Donors Choose - Clustering

Objective : Predict whether teachers' project proposals are accepted

Importing packages

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

from chart_studio import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

Reading the data

In [2]:

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

Project data

In [3]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("Attributes :", project_data.columns.values)
project_data.head(2)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
Attributes : ['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']
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	

Handling Missing Value in "Teacher prefix" column

In [4]:

```
a = project_data['teacher_prefix'].mode().values
```

In [5]:

```
#Replacin nan with the most frequently occured value in that column
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(a[0])
```

Total number of null values in each column

In [6]:

```
#Total number of null values in each column
project_data.isnull().sum(axis = 0)
```

Out[6]:

```
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
```

Resource data

In [7]:

```
print("Number of data points in train data", resource_data.shape)
print('-'*50)
print("Attributes: ", resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
--------------------------------------------------
Attributes:  ['id' 'description' 'quantity' 'price']
```

Out[7]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

Replacing date-time with date

In [8]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]


project_data.head(2)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00:
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	00:

In [9]:

```
# https://matplotlib.org/gallery/pie_and_polar_charts/pie_and_donut_labels.html#sphx-glr-ga

y_value_counts = project_data['project_is_approved'].value_counts()
print("Number of projects thar are approved for funding ", y_value_counts[1], ", (", (y_val
print("Number of projects thar are not approved for funding ", y_value_counts[0], ", (", (y
```

```
Number of projects thar are approved for funding  92706 , ( 84.8583040421792
7 %)
Number of projects thar are not approved for funding  16542 , ( 15.141695957
820739 %)
```

NOTE: This is an imbalance dataset that containes 85% approved project's data and 15% not approved project's data

Considering 10k data points

In [10]:

```
approved_project=project_data[project_data["project_is_approved"]==1].sample(n=7000,random_
rejected_project=project_data[project_data["project_is_approved"]==0].sample(n=3000,random_
project_data=pd.concat([approved_project,rejected_project])
```

Preprocessing Categorical Data

Project Subject Categories

In [11]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# 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", "
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it w
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
        temp = temp.replace('-','_') # we are replacing - & _
        temp = temp.lower()
    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]))
```

In [12]:

```
print(sorted_cat_dict)
```

```
{'warmth': 125, 'care_hunger': 125, 'history_civics': 524, 'music_arts': 96
1, 'appliedlearning': 1194, 'specialneeds': 1262, 'health_sports': 1273, 'ma
th_science': 3853, 'literacy_language': 4685}
```

Project Subject Sub-Categories

In [13]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# 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", "
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it w
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
        temp = temp.replace('-','_') # we are replacing - & _
        temp = temp.lower()
    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]))
```

In [14]:

```
print(sorted_sub_cat_dict)
```

```
{'economics': 24, 'communityservice': 42, 'financialliteracy': 49, 'civics_g
overnment': 64, 'parentinvolvement': 66, 'extracurricular': 76, 'foreignlang
uages': 91, 'warmth': 125, 'care_hunger': 125, 'nutritioneducation': 130, 'p
erformingarts': 172, 'socialsciences': 182, 'teamsports': 203, 'characteredu
cation': 211, 'other': 250, 'college_careerprep': 259, 'history_geography':
284, 'music': 308, 'esl': 396, 'earlydevelopment': 404, 'health_lifescienc
e': 411, 'gym_fitness': 426, 'environmentalscience': 536, 'visualarts': 576,
'health_wellness': 898, 'appliedsciences': 1054, 'specialneeds': 1262, 'lite
rature_writing': 1990, 'mathematics': 2553, 'literacy': 3005}
```

Teacher Prefix

In [15]:

```
prefix = list(project_data['teacher_prefix'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# 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

prefix_list = []
for i in prefix:
    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", "
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('.','')
        temp = temp.lower()
    prefix_list.append(temp.strip())

project_data['prefix_teacher'] = prefix_list
project_data.drop(['teacher_prefix'], 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['prefix_teacher'].values:
    my_counter.update(word.split())

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

In [16]:

```
print(sorted_prefix_dict)
```

```
{'dr': 2, 'teacher': 213, 'mr': 970, 'ms': 3522, 'mrs': 5293}
```

Project Grade categories

In [17]:

```
grades = list(project_data["project_grade_category"].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# 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

grades_list = []
for i in grades:
    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", "
        j = j.replace(' ','_') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('-','_')
        temp = temp.lower()
    grades_list.append(temp.strip())

project_data['project_grade'] = grades_list
project_data.drop(["project_grade_category"], 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['project_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]))
```

In [18]:

```
print(sorted_grade_dict)
```

```
{'grades_9_12': 1023, 'grades_6_8': 1570, 'grades_3_5': 3407, 'grades_prek_
2': 4000}
```

Preprocessing Text Data

Project Essay

In [19]:

```
# 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)
```

Sentiment Score of project Essay

In [20]:

```
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
essays = project_data['essay']
essays_sentiments = []
for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    essays_sentiments.append(res['compound']) #Considering compound as a criteria.
project_data['essay_sentiment'] = essays_sentiments
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\vansh\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
100%|██████████████████████████████████████████████████████████████████████████████
████| 10000/10000 [00:19<00:00, 513.08it/s]
```

In [21]:

```
# https://stackoverflow.com/a/47091490/4084039
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 [22]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"]


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('nannan', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.lower()
    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())
```

```
100%|████████████████████████████████████████████████████████████████
███| 10000/10000 [00:04<00:00, 2395.24it/s]
```

In [23]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_essays'] = preprocessed_essays
project_data.drop(['essay'], 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 [24]:

```
#Printing random cleaned essay
project_data['clean_essays'].values[23]
```

Out[24]:

```
'students attend high poverty school students receive free breakfast majorit
y free reduced lunch despite socioeconomic backgrounds students high achieve
rs progress throughout school years becomes evident successfully closing soc
ioeconomic achievement gap engaged curious always active students love learn
ing world around meaningful ways build foundation future learning no matter
bad day may going always count little friends bring smile face big books pro
vide materials shared reading book activity sets storage center allow studen
ts participate variety literacy experiences students chance manipulate learn
hands comprehension increases mouse paint allow content integrated across cu
rriculum shared reading affords teachers opportunities teach reading strateg
ies larger group students big books activity sets allow provide instruction
group setting well making materials accessible students use literacy centers
build comprehension allow interact text practicing strategies'
```

Project title

In [25]:

```
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.lower()
    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_titles.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████
██| 10000/10000 [00:00<00:00, 50639.88it/s]
```

In [26]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_titles'] = preprocessed_titles
project_data.drop(['project_title'], axis=1, inplace=True)
```

In [27]:

```
#Printing random cleaned title
project_data['clean_titles'].values[12]
```

Out[27]:

```
'learning outside box'
```

Merging Price and quantity data to Project data (left joining price data)

In [28]:

```
# reference : https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexe
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index
price_data.head(2)
```

Out[28]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [29]:

```
# join two dataframes(project_data and price_data) in python
# reference : https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.m
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

Vectorizing Categorical Data

Clean Categories

In [30]:

```
# we use count vectorizer to convert the values into one hot encoded features

# Vectorizing "clean_categories"
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_sbj = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,

categories_one_hot = vectorizer_sbj.fit_transform(project_data['clean_categories'].values)

print("After verctorizing")
print(categories_one_hot.shape)
```

```
After verctorizing
(10000, 9)
```

Clean sub Categories

In [31]:

```
# Vectorizing "clean_subcategories"
vectorizer_sub_sbj = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase

sub_categories_one_hot = vectorizer_sub_sbj.fit_transform(project_data['clean_subcategories


print("After verctorizing")
print(sub_categories_one_hot.shape)
```

```
After verctorizing
(10000, 30)
```

Teacher Prefix

In [32]:

```
# Vectorizing "teacher_prefix"
vectorizer_teacher = CountVectorizer(vocabulary=list(sorted_prefix_dict.keys()), lowercase=


prefix_one_hot = vectorizer_teacher.fit_transform(project_data['prefix_teacher'])


print("After verctorizing")
print(prefix_one_hot.shape)
```

```
After verctorizing
(10000, 5)
```

school state

In [33]:

```
# Vectorizing "school_state"
from collections import Counter
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())

state_dict = dict(my_counter)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))

vectorizer_state = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=Fal

state_one_hot = vectorizer_state.fit_transform(project_data['school_state'].values)

print("After verctorizing")
print(state_one_hot.shape)
```

```
After verctorizing
(10000, 51)
```

Project Grade Category

In [34]:

```
# Vectorizing "project_grade_category"
vectorizer_grade = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=Fal

grade_one_hot = vectorizer_grade.fit_transform(project_data['project_grade'])


print("After verctorizing")
print(grade_one_hot.shape)
```

```
After verctorizing
(10000, 4)
```

Price

In [35]:

```
from sklearn.preprocessing import StandardScaler

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

# Now standardize the data with above maen and variance.
price_std = price_scalar.transform(project_data['price'].values.reshape(-1, 1))

print(price_std.shape)
```

```
Mean : 304.43609100000003, Standard deviation : 338.94645310548646
(10000, 1)
```

Resource Quantity

In [36]:

```
from sklearn.preprocessing import StandardScaler

quantity_scalar = StandardScaler()
quantity_scalar.fit(project_data['quantity'].values.reshape(-1,1)) # finding the mean and s
print(f"Mean : {quantity_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.va

# Now standardize the data with above maen and variance.
quantity_std = price_scalar.transform(project_data['quantity'].values.reshape(-1, 1))

print(quantity_std.shape)
```

```
Mean : 17.5417, Standard deviation : 27.5599611957274
(10000, 1)
```

Sentiment scores

In [37]:

```
from sklearn.preprocessing import StandardScaler

senti_scalar = StandardScaler()
senti_scalar.fit(project_data['essay_sentiment'].values.reshape(-1,1)) # finding the mean a
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(senti_scalar.var_[0])

# Now standardize the data with above maen and variance.
sentiment_std = price_scalar.transform(project_data['essay_sentiment'].values.reshape(-1, 1

print(sentiment_std.shape)
```

```
Mean : 304.43609100000003, Standard deviation : 0.15195749041864273
(10000, 1)
```

Number of previously posted assignmnets by the teachers

In [38]:

```
from sklearn.preprocessing import StandardScaler

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

# Now standardize the data with above maen and variance.
number_projects_std = nop_scalar.transform(project_data['teacher_number_of_previously_poste

print(number_projects_std.shape)
```

```
Mean : 304.43609100000003, Standard deviation : 25.476402420867828
(10000, 1)
```

TFIDF Text Vectorization

Tfidf on Clean Essay

In [39]:

```
from sklearn.feature_extraction.text import TfidfVectorizer


vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)
vectorizer_tfidf_essay.fit(project_data['clean_essays'].values)

essay_tfidf = vectorizer_tfidf_essay.transform(project_data['clean_essays'])

print("After vectorizing")
print(essay_tfidf.shape)
```

```
After vectorizing
(10000, 6070)
```

Tfidf on Clean Title

In [40]:

```
vectorizer_tfidf_title = TfidfVectorizer(min_df=10)
vectorizer_tfidf_title.fit(project_data['clean_titles'].values)

title_tfidf = vectorizer_tfidf_title.transform(project_data['clean_titles'])

print("After vectorizing")
print(title_tfidf.shape)
```

```
After vectorizing
(10000, 612)
```

In [41]:

```
essay_features = (vectorizer_tfidf_essay.get_feature_names(), essay_tfidf.toarray())
```

Function for WordCloud

In [42]:

```
#https://www.geeksforgeeks.org/python-split-dictionary-of-lists-to-list-of-dictionaries/

from wordcloud import WordCloud

def getWordCorpus(essay_features, y_kmeans):
    features, encodings = essay_features  #get feature names and their encodings in tfidf
    encodings_cols = encodings.shape[1]  #number of encodings
    words = {}
    i = 0
    for each_x in tqdm(y_kmeans):
        if each_x not in words: words[each_x] = ''
        for j in range(encodings_cols):  #iterating to each word encoding
            if encodings[i][j] >= 0.5:    #taking features that has tfidf val>=0.5
                words[each_x] = "%s %s"%(words[each_x], features[j].strip())  #addind words
        i += 1

    return words

def getWordCloud(word_corpus, num):
    wordcloud = WordCloud(width = 800, height = 800,
                    background_color ='white',
                    stopwords = stopwords,
                    collocations = False,
                    min_font_size = 10).generate(word_corpus)

    print("Number of words", len(word_corpus)) #printing number of words in word corpus

    # plot the WordCloud image
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.title('The word cloud with essay text: Cluster %s'%(num))
    plt.tight_layout(pad = 0)
    plt.show()
```

K-means Clustering

Hstacking features

In [43]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, essay_tfidf, quantity_std, state_or
y = project_data['project_is_approved']
print('Final matrix')
print(X.shape, y.shape)
```

```
Final matrix
(10000, 6785) (10000,)
```

Selecting 5000 best features using Feature Importance

In [44]:

```
from sklearn.feature_selection import SelectKBest, f_classif

sk = SelectKBest(f_classif, k=5000)
X = sk.fit_transform(X,y)

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

```
(10000, 5000) (10000,)
```

Hyperparameter tuning : Findind the best 'k' - Elbow / keen method

Inertia: It is the sum of squared distances of samples to their closest cluster center.

In [45]:

```
#https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/
#https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist
from sklearn.metrics import silhouette_samples, silhouette_score

inertias = []

K = [2,3,4,5,6,7,8,9,10]

for k in (K):
    #Building and fitting the model
    kmeans = KMeans(n_clusters=k, random_state=10)
    kmeans_labels = kmeans.fit_predict(X)


    silhouette_avg = silhouette_score(X, kmeans_labels)
    print("For n_clusters =", k,
          "The average silhouette_score is :", silhouette_avg)

    inertias.append(kmeans.inertia_)



plt.plot(K, inertias, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Inertia')
plt.title('The Elbow Method using inertia')
plt.show()
```

```
For n_clusters = 2 The average silhouette_score is : 0.36455925496647235
For n_clusters = 3 The average silhouette_score is : 0.1990315241512174
For n_clusters = 4 The average silhouette_score is : 0.08330703140000002
For n_clusters = 5 The average silhouette_score is : 0.08473846913523177
For n_clusters = 6 The average silhouette_score is : 0.08642238148781482
For n_clusters = 7 The average silhouette_score is : 0.0813214295833443
For n_clusters = 8 The average silhouette_score is : 0.07017465441916733
For n_clusters = 9 The average silhouette_score is : 0.07276995912913005
For n_clusters = 10 The average silhouette_score is : 0.06479078761524934
```

The Elbow Method using inertia

Silhouette score for each n_clusters

In [46]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.title = "K Means - Silhouette Score of each n_Clusters"
x.field_names = ["N_clusters","Avg Silhouette value"]

x.add_row(["2","0.364"])
x.add_row(["3","0.199"])
x.add_row(["4","0.083"])
x.add_row(["5","0.084"])
x.add_row(["6","0.086"])
x.add_row(["7","0.081"])
x.add_row(["8","0.070"])
x.add_row(["9","0.072"])
x.add_row(["10","0.064"])


print(x)
```

```
+------------+----------------------+
| N_clusters | Avg Silhouette value |
+------------+----------------------+
|     2      |        0.364         |
|     3      |        0.199         |
|     4      |        0.083         |
|     5      |        0.084         |
|     6      |        0.086         |
|     7      |        0.081         |
|     8      |        0.070         |
|     9      |        0.072         |
|     10     |        0.064         |
+------------+----------------------+
```

In [47]:

```
#best k = 2
```

Training and predicting with best K value

In [48]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, init='k-means++', n_init=10, max_iter=300,
                tol=0.0001, precompute_distances='auto').fit(X)


print("Clusters center", kmeans.cluster_centers_)
```

```
Clusters center [[ 4.71464020e-02  4.71464020e-02  5.95533499e-02 ...  4.220
84086e-02
   3.96203614e+00 -8.95306657e-01]
 [ 1.10451183e-02  1.10451183e-02  5.20996145e-02 ... -1.77242770e-03
  -1.66374968e-01 -8.95353963e-01]]
```

Top 10 terms for each cluster

In [50]:

```
#https://pythonprogramminglanguage.com/kmeans-text-clustering/
from __future__ import print_function

print("Top terms per cluster:")
order_centroids = kmeans.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer_tfidf_essay.get_feature_names()
for i in range(2):
    print("Cluster %d:" % i)
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind])
    print('-'*50)
```

```
Top terms per cluster:
Cluster 0:
 simplest
 relieve
 reluctant
 11
 110
 relies
 relish
 relevant
 22
 10th
--------------------------------------------------
Cluster 1:
 relieve
 110
 reluctant
 11
 relies
 relish
 22
 21st
 21
 religions
--------------------------------------------------
```

Word Cloud for each cluster

In [51]:

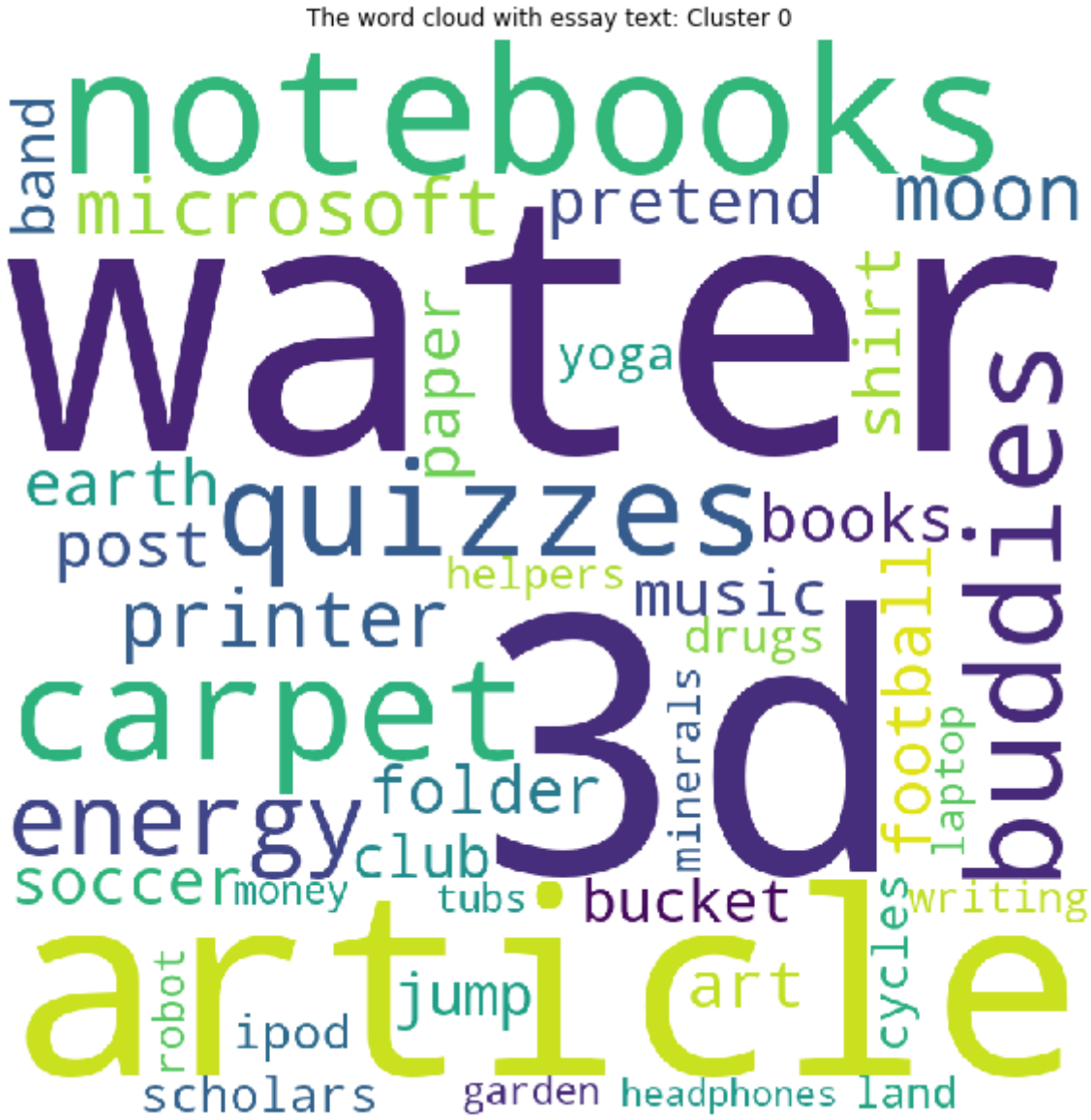
```
word_corpus = getWordCorpus(essay_features, kmeans.labels_)
word_corpus = dict(sorted(list(word_corpus.items()), key=lambda x: x[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 10000/10000 [00:18<00:00, 533.62it/s]
```

In [52]:

```
getWordCloud(word_corpus.get(0), 0)
```

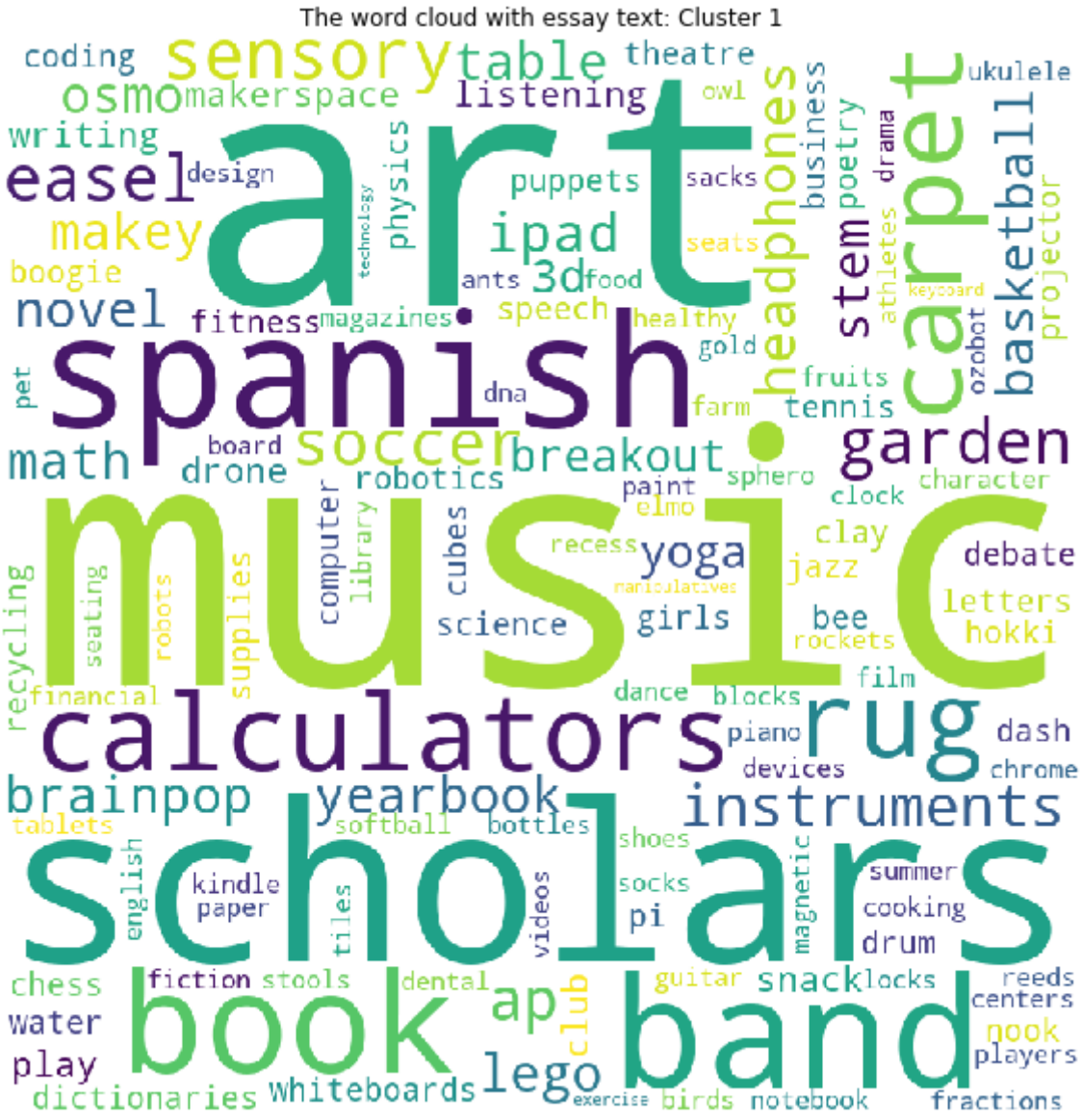
```
Number of words 285
```



In [53]:

```
getWordCloud(word_corpus.get(1), 1)
```

```
Number of words 5591
```



In [54]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.title = "Conclusion for Kmeans Clustering - Optimal number of clusters obtained - 3"
x.field_names = ["Cluster", "Number of words in cluster","Most frequently occured words"]

x.add_row(["Cluster 0",285,"article,3d,water,notebooks,quizzes"])
x.add_row(["Cluster 1",5591,"art,music,scholars,art,spanish,calculator,book,band"])

print(x)
```

```
+-----------+---------------------------+---------------------------------
------------------+
|  Cluster  | Number of words in cluster |            Most frequently occure
d words            |
+-----------+---------------------------+---------------------------------
------------------+
| Cluster 0 |            285            |          article,3d,water,noteboo
ks,quizzes         |
| Cluster 1 |            5591           | art,music,scholars,art,spanish,ca
lculator,book,band |
+-----------+---------------------------+---------------------------------
------------------+
```

Obeservation:

1. Cluster 0 is the smaller of all the 4 clusters.
2. Cluster 1 in the bigger of all clusters.
3. Most frequently occured values of all the clusters are different that means clusters can be thought as well separated form each other.

Agglomerative clustering - 2,3,4 and 5 clusters

Considering 5K datapoints

In [55]:

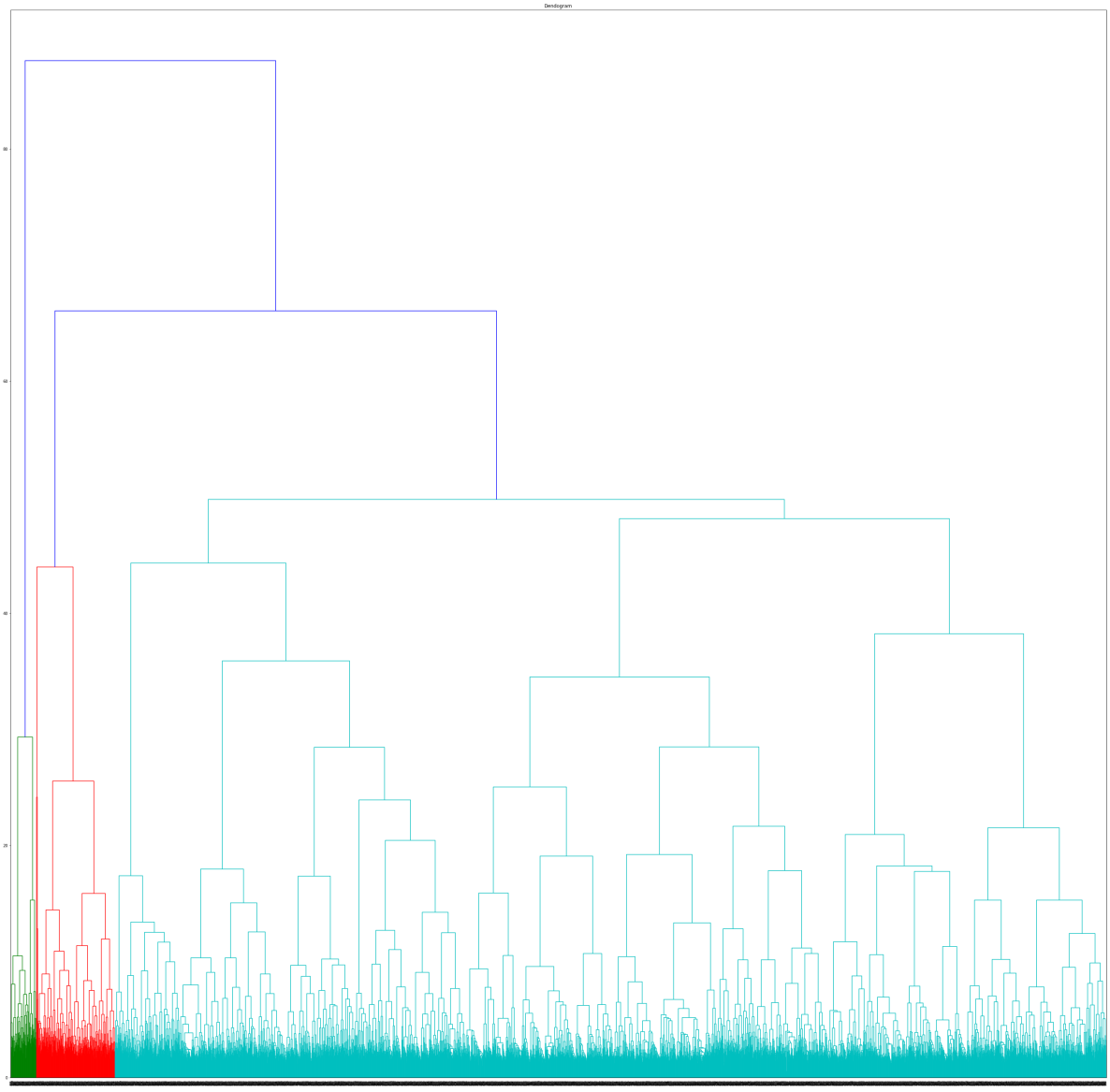
```
X = X[0:5000]

print('Final matrix')
print(X.shape, y.shape)
X = X.todense()[:5000]
```

```
Final matrix
(5000, 5000) (10000,)
```

Plotting Dendogram

In [56]:

```
#http://brandonrose.org/clustering
import scipy.cluster.hierarchy as shc

plt.figure(figsize=(50, 50))
plt.title("Dendogram")
dend = shc.dendrogram(shc.linkage(X, method='ward'))
```



Take away from dendogram:

If we draw a horizontal line crossing the largest veritical line in the dendogram that doesn't collide with any horizontal line, then number of vertical lines it crosses gives the optimal number of clusters. Therefore optimal number of clusers = 3

Agglomertive clustering - Silhouette score for each n_clusters

In [58]:

```
from sklearn.cluster import AgglomerativeClustering

K = [2,3,4,5,6,7,8,9,10]

for k in (K):
    #Building and fitting the model
    agglo = AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                            connectivity=None, distance_threshold=None,
                            linkage='ward', memory=None, n_clusters=k,
                            pooling_func='deprecated').fit(X)

    agglo_labels = agglo.fit_predict(X)


    silhouette_avg = silhouette_score(X, agglo_labels)
    print("For n_clusters =", k,
          "The average silhouette_score is :", silhouette_avg)
```

```
For n_clusters = 2 The average silhouette_score is : 0.4742097097030332
For n_clusters = 3 The average silhouette_score is : 0.21838618435451498
For n_clusters = 4 The average silhouette_score is : 0.06564713086729232
For n_clusters = 5 The average silhouette_score is : 0.05023166345908125
For n_clusters = 6 The average silhouette_score is : 0.05747479257648393
For n_clusters = 7 The average silhouette_score is : 0.06030736876612707
For n_clusters = 8 The average silhouette_score is : 0.05132722834396347
For n_clusters = 9 The average silhouette_score is : 0.060239506261618524
For n_clusters = 10 The average silhouette_score is : 0.05407839301139477
```

In [59]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.title = "Agglomerative clustering - Silhouette Score of each n_Clusters"
x.field_names = ["N_clusters","Avg Silhouette value"]

x.add_row(["2","0.474"])
x.add_row(["3","0.218"])
x.add_row(["4","0.065"])
x.add_row(["5","0.050"])
x.add_row(["6","0.057"])
x.add_row(["7","0.060"])
x.add_row(["8","0.051"])
x.add_row(["9","0.060"])
x.add_row(["10","0.054"])


print(x)
```

```
+------------+----------------------+
| N_clusters | Avg Silhouette value |
+------------+----------------------+
|     2      |        0.474         |
|     3      |        0.218         |
|     4      |        0.065         |
|     5      |        0.050         |
|     6      |        0.057         |
|     7      |        0.060         |
|     8      |        0.051         |
|     9      |        0.060         |
|     10     |        0.054         |
+------------+----------------------+
```

In [60]:

```
#best k = 2
```

Agglomertive clustering (2 Clusters)

In [61]:

```
agglo = AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                                connectivity=None, distance_threshold=None,
                                linkage='ward', memory=None, n_clusters= 2 ,
                                pooling_func='deprecated').fit(X)
```

Word Cloud for Agglomertive clustering - 2 Clusters

In [62]:

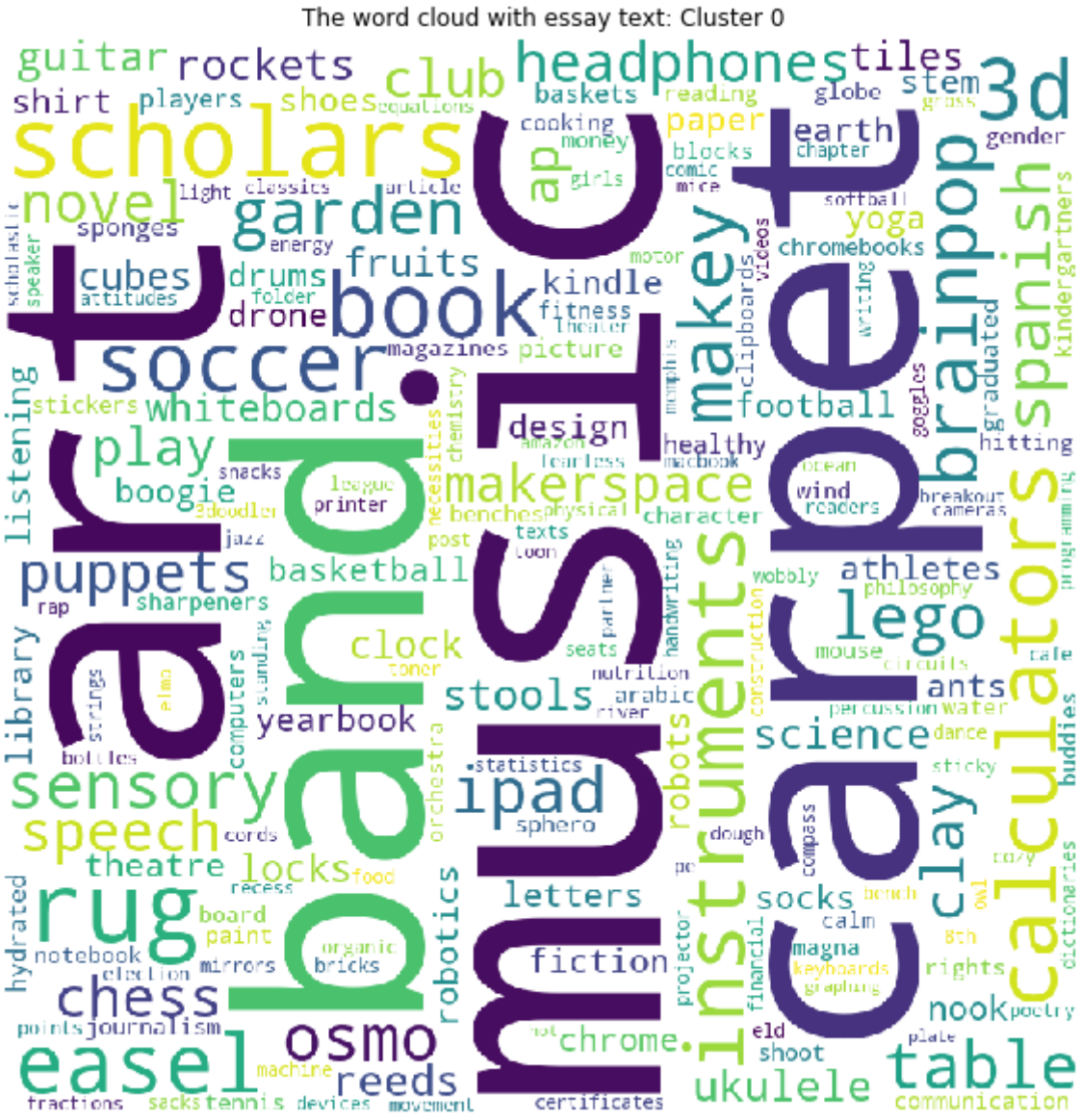
```
word_corpus = getWordCorpus(essay_features, agglo.labels_)
word_corpus = dict(sorted(list(word_corpus.items()), key=lambda x: x[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 5000/5000 [00:09<00:00, 519.72it/s]
```

In [63]:

```
getWordCloud(word_corpus.get(0), 0)
```

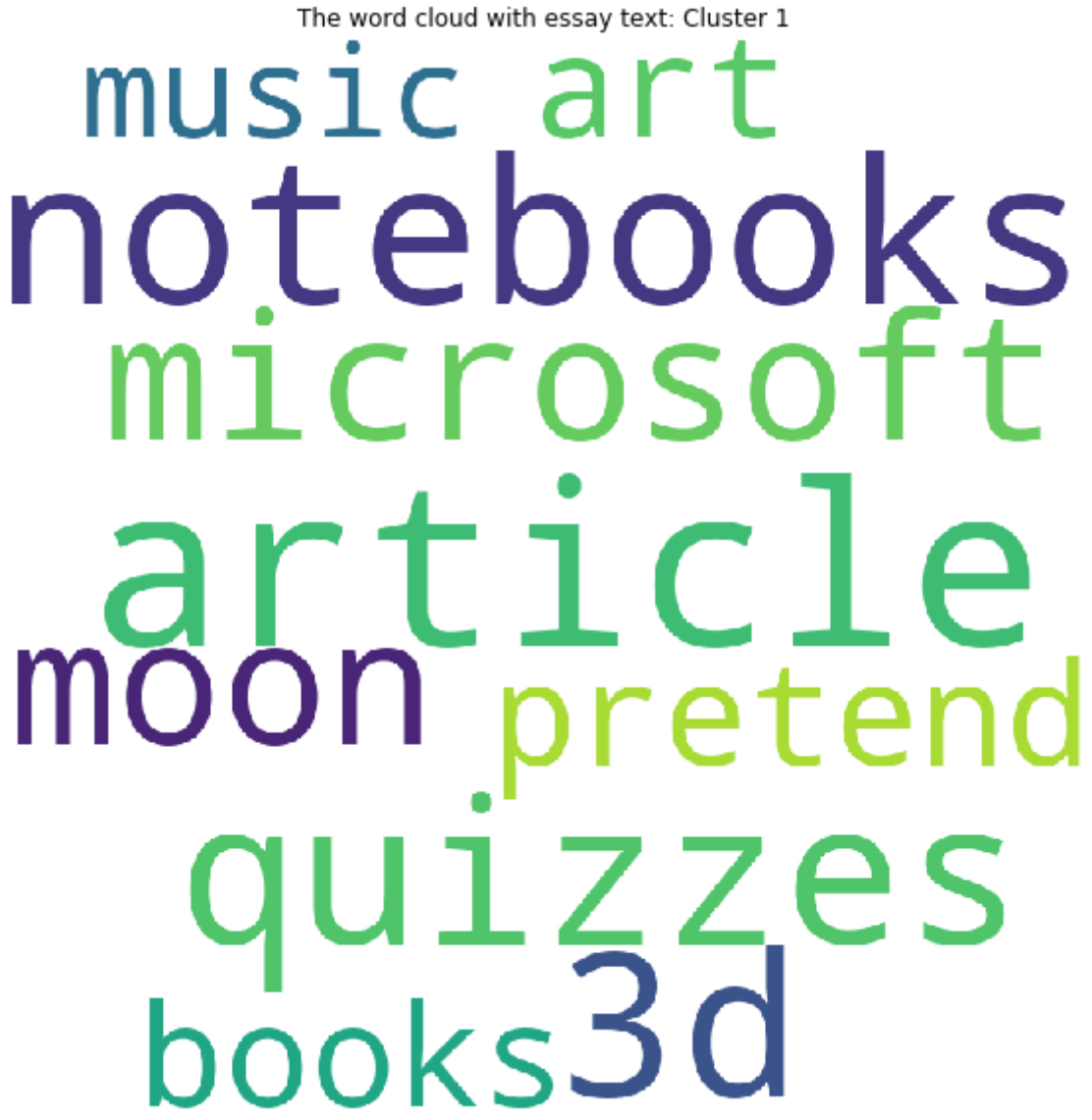
```
Number of words 2852
```



In [64]:

```
getWordCloud(word_corpus.get(1), 1)
```

```
Number of words 68
```



In [66]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.title = "Conclusion for Agglomerative Clustering with 3 Clusters"
x.field_names = ["Cluster", "Number of words in cluster","Most frequently occured words"]

x.add_row(["Cluster 0",2852,"carpet,music,art,band"])
x.add_row(["Cluster 1",68,"microsoft,pretend,music,notebook,quizzes,moon,3d,article"])


print(x)
```

```
+-----------+---------------------------+---------------------------------
------------------------+
|  Cluster  | Number of words in cluster |              Most frequently occu
red words               |
+-----------+---------------------------+---------------------------------
------------------------+
| Cluster 0 |            2852           |                  carpet,music,ar
t,band                  |
| Cluster 1 |             68            | microsoft,pretend,music,notebook,
quizzes,moon,3d,article |
+-----------+---------------------------+---------------------------------
------------------------+
```

Conclusion:

1. Cluster 0 is the small cluster.
2. Cluster 1 is the bigger cluster.
3. Clusters can be thought of well separated as there as very less commonly ocuured frequent words.

Agglomertive clustering (3 Clusters)

In [67]:

```
from sklearn.cluster import AgglomerativeClustering

agglo = AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                                connectivity=None, distance_threshold=None,
                                linkage='ward', memory=None, n_clusters=3,
                                pooling_func='deprecated').fit(X)

agglo.labels_
```

Out[67]:

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

Word Cloud for Agglomertive clustering - 3 Clusters

In [68]:

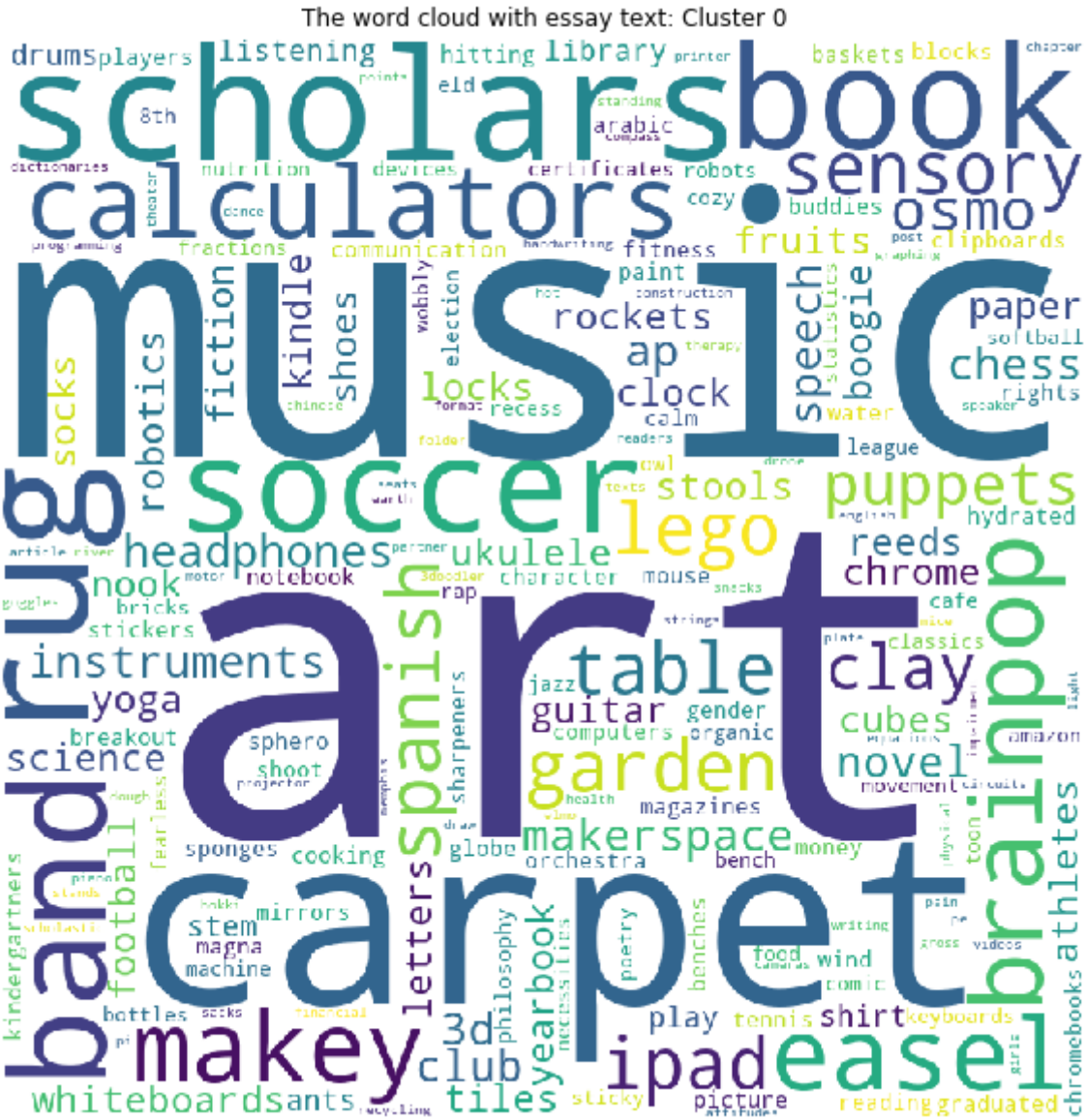
```
word_corpus = getWordCorpus(essay_features, agglo.labels_)
word_corpus = dict(sorted(list(word_corpus.items()), key=lambda x: x[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 5000/5000 [00:09<00:00, 520.08it/s]
```

In [69]:

```
getWordCloud(word_corpus.get(0), 0)
```

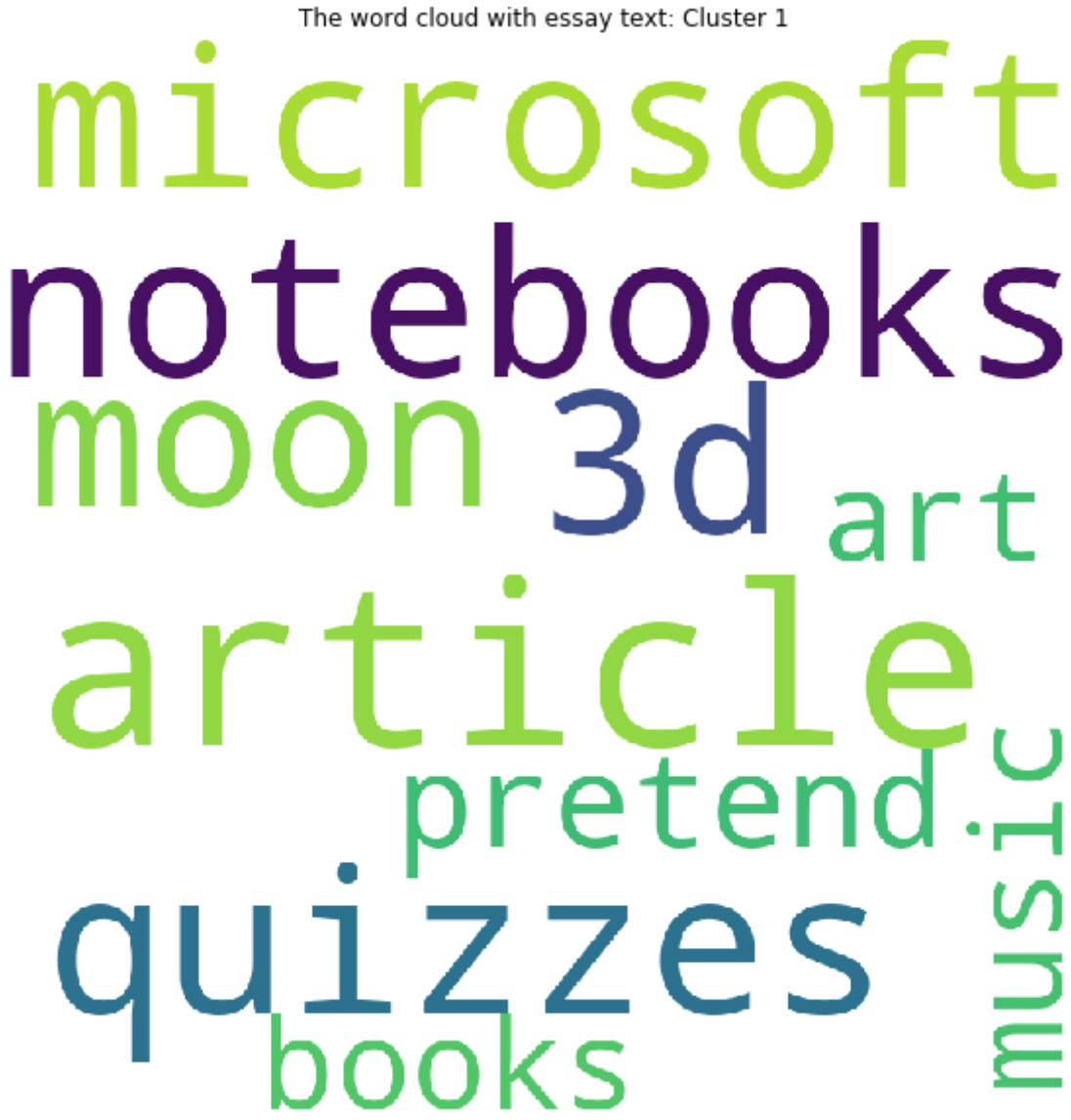
```
Number of words 2608
```



In [70]:

```
getWordCloud(word_corpus.get(1), 1)
```

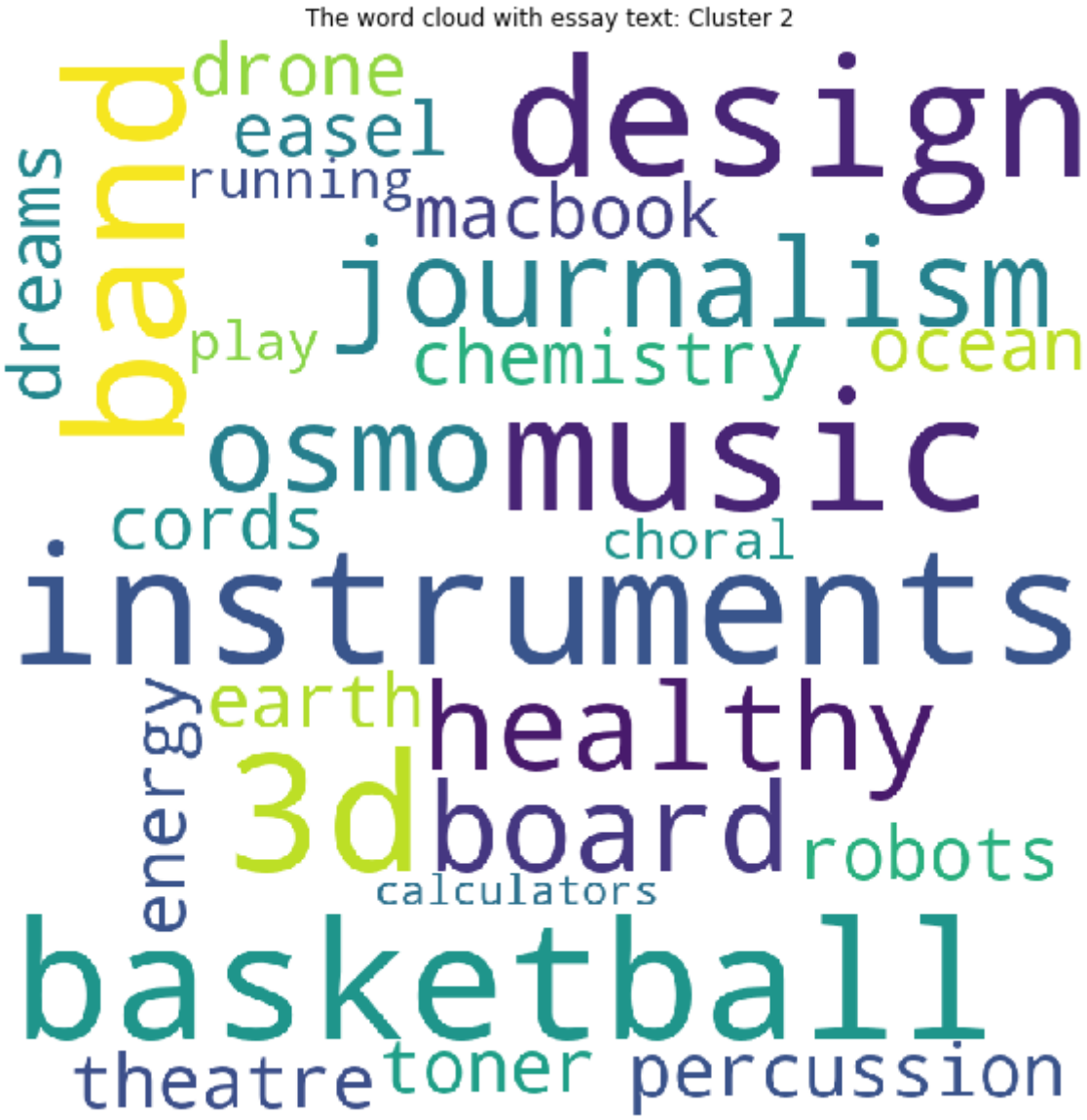
```
Number of words 68
```



In [71]:

```
getWordCloud(word_corpus.get(2), 2)
```

```
Number of words 244
```



In [73]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.title = "Conclusion for Agglomerative Clustering with 3 Clusters"
x.field_names = ["Cluster", "Number of words in cluster","Most frequently occured words"]

x.add_row(["Cluster 0",2608,"carpet,music,art,scholars,rug,book,band"])
x.add_row(["Cluster 1",68,"microsoft,notebook,quizzes,moon,3d,article"])
x.add_row(["Cluster 2",244,"instruments,basketball,music,design,band,3d,journalism,healthy,

print(x)
```

```
+-----------+---------------------------+-----------------------------------
-------------------------------------+
|  Cluster  | Number of words in cluster |                    Most frequentl
y occured words                      |
+-----------+---------------------------+-----------------------------------
-------------------------------------+
| Cluster 0 |            2608           |               carpet,music,art,sc
holars,rug,book,band                 |
| Cluster 1 |             68            |              microsoft,notebook,q
uizzes,moon,3d,article               |
| Cluster 2 |            244            | instruments,basketball,music,desi
gn,band,3d,journalism,healthy,board |
+-----------+---------------------------+-----------------------------------
-------------------------------------+
```

Conclusion:

1. Cluster 0 is the biggest cluster.
2. Cluster 1 is the smalles cluster.
3. Clusters can be thought of well separated as there as very less commonly ocuured frequent words.

Agglomertive clustering (4 Clusters)

In [74]:

```
from sklearn.cluster import AgglomerativeClustering

agglo = AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                                connectivity=None, distance_threshold=None,
                                linkage='ward', memory=None, n_clusters=4,
                                pooling_func='deprecated').fit(X)

agglo.labels_
```

Out[74]:

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

Word Cloud for 4 clusters

In [75]:

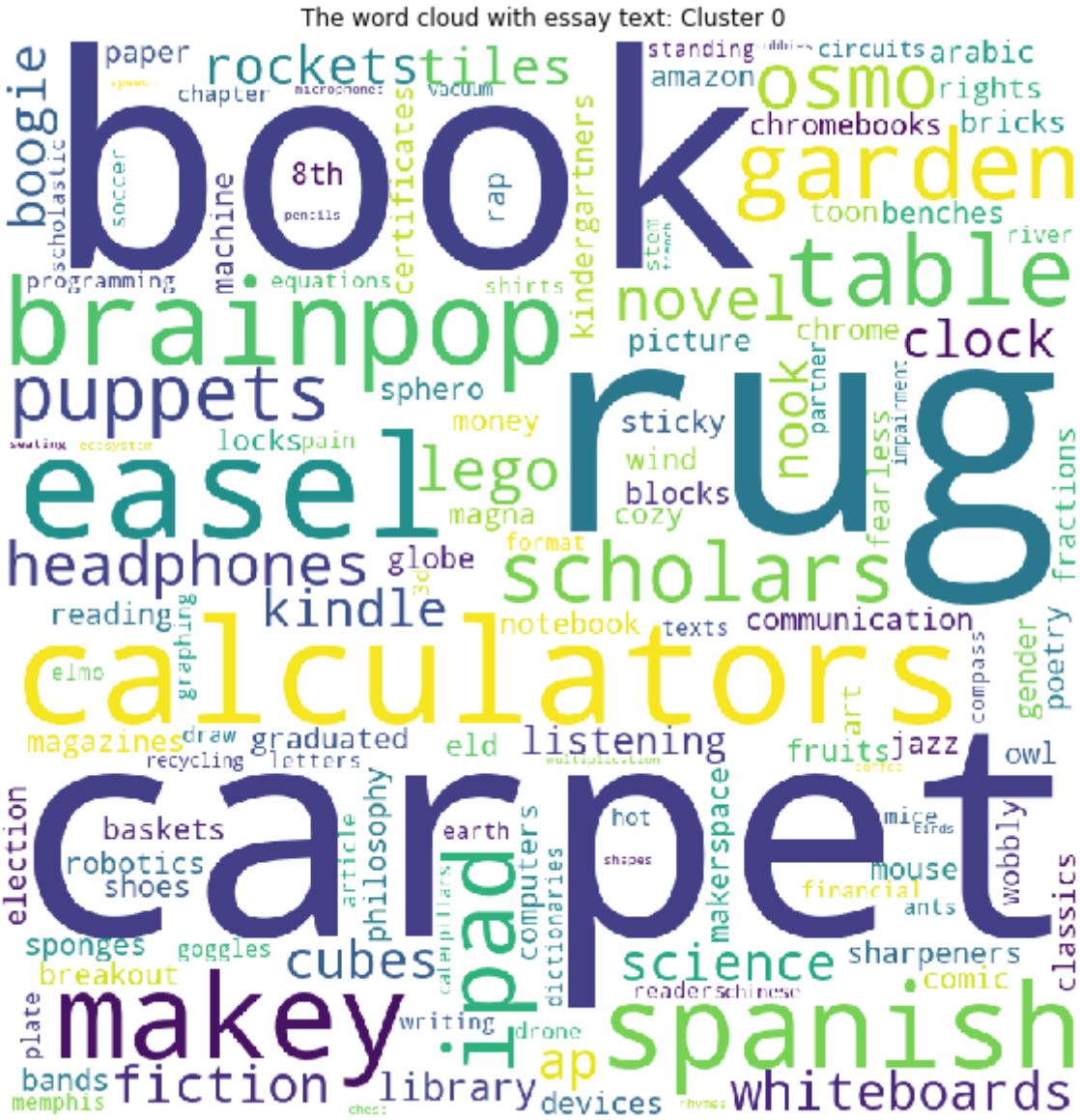
```
word_corpus = getWordCorpus(essay_features, agglo.labels_)
word_corpus = dict(sorted(list(word_corpus.items()), key=lambda x: x[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 5000/5000 [00:09<00:00, 521.20it/s]
```

In [76]:

```
getWordCloud(word_corpus.get(0), 0)
```

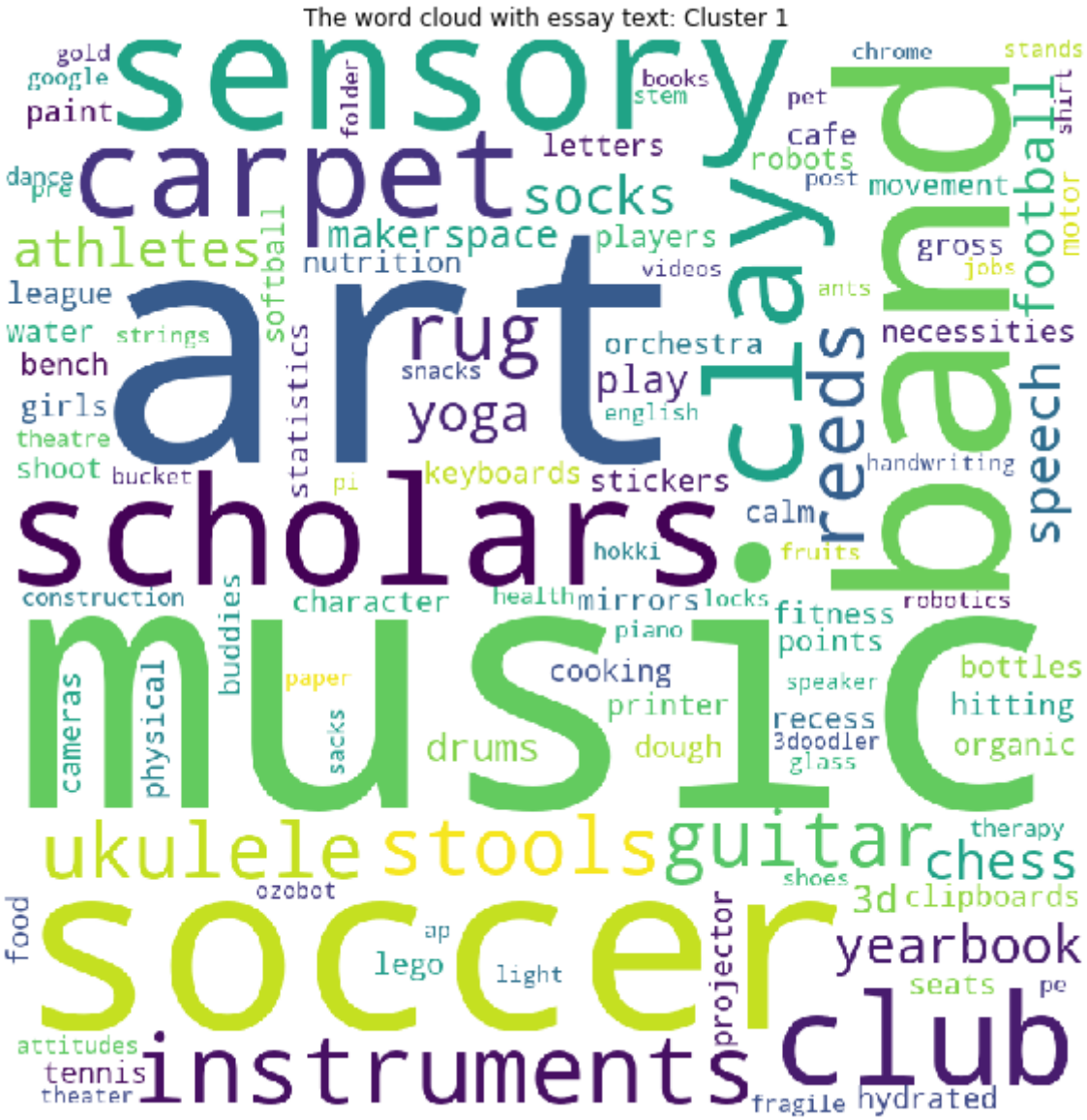
```
Number of words 1469
```



In [77]:

```
getWordCloud(word_corpus.get(1), 1)
```

Number of words 1139



In [78]:

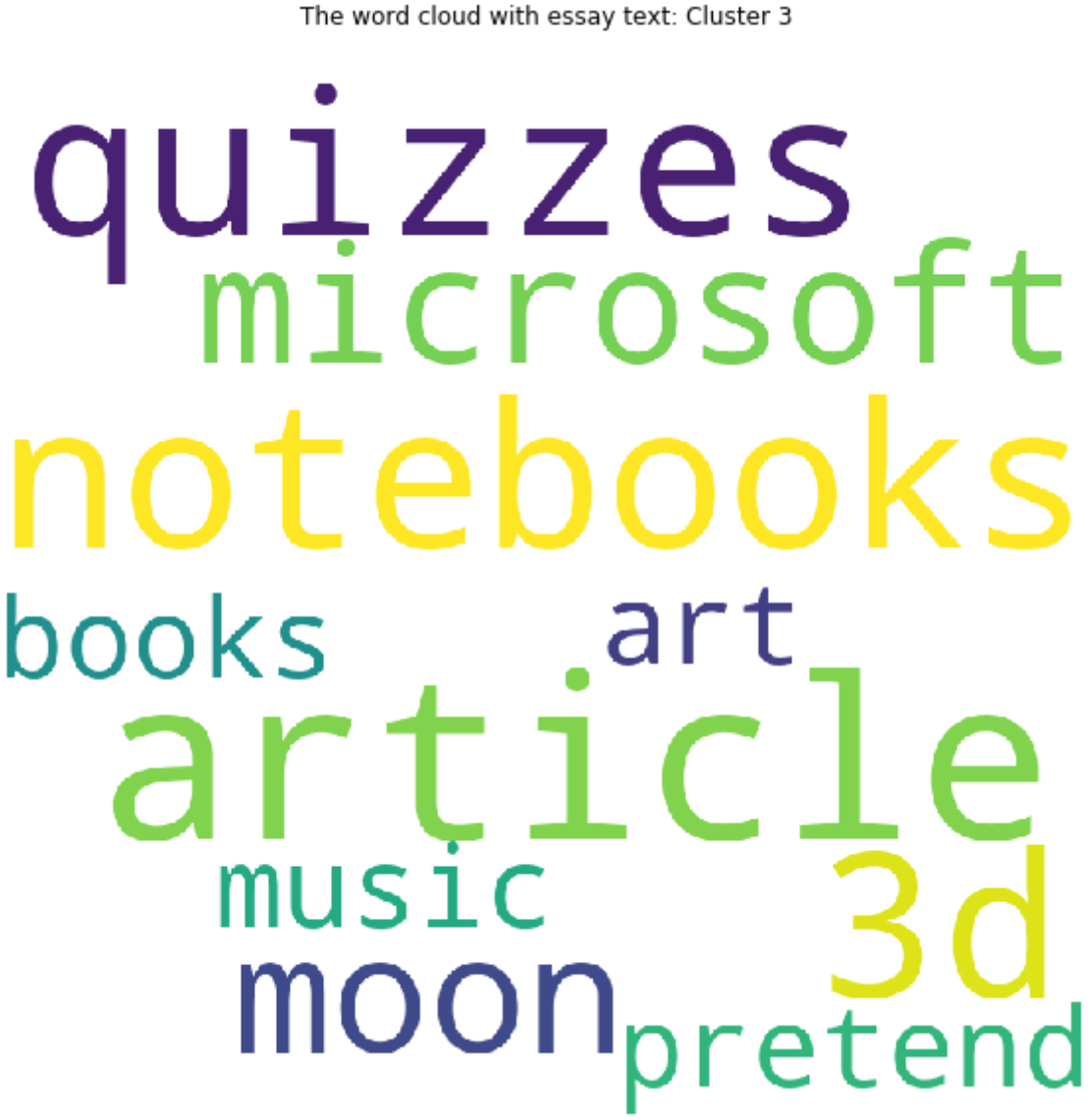
```
getWordCloud(word_corpus.get(2), 2)
```

```
Number of words 244
```

In [79]:

```
getWordCloud(word_corpus.get(3), 3)
```

```
Number of words 68
```



In [80]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.title = "Conclusion for Agglomerative Clustering with 4 Clusters"
x.field_names = ["Cluster", "Number of words in cluster","Most frequently occured words"]

x.add_row(["Cluster 0",1469,"carpet,calculator,easel,brainpop,book,rug,makey"])
x.add_row(["Cluster 1",1139,"art,music,soccor,band"])
x.add_row(["Cluster 2",244,"instruments,basketball,music,design,band,3d,journalism,healthy,
x.add_row(["Cluster 3",68,"microsoft,notebook,quizzes,moon,3d,article"])

print(x)
```

```
+-----------+---------------------------+---------------------------------
-----------------------------------+
|  Cluster  | Number of words in cluster |                    Most frequentl
y occured words                     |
+-----------+---------------------------+---------------------------------
-----------------------------------+
| Cluster 0 |            1469           |           carpet,calculator,ease
l,brainpop,book,rug,makey            |
| Cluster 1 |            1139           |                        art,music,
soccor,band                         |
| Cluster 2 |            244            | instruments,basketball,music,desi
gn,band,3d,journalism,healthy,board |
| Cluster 3 |             68            |              microsoft,notebook,q
uizzes,moon,3d,article              |
+-----------+---------------------------+---------------------------------
-----------------------------------+
```

Conclusion:

1. Cluster 0 is the smallest cluster.
2. Cluster 1 and 2 are relatively bigger clusters.
3. Clusters can be thought of well separated as there as very less commonly ocuured frequent words.

Agglomertive clustering (5 Clusters)

In [81]:

```
from sklearn.cluster import AgglomerativeClustering

agglo = AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                                connectivity=None, distance_threshold=None,
                                linkage='ward', memory=None, n_clusters=5,
                                pooling_func='deprecated').fit(X)

agglo.labels_
```

Out[81]:

```
array([4, 0, 0, ..., 4, 0, 1], dtype=int64)
```

Word Cloud for 5 clusters

In [82]:

```
word_corpus = getWordCorpus(essay_features, agglo.labels_)
word_corpus = dict(sorted(list(word_corpus.items()), key=lambda x: x[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 5000/5000 [00:09<00:00, 513.86it/s]
```

In [83]:

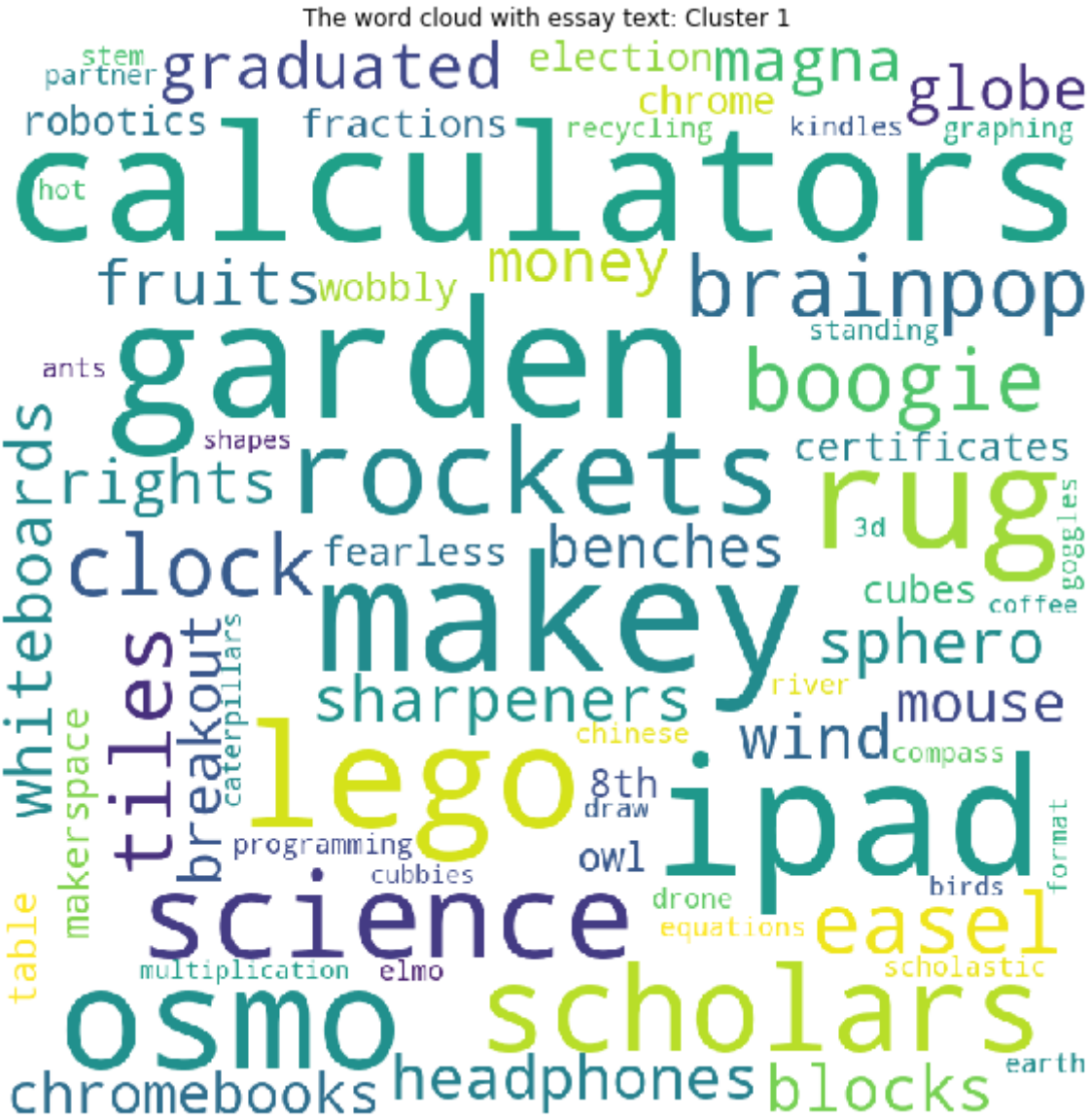
```
getWordCloud(word_corpus.get(0), 0)
```

```
Number of words 1139
```

In [84]:

```
getWordCloud(word_corpus.get(1), 1)
```

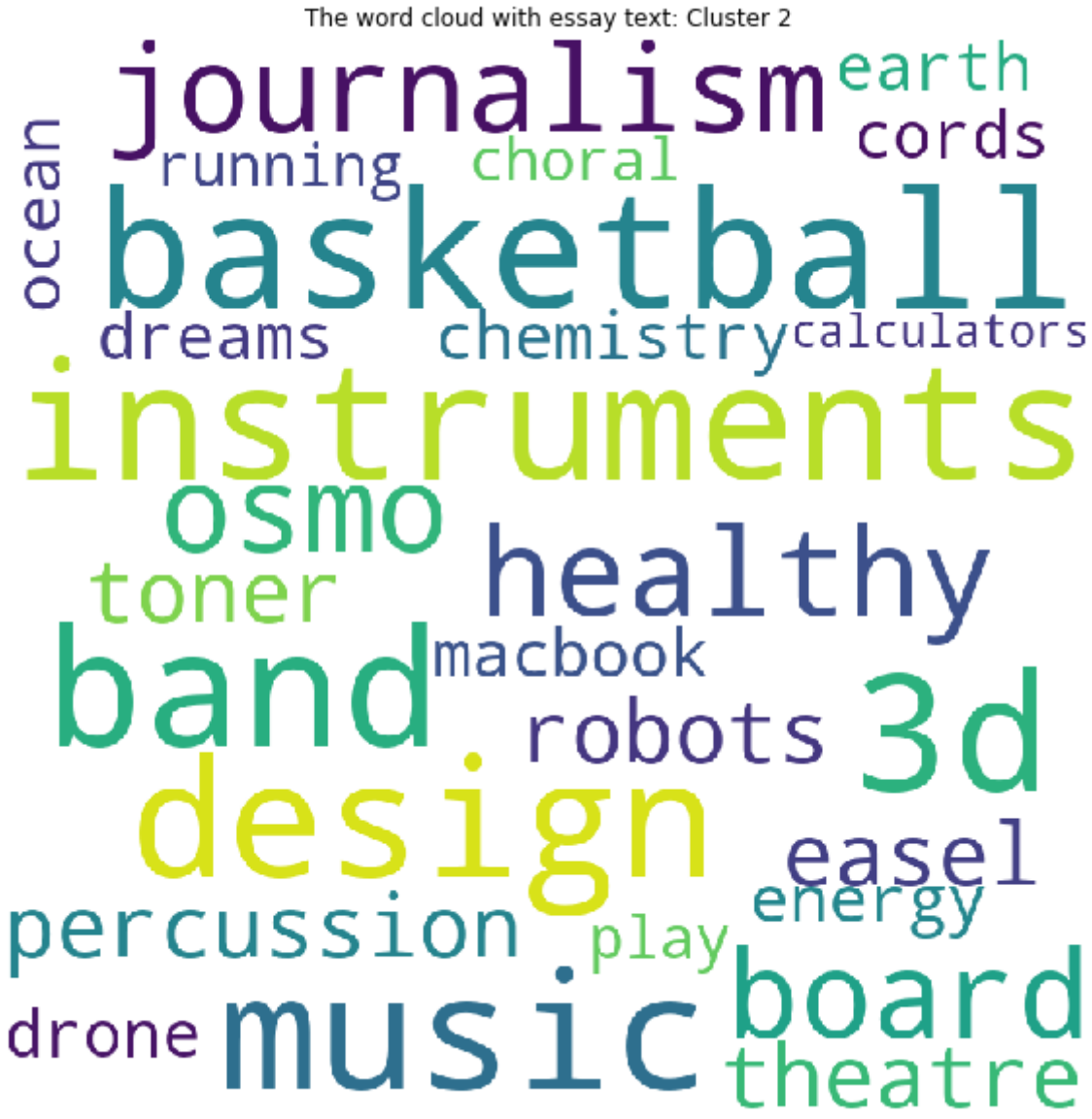
Number of words 719



In [85]:

```
getWordCloud(word_corpus.get(2), 2)
```

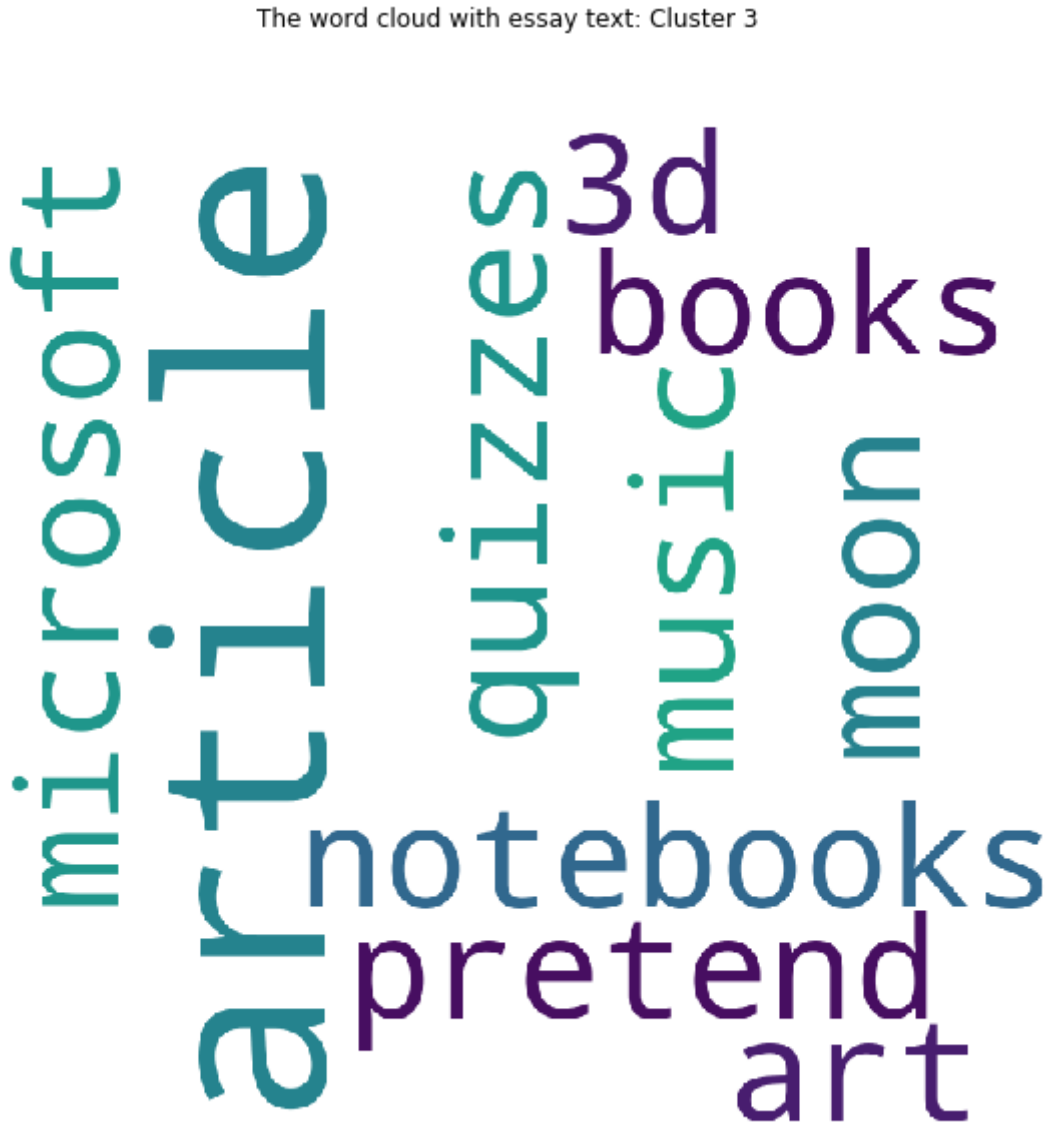
```
Number of words 244
```



In [86]:

```
getWordCloud(word_corpus.get(3), 3)
```

```
Number of words 68
```



In [87]:

```
getWordCloud(word_corpus.get(4), 4)
```

```
Number of words 750
```

In [88]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.title = "Conclusion for Agglomerative Clustering with 4 Clusters"
x.field_names = ["Cluster", "Number of words in cluster","Most frequently occured words"]

x.add_row(["Cluster 0",1139,"art,music,soccor,band"])
x.add_row(["Cluster 1",719,"calculator,graden, lego,ipad,rug,makey"])
x.add_row(["Cluster 2",244,"instruments,basketball,music,design,band,3d,journalism,healthy,
x.add_row(["Cluster 3",68,"microsoft,notebook,quizzes,moon,3d,article"])
x.add_row(["Cluster 3",750,"book, carpet, spanist, puppets"])

print(x)
```

```
+-----------+---------------------------+---------------------------------
-----------------------------------+
|  Cluster  | Number of words in cluster |                    Most frequentl
y occured words                      |
+-----------+---------------------------+---------------------------------
-----------------------------------+
| Cluster 0 |            1139           |                        art,music,
soccor,band                          |
| Cluster 1 |            719            |                calculator,graden,
lego,ipad,rug,makey                  |
| Cluster 2 |            244            | instruments,basketball,music,desi
gn,band,3d,journalism,healthy,board |
| Cluster 3 |             68            |              microsoft,notebook,q
uizzes,moon,3d,article               |
| Cluster 3 |            750            |                    book, carpet,
spanist, puppets                     |
+-----------+---------------------------+---------------------------------
-----------------------------------+
```

Conclusion:

1. Cluster 4 is the smallest cluster.
2. Cluster 0 is relatively bigger cluster.
3. Clusters can be thought of well separated as there as very less commonly ocuured frequent words.

DBSCAN

Range query - KD tree implementation

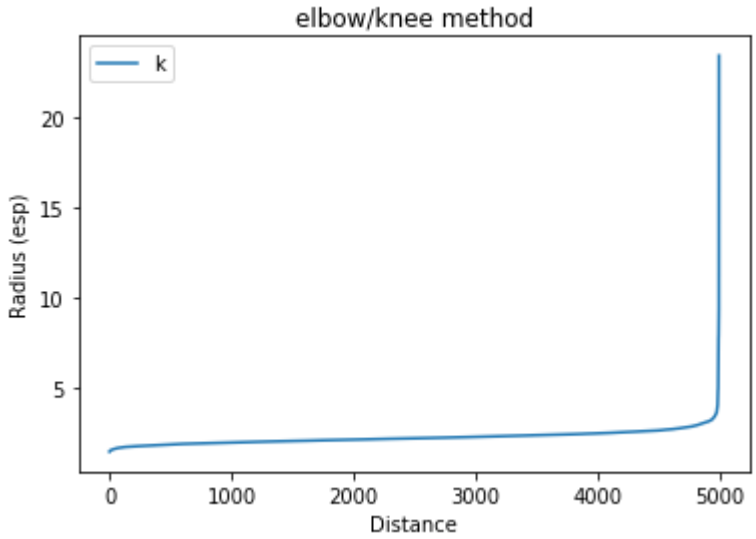
In [91]:

```
#http://brandonrose.org/clustering
from sklearn.neighbors import KDTree

#Minpts are approximately equal to 2*d (d is dimension)
minPts = 10
index = 0
dist = []

#Kd tree and rangeserach
tree = KDTree(X)

# Elbow/knee plot for getting the best value of Radius 'eps'
for i in tqdm(X):
    dist.append(tree.query(i, return_distance=True, k=minPts)[0][0][-1])


dist.sort()

#Plotting elbow /knee plot
plt.plot(range(0,5000), dist[:5000])
plt.title("elbow/knee method")
plt.xlabel('Distance')
plt.ylabel('Radius (esp)')
plt.legend('knee Plot')
```

```
100%|████████████████████████████████████████████████████████████████
████████| 5000/5000 [02:28<00:00, 33.77it/s]
```

Out[91]:

```
<matplotlib.legend.Legend at 0x236f7599c18>
```



In [92]:

```
from sklearn.cluster import DBSCAN

db = DBSCAN(eps = 3 , min_samples=minPts).fit(X)

silhouette_avg = silhouette_score(X, db.labels_)
print("For n_clusters =", 2,
          "The average silhouette_score is :", silhouette_avg)
```

```
For n_clusters = 2 The average silhouette_score is : 0.6475367907366139
```

In [94]:

```
print("Number of clusters got from DBSCAN at optimal radius (eps = 3)", np.unique(db.labels
```

```
Number of clusters got from DBSCAN at optimal radius (eps = 3) [-1  0]
```

In [97]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.title = "DBSCAN - Silhouette Score of each n_Clusters"
x.field_names = ["N_clusters","Avg Silhouette value"]

x.add_row(["2","0.647"])


print(x)
```

```
+------------+----------------------+
| N_clusters | Avg Silhouette value |
+------------+----------------------+
|     2      |        0.647         |
+------------+----------------------+
```

word cloud for each cluster

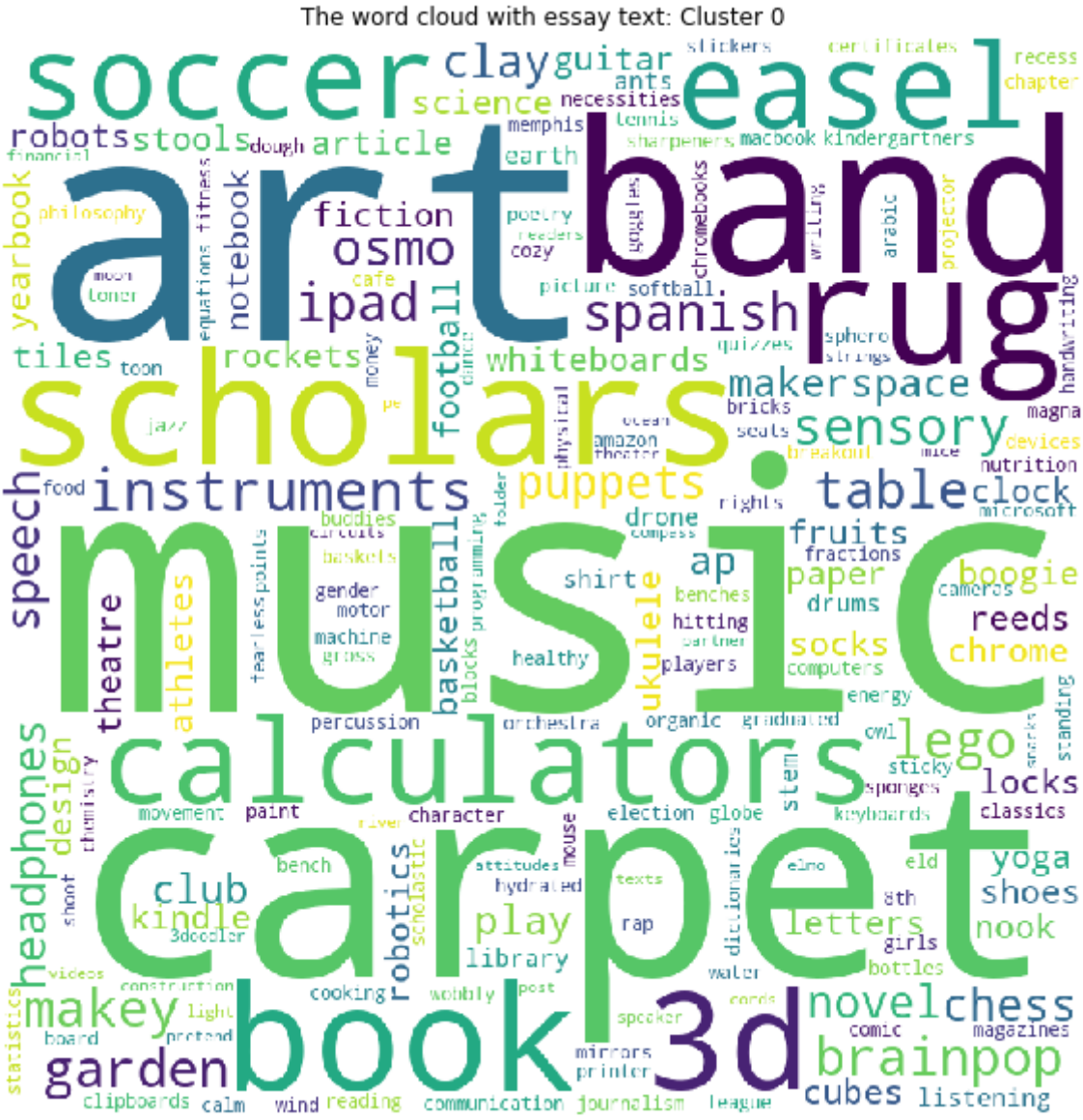
In [95]:

```
word_corpus = getWordCorpus(essay_features, db.labels_)
word_corpus = dict(sorted(list(word_corpus.items()), key=lambda x: x[0]))
for num, corpus in word_corpus.items():
    getWordCloud(corpus, num)
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 5000/5000 [00:19<00:00, 263.01it/s]

Number of words 10
```

The word cloud with essay text: Cluster -1

```
Number of words 2910
```



In [96]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.title = "Conclusion for Agglomerative Clustering with 4 Clusters"
x.field_names = ["Cluster", "Number of words in cluster","Most frequently occured words"]

x.add_row(["Cluster -1",10,"music,art"])
x.add_row(["Cluster 0",2910,"music,carpet,calculator,3d,scholars,rug,book,art"])

print(x)
```

```
+------------+---------------------------+---------------------------------
-----------------+
|  Cluster   | Number of words in cluster |          Most frequently occured
words           |
+------------+---------------------------+---------------------------------
-----------------+
| Cluster -1 |             10            |                    music,art
|
| Cluster 0  |            2910           | music,carpet,calculator,3d,schol
ars,rug,book,art |
+------------+---------------------------+---------------------------------
-----------------+
```

Conclusion:

1. Two clusters has been formed
2. It is so much sensitive to the hyperparameters
3. One cluster is very big while other is too small.

Obeservations from all the models:

1. Elbow method in selecting number of clusters doesn't usually work because the error function is monotonically decreasing for all ks.
2. Kmeans gives more weight to the bigger clusters.
3. Agglomerative has performed well but DBSCAN has performed better than other two.
4. DBSCAN is very sensitive to hyperparameters.
5. DBSCAN has shown best sihouette score of its optimal cluster value that means points in the clusters formed by DBSCAN are more closer to points in neighbouring clusters.

In [98]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.title = "Silhouette Score of each best n_Clusters for each Clustering model"
x.field_names = ["model","N_clusters","Avg Silhouette value"]

x.add_row(["Kmeans","2","0.364"])
x.add_row(["Agglomerative","2","0.474"])
x.add_row(["DBSCAN","2","0.647"])


print(x)
```

```
+---------------+------------+----------------------+
|     model     | N_clusters | Avg Silhouette value |
+---------------+------------+----------------------+
|     Kmeans    |     2      |        0.364         |
| Agglomerative |     2      |        0.474         |
|     DBSCAN    |     2      |        0.647         |
+---------------+------------+----------------------+
```