Assignment 10: Clustering

Data splitting and pre-processing

In [1]:

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


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.model_selection import train_test_split
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
from wordcloud import WordCloud, STOPWORDS
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


import pickle
from tqdm import tqdm
import os

from collections import Counter
```

In [2]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:

```
project_data.isnull().sum()
```

Out[3]:

```
Unnamed: 0                                      0
id                                              0
teacher_id                                      0
teacher_prefix                                  3
school_state                                    0
project_submitted_datetime                      0
project_grade_category                          0
project_subject_categories                      0
project_subject_subcategories                   0
project_title                                   0
project_essay_1                                 0
project_essay_2                                 0
project_essay_3                            105490
project_essay_4                            105490
project_resource_summary                        0
teacher_number_of_previously_posted_projects    0
project_is_approved                             0
```

```
project_is_approved                                    0
dtype: int64
```

In [4]:

```
#filling 3 null teacher prefix values with Teacher

project_data["teacher_prefix"].fillna("Teacher",inplace = True)
project_data.isnull().sum()
```

Out[4]:

```
Unnamed: 0                                             0
id                                                     0
teacher_id                                             0
teacher_prefix                                         0
school_state                                           0
project_submitted_datetime                             0
project_grade_category                                 0
project_subject_categories                             0
project_subject_subcategories                          0
project_title                                          0
project_essay_1                                        0
project_essay_2                                        0
project_essay_3                                   105490
project_essay_4                                   105490
project_resource_summary                               0
teacher_number_of_previously_posted_projects           0
project_is_approved                                    0
dtype: int64
```

In [5]:

```
# merge two column text dataframe:
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 [6]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [7]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109248 entries, 0 to 109247
Data columns (total 20 columns):
Unnamed: 0                                        109248 non-null int64
id                                                109248 non-null object
teacher_id                                        109248 non-null object
teacher_prefix                                    109248 non-null object
school_state                                      109248 non-null object
project_submitted_datetime                        109248 non-null object
project_grade_category                            109248 non-null object
project_subject_categories                        109248 non-null object
project_subject_subcategories                     109248 non-null object
project_title                                     109248 non-null object
project_essay_1                                   109248 non-null object
project_essay_2                                   109248 non-null object
project_essay_3                                   3758 non-null object
project_essay_4                                   3758 non-null object
project_resource_summary                          109248 non-null object
teacher_number_of_previously_posted_projects      109248 non-null int64
project_is_approved                               109248 non-null int64
essay                                             109248 non-null object
price                                             109248 non-null float64
quantity                                          109248 non-null int64
dtypes: float64(1), int64(4), object(15)
memory usage: 17.5+ MB
```

memory usage: 17.5+ MB

Considering only 10k samples.

In [12]:

```
from sklearn.utils import resample
X = resample(project_data,n_samples = 10000 )

X["project_is_approved"].value_counts()
```

In [15]:

```
#splitting data as 30% to test
# dropping project is approved as we dont need it.
#X = project_data.drop("project_is_approved",axis = 1)
X_train, X_test = train_test_split(X, test_size=0.20, random_state=42)
```

In [18]:

```
print(X_train.shape,"\t",X_test.shape)
```

```
(8000, 20)    (2000, 20)
```

Preprocessing categorical Features

1. project subject categories

In [ ]:

```
#using code from assignment
# project subject categories
catogories = list(X_train['project_subject_categories'].values)

cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

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

from collections import Counter
my_counter = Counter()
for word in X_train['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]))


# project subject categories for test data

catogories = list(X_test['project_subject_categories'].values)

cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
```

```
        temp = temp.replace('&', '_')
    cat_list.append(temp.strip())

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

1. project subject sub_categories

In [20]:

```
sub_catogories = list(X_train['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

X_train['clean_subcategories'] = sub_cat_list
X_train.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 X_train['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]))


sub_catogories = list(X_test['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

1. Teacher Prefix

In [21]:

```
#preprocessing teacher prefix
prefix = list(X_train['teacher_prefix'].values)
prefix_list = []
for i in prefix:
    temp = ""
    if "." in i:
            i=i.replace('.','')
    temp+=i.strip()+" "
    prefix_list.append(temp.strip())

X_train['clean_prefix'] = prefix_list

my_counter = Counter()
for word in X_train['clean_prefix'].values:
  my_counter.update(word.split())
```

```
prefix_dict = dict(my_counter)
sorted_prefix_dict = dict(sorted(prefix_dict.items(), key=lambda kv: kv[1]))
print(sorted_prefix_dict)


#preprocessing teacher prefix for test data
prefix = list(X_test['teacher_prefix'].values)
prefix_list = []
for i in prefix:
    temp = ""
    if "." in i:
            i=i.replace('.','')
    temp+=i.strip()+" "
    prefix_list.append(temp.strip())

X_test['clean_prefix'] = prefix_list
```

```
{'Dr': 2, 'Teacher': 178, 'Mr': 790, 'Ms': 2888, 'Mrs': 4142}
```

1. Project Grade Category

In [22]:

```
# preprocessing of grade category for train data

grade = list(X_train['project_grade_category'].values)
grade_list = []
for i in grade:
    temp = ""
    if "Grades" in i:
      i = i.replace("Grades","")
    if "6-8" in i:
      i = i.replace("6-8","six_eight")
    if "3-5" in i:
      i = i.replace("3-5","three_five")
    if "9-12" in i:
      i = i.replace("9-12","nine_twelve")
    if "PreK-2" in i:
      i = i.replace("PreK-2","prek_two")
    temp+=i.strip()+" "
    grade_list.append(temp.strip())

X_train['clean_grade'] = grade_list

my_counter = Counter()
for word in X_train['clean_grade'].values:
  my_counter.update(word.split())

grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))
print(sorted_grade_dict)

# preprocessing of grade category for test data

grade = list(X_test['project_grade_category'].values)
grade_list = []
for i in grade:
    temp = ""
    if "Grades" in i:
      i = i.replace("Grades","")
    if "6-8" in i:
      i = i.replace("6-8","six_eight")
    if "3-5" in i:
      i = i.replace("3-5","three_five")
    if "9-12" in i:
      i = i.replace("9-12","nine_twelve")
    if "PreK-2" in i:
      i = i.replace("PreK-2","prek_two")
    temp+=i.strip()+" "
    grade_list.append(temp.strip())

X_test['clean_grade'] = grade_list
```

```
{'nine_twelve': 704, 'six_eight': 1205, 'three_five': 2675, 'prek_two': 3246}
```

{'nine_twelve': 794, 'six_eight': 1285, 'three_five': 2675, 'prek_two': 3246}

1. School State

In [23]:

```
#no need of preprocessing on school state

state = X_train["school_state"].value_counts()
sorted_state = dict(state)
sorted_state_dict = dict(sorted(sorted_state.items(), key=lambda kv: kv[1]))
X_train["clean_state"] = X_train["school_state"]

#similarly for X_test
X_test["clean_state"] = X_test["school_state"]
```

Preprocessing Numerical Feature

1. Standardizing price

In [24]:

```
from sklearn.preprocessing import StandardScaler


price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1))
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

#train data price standardization
price_standardized = price_scalar.transform(X_train['price'].values.reshape(-1, 1))

#test data price stanardization. Fit method applied on X_train
test_price_standardized = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

1. Standardizing quantity

In [25]:

```
price_scalar = StandardScaler()
price_scalar.fit(X_train["quantity"].values.reshape(-1, 1))
print(f"Mean of Quantity : {price_scalar.mean_[0]}, Standard deviation of Quantity :
{np.sqrt(price_scalar.var_[0])}")

#train data quantity standardization
quantity_standardized = price_scalar.transform(X_train["quantity"].values.reshape(-1, 1))

#test data quantity stanardization. Fit method applied on X_train
test_quantity_standardized = price_scalar.transform(X_test["quantity"].values.reshape(-1, 1))
```

```
C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

```
Mean of Quantity : 16.7425, Standard deviation of Quantity : 25.993676033797144
```

```
C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

1. Standardizing number of ppp

In [26]:

```
price_scalar = StandardScaler()
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

#train data ppp standardization
number_ppp_standardized =
price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))

#test data price stanardization. Fit method applied on X_train
test_number_ppp_standardized =
price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)
)
```

```
C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

```
Mean : 11.2885, Standard deviation : 27.839616875057747
```

```
C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning: Data with input dtype int64 was c
onverted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Preprocessing of Text Feature for both test and train data

In [27]:

```
#using function and stopwords form assignemnt

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

# we are removing the words from the stop words list: 'no', 'nor', 'not'
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', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            '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', "d
esn'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', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

1. preprocessing of project essay

In [28]:

```
from tqdm import tqdm

#for train data
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['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())

test_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['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)
    test_preprocessed_essays.append(sent.lower().strip())
```

```
100%|
|
8000/8000 [00:04<00:00, 1676.60it/s]
100%|
|
2000/2000 [00:01<00:00, 1679.07it/s]
```

1. preprocessing of project title

In [29]:

```
preprocessed_title = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['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())

# for test data
test_preprocessed_title = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['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)
    test_preprocessed_title.append(sent.lower().strip())
```

```
100%|
|
8000/8000 [00:00<00:00, 36684.90it/s]
100%|
|
2000/2000 [00:00<00:00, 37010.80it/s]
```

Vectorizing of Categorical data

In [30]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)

# fitting on train data
vectorizer.fit(X_train['clean_categories'].values)
print(vectorizer.get_feature_names())
categories_feature = vectorizer.get_feature_names()

# for train data
categories_one_hot = vectorizer.transform(X_train['clean_categories'].values)

print("Shape of matrix after one hot encodig ",categories_one_hot.shape)

# for test data
test_categories_one_hot = vectorizer.transform(X_test['clean_categories'].values)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (8000, 9)
```

1. Vectorizing project subcategories

In [31]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)

# fitting on train data
vectorizer.fit(X_train['clean_subcategories'].values)
print(vectorizer.get_feature_names())
subcategories_feature = vectorizer.get_feature_names()

# for train data
sub_categories_one_hot = vectorizer.transform(X_train['clean_subcategories'].values)
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)

# for test data
test_sub_categories_one_hot = vectorizer.transform(X_test['clean_subcategories'].values)
```

```
['Economics', 'CommunityService', 'Civics_Government', 'ParentInvolvement', 'FinancialLiteracy', '
Extracurricular', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
```

```
'PerformingArts', 'SocialSciences', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'History_Geography', 'Music', 'Health_LifeScience', 'Gym_Fitness',
'EarlyDevelopment', 'ESL', 'VisualArts', 'EnvironmentalScience', 'Health_Wellness',
'AppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (8000, 30)
```

1. vectorizing teacher prefix

In [32]:

```
vectorizer = CountVectorizer(vocabulary=list(prefix_dict.keys()), lowercase=False, binary=True)

# fitting on train data
vectorizer.fit(X_train['clean_prefix'].values)
print(vectorizer.get_feature_names())
prefix_feature = vectorizer.get_feature_names()
# for train data
prefix_one_hot = vectorizer.transform(X_train['clean_prefix'].values)
print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)

# for test data
test_prefix_one_hot = vectorizer.transform(X_test['clean_prefix'].values)
```

```
['Mr', 'Mrs', 'Ms', 'Teacher', 'Dr']
Shape of matrix after one hot encodig  (8000, 5)
```

1. Vectorizing school state and grade

In [33]:

```
vectorizer = CountVectorizer(vocabulary=list(grade_dict.keys()), lowercase=False, binary=True)

# fitting on train data
vectorizer.fit(X_train['clean_grade'].values)
print(vectorizer.get_feature_names())
grade_feature = vectorizer.get_feature_names()
# for train data
grade_one_hot = vectorizer.transform(X_train['clean_grade'].values)
print("Shape of matrix after one hot encodig ",grade_one_hot.shape)

# for test data
test_grade_one_hot = vectorizer.transform(X_test['clean_grade'].values)


vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(X_train['clean_state'].values)
print(vectorizer.get_feature_names())
state_one_hot = vectorizer.transform(X_train['clean_state'].values)
state_feature = vectorizer.get_feature_names()
test_state_one_hot = vectorizer.transform(X_test['clean_state'].values)
```

```
['three_five', 'prek_two', 'nine_twelve', 'six_eight']
Shape of matrix after one hot encodig  (8000, 4)
['VT', 'WY', 'ND', 'AK', 'MT', 'RI', 'SD', 'NE', 'NH', 'HI', 'NM', 'DE', 'WV', 'ME', 'IA', 'DC', 'K
S', 'ID', 'AR', 'MS', 'OR', 'CO', 'MD', 'KY', 'NV', 'MN', 'CT', 'AL', 'TN', 'OK', 'WI', 'VA', 'UT',
'LA', 'NJ', 'WA', 'MO', 'MA', 'AZ', 'OH', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
```

Vectorizing Text Feature

1. TFIDF

In [34]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(2,2),max_features=5000)
#fit using train data
vectorizer.fit(preprocessed_essays)
```

```
essay_feature_tfidf = vectorizer.get_feature_names()
# for train data
text_tfidf = vectorizer.transform(preprocessed_essays)
print("Shape of train matrix : ",text_tfidf.shape)
# for test data
test_text_tfidf = vectorizer.transform(test_preprocessed_essays)
print("Shape of test matrix : ",test_text_tfidf.shape)


# for title
vectorizer.fit(preprocessed_title)
title_feature_tfidf = vectorizer.get_feature_names()

# for train data
title_tfidf = vectorizer.transform(preprocessed_title)
print("Shape of train matrix : ",title_tfidf.shape)
# for test data
test_title_tfidf = vectorizer.transform(test_preprocessed_title)
print("Shape of test matrix : ",test_title_tfidf.shape)
```

```
Shape of train matrix :  (8000, 5000)
Shape of test matrix :  (2000, 5000)
Shape of train matrix :  (8000, 155)
Shape of test matrix :  (2000, 155)
```

Printing all

In [35]:

```
print("*"*70)
print("Categorical Features that are considered :- ")
print("Subject Categories :- ",categories_one_hot.shape)
print("Subject Sub-Categories :- ",sub_categories_one_hot.shape)
print("Sudent Grade :- ",grade_one_hot.shape)
print("School State :- ",state_one_hot.shape)
print("Teacher Prefix :- ",prefix_one_hot.shape)
print("*"*70)
```

```
**********************************************************************
Categorical Features that are considered :-
Subject Categories :-  (8000, 9)
Subject Sub-Categories :-  (8000, 30)
Sudent Grade :-  (8000, 4)
School State :-  (8000, 51)
Teacher Prefix :-  (8000, 5)
**********************************************************************
```

In [36]:

```
print("Text Features that are considered :- ")
print("*"*70)
print("Project Essay TFIDF:- ",text_tfidf.shape)
print("Project Title TFIDF:- ",title_tfidf.shape)
print("*"*70)
```

```
Text Features that are considered :-
**********************************************************************
Project Essay TFIDF:-  (8000, 5000)
Project Title TFIDF:-  (8000, 155)
**********************************************************************
```

sets

In [37]:

```
#combining all feature into one
from scipy.sparse import hstack

set_ =
```

```
hstack((categories_one_hot,sub_categories_one_hot,prefix_one_hot,state_one_hot,grade_one_hot,text_t
fidf,title_tfidf,price_standardized,quantity_standardized,number_ppp_standardized))
set_t =
hstack((test_categories_one_hot,test_sub_categories_one_hot,test_prefix_one_hot,test_state_one_hot
,test_grade_one_hot,test_text_tfidf,test_title_tfidf,test_price_standardized,test_quantity_standard
ized,test_number_ppp_standardized))

print(set_.shape,"\t",set_t.shape)
```

```
(8000, 5257)    (2000, 5257)
```

In [38]:

```
set_feature = categories_feature + subcategories_feature + prefix_feature + grade_feature + state_f
eature + essay_feature_tfidf + title_feature_tfidf


set_feature.append("price")
set_feature.append("quantity")
set_feature.append("number")
print(len(set_feature))
```

```
5257
```

In [39]:

```
from nltk.corpus import stopwords
import nltk
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\rdbz3b\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[39]:

```
True
```

In [40]:

```
chachedWords = stopwords.words('english')
def Plot_wordcloud(cluster):

    """
     Function for plotting wordcloud.
    """
    words = " "
    for ew in cluster:
        tokens = ew.split()
        for w in tokens:
            words = words+ " "+ w

    wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = chached
Words,
                          min_font_size = 10).generate(words)
    # plot the WordCloud image
    plt.figure(figsize = (8,8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.title("Word Cloud Plot")
    plt.show()
```

In [41]:

```
from sklearn.cluster import KMeans
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.decomposition import PCA
```

Selecting top 5000 components

Selecting top 5000 components...

In [42]:

```
pca = PCA(n_components=5000)
set_ = pca.fit_transform(set_.toarray())
set_t = pca.transform(set_t.toarray())
print(set_.shape)
print(set_t.shape)
```

```
(8000, 5000)
(2000, 5000)
```

SET1 - K-Means

In [45]:

```
k = [1,2,3,5,7,8]
k_info = dict()

for i in k:
    temp = dict()
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(set_)
    temp["inertia"] = kmeans.inertia_
    temp["labels"] = kmeans.labels_
    k_info[i] = temp
```

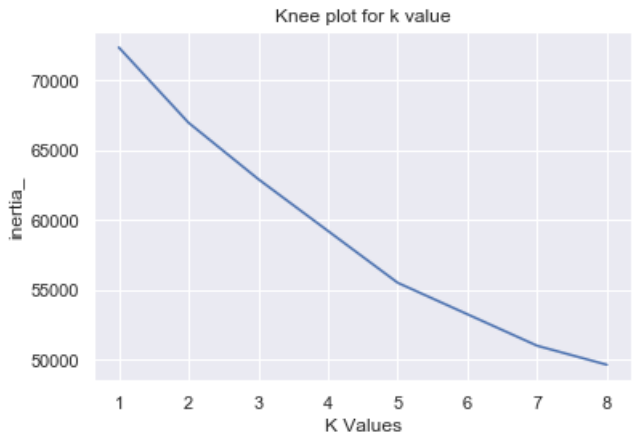
In [46]:

```
inertia = [x["inertia"] for x in k_info.values()]
```

Elbow plot

In [47]:

```
sns.set()
plt.plot(k,inertia)
plt.xlabel("K Values")
plt.ylabel("inertia_")
plt.title("Knee plot for k value")
plt.show()
```



Why k = 2 as best value...?

- Drop in loss at k = 2 is maximum as compared to other

Lets predict the labels of train set data points for k = 2

In [48]:

```
kmeans = KMeans(n_clusters=2).fit(set_)
```

Visualization of cluster

cluster = 2

In [52]:

```
# converting set_ to two dimetional using pca for visiualization.
pca = PCA(n_components=2)
set_v = pca.fit_transform(set_)

sns.set()
plt.figure(figsize=(8,8))
sns.scatterplot(x=set_v[:,0], y=set_v[:,1],hue=kmeans.labels_,palette="Set1")
plt.title("Train data cluster plot")
```

Out[52]:

```
Text(0.5, 1.0, 'Train data cluster plot')
```

Word Plot

In [53]:

```
test_predicted = []
for i in range(set_t.shape[0]):
    t = np.expand_dims(set_[i],axis = 0)

    test_predicted.append(kmeans.predict(t)[0])
```

In [55]:

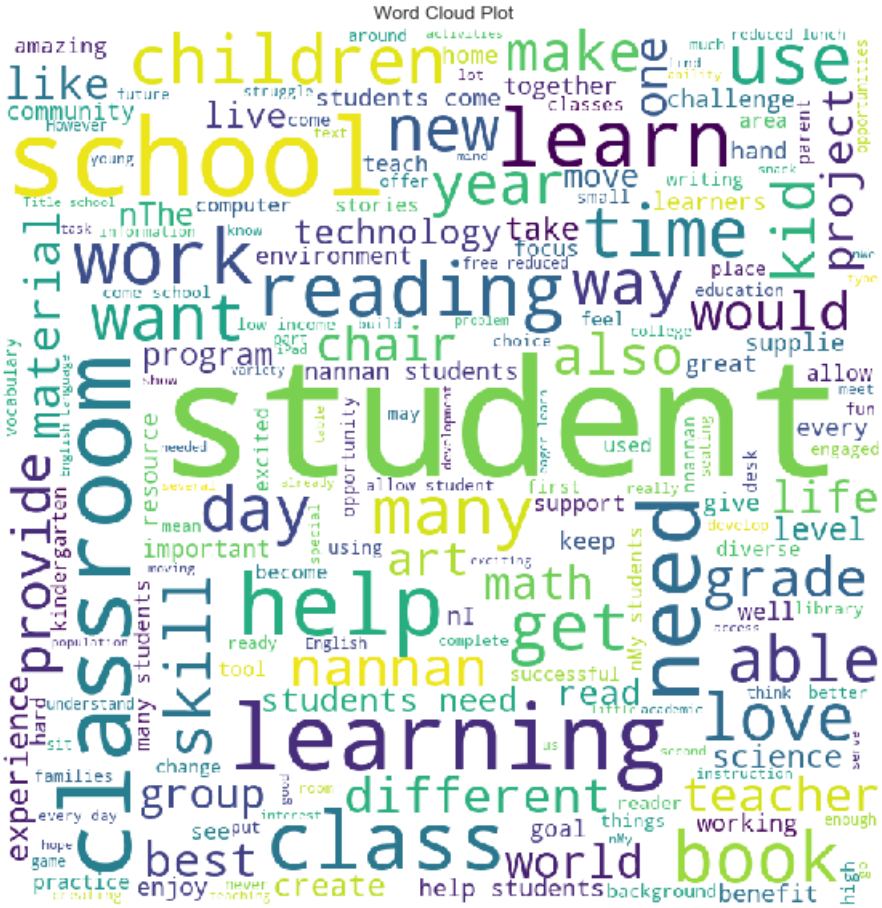
```
cluster_1 = []
cluster_0 = []


for i,l in enumerate(test_predicted):
    if l:
        cluster_1.append(X_test["essay"].iloc[i])
    else:
```

```
        cluster_0.append(X_test["essay"].iloc[i])
```

word plot For +ve class

In [56]:

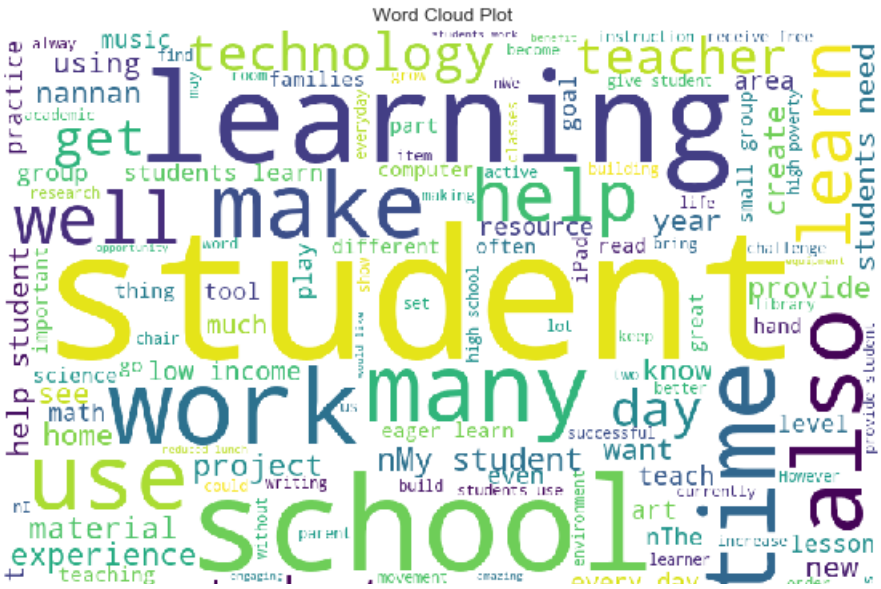
```
Plot_wordcloud(cluster_1)
```

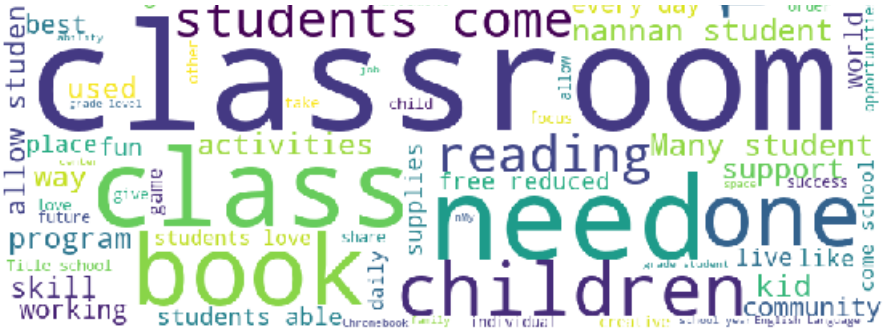


word plot for -ve class

In [57]:

```
Plot_wordcloud(cluster_0)
```





Summary:

- For +ve class student,slassroom,school,learning and reading are some most occuring words.
- For -ve class Studnet,work,need,time are some of the most occuring words.
- Student word occurs in both the word plot.
- Best value of k = 5 found when considered all the data points.
- Two clusters are seperate as seen from above scatter plot.

Set2 - Agglomerative Clustering

In [61]:

```
from sklearn.cluster import AgglomerativeClustering
import numpy as np
```

Agglomerative

Data visiualization for various clusters

- considering clusters from 2 to 5

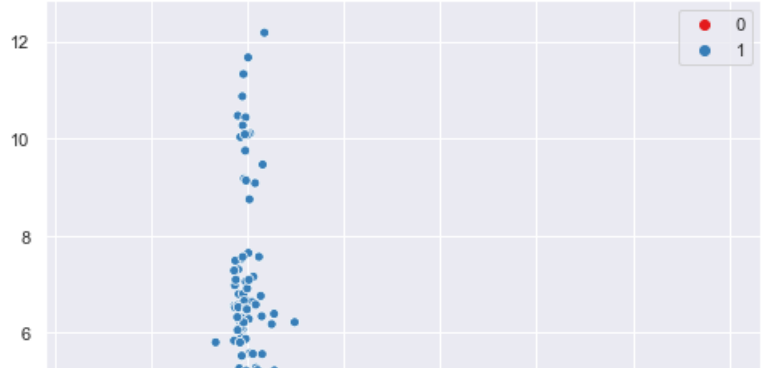
n_clusters = 2

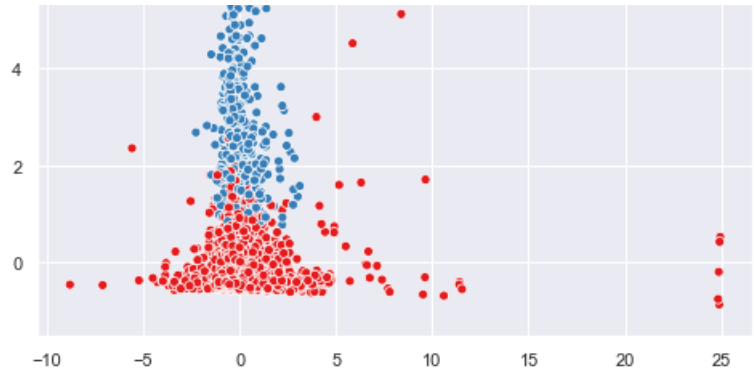
In [62]:

```
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
cluster.fit_predict(set_)

# plotting the cluster
sns.set()
plt.figure(figsize=(8,8))
sns.scatterplot(x=set_v[:,0], y=set_v[:,1],hue=cluster.labels_,palette="Set1")
```

Out[62]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x290b7599b00>
```





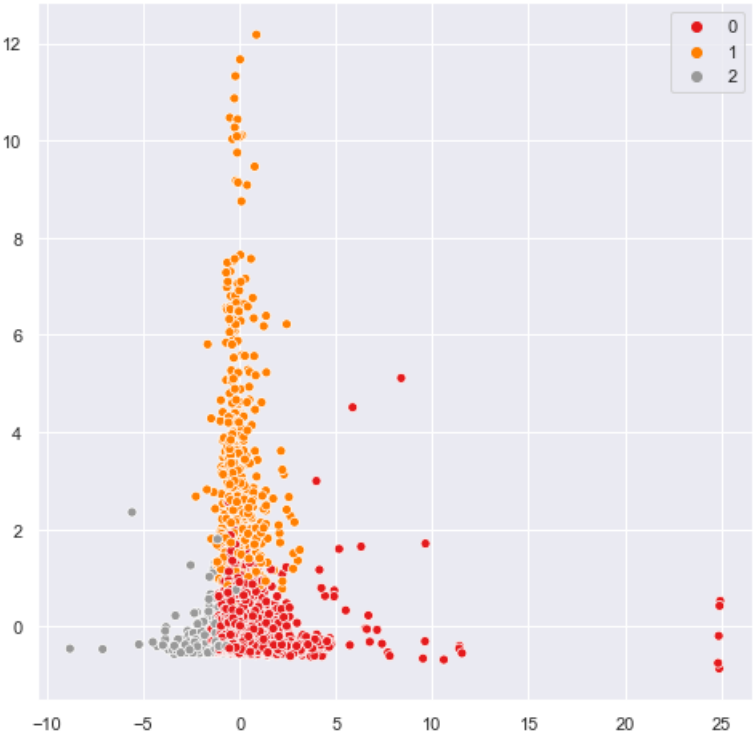
n_clusters = 3

In [63]:

```
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
cluster.fit_predict(set_)
plt.figure(figsize=(8,8))
sns.scatterplot(x=set_v[:,0], y=set_v[:,1],hue=cluster.labels_,palette="Set1")
```

Out[63]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x290a4f000b8>
```



n_clusters = 4

In [64]:

```
cluster = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='ward')
cluster.fit_predict(set_)
plt.figure(figsize=(8,8))
sns.scatterplot(x=set_v[:,0], y=set_v[:,1],hue=cluster.labels_,palette="Set1")
```

Out[64]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x290b4751358>
```

n_clusters = 5

In [65]:

```
cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
cluster.fit_predict(set_)
sns.set()
plt.figure(figsize=(8,8))
sns.scatterplot(x=set_v[:,0], y=set_v[:,1],hue=cluster.labels_,palette="Set1")
```

Out[65]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x290b3f10eb8>
```

World plot fot agglomerative

- as our dataset contains only two classes. we will be considering two clusters.

In [68]:

```
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
cluster.fit_predict(set_t)
```

Out[68]:

```
array([0, 0, 0, ..., 0, 1, 0], dtype=int64)
```

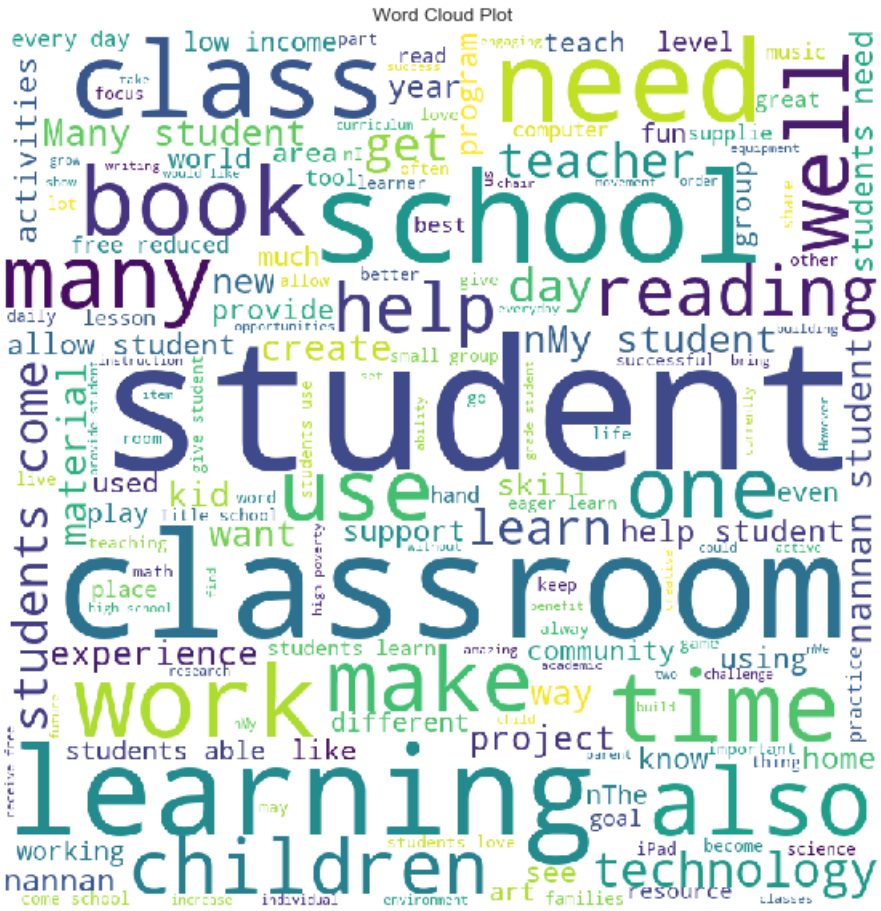
In [69]:

```
cluster_1 = []
cluster_0 = []


for i,l in enumerate(cluster.labels_):
    if l:
        cluster_1.append(X_test["essay"].iloc[i])
    else:
        cluster_0.append(X_test["essay"].iloc[i])
```

-ve data points

In [70]:

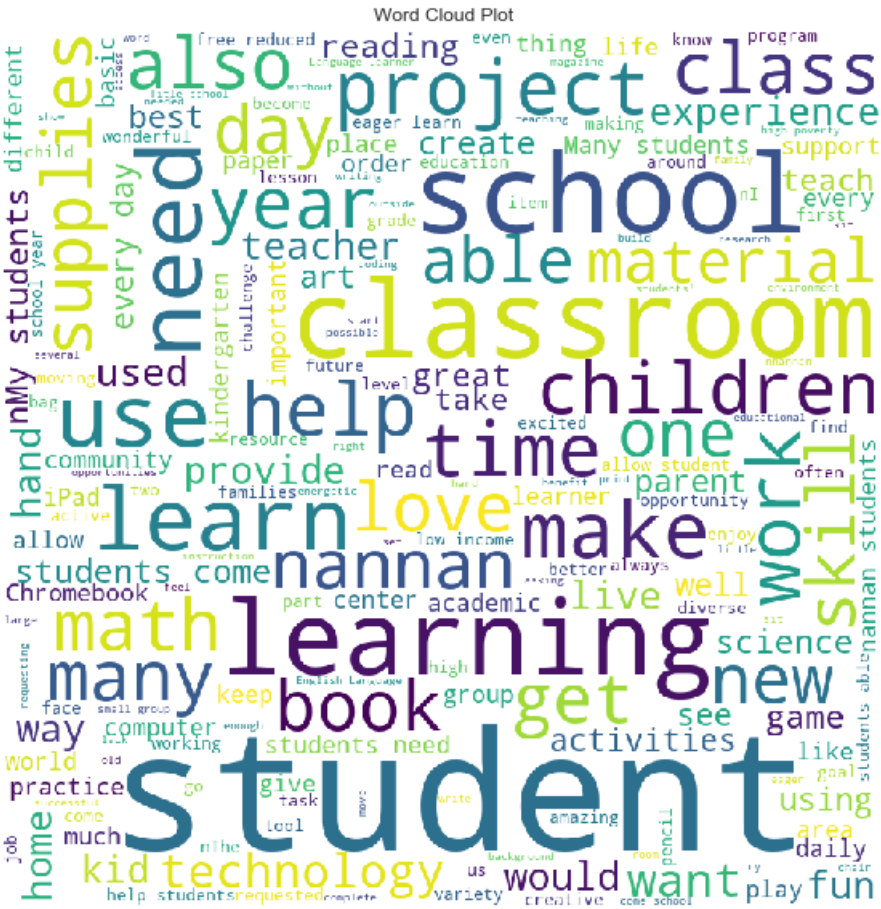
```
Plot_wordcloud(cluster_0)
```



+ve Data points

In [72]:

```
Plot_wordcloud(cluster_1)
```



Summary:

- For +ve cluster, classroom,student,learning are some of the most occurint words.
- For -ve class cluster classroom,student,time,work,need are some of the most occuring words.
- **Clusters are well seprated when n_cluster = 2**
- As n_clusters imcreases, they become in seprable.

Set3 - DBSCAN Clustering

- We are considering **min_pts = 7** as per **ln(n)**
- Where n = number of data points

In [73]:

```
# Distance Function to get distance between two vectors.
# took referance form https://www.python-course.eu/k_nearest_neighbor_classifier.php

def distance(instance1, instance2):
    # just in case, if the instances are lists or tuples:
    instance1 = np.array(instance1)
    instance2 = np.array(instance2)

    return np.linalg.norm((instance1 - instance2),ord = 2)


# get_neighbours function returns distance of kth nearest neighbours.
def get_neighbors(training_set,test_instance, k, distance=distance):
    """
    get_neighors calculates a list of the k nearest neighbors
    of an instance 'test_instance'.
    The list neighbors contains 3-tuples with
    (index, dist, label)
    """
```

```
    distances = []
    for index in range(len(training_set)):
        dist = distance(test_instance, training_set[index])
        distances.append(dist)
    distances.sort()
    neighbors = distances[k]
    return neighbors
```

In [74]:

```
print(distance([3, 5], [1, 1]))
```

```
4.47213595499958
```

In [75]:

```
# iterate over set of all data points and collect distances in eps.
eps = []

for i in range(set_.shape[0]):
    eps.append(get_neighbors(set_,set_[i],7))
```

In [76]:

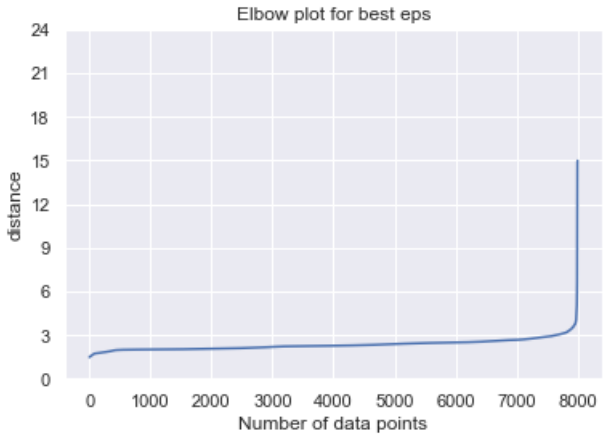
```
sorted_eps = sorted(eps)
```

In [77]:

```
sns.set()
plt.plot(sorted_eps)
plt.xlabel("Number of data points")
plt.ylabel("distance")
plt.title("Elbow plot for best eps")
plt.yticks([x for x in range(0,25,3)])
```

Out[77]:

```
([<matplotlib.axis.YTick at 0x290b7b30a58>,
  <matplotlib.axis.YTick at 0x290b3f109b0>,
  <matplotlib.axis.YTick at 0x290a5a78710>,
  <matplotlib.axis.YTick at 0x290b78a1320>,
  <matplotlib.axis.YTick at 0x290b78a17b8>,
  <matplotlib.axis.YTick at 0x290b78a1c88>,
  <matplotlib.axis.YTick at 0x290b787e208>,
  <matplotlib.axis.YTick at 0x290b787e668>,
  <matplotlib.axis.YTick at 0x290b787eb38>],
 <a list of 9 Text yticklabel objects>)
```



Why best eps is 4....?

- From above we can say that the best eps is 4.

- There is sudden increase in distance ,which represent to noisy points.

In [78]:

```
from sklearn.cluster import DBSCAN
```

In [79]:

```
db = DBSCAN(eps = 4,min_samples=7).fit(set_)
labels = db.labels_
```

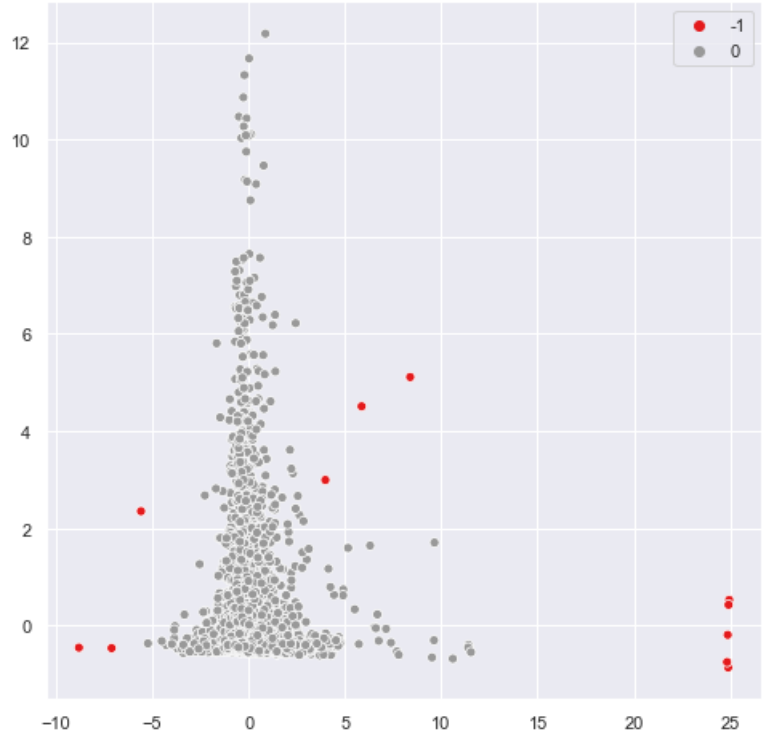
- Let's visiualize data points

In [80]:

```
plt.figure(figsize=(8,8))
sns.scatterplot(x=set_v[:,0], y=set_v[:,1],hue=labels,palette="Set1")
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x290b78aa1d0>
```



Word Cloud

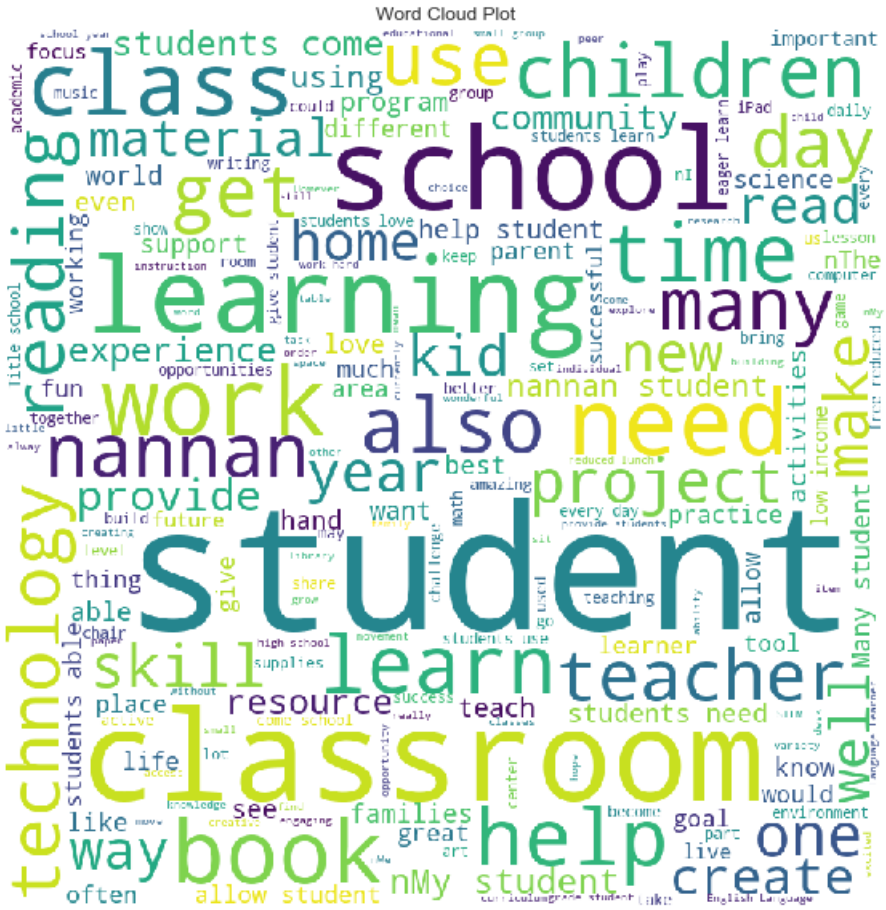
In [82]:

```
cluster_1 = []
cluster_0 = []


for i,l in enumerate(labels):
    if l:
        cluster_1.append(X_train["essay"].iloc[i])
    else:
        cluster_0.append(X_train["essay"].iloc[i])
```

1. For 1st cluster

In [86]:

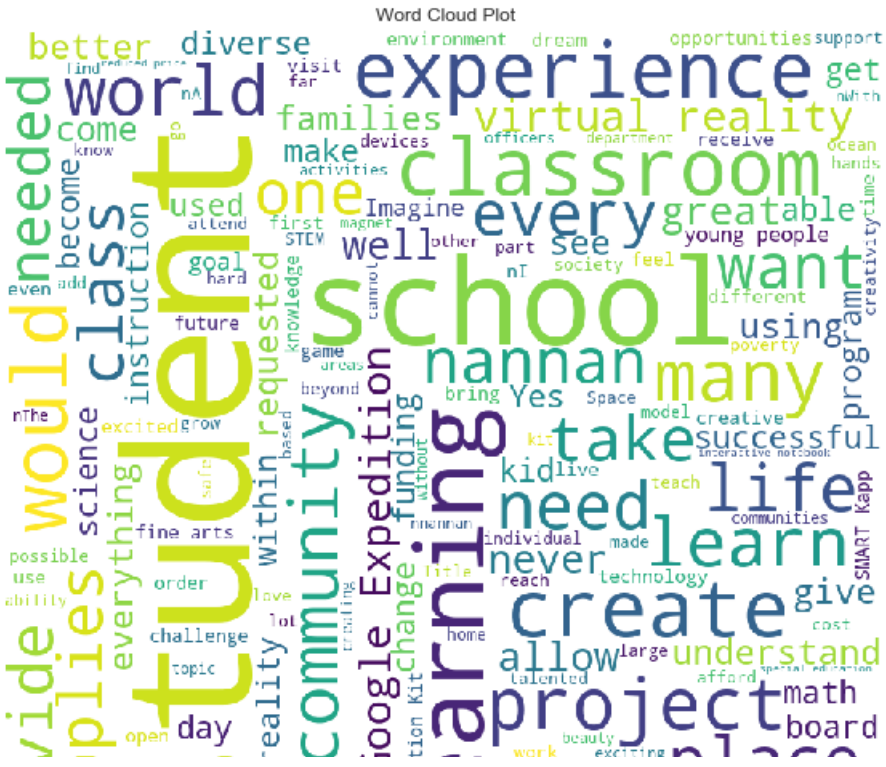
```
Plot_wordcloud(cluster_0[:1000])
```



1. For 2nd cluster

In [83]:

```
Plot_wordcloud(cluster_1)
```



Summary

- DBSCAN gave the most different results.
- DBSCAN assigned most of the data points to cluster 0
- For +ve class student,learning,school,classroom,experience are among the top words in word clous.
- While children,need,school,help are among the top words for -ve class.

In [ ]: