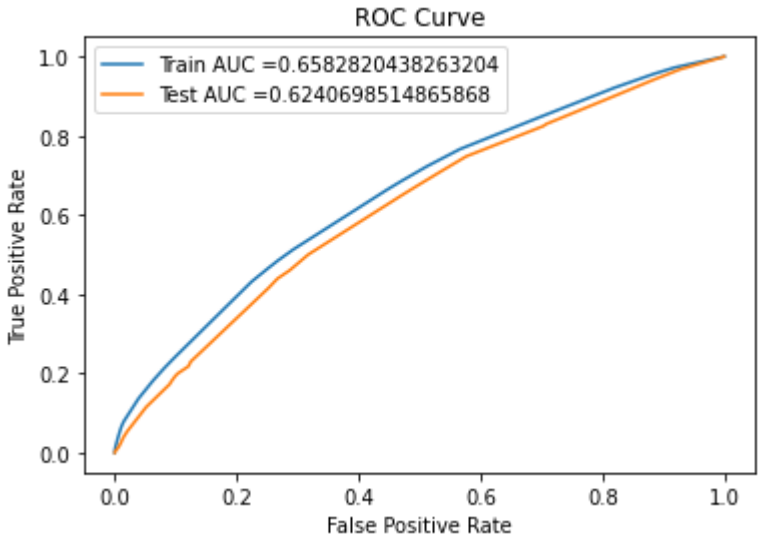


In [228]:

```
y_pred_prob_train_tfidf_w2v = best_model_dt_tfidf_w2v.predict_proba(X_Train_tfid
f_w2v)
y_pred_prob_test_tfidf_w2v = best_model_dt_tfidf_w2v.predict_proba(X_Test_tfidf_
w2v)
```

In [229]:

```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_prob_train_tfidf
_w2v[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_prob_test_tfidf_w2v
[:,1])

train_auc_dt_tfidf_w2v = auc(train_fpr, train_tpr)
test_auc_dt_tfidf_w2v = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(train_auc_tfidf_w2v))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(test_auc_tfidf_w2v))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```



1.8.7 Plot the confusion Matrix

In [230]:

```
from sklearn.metrics import confusion_matrix
y_pred = best_model_dt.predict(X_Test)
```

In [231]:

```
cm = confusion_matrix(y_test,y_pred)
cm
```

Out[231]:

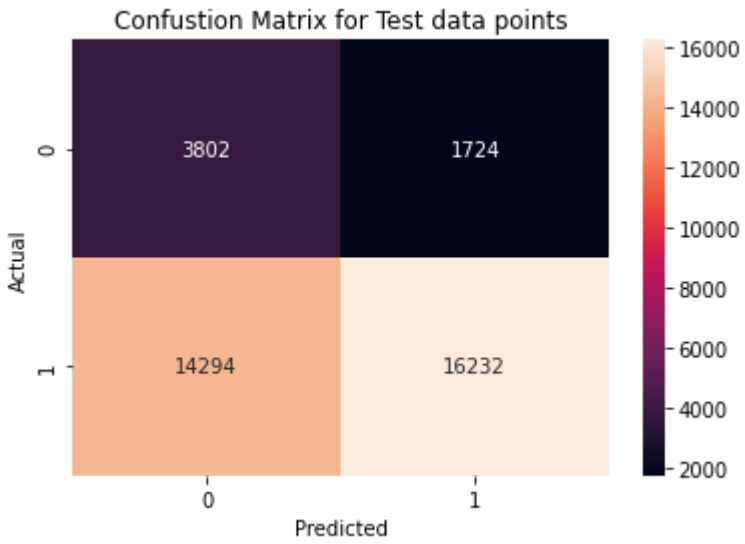
```
array([[ 3802,  1724],
       [14294, 16232]])
```

In [232]:

```
sns.heatmap(cm,annot=True,fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confustion Matrix for Test data points')
```

Out[232]:

```
Text(0.5, 1.0, 'Confustion Matrix for Test data points')
```



In [233]:

```
y_pred_tfidf_w2v = best_model_dt_tfidf_w2v.predict(X_Test_tfidf_w2v)
```

In [234]:

```
cm_tfidf_w2v = confusion_matrix(y_test,y_pred_tfidf_w2v)
cm_tfidf_w2v
```

Out[234]:

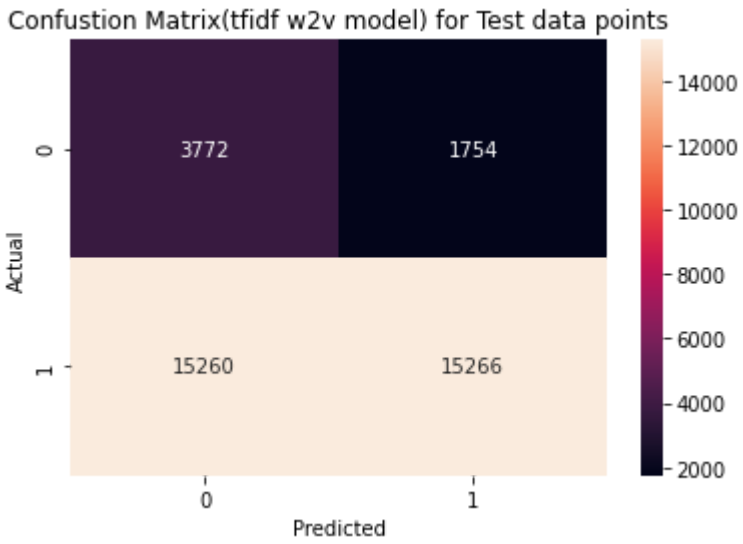
```
array([[ 3772,  1754],
       [15260, 15266]])
```

In [235]:

```
sns.heatmap(cm_tfidf_w2v,annot=True,fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confustion Matrix(tfidf w2v model) for Test data points')
```

Out[235]:

```
Text(0.5, 1.0, 'Confustion Matrix(tfidf w2v model) for Test data po
ints')
```



1.8.8 Getting all the False positive points

In [236]:

```
df = pd.DataFrame({'Actual': y_test.values,'Predicted': y_pred})
```

In [237]:

```
df
```

Out[237]:

	Actual	Predicted
0	**1**	**0**
1	**0**	**1**
2	**1**	**0**
3	**1**	**1**
4	**1**	**1**
...	**...**	**...**
36047	**1**	**1**
36048	**1**	**1**
36049	**1**	**1**
36050	**1**	**0**
36051	**0**	**0**

36052 rows × 2 columns

In [238]:

```
fp = df.loc[(df.loc[:,'Actual']==0) & (df.loc[:,'Predicted']==1)]
fp
```

Out[238]:

	Actual	Predicted
1	**0**	**1**
15	**0**	**1**
43	**0**	**1**
92	**0**	**1**
113	**0**	**1**
...	...	...
35936	**0**	**1**
35979	**0**	**1**
36014	**0**	**1**
36025	**0**	**1**
36038	**0**	**1**

1724 rows × 2 columns

In [239]:

```
false_positive_points_test_data = x_test.iloc[fp.index,:]
```

In [240]:

```
false_positive_points_test_data.head()
```

Out[240]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_po
20492	**mi**	**mrs**	**grades_3_5**	**15**
6405	**fl**	**ms**	**grades_3_5**	**47**
13864	**tx**	**mrs**	**grades_prek_2**	**25**
32821	**mi**	**ms**	**grades_prek_2**	**9**
88186	**il**	**mrs**	**grades_prek_2**	**5**

In [241]:

```
len(false_positive_points_test_data)
```

Out[241]:

1724

In [242]:

```
df_tfidf_w2v = pd.DataFrame({'Actual': y_test.values,'Predicted': y_pred_tfidf_w
2v})
fp_tfidf_w2v = df_tfidf_w2v.loc[(df_tfidf_w2v.loc[:,'Actual']==0) & (df_tfidf_w2
v.loc[:,'Predicted']==1)]
false_positive_points_test_data_tfidf_w2v = x_test.iloc[fp_tfidf_w2v.index,:]
```

In [243]:

```
false_positive_points_test_data_tfidf_w2v.head()
```

Out[243]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_po
20492	**mi**	**mrs**	**grades_3_5**	**15**
6405	**fl**	**ms**	**grades_3_5**	**47**
68169	**ut**	**mr**	**grades_6_8**	**1**
13864	**tx**	**mrs**	**grades_prek_2**	**25**
35538	**ca**	**mrs**	**grades_prek_2**	**10**

In [244]:

```
len(false_positive_points_test_data_tfidf_w2v)
```

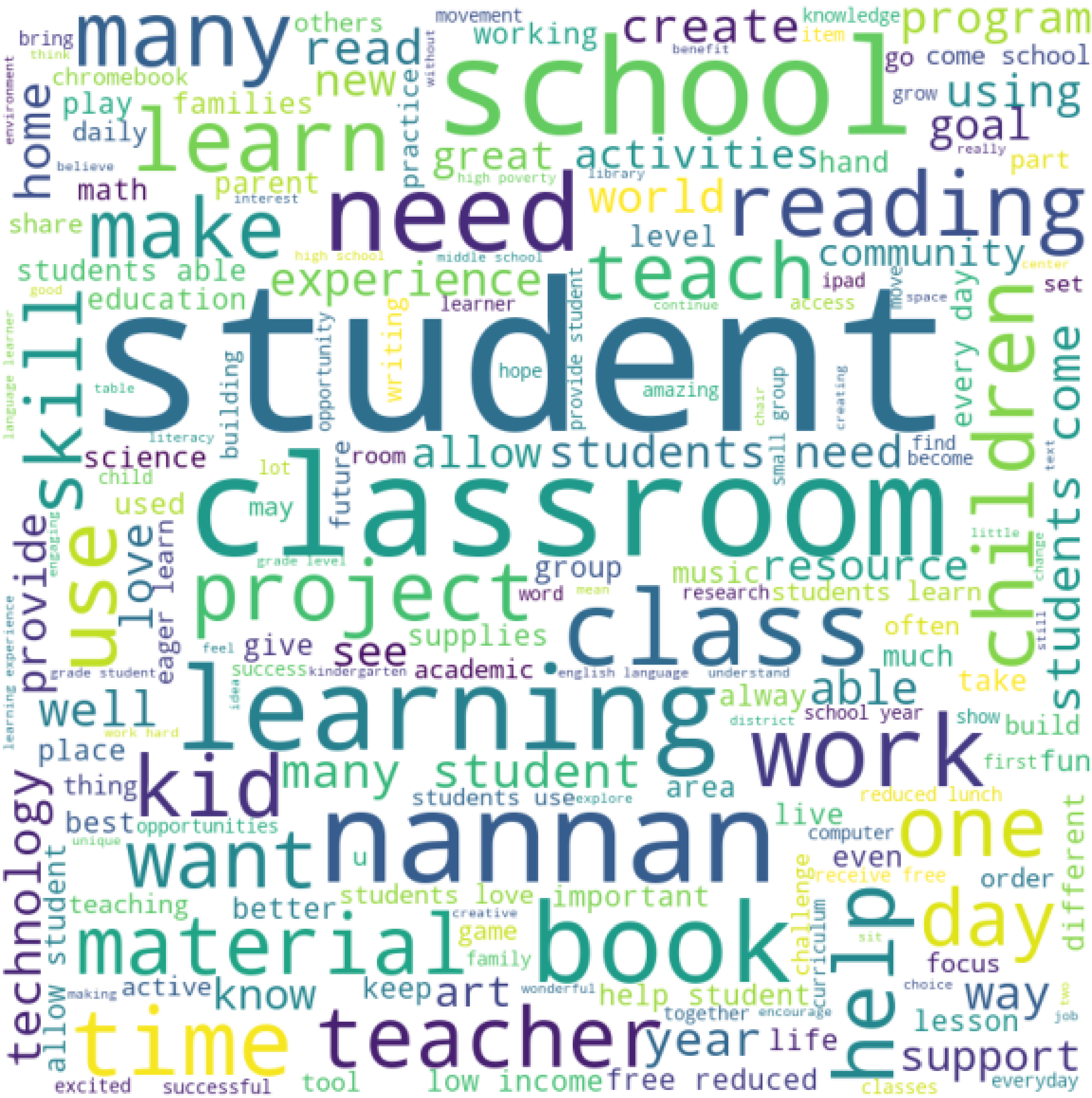
Out[244]:

1754

1.8.8.1 Plotting word cloud

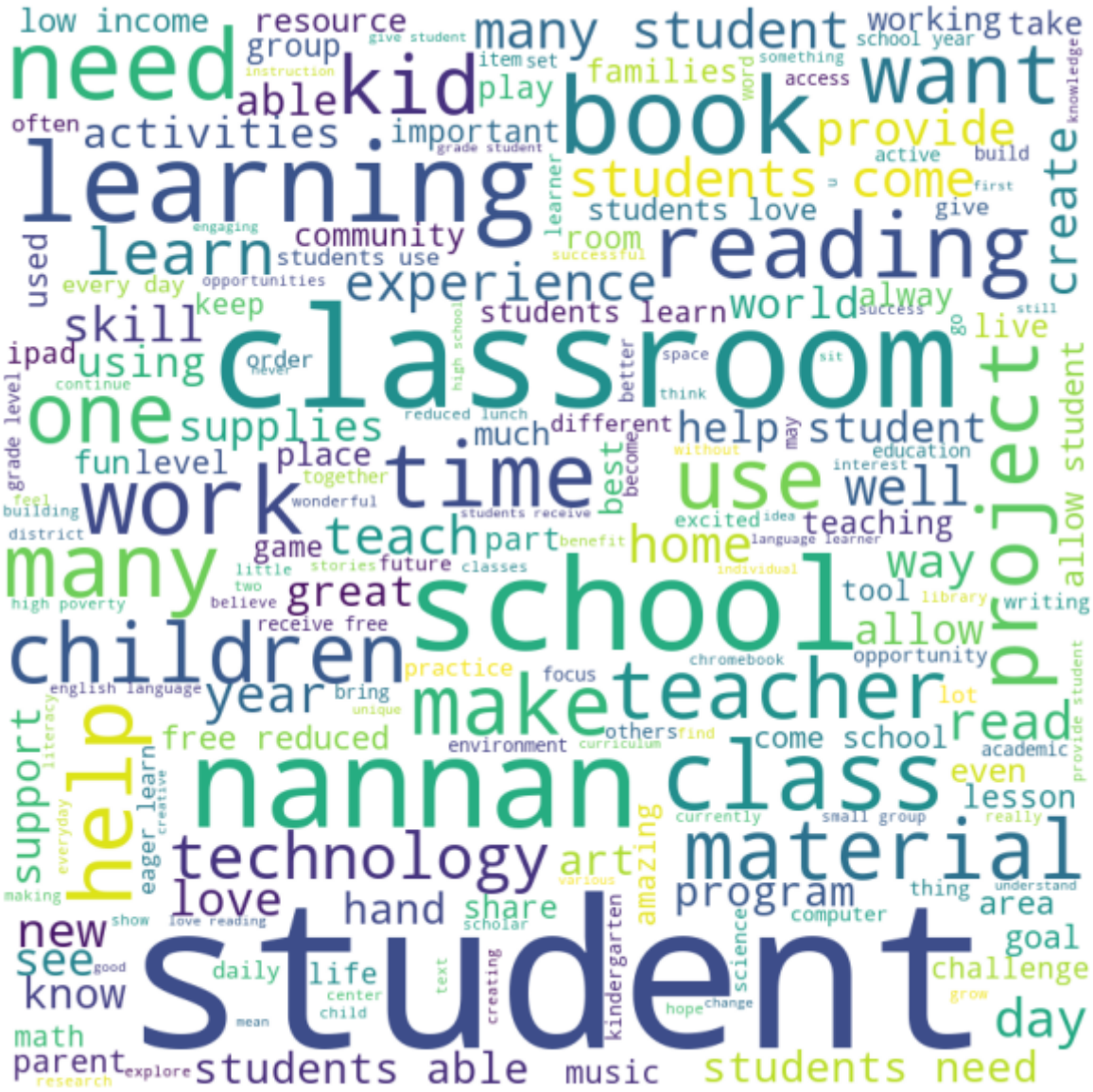
In [245]:

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

fp_essay = false_positive_points_test_data.loc[:,'essay']
comment_words = ''
stopwords = set(STOPWORDS)
for val in fp_essay:

    # typecaste each val to string
    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)

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

plt.show()
```


many
others
read
movement
working
create
knowledge
program
bring
school
go come school
chromebook
new
practice
using
play
families
grow
goal
daily
learn
home
great
activities
hand
part
math
parent
need
world
reading
share
make
level
students able
experience
teach
community
education
ipad
set
access
student
skill
writing
hope
opportunity
provide student
every day
children
come
building
amazing
group
science
room
allow
students
need
find
become
child
lot
classroom
used
may
future
provide
use
love
eager learn
project
group
music
resource
word
research
students learn
class
supplies
often
give
see
academic
much
take
success
kindergarten
english language
understand
alway
able
students
well
learning
school year
show
build
place
work
first
fun
thing
kid
many student
area
students use
reduced lunch
technology
best
opportunities
live
one
computer
want
nannan
even
u
receive free
order
teaching
students love
important
day
different
better
student
challenge
material
book
game
family
focus
active
know
keep
art
help student
way
allow
time
together
lesson
curriculum
teacher
year
life
help
support
excited
successful
tool
low income
free reduced
classes
everyday

In [246]:

```
fp_essay = false_positive_points_test_data_tfidf_w2v.loc[:,'essay']
comment_words = ''
stopwords = set(STOPWORDS)
for val in fp_essay:

    # typecaste each val to string
    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)

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

plt.show()
```



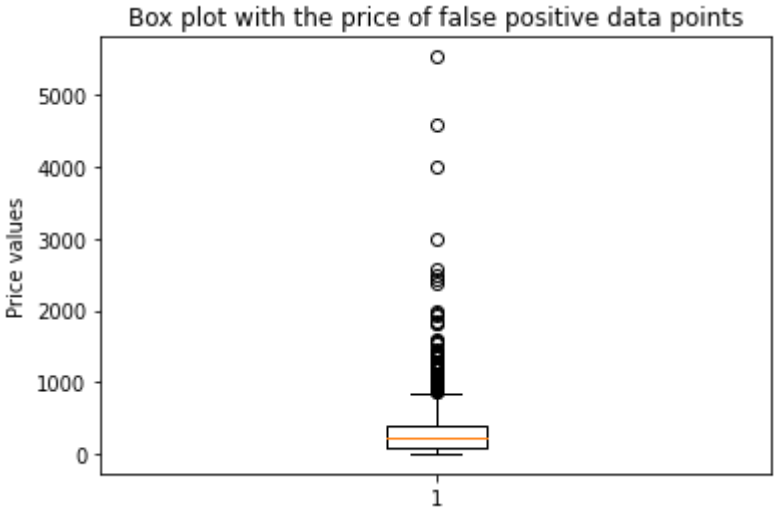
1.8.8.2 Plot the box plot with the `price` of these `false positive data points`

In [247]:

```
plt.boxplot(false_positive_points_test_data.loc[:,'price'])
plt.title("Box plot with the price of false positive data points")
plt.ylabel("Price values")
```

Out[247]:

```
Text(0, 0.5, 'Price values')
```

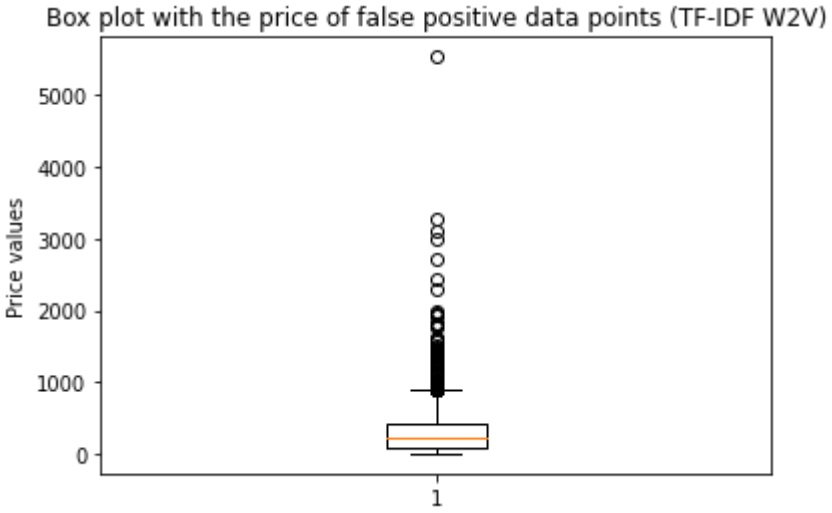


In [248]:

```
plt.boxplot(false_positive_points_test_data_tfidf_w2v.loc[:,'price'])
plt.title("Box plot with the price of false positive data points (TF-IDF W2V)")
plt.ylabel("Price values")
```

Out[248]:

```
Text(0, 0.5, 'Price values')
```



1.8.8.3 Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

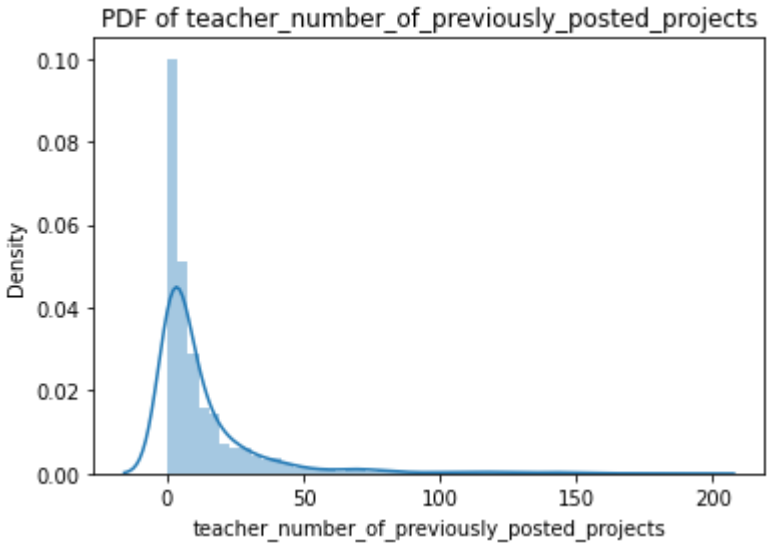
In [249]:

```
sns.distplot(false_positive_points_test_data.loc[:,'teacher_number_of_previously_posted_projects'])
plt.title("PDF of teacher_number_of_previously_posted_projects")
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[249]:

```
Text(0.5, 1.0, 'PDF of teacher_number_of_previously_posted_projects')
```



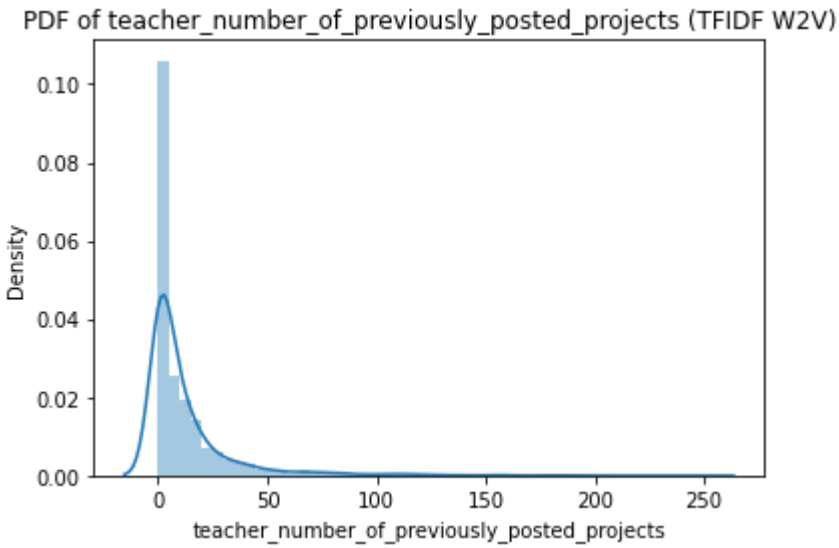
In [250]:

```
sns.distplot(false_positive_points_test_data_tfidf_w2v.loc[:,'teacher_number_of_
previously_posted_projects'])
plt.title("PDF of teacher_number_of_previously_posted_projects (TFIDF W2V)")
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:255
7: FutureWarning: `distplot` is a deprecated function and will be r
emoved in a future version. Please adapt your code to use either `d
isplot` (a figure-level function with similar flexibility) or `hist
plot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[250]:

```
Text(0.5, 1.0, 'PDF of teacher_number_of_previously_posted_projects
(TFIDF W2V)')
```



Task 2: Decision Trees with Non-Zero Feature Importances

2.1 Finding Non-Zero Features trained on DecisionTree with max_depth=None

In [251]:

```
DecisionTree = DecisionTreeClassifier(class_weight='balanced')
DecisionTree.fit(X_Train,y_train)
```

Out[251]:

```
DecisionTreeClassifier(class_weight='balanced')
```

In [252]:

```
#Non-zero feature importances
X_Train_new = X_Train[:,DecisionTree.feature_importances_!=0]
X_Test_new = X_Test[:,DecisionTree.feature_importances_!=0]
```

In [253]:

```
print(X_Train_new.shape,y_train.shape)
print(X_Test_new.shape,y_test.shape)
```

```
(73196, 2873) (73196,)
(36052, 2873) (36052,)
```

2.2 Tuning Hyperparameters of Decision Tree with GridSearchCV

In [254]:

```
dt_with_nonzero_features= DecisionTreeClassifier(class_weight='balanced')
params = {'max_depth':[1, 5, 10, 50],'min_samples_split':[5, 10, 100, 500]}
clf_3 = GridSearchCV(dt_with_nonzero_features, params, cv= 3, scoring='roc_auc',
verbose=1,return_train_score=True,n_jobs=-1)
clf_3.fit(X_Train_new,y_train)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed:  4.6min finis
hed
```

Out[254]:

```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(class_weight='b
alanced'),
             n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

In [255]:

```
clf_3.cv_results_
```

Out[255]:


```
{'mean_fit_time': array([ 1.35922956,  1.36848497,  1.35858226,  1.
41409461,  5.83518648,
         5.43844326,  5.20009534,  5.04963986, 14.46034447, 14.5523
0792,
        13.26966127, 11.09823418, 86.83702747, 82.37374322, 62.1867
0996,
        37.50462826]),
 'std_fit_time': array([0.07805345, 0.10551094, 0.08632007, 0.12155
724, 0.05793359,
        0.28869056, 0.07128091, 0.00281187, 0.04871839, 0.1619503 ,
        0.21567284, 0.18056875, 1.5002921 , 1.14660136, 2.08531456,
        0.61901644]),
 'mean_score_time': array([0.03880843, 0.04997245, 0.0388782 , 0.03
429063, 0.03716032,
        0.03726355, 0.0357732 , 0.03614767, 0.03870114, 0.03803476,
        0.03948577, 0.03772489, 0.04488436, 0.04303018, 0.03936815,
        0.03344385]),
 'std_score_time': array([0.00377975, 0.00844233, 0.00331217, 0.000
91612, 0.00113276,
        0.00078364, 0.00026635, 0.00046015, 0.00020514, 0.00038321,
        0.00243496, 0.0010495 , 0.00114296, 0.00037229, 0.00592585,
        0.00437531]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10,
10, 10, 10, 50, 50, 50, 50],
              mask=[False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 100, 500, 5,
10, 100, 500, 5, 10, 100, 500, 5,
                    10, 100, 500],
              mask=[False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 1, 'min_samples_split': 5},
  {'max_depth': 1, 'min_samples_split': 10},
  {'max_depth': 1, 'min_samples_split': 100},
  {'max_depth': 1, 'min_samples_split': 500},
```

```
  {'max_depth': 5, 'min_samples_split': 5},
  {'max_depth': 5, 'min_samples_split': 10},
  {'max_depth': 5, 'min_samples_split': 100},
  {'max_depth': 5, 'min_samples_split': 500},
  {'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 10, 'min_samples_split': 500},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 500}],
 'split0_test_score': array([0.55014719, 0.55014719, 0.55014719, 0.
55014719, 0.63327489,
        0.63322668, 0.6332515 , 0.6338196 , 0.64903817, 0.64881046,
        0.65166494, 0.66069029, 0.56630014, 0.57627561, 0.59216848,
        0.63266962]),
 'split1_test_score': array([0.54377348, 0.54377348, 0.54377348, 0.
54377348, 0.62496138,
        0.62496138, 0.62475339, 0.62594997, 0.6427734 , 0.64390237,
        0.64119433, 0.6512898 , 0.58345895, 0.58039864, 0.6047903 ,
        0.62017995]),
 'split2_test_score': array([0.5480099 , 0.5480099 , 0.5480099 , 0.
5480099 , 0.6312028 ,
        0.6312028 , 0.63117517, 0.6307491 , 0.6496485 , 0.64989456,
        0.65415975, 0.65911421, 0.56652799, 0.57013419, 0.60076925,
        0.62204995]),
 'mean_test_score': array([0.54731019, 0.54731019, 0.54731019, 0.54
731019, 0.62981302,
        0.62979695, 0.62972669, 0.63017289, 0.64715336, 0.6475358 ,
        0.64900634, 0.65703144, 0.57209569, 0.57560281, 0.59924268,
        0.62496651]),
 'std_test_score': array([0.00264868, 0.00264868, 0.00264868, 0.002
64868, 0.00353338,
        0.00351768, 0.00361737, 0.0032385 , 0.00310711, 0.00260707,
        0.00561704, 0.00411062, 0.00803557, 0.00421736, 0.00526469,
        0.00550016]),
 'rank_test_score': array([13, 13, 13, 13,  6,  7,  8,  5,  4,  3,
2,  1, 12, 11, 10,  9],
       dtype=int32),
 'split0_train_score': array([0.5486728 , 0.5486728 , 0.5486728 ,
0.5486728 , 0.64858327,
        0.64858327, 0.64790519, 0.64614257, 0.75242694, 0.75076475,
        0.72902179, 0.71225487, 0.98645142, 0.97817261, 0.92103881,
        0.83230188]),
```

```
 'split1_train_score': array([0.54836648, 0.54836648, 0.54836648,
0.54836648, 0.65189596,
        0.65189596, 0.651467  , 0.64988036, 0.75382792, 0.75274745,
        0.73632556, 0.71540803, 0.98200569, 0.97493587, 0.91467018,
        0.84676857]),
 'split2_train_score': array([0.54806949, 0.54806949, 0.54806949,
0.54806949, 0.64692685,
        0.64692685, 0.64608615, 0.64467379, 0.74948077, 0.74809485,
        0.73089217, 0.71090201, 0.98268866, 0.975978  , 0.91521861,
        0.83719535]),
 'mean_train_score': array([0.54836959, 0.54836959, 0.54836959, 0.5
4836959, 0.64913536,
        0.64913536, 0.64848611, 0.64689891, 0.75191188, 0.75053568,
        0.73207984, 0.71285497, 0.98371526, 0.97636216, 0.91697587,
        0.83875527]),
 'std_train_score': array([0.00024631, 0.00024631, 0.00024631, 0.00
024631, 0.00206585,
        0.00206585, 0.0022348 , 0.00219182, 0.0018117 , 0.00190631,
        0.00309776, 0.00188788, 0.00195475, 0.00134902, 0.00288164,
        0.00600812])}
```

2.3 Converting GridSearchCV results to dataframe and then group by hyperparameters to return train_score and CV_score dfs

In [256]:

```
clf_3_df = pd.DataFrame(clf_3.cv_results_)
clf_3_df
```

Out[256]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param
0	**1.359230**	**0.078053**	**0.038808**	**0.003780**	**1**	**5**
1	**1.368485**	**0.105511**	**0.049972**	**0.008442**	**1**	**10**
2	**1.358582**	**0.086320**	**0.038878**	**0.003312**	**1**	**100**
3	**1.414095**	**0.121557**	**0.034291**	**0.000916**	**1**	**500**
4	**5.835186**	**0.057934**	**0.037160**	**0.001133**	**5**	**5**
5	**5.438443**	**0.288691**	**0.037264**	**0.000784**	**5**	**10**
6	**5.200095**	**0.071281**	**0.035773**	**0.000266**	**5**	**100**
7	**5.049640**	**0.002812**	**0.036148**	**0.000460**	**5**	**500**
8	**14.460344**	**0.048718**	**0.038701**	**0.000205**	**10**	**5**
9	**14.552308**	**0.161950**	**0.038035**	**0.000383**	**10**	**10**
10	**13.269661**	**0.215673**	**0.039486**	**0.002435**	**10**	**100**
11	**11.098234**	**0.180569**	**0.037725**	**0.001050**	**10**	**500**
12	**86.837027**	**1.500292**	**0.044884**	**0.001143**	**50**	**5**
13	**82.373743**	**1.146601**	**0.043030**	**0.000372**	**50**	**10**
14	**62.186710**	**2.085315**	**0.039368**	**0.005926**	**50**	**100**
15	**37.504628**	**0.619016**	**0.033444**	**0.004375**	**50**	**500**

In [257]:

```
scores_nonzero_ft = clf_3_df.groupby(['param_max_depth','param_min_samples_split']).max().unstack()[['mean_train_score','mean_test_score']]
scores_nonzero_ft
```

Out[257]:

	mean_train_score				mean_test_score	
param_min_samples_split	5	10	100	500	5	10
param_max_depth						
1	**0.548370**	**0.548370**	**0.548370**	**0.548370**	**0.547310**	**0.547**
5	**0.649135**	**0.649135**	**0.648486**	**0.646899**	**0.629813**	**0.629**
10	**0.751912**	**0.750536**	**0.732080**	**0.712855**	**0.647153**	**0.647**
50	**0.983715**	**0.976362**	**0.916976**	**0.838755**	**0.572096**	**0.575**

In [258]:

```
train_score_ft = scores_nonzero_ft['mean_train_score']
train_score_ft
```

Out[258]:

param_min_samples_split	5	10	100	500
param_max_depth				
1	**0.548370**	**0.548370**	**0.548370**	**0.548370**
5	**0.649135**	**0.649135**	**0.648486**	**0.646899**
10	**0.751912**	**0.750536**	**0.732080**	**0.712855**
50	**0.983715**	**0.976362**	**0.916976**	**0.838755**

In [259]:

```
test_score_ft = scores_nonzero_ft['mean_test_score']
test_score_ft
```

Out[259]:

param_min_samples_split	5	10	100	500
param_max_depth				
1	**0.547310**	**0.547310**	**0.547310**	**0.547310**
5	**0.629813**	**0.629797**	**0.629727**	**0.630173**
10	**0.647153**	**0.647536**	**0.649006**	**0.657031**
50	**0.572096**	**0.575603**	**0.599243**	**0.624967**

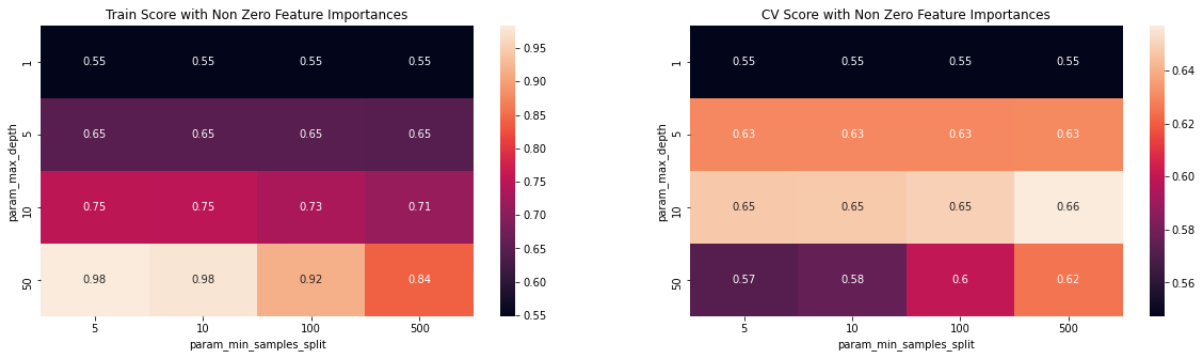
2.4 Plotting Heatmap of Trainscore and cv scores

In [260]:

```
fig,ax = plt.subplots(ncols=2,figsize=(20,5))
sns.heatmap(train_score_ft,annot=True,ax=ax[0])
sns.heatmap(test_score_ft,annot=True,ax=ax[1])
ax[0].set_title("Train Score with Non Zero Feature Importances")
ax[1].set_title("CV Score with Non Zero Feature Importances")
```

Out[260]:

```
Text(0.5, 1.0, 'CV Score with Non Zero Feature Importances')
```



2.5 Finding Best Hyperparameter

In [261]:

```
best_model_dt_nonzero_features = clf_3.best_estimator_
best_model_dt_nonzero_features.fit(X_Train_new,y_train)
```

Out[261]:

```
DecisionTreeClassifier(class_weight='balanced', max_depth=10,
                       min_samples_split=500)
```

In [262]:

```
best_model_dt_nonzero_features_max_depth = clf_3.best_params_['max_depth']
best_model_dt_nonzero_features_min_sample_split = clf_3.best_params_['min_sample
s_split']
```

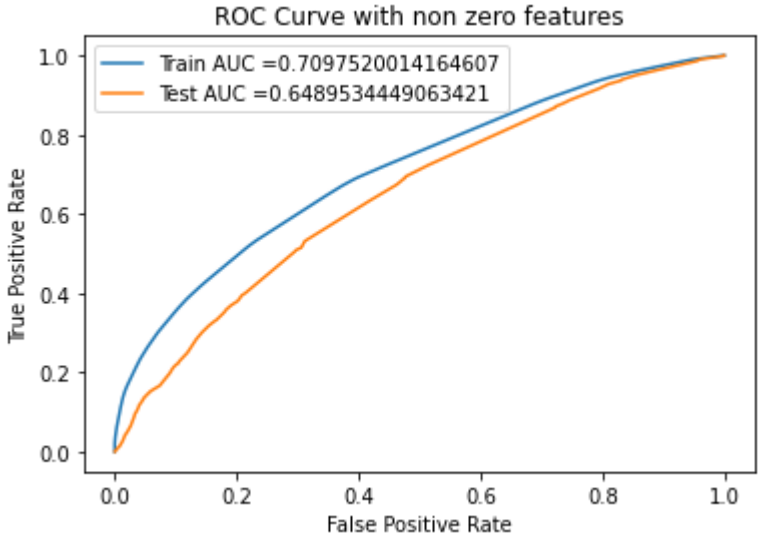
2.6 Plotting ROC

In [263]:

```
y_pred_train_nonzero_features = best_model_dt_nonzero_features.predict_proba(X_T
rain_new)
y_pred_test_nonzero_features = best_model_dt_nonzero_features.predict_proba(X_Te
st_new)
```

In [264]:

```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_pred_train_nonzero_features[:,1])
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_pred_test_nonzero_features[:,1])

train_auc_dt_nonzero_features = auc(train_fpr, train_tpr)
test_auc_dt_nonzero_features = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(train_auc_dt_nonzero_features))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(test_auc_dt_nonzero_features))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve with non zero features")
plt.legend()
plt.show()
```



2.7 Plotting Confusion matrix

In [265]:

```
y_pred_nonzero_features = best_model_dt_nonzero_features.predict(X_Test_new)
cm_nonzero_features = confusion_matrix(y_test,y_pred_nonzero_features)
```

In [266]:

```
cm_nonzero_features
```

Out[266]:

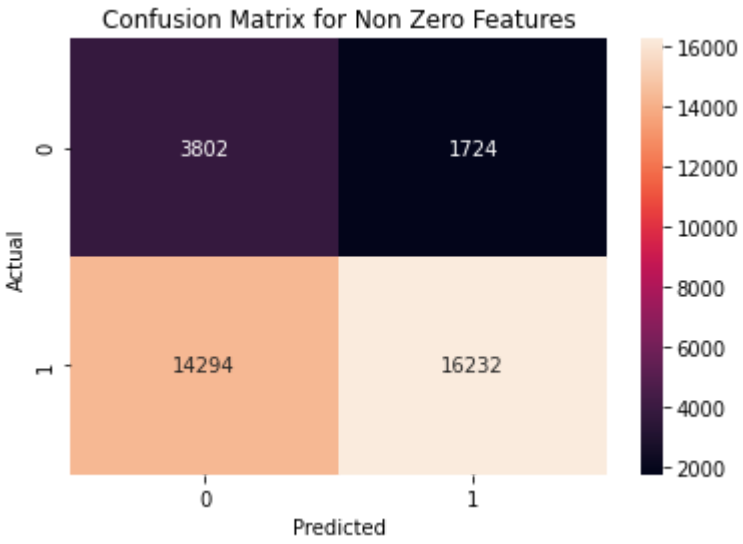
```
array([[ 3802,  1724],
       [14294, 16232]])
```

In [267]:

```
sns.heatmap(cm_nonzero_features,annot=True,fmt="g")
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("Confusion Matrix for Non Zero Features")
```

Out[267]:

```
Text(0.5, 1.0, 'Confusion Matrix for Non Zero Features')
```



2.8 Getting False Positive Points

In [268]:

```
df_nonzero_features = pd.DataFrame({'Actual':y_test.values,
                                    'Predicted':y_pred_nonzero_features})
```

In [269]:

```
fp_nonzero_features = df_nonzero_features.loc[(df_nonzero_features.loc[:,'Actua
l']==0) & (df_nonzero_features.loc[:,'Predicted']==1)]
```

In [270]:

```
fp_nonzero_features
```

Out[270]:

	Actual	Predicted
1	**0**	**1**
15	**0**	**1**
43	**0**	**1**
92	**0**	**1**
113	**0**	**1**
...	...	...
35936	**0**	**1**
35979	**0**	**1**
36014	**0**	**1**
36025	**0**	**1**
36038	**0**	**1**

1724 rows × 2 columns

In [271]:

```
false_positive_points_test_data_nonzero_features = x_test.iloc[fp_nonzero_features.index,:]
```

In [272]:

```
false_positive_points_test_data_nonzero_features.head()
```

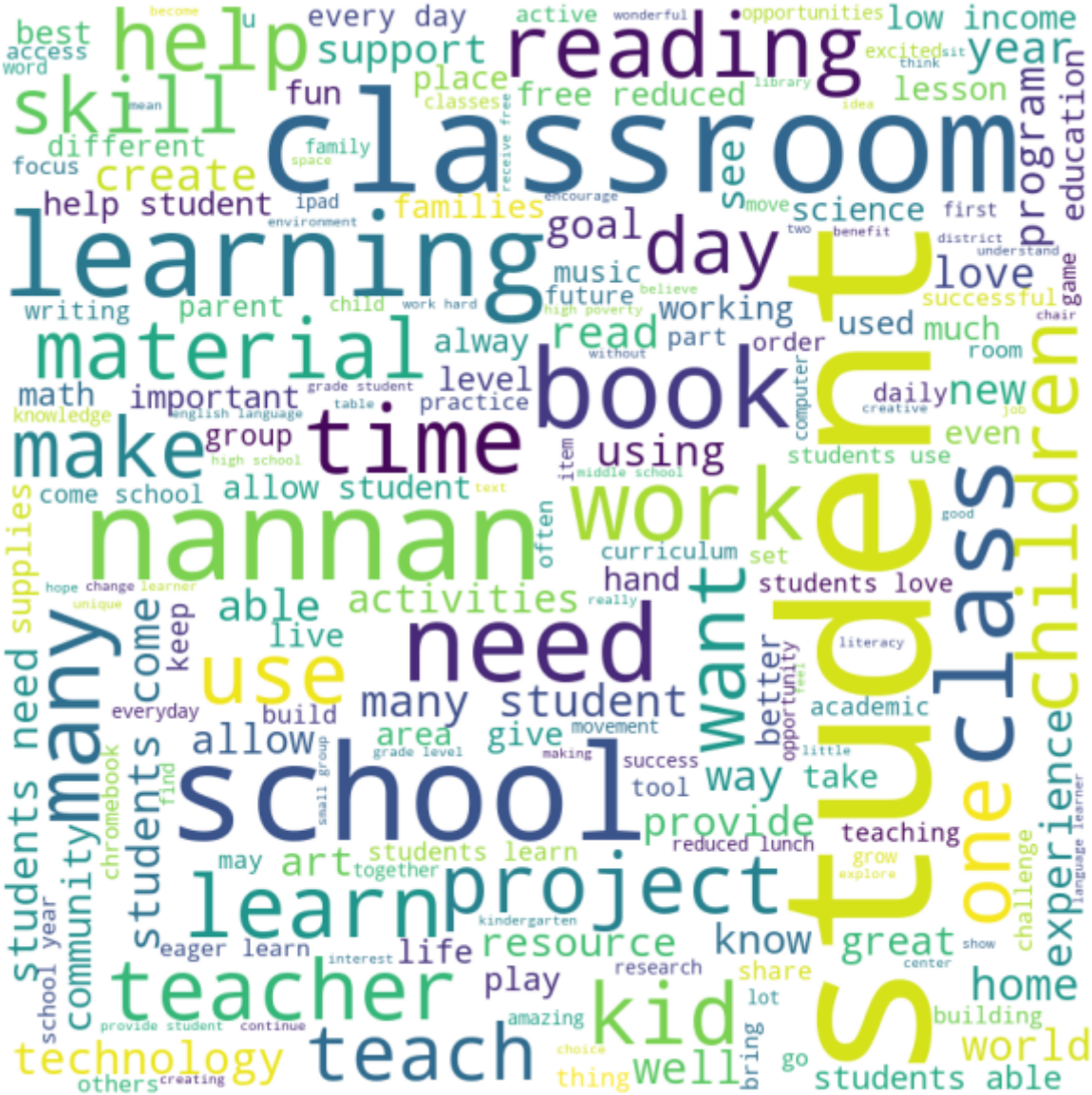
Out[272]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_po
20492	**mi**	**mrs**	**grades_3_5**	**15**
6405	**fl**	**ms**	**grades_3_5**	**47**
13864	**tx**	**mrs**	**grades_prek_2**	**25**
32821	**mi**	**ms**	**grades_prek_2**	**9**
88186	**il**	**mrs**	**grades_prek_2**	**5**

2.8.1 Printing word cloud

In [273]:

```
fp_essay = false_positive_points_test_data_nonzero_features.loc[:,'essay']
comment_words = ''
stopwords = set(STOPWORDS)
for val in fp_essay:

    # typecaste each val to string
    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)

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

plt.show()
```



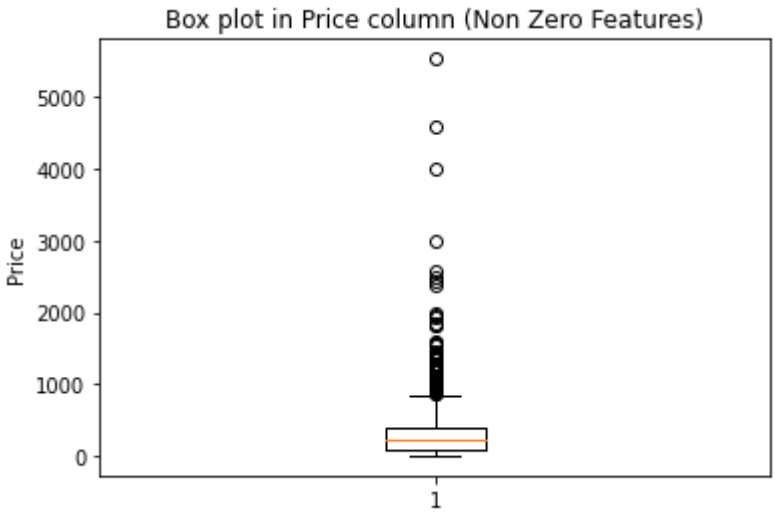
2.8.2 Plotting box plot

In [274]:

```
plt.boxplot(false_positive_points_test_data_nonzero_features.loc[:,'price'])
plt.ylabel('Price')
plt.title('Box plot in Price column (Non Zero Features)')
```

Out[274]:

```
Text(0.5, 1.0, 'Box plot in Price column (Non Zero Features)')
```



2.8.3 Plotting pdf on teacher_number_of_previously_posted_projects

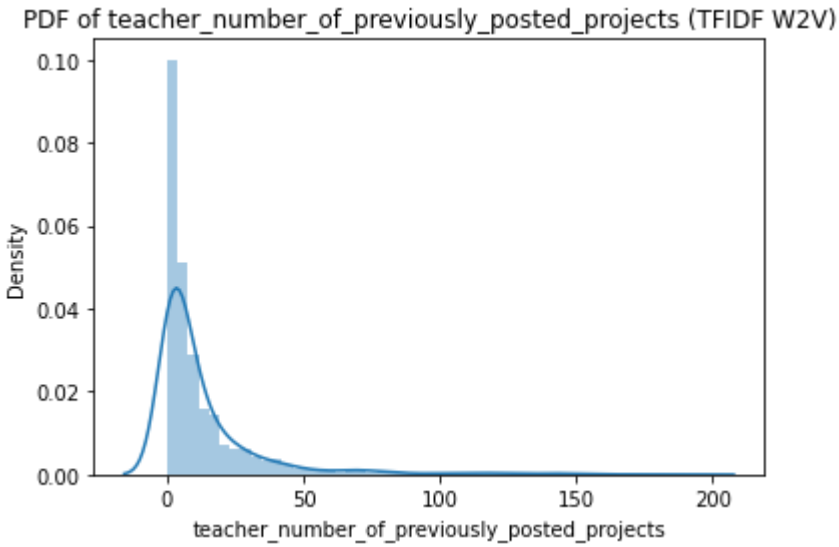
In [275]:

```
sns.distplot(false_positive_points_test_data_nonzero_features.loc[:,'teacher_number_of_previously_posted_projects'])
plt.title("PDF of teacher_number_of_previously_posted_projects (TFIDF W2V)")
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[275]:

```
Text(0.5, 1.0, 'PDF of teacher_number_of_previously_posted_projects (TFIDF W2V)')
```



Summary

In [277]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer","Model","Max Depth","Min Samples Split","Train AUC","Test AUC"]

x.add_row(["TF-IDF","Decision Tree", best_model_dt_max_depth,best_model_dt_min_sample_split,train_auc_dt,test_auc_dt])
x.add_row(["TF-IDF W2V","Decision Tree", best_model_dt_tfidf_w2v_max_depth,best_model_dt_tfidf_w2v_min_sample_split,train_auc_dt_tfidf_w2v,test_auc_dt_tfidf_w2v])
x.add_row(["TF-IDF Non Zero Features","Decision Tree", best_model_dt_nonzero_features_max_depth,best_model_dt_nonzero_features_min_sample_split,train_auc_dt_nonzero_features,test_auc_dt_nonzero_features])


print(x)
```

```
+--------------------------+---------------+-----------+-----------
--------+--------------------+--------------------+
|        Vectorizer        |     Model     | Max Depth | Min Sample
s Split |     Train AUC      |      Test AUC      |
+--------------------------+---------------+-----------+-----------
--------+--------------------+--------------------+
|          TF-IDF          | Decision Tree |     10    |        500
| 0.7097520014164607 | 0.6489534449063421 |
|        TF-IDF W2V        | Decision Tree |     5     |        500
| 0.6535535536719245 | 0.6211898680130493 |
| TF-IDF Non Zero Features | Decision Tree |     10    |        500
| 0.7097520014164607 | 0.6489534449063421 |
+--------------------------+---------------+-----------+-----------
--------+--------------------+--------------------+
```

In [ ]: