K means Agglomerative DBSCAN clustering algorithms on Donors Choose dataset

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website. Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve: How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible How to increase the consistency of project vetting across different volunteers to improve the experience for teachers How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
```

```
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
import chart_studio.plotly as py
from scipy.sparse import hstack
import chart_studio.plotly as py


from collections import Counter
```

1. LOAD AND PROCESS DATA

1.1 Reading Data

In [2]:
```
data=pd.read_csv("train_data.csv",nrows=50000)
resource_data=pd.read_csv("resources.csv")
data.columns
```

Out[2]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_subject_categories', 'project_subject_subcategories',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved'],
      dtype='object')
```

In [3]:
```
price_data=resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
```

In [4]:
```
project_data=pd.merge(data, price_data, on='id', how='left')
```

In [5]:
```
project_data.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_subject_categories', 'project_subject_subcategories',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity'],
      dtype='object')
```

1.2 process Project Essay

In [6]:
```
project_data.head(3)
```

Out[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Histo
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Grades 6-8	

In [7]:
```
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [8]:
```
import re
```

```
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [9]:

```
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'furthe
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mo
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "were
            'won', "won't", 'wouldn', "wouldn't"]
```

In [10]:

```
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
```

```
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
project_data['cleaned_essay']=preprocessed_essays
```

```
100%|██████████| 50000/50000 [00:27<00:00, 1818.44it/s]
```

1.2 process Project Title

In [11]:
```
# https://stackoverflow.com/a/47091490/4084039
from tqdm import tqdm
preprocessed_title = []
# tqdm is for printing the status bar
for sentance in tqdm(data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_title.append(sent.lower().strip())
project_data['cleaned_project_title']=preprocessed_title
```

```
100%|██████████| 50000/50000 [00:01<00:00, 37055.84it/s]
```

1.3 teacher_prefix

In [12]:
```
temp1=data.teacher_prefix.apply(lambda x: str(x).replace('.', ''))
project_data['teacher_prefix']=temp1
project_data['teacher_prefix'].value_counts()
```

Out[12]:
```
Mrs        26140
Ms         17936
Mr          4859
Teacher     1061
Dr             2
nan            2
Name: teacher_prefix, dtype: int64
```

1.4 project grade

In [13]:
```
project_data.project_grade_category.value_counts()
```

Out[13]:
```
Grades PreK-2    20316
Grades 3-5       16968
Grades 6-8        7750
Grades 9-12       4966
Name: project_grade_category, dtype: int64
```

In [14]:
```
grade_list=[]
for i in project_data['project_grade_category'].values:
    i=i.replace(' ','_')
    i=i.replace('-','_')
    grade_list.append(i.strip())


project_data['project_grade_category']=grade_list
```

In [15]:
```
project_data['project_grade_category'].value_counts()
```

Out[15]:
```
Grades_PreK_2    20316
Grades_3_5       16968
Grades_6_8        7750
Grades_9_12       4966
Name: project_grade_category, dtype: int64
```

1.5 project_subject_categories

In [16]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", '
```

```
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

1.6 project_subject_subcategories

In [17]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
```

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

1.7 counting words in title

In [18]:
```
#https://stackoverflow.com/questions/49984905/count-number-of-words-per-row
project_data['totalwords_title'] = project_data['cleaned_project_title'].str.split().str.len()
```

1.8 number of words in the essay

In [19]:
```
project_data['totalwords_essay'] = project_data['cleaned_essay'].str.split().str.len()
```

1.9 sentiment score's of each of the essay

In [20]:
```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
neg=[]
compound=[]
pos=[]
neu=[]
for sent in (project_data['cleaned_essay'].values):
    score = analyser.polarity_scores(sent)
    neg.append(score.get('neg'))
    neu.append(score.get('neu'))
    pos.append(score.get('pos'))
    compound.append(score.get('compound'))
project_data['neg']=neg
project_data['neu']=neu
project_data['pos']=pos
project_data['compound']=compound
```

1.10 droping unnecesarry columns

In [21]:
```
project_data.drop(['project_title'], axis=1, inplace=True)
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [22]:
```
project_data.head(3)
```

Out[22]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs	IN	2016-12-05 13:43:57	Grades_PreK_2	op
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr	FL	2016-10-25 09:22:10	Grades_6_8	My stud
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms	AZ	2016-08-31 12:03:56	Grades_6_8	My gua

3 rows × 23 columns

1.11 Making dependant(label) and independant variables

In [23]:
```
y = project_data['project_is_approved'].values

project_data.head(1)
x=project_data
x.head(3)
```

Out[23]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs	IN	2016-12-05 13:43:57	Grades_PreK_2	opp
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr	FL	2016-10-25 09:22:10	Grades_6_8	My stud
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms	AZ	2016-08-31 12:03:56	Grades_6_8	My gua

3 rows × 23 columns

1.12 Traing and Test split

In [24]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.33, stratify=y,random_state=42)

#X_train, X_cv, Y_train, Y_cv = train_test_split(X_train,Y_train, test_size=0.33, stratify=Y_train,random_state=42)
```

2.Text Vectorization and encoding catagories,normalization numerical features

2.1 converting the essay to vectors using BOW

In [25]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['cleaned_essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['cleaned_essay'].values)
#X_cv_essay_bow = vectorizer.transform(X_cv['cleaned_essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['cleaned_essay'].values)
```

```
print("After vectorizations")
print(X_train_essay_bow.shape, Y_train.shape)
print(X_test_essay_bow.shape, Y_test.shape)
#print(X_cv_essay_bow.shape, Y_cv.shape)
print("="*100)
```

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

2.2 converting the title to vectors using BOW

In [26]:
```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['cleaned_project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['cleaned_project_title'].values)
#X_cv_title_bow = vectorizer.transform(X_cv['cleaned_project_title'].values)
X_test_title_bow = vectorizer.transform(X_test['cleaned_project_title'].values)

print("After vectorizations")
print(X_train_title_bow.shape, Y_train.shape)
#print(X_cv_title_bow.shape, Y_cv.shape)
print(X_test_title_bow.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 2902) (33500,)
(16500, 2902) (16500,)
====================================================================================================
```

2.3 converting the title to vectors using TFIDF

In [27]:
```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['cleaned_project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['cleaned_project_title'].values)
#X_cv_title_tfidf = vectorizer.transform(X_cv['cleaned_project_title'].values)
X_test_title_tfidf = vectorizer.transform(X_test['cleaned_project_title'].values)
```

```
print("After vectorizations")
print(X_train_title_tfidf.shape, Y_train.shape)
#print(X_cv_title_tfidf.shape, Y_cv.shape)
print(X_test_title_tfidf.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1629) (33500,)
(16500, 1629) (16500,)
====================================================================================================
```

2.4 converting the essay to vectors using TFIDF

In [28]:

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['cleaned_essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['cleaned_essay'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['cleaned_essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['cleaned_essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, Y_train.shape)
#print(X_cv_essay_tfidf.shape, Y_cv.shape)
print(X_test_essay_tfidf.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 10434) (33500,)
(16500, 10434) (16500,)
====================================================================================================
```

2.5 load glove mode

In [29]:

```
# 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!
'''
# ==============================
```

```
754it [00:00, 7538.50it/s]
Loading Glove Model
1917495it [03:57, 8067.29it/s]
Done. 1917495  words loaded!
```

Out[29]:
```
'Output:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n'
```

In [30]:
```
words = []
for i in X_train['cleaned_essay'].values:
    words.extend(i.split(' '))

for i in X_train['cleaned_project_title'].values:
    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-variabl

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

```
all the words in the coupus 5220928
the unique words in the coupus 36812
The number of words that are present in both glove vectors and our coupus 34156 ( 92.785 %)
word 2 vec length 34156
```

In [31]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variabl
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

2.6 Avg w2v on essay

In [32]:
```
Text_avg_w2v_train_essay= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    Text_avg_w2v_train_essay.append(vector)

print(len(Text_avg_w2v_train_essay))
print(len(Text_avg_w2v_train_essay[0]))
```

```
100%|██████████| 33500/33500 [00:11<00:00, 3031.38it/s]
33500
300
```

In [33]:
```
""" Text_avg_w2v_cv_essay= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    Text_avg_w2v_cv_essay.append(vector)

print(len(Text_avg_w2v_cv_essay))
print(len(Text_avg_w2v_cv_essay[0])) """
```

Out[33]:
```
" Text_avg_w2v_cv_essay= []; # the avg-w2v for each sentence/review is stored in this list\nfor sentence in tqdm(X_cv
['cleaned_essay'].values): # for each review/sentence\n    vector = np.zeros(300) # as word vectors are of zero lengt
h\n    cnt_words =0; # num of words with a valid vector in the sentence/review\n    for word in sentence.split(): # f
or each word in a review/sentence\n        if word in glove_words:\n            vector += model[word]\n            cn
t_words += 1\n    if cnt_words != 0:\n        vector /= cnt_words\n    Text_avg_w2v_cv_essay.append(vector)\n\nprint
(len(Text_avg_w2v_cv_essay))\nprint(len(Text_avg_w2v_cv_essay[0])) "
```

In [34]:
```
Text_avg_w2v_test_essay= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    Text_avg_w2v_test_essay.append(vector)

print(len(Text_avg_w2v_test_essay))
print(len(Text_avg_w2v_test_essay[0]))
```

```
100%|██████████| 16500/16500 [00:05<00:00, 3050.14it/s]
16500
300
```

In [ ]:

2.7 Avg w2v on title

In [35]:
```
Text_avg_w2v_train_title= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['cleaned_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    Text_avg_w2v_train_title.append(vector)

print(len(Text_avg_w2v_train_title))
print(len(Text_avg_w2v_train_title[0]))
```

```
100%|██████████| 33500/33500 [00:00<00:00, 65437.90it/s]
33500
300
```

In [36]:
```
"""Text_avg_w2v_cv_title= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['cleaned_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    Text_avg_w2v_cv_title.append(vector)

print(len(Text_avg_w2v_cv_title))
print(len(Text_avg_w2v_cv_title[0])) """
```

Out[36]:
```
"Text_avg_w2v_cv_title= []; # the avg-w2v for each sentence/review is stored in this list\nfor sentence in tqdm(X_cv
['cleaned_project_title'].values): # for each review/sentence\n    vector = np.zeros(300) # as word vectors are of ze
ro length\n    cnt_words =0; # num of words with a valid vector in the sentence/review\n    for word in sentence.spli
```

```
t(): # for each word in a review/sentence\n        if word in glove_words:\n            vector += model[word]\n
cnt_words += 1\n    if cnt_words != 0:\n        vector /= cnt_words\n    Text_avg_w2v_cv_title.append(vector)\n\nprin
t(len(Text_avg_w2v_cv_title))\nprint(len(Text_avg_w2v_cv_title[0])) "
```

In [37]:

```
Text_avg_w2v_test_title= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['cleaned_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    Text_avg_w2v_test_title.append(vector)

print(len(Text_avg_w2v_test_title))
print(len(Text_avg_w2v_test_title[0]))
```

```
100%|██████████| 16500/16500 [00:00<00:00, 62936.13it/s]
16500
300
```

2.4 TFIDF weighted W2V on essay

In [38]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['cleaned_essay'].values)
# 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 [39]:

```
Text_tfidf_w2v_train_essay= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['cleaned_essay'].values): # 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(sentence
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each
```

```
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_train_essay.append(vector)

print(len(Text_tfidf_w2v_train_essay))
print(len(Text_tfidf_w2v_train_essay[0]))
```

```
100%|██████████| 33500/33500 [01:24<00:00, 396.21it/s]
33500
300
```

In [40]:

```
"""Text_tfidf_w2v_cv_essay= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['cleaned_essay'].values): # 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(sentence
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_cv_essay.append(vector)

print(len(Text_tfidf_w2v_cv_essay))
print(len(Text_tfidf_w2v_cv_essay[0]))"""
```

Out[40]:

```
"Text_tfidf_w2v_cv_essay= []; # the avg-w2v for each sentence/review is stored in this list\nfor sentence in tqdm(X_c
v['cleaned_essay'].values): # for each review/sentence\n    vector = np.zeros(300) # as word vectors are of zero leng
th\n    tf_idf_weight =0; # num of words with a valid vector in the sentence/review\n    for word in sentence.split
(): # for each word in a review/sentence\n        if (word in glove_words) and (word in tfidf_words):\n            ve
c = model[word] # getting the vector for each word\n            # here we are multiplying idf value(dictionary[word])
and the tf value((sentence.count(word)/len(sentence.split())))\n            tf_idf = dictionary[word]*(sentence.count
(word)/len(sentence.split())) # getting the tfidf value for each word\n            vector += (vec * tf_idf) # calcula
ting tfidf weighted w2v\n            tf_idf_weight += tf_idf\n    if tf_idf_weight != 0:\n        vector /= tf_idf_we
ight\n    Text_tfidf_w2v_cv_essay.append(vector)\n\nprint(len(Text_tfidf_w2v_cv_essay))\nprint(len(Text_tfidf_w2v_cv_
essay[0]))"
```

In [41]:
```
Text_tfidf_w2v_test_essay= [];
for sentence in tqdm(X_test['cleaned_essay'].values): # 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(sentence
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_test_essay.append(vector)

print(len(Text_tfidf_w2v_test_essay))
print(len(Text_tfidf_w2v_test_essay[0]))
```

```
100%|██████████| 16500/16500 [00:40<00:00, 403.02it/s]
16500
300
```

2.5 TFIDF weighted W2V on title

In [42]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['cleaned_project_title'].values)
# 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 [43]:
```
Text_tfidf_w2v_train_title= [];
for sentence in tqdm(X_train['cleaned_project_title'].values): # 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(sentence
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
```

```
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_train_title.append(vector)

print(len(Text_tfidf_w2v_train_title))
print(len(Text_tfidf_w2v_train_title[0]))
```

```
100%|██████████| 33500/33500 [00:01<00:00, 28519.60it/s]
33500
300
```

In [44]:

```
"""Text_tfidf_w2v_cv_title= [];
for sentence in tqdm(X_cv['cleaned_project_title'].values): # 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(sentence
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_cv_title.append(vector)

print(len(Text_tfidf_w2v_cv_title))
print(len(Text_tfidf_w2v_cv_title[0])) """
```

Out[44]:

```
"Text_tfidf_w2v_cv_title= []; \nfor sentence in tqdm(X_cv['cleaned_project_title'].values): # for each review/sentenc
e\n    vector = np.zeros(300) # as word vectors are of zero length\n    tf_idf_weight =0; # num of words with a valid
vector in the sentence/review\n    for word in sentence.split(): # for each word in a review/sentence\n        if (wo
rd in glove_words) and (word in tfidf_words):\n            vec = model[word] # getting the vector for each word\n
# here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
\n            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for ea
ch word\n            vector += (vec * tf_idf) # calculating tfidf weighted w2v\n            tf_idf_weight += tf_idf\n
if tf_idf_weight != 0:\n        vector /= tf_idf_weight\n    Text_tfidf_w2v_cv_title.append(vector)\n\nprint(len(Text
_tfidf_w2v_cv_title))\nprint(len(Text_tfidf_w2v_cv_title[0])) "
```

In [45]:

```
Text_tfidf_w2v_test_title= [];
for sentence in tqdm(X_test['cleaned_project_title'].values): # 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(sentence
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_test_title.append(vector)

print(len(Text_tfidf_w2v_test_title))
print(len(Text_tfidf_w2v_test_title[0]))
```

```
100%|██████████| 16500/16500 [00:00<00:00, 29576.28it/s]
16500
300
```

In [46]:
```
X_train.columns
```

Out[46]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity', 'essay', 'cleaned_essay', 'cleaned_project_title',
       'clean_categories', 'clean_subcategories', 'totalwords_title',
       'totalwords_essay', 'neg', 'neu', 'pos', 'compound'],
      dtype='object')
```

2.6 Categories with response coding

In [47]:
```
def Responsetable(table, col) :
    cat = table[col].unique()
    alpha=1
    freq_Pos = []
    for i in cat :
        freq_Pos.append(len(table.loc[(table[col] == i) & (table['project_is_approved'] == 1)]))
```

```
    freq_Neg = []
    for i in cat :
        freq_Neg.append(len(table.loc[(table[col] == i) & (table['project_is_approved'] == 0)]))


    encoded_Pos = []
    for i in range(len(cat)) :
        encoded_Pos.append(((freq_Pos[i]+alpha)/(freq_Pos[i] + freq_Neg[i]+alpha)))

    encoded_Neg = []
    encoded_Neg[:] = [1 - x for x in encoded_Pos]

    encoded_Pos_val = dict(zip(cat, encoded_Pos))
    encoded_Neg_val = dict(zip(cat, encoded_Neg))

    return encoded_Pos_val, encoded_Neg_val
```

In [48]:

```
def Responsecode(table) :
    pos_cleancat, neg_cleancat = Responsetable(table,'clean_categories')
    pos_cleansubcat, neg_cleansubcat = Responsetable(table,'clean_subcategories')
    pos_schoolstate, neg_schoolstate = Responsetable(table, 'school_state')
    pos_teacherprefix, neg_teacherprefix = Responsetable(table, 'teacher_prefix')
    pos_projgradecat, neg_projgradecat = Responsetable(table, 'project_grade_category')

    df = pd.DataFrame()
    df['clean_cat_pos'] = table['clean_categories'].map(pos_cleancat)
    df['clean_cat_neg'] = table['clean_categories'].map(neg_cleancat)
    df['clean_subcat_pos'] = table['clean_subcategories'].map(pos_cleansubcat)
    df['clean_subcat_neg'] = table['clean_subcategories'].map(neg_cleansubcat)
    df['school_state_pos'] = table['school_state'].map(pos_schoolstate)
    df['school_state_neg'] = table['school_state'].map(neg_schoolstate)
    df['teacher_prefix_pos'] = table['teacher_prefix'].map(pos_teacherprefix)
    df['teacher_prefix_neg'] = table['teacher_prefix'].map(neg_teacherprefix)
    df['proj_grade_cat_pos'] = table['project_grade_category'].map(pos_projgradecat)
    df['proj_grade_cat_neg'] = table['project_grade_category'].map(neg_projgradecat)

    return df
```

In [49]:

```
newTrain = Responsecode(X_train)
newTest = Responsecode(X_test)
#newCv=Responsecode(X_cv)
```

In [50]:
```
def mergeEncoding(table, p, n) :
    lstPos = table[p].values.tolist()
    lstNeg = table[n].values.tolist()
    frame = pd.DataFrame(list(zip(lstNeg, lstPos)))

    return frame
```

2.7 response code of clean_categories

In [51]:
```
X_train_clean_cat_resposecode = mergeEncoding(newTrain, 'clean_cat_pos', 'clean_cat_neg')
X_test_clean_cat_resposecode = mergeEncoding(newTest, 'clean_cat_pos', 'clean_cat_neg')
#X_cv_clean_cat_resposecode=mergeEncoding(newCv, 'clean_cat_pos', 'clean_cat_neg')
print(X_train_clean_cat_resposecode.shape)
```

```
(33500, 2)
```

2.8 response code of clean_sub_categories

In [52]:
```
X_train_clean_subcat_resposecode = mergeEncoding(newTrain, 'clean_subcat_pos', 'clean_subcat_neg')
X_test_clean_subcat_resposecode = mergeEncoding(newTest, 'clean_subcat_pos', 'clean_subcat_neg')
#X_cv_clean_subcat_resposecode = mergeEncoding(newCv, 'clean_subcat_pos', 'clean_subcat_neg')
print(X_train_clean_subcat_resposecode.shape)
print(X_test_clean_subcat_resposecode.shape)
#print(X_cv_clean_subcat_resposecode.shape)
```

```
(33500, 2)
(16500, 2)
```

2.9 response code of project grade

In [53]:
```
X_train_grade_resposecode = mergeEncoding(newTrain, 'proj_grade_cat_pos', 'proj_grade_cat_neg')
X_test_grade_resposecode = mergeEncoding(newTest, 'proj_grade_cat_pos', 'proj_grade_cat_neg')
#X_cv_grade_resposecode = mergeEncoding(newCv, 'proj_grade_cat_pos', 'proj_grade_cat_neg')
print(X_train_grade_resposecode.shape)
print(X_test_grade_resposecode.shape)
#print(X_cv_grade_resposecode.shape)
```

```
(33500, 2)
(16500, 2)
```

2.10 response code of school state

In [54]:
```
X_train_state_resposecode = mergeEncoding(newTrain, 'school_state_pos', 'school_state_neg')
X_test_state_resposecode = mergeEncoding(newTest, 'school_state_pos', 'school_state_neg')
#X_cv_state_resposecode = mergeEncoding(newCv, 'school_state_pos', 'school_state_neg')
print(X_train_state_resposecode.shape)
print(X_test_state_resposecode.shape)
#print(X_cv_state_resposecode.shape)
```

```
(33500, 2)
(16500, 2)
```

2.11 response code of teacher prefix

In [55]:
```
X_train_teacher_resposecode = mergeEncoding(newTrain, 'teacher_prefix_pos', 'teacher_prefix_neg')
X_test_teacher_resposecode = mergeEncoding(newTest, 'teacher_prefix_pos', 'teacher_prefix_neg')
#X_cv_teacher_resposecode = mergeEncoding(newCv, 'teacher_prefix_pos', 'teacher_prefix_neg')
print(X_train_teacher_resposecode.shape)
print(X_test_teacher_resposecode.shape)
#print(X_cv_teacher_resposecode.shape)
```

```
(33500, 2)
(16500, 2)
```

2.12 Normalizing the numerical features: Price

In [56]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

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

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

#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, Y_train.shape)
#print(X_cv_price_norm.shape, Y_cv.shape)
```

```
print(X_test_price_norm.shape, Y_test.shape)
print("="*100)
```

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

2.13 Normalizing the numerical features:teacher_number_of_previously_posted_projects

In [57]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_TPPP_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
#X_cv_TPPP_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_TPPP_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_TPPP_norm.shape, Y_train.shape)
#print(X_cv_TPPP_norm.shape, Y_cv.shape)
print(X_test_TPPP_norm.shape, Y_test.shape)
print("="*100)
```

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

2.14 Normalizing the numerical features: quantity

In [58]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['quantity'].values.reshape(-1,1))

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
#X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
```

```
print(X_train_quantity_norm.shape, Y_train.shape)
#print(X_cv_quantity_norm.shape, Y_cv.shape)
print(X_test_quantity_norm.shape, Y_test.shape)
print("="*100)
```

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

2.15 Normalizing the numerical features: totalwords_title

In [59]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['totalwords_title'].values.reshape(-1,1))

X_train_totalwords_title_norm = normalizer.transform(X_train['totalwords_title'].values.reshape(-1,1))

X_test_totalwords_title_norm = normalizer.transform(X_test['totalwords_title'].values.reshape(-1,1))
#X_cv_totalwords_title_norm = normalizer.transform(X_cv['totalwords_title'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_totalwords_title_norm.shape, Y_train.shape)
#print(X_cv_totalwords_title_norm.shape, Y_cv.shape)
print(X_test_totalwords_title_norm.shape, Y_test.shape)
print("="*100)
```

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

2.17 Normalizing the numerical features: totalwords_essay

In [60]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['totalwords_essay'].values.reshape(-1,1))

X_train_totalwords_essay_norm = normalizer.transform(X_train['totalwords_essay'].values.reshape(-1,1))
#X_cv_totalwords_essay_norm = normalizer.transform(X_cv['totalwords_essay'].values.reshape(-1,1))
X_test_totalwords_essay_norm = normalizer.transform(X_test['totalwords_essay'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_totalwords_essay_norm.shape, Y_train.shape)
#print(X_cv_totalwords_essay_norm.shape, Y_cv.shape)
print(X_test_totalwords_essay_norm.shape, Y_test.shape)
print("="*100)
```

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

3. Best Auc found on TFIDF

3.1 TFIDF:Concatinating all the features

In [61]:
```
X_tr_tfidf = hstack((X_train_essay_tfidf,X_train_title_tfidf,X_train_clean_cat_resposecode,X_train_clean_subcat_respo

X_te_tfidf = hstack((X_test_essay_tfidf,X_test_title_tfidf,X_test_clean_cat_resposecode,X_test_clean_subcat_resposeco

print("Final Data matrix")
print(X_tr_tfidf.shape, Y_train.shape)

print(X_te_tfidf.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 12076) (33500,)
(16500, 12076) (16500,)
====================================================================================================
```

3.2 Model with best AUC

In [62]:
```
X_tr_tfidf = hstack((X_train_essay_tfidf,X_train_title_tfidf,X_train_clean_cat_resposecode,X_train_clean_subcat_respo

X_te_tfidf = hstack((X_test_essay_tfidf,X_test_title_tfidf,X_test_clean_cat_resposecode,X_test_clean_subcat_resposeco

print("Final Data matrix")
print(X_tr_tfidf.shape, Y_train.shape)

print(X_te_tfidf.shape, Y_test.shape)
```

```
print("="*100)
```

```
Final Data matrix
(33500, 12076) (33500,)
(16500, 12076) (16500,)
====================================================================================================
```

3.3 Feature selection

In [63]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import mutual_info_classif

select_func = SelectKBest(chi2, k=5000).fit(X_tr_tfidf, Y_train)


X_train_fe_5000 = select_func.transform(X_tr_tfidf)

print("Final Data matrix")
print(X_train_fe_5000.shape)
print("="*100)
```

```
Final Data matrix
(33500, 5000)
====================================================================================================
```

4 Apply kmean

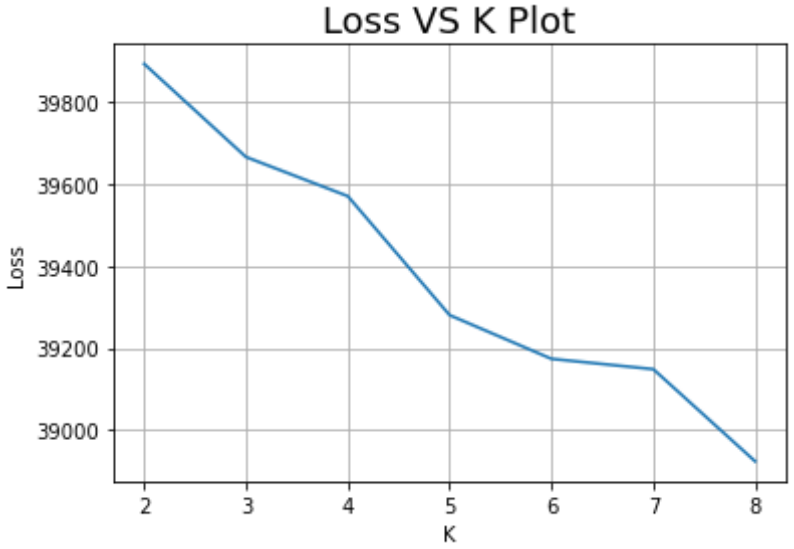
4.1 apply kmeans

In [64]:
```
%%time
from sklearn.cluster import KMeans
```

```
k_values = [2, 3, 4, 5, 6, 7, 8]
loss = []
for i in k_values:
    kmeans = KMeans(n_clusters=i, n_jobs=-1).fit(X_train_fe_5000)
    loss.append(kmeans.inertia_)
```

```
Wall time: 1min 42s
```

In [65]:
```
plt.plot(k_values, loss)
plt.xlabel('K')
plt.ylabel('Loss')
plt.title('Loss VS K Plot',size=18)
plt.grid()
plt.show()
```



observations:

1.we found optimal cluster value is 6

In [66]:
```
optimal_K=6
kmeans = KMeans(n_clusters=optimal_K, n_jobs=-1).fit(X_train_fe_5000)
```

In [67]:
```
kmeans.n_clusters
```

Out[67]: 6

In [68]:
```
kmeans.labels_
```

Out[68]: array([3, 3, 2, ..., 2, 2, 3])

In [69]:
```
print(len(kmeans.labels_))
```

33500

In [70]:
```
clusters_set = {i: np.where(kmeans.labels_ == i)[0] for i in range(kmeans.n_clusters)}
clusters_set
```

Out[70]:
```
{0: array([    9,    27,    33,    50,    53,    77,   142,   145,   159,
            185,   208,   214,   215,   231,   286,   313,   319,   326,
            358,   452,   537,   568,   596,   614,   686,   697,   739,
            741,   750,   756,   774,   923,  1013,  1016,  1020,  1052,
           1147,  1200,  1228,  1231,  1249,  1267,  1360,  1393,  1426,
           1460,  1486,  1539,  1550,  1606,  1607,  1622,  1646,  1683,
           1700,  1873,  1880,  1895,  1918,  1975,  1979,  1995,  2003,
           2054,  2055,  2101,  2143,  2191,  2206,  2218,  2259,  2307,
           2312,  2352,  2361,  2378,  2404,  2506,  2540,  2584,  2602,
           2607,  2617,  2625,  2661,  2666,  2719,  2752,  2754,  2872,
           2892,  2945,  2974,  3021,  3082,  3136,  3139,  3149,  3289,
           3337,  3464,  3480,  3501,  3624,  3686,  3763,  3785,  3805,
           3810,  3930,  3942,  3944,  3993,  3995,  4002,  4048,  4056,
           4117,  4125,  4141,  4162,  4164,  4235,  4243,  4266,  4274,
           4313,  4343,  4411,  4589,  4724,  4731,  4752,  4767,  4805,
           4886,  4934,  4950,  4957,  4979,  4984,  5095,  5102,  5148,
           5165,  5175,  5230,  5231,  5280,  5314,  5382,  5486,  5512,
           5587,  5612,  5642,  5648,  5694,  5755,  5896,  5900,  5954,
           6052,  6077,  6100,  6150,  6168,  6230,  6272,  6289,  6314,
           6322,  6419,  6542,  6547,  6615,  6619,  6623,  6678,  6689,
           6776,  6791,  6865,  6906,  6955,  6969,  7101,  7103,  7116,
           7137,  7154,  7161,  7163,  7212,  7245,  7275,  7352,  7440,
           7575,  7610,  7625,  7655,  7741,  7751,  7766,  7850,  7940,
           7984,  7989,  8001,  8054,  8144,  8178,  8193,  8197,  8208,
           8269,  8317,  8361,  8400,  8421,  8496,  8509,  8557,  8644,
           8678,  8683,  8684,  8704,  8740,  8795,  8797,  8807,  8916,
           9085,  9170,  9301,  9318,  9389,  9442,  9443,  9501,  9541,
           9588,  9629,  9635,  9643,  9652,  9664,  9682,  9699,  9748,
           9870,  9920,  9960,  9968, 10018, 10052, 10063, 10095, 10096,
          10117, 10152, 10167, 10179, 10218, 10221, 10297, 10333, 10372,
```

```
10384, 10434, 10477, 10490, 10563, 10630, 10719, 10730, 10741,
10825, 10983, 11029, 11112, 11139, 11207, 11224, 11301, 11311,
11312, 11358, 11385, 11520, 11636, 11650, 11683, 11739, 11748,
11755, 11765, 11791, 11905, 12036, 12069, 12085, 12090, 12137,
12156, 12162, 12178, 12180, 12211, 12228, 12342, 12349, 12406,
12410, 12411, 12413, 12428, 12463, 12480, 12494, 12501, 12508,
12534, 12599, 12669, 12706, 12718, 12747, 12814, 12842, 12899,
12905, 12914, 12946, 13030, 13094, 13110, 13127, 13130, 13225,
13296, 13375, 13429, 13434, 13552, 13574, 13588, 13703, 13749,
13854, 13855, 13899, 13960, 14038, 14048, 14089, 14097, 14130,
14299, 14395, 14404, 14474, 14488, 14596, 14618, 14668, 14689,
14731, 14767, 14770, 14779, 14785, 14790, 14839, 14877, 14891,
14937, 14984, 14992, 15011, 15021, 15041, 15073, 15144, 15180,
15204, 15205, 15212, 15265, 15295, 15434, 15443, 15482, 15501,
15507, 15563, 15567, 15595, 15653, 15674, 15681, 15686, 15698,
15770, 15879, 15880, 15916, 16001, 16009, 16076, 16128, 16190,
16194, 16196, 16216, 16223, 16230, 16276, 16292, 16293, 16312,
16315, 16342, 16354, 16397, 16408, 16435, 16448, 16453, 16485,
16513, 16542, 16574, 16612, 16655, 16705, 16725, 16734, 16756,
16780, 16860, 16946, 16956, 17044, 17045, 17072, 17094, 17110,
17113, 17136, 17187, 17195, 17196, 17198, 17201, 17211, 17232,
17271, 17273, 17275, 17327, 17371, 17431, 17453, 17502, 17512,
17545, 17551, 17613, 17733, 17748, 17775, 17778, 17804, 17861,
17872, 17922, 17979, 18008, 18134, 18144, 18174, 18196, 18219,
18222, 18302, 18316, 18403, 18445, 18466, 18469, 18496, 18528,
18542, 18626, 18667, 18728, 18729, 18747, 18774, 18779, 18836,
18851, 18861, 18913, 18948, 18976, 19037, 19041, 19070, 19107,
19135, 19162, 19253, 19254, 19277, 19325, 19442, 19529, 19608,
19657, 19675, 19683, 19740, 19796, 19812, 19833, 19886, 19888,
19903, 19920, 19963, 19984, 20017, 20071, 20108, 20115, 20127,
20140, 20160, 20196, 20197, 20199, 20212, 20270, 20318, 20338,
20353, 20369, 20374, 20401, 20430, 20473, 20477, 20492, 20586,
20608, 20614, 20618, 20626, 20687, 20706, 20755, 20759, 20813,
20825, 20900, 20960, 20995, 21006, 21039, 21102, 21105, 21161,
21178, 21239, 21268, 21278, 21284, 21291, 21292, 21318, 21357,
21388, 21398, 21492, 21495, 21571, 21590, 21657, 21724, 21770,
21774, 21845, 21857, 21887, 21893, 21924, 21946, 21948, 21965,
21988, 22089, 22126, 22133, 22154, 22185, 22187, 22219, 22239,
22246, 22276, 22326, 22342, 22371, 22486, 22530, 22561, 22625,
22639, 22660, 22701, 22704, 22706, 22731, 22766, 22849, 22909,
22945, 22962, 23010, 23143, 23250, 23258, 23325, 23358, 23378,
23525, 23559, 23565, 23646, 23663, 23691, 23702, 23797, 23811,
23840, 23866, 23871, 23922, 23928, 23949, 23970, 23982, 24001,
24023, 24056, 24057, 24064, 24127, 24177, 24243, 24248, 24283,
24422, 24470, 24555, 24571, 24617, 24650, 24669, 24773, 24783,
```

```
       24806, 24811, 24923, 24936, 24942, 24983, 24985, 25001, 25037,
       25065, 25112, 25114, 25115, 25117, 25144, 25205, 25218, 25246,
       25247, 25257, 25270, 25275, 25341, 25349, 25366, 25375, 25422,
       25430, 25474, 25500, 25506, 25527, 25540, 25549, 25606, 25690,
       25701, 25723, 25735, 25737, 25863, 25866, 25908, 25924, 25927,
       25931, 25933, 26079, 26083, 26099, 26120, 26164, 26240, 26252,
       26301, 26325, 26366, 26375, 26391, 26409, 26427, 26446, 26475,
       26486, 26525, 26536, 26611, 26655, 26660, 26687, 26742, 26753,
       26766, 26776, 26783, 26801, 26807, 26821, 26826, 26895, 26917,
       26925, 26928, 26944, 26946, 26990, 26996, 27008, 27009, 27078,
       27081, 27136, 27159, 27222, 27241, 27277, 27291, 27299, 27303,
       27315, 27325, 27327, 27348, 27374, 27386, 27396, 27454, 27478,
       27494, 27508, 27578, 27592, 27598, 27622, 27713, 27741, 27812,
       27838, 27849, 27871, 27879, 27882, 27897, 27899, 27901, 27955,
       27983, 28049, 28192, 28220, 28232, 28237, 28340, 28399, 28419,
       28438, 28472, 28476, 28537, 28574, 28585, 28627, 28629, 28641,
       28645, 28684, 28693, 28725, 28744, 28778, 28812, 28941, 28963,
       28974, 28977, 29036, 29047, 29078, 29097, 29098, 29126, 29140,
       29141, 29145, 29254, 29263, 29293, 29310, 29324, 29329, 29334,
       29389, 29404, 29457, 29483, 29507, 29520, 29563, 29566, 29573,
       29601, 29606, 29616, 29634, 29636, 29653, 29840, 29883, 29896,
       29906, 29916, 29958, 30036, 30169, 30240, 30241, 30252, 30276,
       30287, 30309, 30380, 30486, 30487, 30567, 30570, 30574, 30587,
       30588, 30596, 30657, 30662, 30677, 30690, 30706, 30744, 30767,
       30824, 30891, 30893, 30913, 30983, 31023, 31046, 31068, 31105,
       31145, 31158, 31169, 31190, 31220, 31225, 31228, 31296, 31300,
       31315, 31369, 31388, 31444, 31539, 31599, 31612, 31621, 31641,
       31678, 31737, 31742, 31769, 31830, 31876, 31992, 32072, 32080,
       32088, 32120, 32139, 32141, 32160, 32210, 32274, 32336, 32370,
       32405, 32428, 32449, 32530, 32601, 32615, 32707, 32709, 32746,
       32774, 32843, 32857, 32865, 32869, 32888, 32944, 32956, 32960,
       33008, 33069, 33078, 33105, 33107, 33160, 33184, 33192, 33193,
       33198, 33203, 33287, 33298, 33313, 33328, 33363, 33444, 33455,
       33487], dtype=int64),
 1: array([   26,    34,    49, ..., 33463, 33473, 33494], dtype=int64),
 2: array([    2,     5,    13, ..., 33493, 33497, 33498], dtype=int64),
 3: array([    0,     1,     3, ..., 33495, 33496, 33499], dtype=int64),
 4: array([   38,   108,   229,   265,   429,   434,   647,   689,   695,
          733,   769,   776,   789,   839,   957,  1002,  1023,  1211,
         1254,  1260,  1304,  1320,  1479,  1551,  1621,  1636,  1647,
         1707,  1748,  1863,  1952,  2004,  2009,  2048,  2215,  2272,
         2277,  2310,  2384,  2458,  2500,  2509,  2567,  2569,  2649,
         2697,  2707,  2753,  2775,  2776,  2800,  2822,  2835,  2841,
         3038,  3040,  3041,  3091,  3102,  3104,  3171,  3240,  3312,
         3366,  3421,  3469,  3507,  3640,  3820,  3925,  3969,  4022,
```

```
 4220,  4226,  4238,  4273,  4302,  4314,  4431,  4591,  4605,
 4668,  4716,  4754,  4846,  4872,  4988,  5101,  5193,  5228,
 5264,  5275,  5418,  5505,  5530,  5580,  5636,  5800,  5821,
 5838,  6002,  6101,  6126,  6148,  6300,  6363,  6383,  6389,
 6441,  6448,  6509,  6676,  6774,  6818,  6850,  6863,  7034,
 7068,  7085,  7110,  7149,  7150,  7183,  7185,  7295,  7301,
 7328,  7373,  7411,  7439,  7512,  7593,  7632,  7666,  7675,
 7692,  7696,  7739,  7799,  7855,  7864,  7931,  7934,  7955,
 8014,  8058,  8081,  8108,  8122,  8128,  8219,  8243,  8252,
 8291,  8320,  8398,  8414,  8450,  8506,  8516,  8522,  8524,
 8528,  8554,  8601,  8619,  8687,  8710,  8822,  8834,  8847,
 8849,  8883,  8974,  8991,  9027,  9036,  9083,  9088,  9093,
 9243,  9340,  9342,  9355,  9457,  9462,  9487,  9517,  9524,
 9631,  9673,  9718,  9779,  9799,  9927,  9961,  9967,  9987,
10177, 10332, 10489, 10543, 10560, 10617, 10618, 10650, 10674,
10793, 10808, 10848, 10930, 10942, 11135, 11256, 11260, 11307,
11313, 11368, 11451, 11512, 11565, 11653, 11800, 11808, 11854,
11872, 11910, 11931, 11942, 11965, 12038, 12049, 12108, 12165,
12191, 12216, 12398, 12434, 12519, 12526, 12671, 12751, 12768,
12775, 12779, 12896, 12921, 13000, 13074, 13396, 13499, 13623,
13907, 14009, 14031, 14041, 14131, 14235, 14276, 14289, 14320,
14383, 14390, 14443, 14569, 14600, 14698, 14709, 14726, 14732,
14760, 14769, 14773, 14782, 14854, 14951, 14956, 15040, 15042,
15087, 15105, 15314, 15353, 15378, 15405, 15549, 15597, 15655,
15705, 15712, 15799, 15847, 15851, 15881, 15904, 15952, 15997,
15998, 16011, 16070, 16101, 16242, 16267, 16288, 16313, 16320,
16332, 16424, 16491, 16770, 16802, 16872, 16951, 16962, 17067,
17138, 17185, 17216, 17247, 17329, 17353, 17444, 17664, 17665,
17678, 17704, 17800, 18005, 18069, 18273, 18373, 18386, 18389,
18481, 18498, 18607, 18619, 18650, 18715, 18735, 18799, 19039,
19049, 19081, 19189, 19222, 19330, 19336, 19338, 19347, 19424,
19476, 19495, 19558, 19595, 19628, 19676, 19726, 19739, 19847,
19851, 19972, 19996, 20015, 20056, 20106, 20109, 20248, 20366,
20417, 20420, 20495, 20549, 20655, 20690, 20702, 20795, 20829,
20858, 20935, 20945, 20947, 20974, 20999, 21036, 21171, 21205,
21308, 21366, 21420, 21460, 21594, 21641, 21738, 21891, 21904,
22058, 22060, 22090, 22148, 22159, 22167, 22171, 22337, 22373,
22522, 22632, 22916, 22949, 22957, 23055, 23077, 23082, 23123,
23227, 23288, 23295, 23318, 23349, 23381, 23440, 23448, 23457,
23488, 23510, 23530, 23713, 23714, 23724, 23785, 23818, 23876,
23944, 23993, 24033, 24062, 24076, 24117, 24153, 24198, 24259,
24331, 24364, 24466, 24510, 24516, 24530, 24633, 24699, 24792,
24871, 24912, 24928, 25043, 25064, 25153, 25154, 25158, 25231,
25253, 25274, 25312, 25350, 25445, 25466, 25573, 25586, 25613,
25617, 25669, 25712, 25714, 25738, 25760, 25869, 25918, 25964,
```

```
        26004, 26084, 26103, 26111, 26321, 26397, 26420, 26592, 26613,
        26711, 26722, 26725, 26808, 26958, 26960, 27072, 27168, 27197,
        27213, 27357, 27400, 27416, 27426, 27459, 27500, 27620, 27648,
        27682, 27701, 27725, 27768, 27815, 28073, 28107, 28154, 28166,
        28201, 28276, 28291, 28317, 28332, 28356, 28372, 28398, 28511,
        28567, 28605, 28633, 28734, 28758, 28848, 28909, 28924, 28933,
        28996, 29042, 29066, 29100, 29190, 29218, 29252, 29271, 29291,
        29319, 29393, 29403, 29441, 29554, 29556, 29597, 29646, 29780,
        29793, 29856, 29877, 29924, 29968, 30010, 30248, 30291, 30302,
        30316, 30323, 30354, 30444, 30494, 30530, 30563, 30647, 30656,
        30667, 30718, 30766, 30850, 30880, 30896, 31002, 31086, 31143,
        31164, 31174, 31191, 31199, 31200, 31219, 31231, 31266, 31286,
        31331, 31461, 31480, 31573, 31707, 31749, 31795, 31925, 31964,
        31986, 32030, 32033, 32056, 32061, 32075, 32098, 32185, 32220,
        32270, 32305, 32308, 32349, 32406, 32444, 32670, 32703, 32800,
        32961, 32986, 33015, 33100, 33116, 33118, 33195, 33259, 33262,
        33318, 33343, 33392, 33404], dtype=int64),
5: array([  121,   209,   220,   312,   389,   390,   419,   433,   464,
         496,   515,   561,   869,   933,   987,  1072,  1127,  1133,
        1137,  1139,  1219,  1263,  1442,  1529,  1583,  1659,  1668,
        1712,  1794,  1908,  1922,  1956,  1965,  2112,  2120,  2202,
        2243,  2295,  2398,  2653,  2680,  2786,  2839,  2840,  2906,
        2960,  2968,  3026,  3222,  3265,  3383,  3416,  3509,  3529,
        3554,  3614,  3690,  3699,  3879,  3991,  4036,  4049,  4264,
        4417,  4418,  4544,  4598,  4696,  4808,  4946,  5126,  5141,
        5174,  5257,  5274,  5284,  5338,  5350,  5390,  5495,  5533,
        5566,  5690,  5740,  5879,  5915,  5960,  6283,  6372,  6424,
        6553,  6584,  6657,  6669,  6900,  6920,  7043,  7133,  7187,
        7270,  7302,  7333,  7368,  7403,  7478,  7566,  7687,  7768,
        7772,  7773,  7816,  7920,  7937,  7972,  8021,  8035,  8062,
        8129,  8186,  8198,  8214,  8300,  8425,  8451,  8758,  8859,
        8867,  8871,  8961,  8997,  9062,  9115,  9146,  9303,  9358,
        9362,  9387,  9441,  9469,  9528,  9563,  9569,  9576,  9923,
        9938, 10047, 10125, 10210, 10246, 10321, 10324, 10354, 10453,
       10482, 10544, 10577, 10642, 10744, 10821, 10835, 11060, 11184,
       11254, 11295, 11333, 11346, 11424, 11441, 11464, 11554, 11691,
       11776, 11853, 11865, 11930, 11948, 11968, 11986, 11998, 12268,
       12379, 12457, 12539, 12690, 12765, 12810, 12931, 13015, 13071,
       13172, 13254, 13340, 13408, 13421, 13605, 13849, 14189, 14246,
       14274, 14408, 14418, 14567, 14594, 14733, 14789, 14829, 14841,
       14872, 14915, 14916, 15188, 15340, 15344, 15453, 15472, 15495,
       15544, 15827, 15891, 15913, 15963, 16186, 16298, 16401, 16416,
       16511, 16556, 16600, 16632, 16707, 16746, 16873, 16912, 16913,
       16974, 17002, 17035, 17206, 17314, 17354, 17446, 17499, 17684,
       17745, 17816, 17845, 17918, 17950, 17993, 18003, 18065, 18067,
```

```
 18077, 18130, 18242, 18247, 18255, 18343, 18397, 18546, 18547,
 18549, 18560, 18717, 18726, 18760, 18786, 18838, 18842, 18902,
 18987, 19068, 19168, 19208, 19268, 19511, 19524, 19573, 19583,
 19624, 19630, 19642, 20037, 20362, 20363, 20387, 20679, 20698,
 20776, 20797, 20867, 20944, 21074, 21294, 21297, 21312, 21440,
 21474, 21503, 21534, 21554, 21824, 21865, 21880, 21958, 22002,
 22051, 22076, 22101, 22235, 22247, 22352, 22365, 22416, 22458,
 22525, 22570, 22717, 22739, 22829, 23042, 23152, 23159, 23161,
 23162, 23217, 23404, 23483, 23726, 23774, 23829, 23874, 23926,
 23940, 23951, 23991, 24058, 24065, 24094, 24101, 24150, 24161,
 24192, 24301, 24314, 24577, 24698, 24713, 24859, 24877, 24898,
 24937, 24945, 24956, 25019, 25075, 25094, 25104, 25174, 25277,
 25280, 25286, 25304, 25340, 25346, 25391, 25488, 25492, 25510,
 25516, 25522, 25605, 25661, 25713, 25783, 25843, 25846, 25859,
 25877, 26022, 26094, 26109, 26182, 26394, 26593, 26606, 26754,
 26765, 26791, 26862, 26981, 26998, 27096, 27249, 27263, 27326,
 27476, 27621, 27669, 27681, 27686, 27807, 27816, 27998, 28035,
 28041, 28131, 28308, 28338, 28393, 28394, 28395, 28457, 28483,
 28494, 28550, 28764, 28813, 28829, 28863, 28873, 28954, 29275,
 29450, 29511, 29522, 29650, 29745, 29820, 29967, 30049, 30057,
 30093, 30128, 30342, 30438, 30505, 30516, 30605, 30762, 30779,
 30907, 30919, 31015, 31115, 31152, 31250, 31417, 31433, 31483,
 31536, 31559, 31647, 31714, 31924, 31926, 32123, 32157, 32353,
 32403, 32422, 32454, 32514, 32551, 32566, 32570, 32579, 32683,
 32747, 32779, 32891, 32895, 33110, 33235, 33272, 33417, 33486],
dtype=int64)}
```

In [71]:

```
essays = preprocessed_essays

cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []
cluster5 = []
cluster6 = []
for i in range(kmeans.labels_.shape[0]):
    if kmeans.labels_[i] == 0:
        cluster1.append(essays[i])
    elif kmeans.labels_[i] == 1:
        cluster2.append(essays[i])
    elif kmeans.labels_[i] == 2:
        cluster3.append(essays[i])
    elif  kmeans.labels_[i] == 3:
        cluster4.append(essays[i])
```

```
    elif  kmeans.labels_[i] == 4:
        cluster5.append(essays[i])
    elif  kmeans.labels_[i] == 5:
        cluster6.append(essays[i])
```

In [72]:
```
#https://stackoverflow.com/a/306417
import random
print('%s'%(random.choice(cluster1)))
```

```
my class diverse i group students benefit additional intervention not common techniques we also steam classroom parti
cipate many hands activities really keep kids involved students class able take responsibility learning we not strict
ly book class they able help guide go year this makes learning student centered exciting kids these tools used regula
rly classroom every day the osmo system multifaceted give kids opportunity would not otherwise creative technology th
e building supplies much needed my students able build create using items help retain information they excited learn
i want kids look forward coming class day i want classroom tools necessary encourage stay positive let know much i be
lieve nannan
```

In [73]:
```
#https://www.datacamp.com/community/tutorials/wordcloud-python

from PIL import Image
wine_mask = np.array(Image.open("wine_mask.png"))
wine_mask
```

Out[73]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

In [74]:
```
def transform_format(val):

    if val == 0:
        return 255
    else:
        return val
```

In [75]:
```
transformed_wine_mask = np.ndarray((wine_mask.shape[0],wine_mask.shape[1]), np.int32)

for i in range(len(wine_mask)):
    transformed_wine_mask[i] = list(map(transform_format, wine_mask[i]))
```

In [76]:

```
words=''
for i in cluster1:
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [77]:
```
print('%s'%(random.choice(cluster2)))
```

```
imagine first generation family receive formal education now imagine also english language learner these kids some ne
w comers country no prior education english our school made 94 economically disadvantaged families the majority stude
nts limited support resources home this makes time class even precious however despite obstacles bright determined ex
cited learn research shows students develop second language proficiency much faster strong foundation maternal langua
ge for class 19 bilingual kids means reading listening stories spanish over years i acquired quite classroom library
lacking spanish high interest books with books kids fully enthralled read alouds they excited shop books collection p
lace independent reading bags then throughout week i certain eager focused reading fun books independently from readi
ng fluency comprehension skyrocket nannan
```

In [78]:
```
words=''
```

```
for i in cluster2:
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [79]:
```
print('%s'%(random.choice(cluster3)))
```

```
i work small fully inclusive charter school almost 90 students receive free reduced lunch i work students labeled man
y ways ell ld id bip iep slp apraxic autistic add adhd odd list goes i working help students remove labels find feet
well voices my students come variety backgrounds one thing common thirst creative learning opportunities no one ever
accuse artist my lack ability not deter search creative art activities integrate multiple subject areas origami conne
ct fine motor art activities math science research projects enhance outcomes energize connections students make when
thinking integrating steam activities round classroom instruction i keep mind seating center organization the table s
tools offer storage allow create vibrant learning environment paper art collide coming alive hands students nannan
```

In [80]:
```
words=''
for i in cluster3:
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [81]:
```
print('%s'%(random.choice(cluster4)))
```

```
my 5th graders awesome kids the students variety education levels interests my kiddos gain education hands activities
active lessons i love reading i teach love read i teach high poverty community kids arrive school excited routine day
my students experience world books i teach students question situations think outside box push limits i want students
want school they come school not learn love loved fellow classmates we strive family respects ideas inspire dream big
these books improve lives giving better understanding grammar techniques as well giving books read aloud school day i
use read alouds reinforce reading grammar lessons these books allow students hear story well see examples listening s
tory i use chapter books help teach reading lessons the students enjoy listening characters learning speaking listeni
ng standards reading also great way teach ela standards literature standards nannan
```

In [82]:
```
words=''
```

```
for i in cluster4:
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [83]:
```
print('%s'%(random.choice(cluster5)))
```

```
my students diverse group children various interests learning styles many students english language learners they amb
itious hard working they curious variety topics vibrant imaginations my students look reading learn interests read ch
aracters identify i hope nurture students love reading providing books thoroughly enjoy appreciate the books donated
project help students fall love learning allowing read stories interested a lot time students not enjoy reading simpl
y not found right book by collecting library books appeals variety interests diversity students students able better
identify characters read the wide variety books included project ensure students represented classroom library nannan
```

In [84]:
```
words=''
for i in cluster5:
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [85]:
```
print('%s'%(random.choice(cluster6)))
```

```
my population unique living hospital i teach they hospitalized emotional intellectual disabilities most students come
broken homes foster care their level safety coping skills day day life compromised due prolonged abuse witnessing dom
estic violence drug alcohol abuse mental illness my students need materials help express discover academics fun the s
upplies i requested already direction they beaiding students furthering artistiic voice they learn art ever changing
narrative multiple directions supplies help substantiate creativity the supplies help support students come classroom
starting paint different masks learn use colors textures create furthering artistic scope the extra help supplies als
o generates avenues create inspirational projects especially since students transitional change frequently due school
hospital setting nannan
```

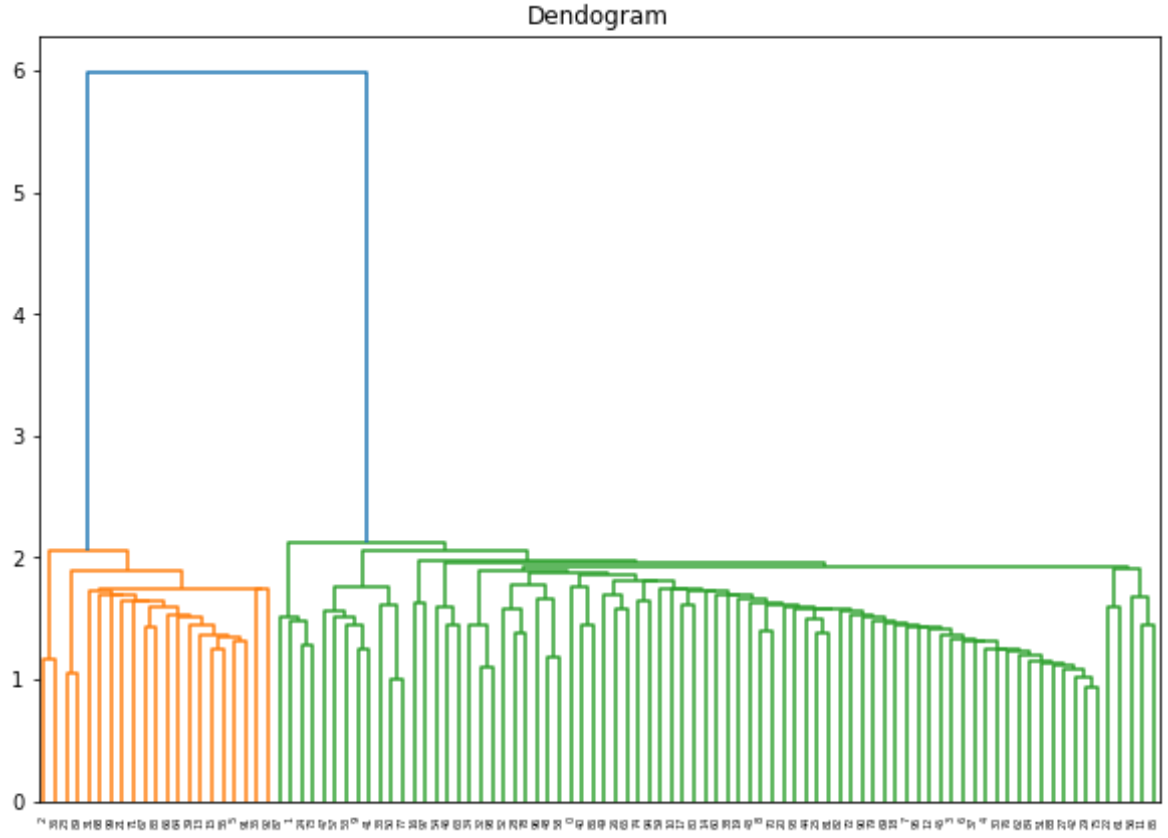
In [86]:
```
words=''
```

```
for i in cluster6:
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

5.Apply Agglomerative Clustering

5.1 view of Agglomerative Clustering

In [87]:
```
import scipy.cluster.hierarchy as shc
X_train = X_train_fe_5000[:100]
X_tr = X_train.toarray()

algo_title = 'Agglomerative Clustering'
plt.figure(figsize=(10, 7))
plt.title("Dendogram")
dend = shc.dendrogram(shc.linkage(X_tr, method='ward'))
```



In [88]:
```
X_train_fe_5000_new = X_train_fe_5000[:2500]
```

In [89]:
```
X_train_fe_5000_new.shape
```

Out[89]:
```
(2500, 5000)
```

5.2 apply Agglomerative Clustering

In [90]:
```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
clusters=[2,3,5,7]
scores = []
for i in clusters:
    aggcl=AgglomerativeClustering(n_clusters=i).fit(X_train_fe_5000_new.toarray())
    score=silhouette_score(X_train_fe_5000_new, aggcl.labels_, random_state=42)
    scores.append(score)
```

In [91]:
```
plt.plot(clusters, scores)
plt.xlabel('Clusters')
plt.ylabel('Silhoutte Score')
plt.title('Elbow Knee')
plt.legend('Knee')
plt.show()
```

observations:

1.Best value for cluster is 3

In [92]:
```
from sklearn.cluster import AgglomerativeClustering

aggcl=AgglomerativeClustering(n_clusters=3).fit(X_train_fe_5000_new.toarray())
```

In [93]:
```
clustera0=[]
clustera1=[]
clustera2=[]

essays = preprocessed_essays
for i in range(aggcl.labels_.shape[0]):
    if aggcl.labels_[i] == 0:
        clustera0.append(essays[i])
    elif aggcl.labels_[i] == 1:
        clustera1.append(essays[i])
    elif aggcl.labels_[i] == 2:
        clustera2.append(essays[i])
```

In [94]:
```
print('%s'%(random.choice(clustera0)))
```

```
remember saying a picture is worth thousand words this using technology study math science students i 40 hard working
willing learn students delta they attend rural title i school delta limited outdated technology classroom they love w
orking technology order compete today society need use updated technology work high school advanced science fuses tog
ether extensive curriculum bursting seams historical breakthroughs abstract ideas to effectively investigate topics e
ssential employ multitude practical hands activities full enriching inspiring activities the laptop essential tool as
sociated accomplishing goals unlock limitless techniques accommodate diverse learning styles among students each scie
nce content standards strengthened incorporating advantageous technology there thousands educational apps enlighten s
tudents eco friendly apps space exploration dissection human body weather geography mathematics reading textbooks var
ious apps appeal virtually student interest with valuable resource i equipped assist students special needs many apps
help socialization communication nannan
```

Wordcloud for clusters

In [95]:
```
words=''
for i in clustera0:
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [96]:
```
print('%s'%(random.choice(clustera1)))
```

```
i work title one school we provide free breakfast lunch students i 27 students class list my students combination eng
lish ell students i privilege teaching kindergartners many students come low income families many come not prepare sc
hool due lack funds home they sometimes unable complete homework not supplies home as teacher i try provide much i or
der succeed i requesting school supplies it difficult complete class work homework without basic school supplies our
class use glue sticks every day complete projects class work pencils must students write every day difficult complete
assignments lost taken home pencils crayons it important basic necessities sometimes difficult buy money issue in ord
er students succeed basic needs must met please help class funding project my students greatly appreciate nannan
```

In [97]:
```
words=''
for i in clustera1:
```

```
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [98]:
```
print('%s'%(random.choice(clustera2)))
```

```
my students blended classroom consists diverse crowd special needs students english language learners students genera
l education population their age range three five years old as special education teacher i serve diverse crowd studen
ts special needs the students i serve mostly african american hispanic my students come low socioeconomic circumstanc
es the parents community try best support students these students inspire better teacher person i learn daily the uni
queness students brings class unimaginable dr seuss quotes the read things know the learn places go i could not agree
statement reading important part life when read young children not opens imaginations also expands knowledge world it
helps develop necessary language listening skills need successful school well prepares understand importance written
words that everyday make one main priorities blended preschool program read the amount knowledge students develop sho
rt amount time taking part read aloud book browsing astonishes i absolutely love creative excited students answer bas
ic comprehension questions confidence feel able answer higher level thinking questions it not puts smile face well i
love not able share stories students also learning stories tell recreate in end resorts back vocabulary riched studen
t centered themed studies creative curriculum preschool i requesting variety themed books go along community helpers
unit building study well farm study order knowledge fun interesting way nannan
```

In [99]:
```
words=''
for i in clustera2:
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

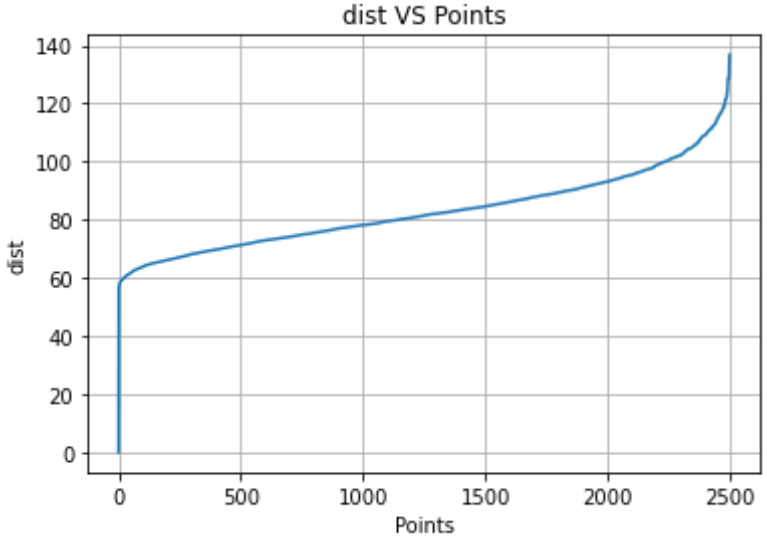
6 Apply DBSCAN

In [100…

```
min_points = 1000
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import euclidean_distances

dbScanData=StandardScaler().fit_transform(X_train_fe_5000_new.toarray())

distance=[]
for point in tqdm(dbScanData):
    temp = euclidean_distances(dbScanData, point.reshape(1, -1))
```

```
    distance.append(temp[min_points])
sorted_distance = np.sort(np.array(distance))

sorted_dist = np.sort(sorted_distance.reshape(1,-1)[0])
points = [i for i in range(len(dbScanData))]


# Draw distances(d_i) VS points(x_i) plot
plt.plot(points, sorted_dist)
plt.xlabel('Points')
plt.ylabel('dist')
plt.title('dist VS Points')
plt.grid()
plt.show()
```

```
100%|██████████| 2500/2500 [01:14<00:00, 33.51it/s]
```



In [101…

```
#we can see that point of inflexion is at eps=65
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=65,n_jobs=-1)
dbscan.fit(dbScanData)
print('No of clusters: ',len(set(dbscan.labels_)))
print('Cluster are including noise i.e -1: ',set(dbscan.labels_))
```

```
No of clusters:  2
```

Cluster are including noise i.e -1: {0, -1}

In [102…
```
#ignoring -1 as it is for noise
cluster1=[]
noisecluster1=[]
for i in range(dbscan.labels_.shape[0]):
    if dbscan.labels_[i] == 0:
        cluster1.append(essays[i])
    elif dbscan.labels_[i] == -1:
        noisecluster1.append(essays[i])
```

In [103…
```
print('%s'%(random.choice(cluster1)))
```

```
some students difficult time staying task small groups especially guided reading math these wobble chairs might key u
nlocking energy wiggle release would allow move focus i work small inclusive charter school small playground virtuall
y no fields run i work using movement classroom allow students bit energy release my school allows lot creative visio
n i hope expand classroom creating different types learning stations the wobble chairs would add kinesthetic element
guided reading math my guided reading math station need reenergized i think wobble chairs help build reading stamina
better focus 3rd graders endless energy a recess not enough time release amount energy students come class i think sm
all limited amount movement i see increase focus interest also math reading acquisition they not dread guided reading
guided math station longer i think donation 6 wobble chairs help students acquire necessary math reading skills incre
ase focus stamina interest my classroom sensory experience my students need anticipate station movement not allowed i
nvited wobble chairs would great focal point classroom
```

In [104…
```
words=''
for i in cluster1:
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [105…
```
print('%s'%(random.choice(noisecluster1)))
```

```
our school located rural area approximately 650 students grades 6 8 the library meeting place student morning well lu
nchtime we trying modernize library meet needs students students love come library hang using different seating arran
gements currently two high boy tables students requested cool places please consider helping us make reality students
last year decided overhaul library one popular projects taking old card catalog turning highboy table students this b
ecome favorite spot students process creating another one an additional project taken adding overhang onto 24 foot lo
ng book shelf this another place students sit work your donation would allow us purchase stools used two new seating
areas we want make library place students want middle school tough many students having safe comfortable welcoming pl
ace go something students need nannan
```

In [106…
```
words=''
```

```
for i in noisecluster1:
    words+=str(i)
from wordcloud import WordCloud
wordcloud = WordCloud(background_color="white", max_words=1000, mask=transformed_wine_mask,
               stopwords=stopwords, contour_width=3, contour_color='firebrick').generate(words)

# Display the generated image:
plt.figure(figsize=[16,8])
plt.title("Word cloud of cluster")
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

7.Pretty Table

In [110…

```
#prettytable for kmeans
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","BEST K","Eps","Number of clusters(INCLUDING NOISE)"]

x.add_row(['KMEANS','6','NA','NA'])
x.add_row(['AGGLOMERATIVE','3','NA','NA'])

x.add_row(['DBSCAN','2',65,2])

print(x)
```

```
+---------------+--------+-----+-----------------------------------+
|     Model     | BEST K | Eps | Number of clusters(INCLUDING NOISE) |
+---------------+--------+-----+-----------------------------------+
|     KMEANS    |   6    |  NA |                 NA                |
| AGGLOMERATIVE |   3    |  NA |                 NA                |
|     DBSCAN    |   2    |  65 |                 2                 |
+---------------+--------+-----+-----------------------------------+
```

8.Conclusion

1.Plot word cloud for every cluster and show top words from each cluster.

2.Some clusters are dense

3.Found optimal clusters 3 and 6 in Kmeans and Agglomrtative

4.We use euclidean distance to find best eps= in DBSCAN