DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

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

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`

Feature	Description
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

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
```

1.1 Reading Data

In [2]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'schoo
l_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']
```

In [4]:
```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[4]:

	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

1.2 preprocessing of `project_subject_categories`

In [5]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.co
m/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-
a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-i
n-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science",
 "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on sp
ace "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to re
place it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the traili
ng spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.co
m/a/47301924/4084039

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

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

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

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

1.3 Text preprocessing

In [7]:
```
# 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 [8]: project_data.head(2)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_da
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 1
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 0

```
In [9]: # 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 [10]: sent = decontracted(project_data['essay'].values[20000])
         print(sent)
         print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and langua
ge delays, cognitive delays, gross/fine motor delays, to autism. They are eager
beavers and always strive to work their hardest working past their limitations.
\r\n\r\nThe materials we have are the ones I seek out for my students. I teach i
n a Title I school where most of the students receive free or reduced price lunc
h.  Despite their disabilities and limitations, my students love coming to schoo
l and come eager to learn and explore.Have you ever felt like you had ants in yo
ur pants and you needed to groove and move as you were in a meeting? This is how
my kids feel all the time. The want to be able to move as they learn or so they
say.Wobble chairs are the answer and I love then because they develop their cor
e, which enhances gross motor and in Turn fine motor skills. \r\nThey also want
to learn through games, my kids do not want to sit and do worksheets. They want
to learn to count by jumping and playing. Physical engagement is the key to our
success. The number toss and color and shape mats can make that happen. My stude
nts will forget they are doing work and just have the fun a 6 year old deserves.
nannan
==================================================
```

```
In [11]: # \r \n \t remove from string python: http://texthandler.com/info/remove-line-bre
         aks-python/
         sent = sent.replace('\\r', ' ')
         sent = sent.replace('\\"', ' ')
         sent = sent.replace('\\n', ' ')
         print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and langua
ge delays, cognitive delays, gross/fine motor delays, to autism. They are eager
beavers and always strive to work their hardest working past their limitations.
The materials we have are the ones I seek out for my students. I teach in a Titl
e I school where most of the students receive free or reduced price lunch.  Desp
ite their disabilities and limitations, my students love coming to school and co
me eager to learn and explore.Have you ever felt like you had ants in your pants
and you needed to groove and move as you were in a meeting? This is how my kids
feel all the time. The want to be able to move as they learn or so they say.Wobb
le chairs are the answer and I love then because they develop their core, which
enhances gross motor and in Turn fine motor skills.   They also want to learn th
rough games, my kids do not want to sit and do worksheets. They want to learn to
count by jumping and playing. Physical engagement is the key to our success. The
number toss and color and shape mats can make that happen. My students will forg
et they are doing work and just have the fun a 6 year old deserves.nannan
```

In [12]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and langua
ge delays cognitive delays gross fine motor delays to autism They are eager beav
ers and always strive to work their hardest working past their limitations The m
aterials we have are the ones I seek out for my students I teach in a Title I sc
hool where most of the students receive free or reduced price lunch Despite thei
r disabilities and limitations my students love coming to school and come eager
to learn and explore Have you ever felt like you had ants in your pants and you
needed to groove and move as you were in a meeting This is how my kids feel all
the time The want to be able to move as they learn or so they say Wobble chairs
are the answer and I love then because they develop their core which enhances gr
oss motor and in Turn fine motor skills They also want to learn through games my
kids do not want to sit and do worksheets They want to learn to count by jumping
and playing Physical engagement is the key to our success The number toss and co
lor and shape mats can make that happen My students will forget they are doing w
ork and just have the fun a 6 year old deserves nannan
```

In [13]:
```
# https://gist.github.com/sebleier/554280
# 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', 'itsel
f', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
t', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
s', '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', 'th
rough', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all'
, 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should'v
e", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
dn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

```
In [14]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_essays = []
         # tqdm is for printing the status bar
         for sentance in tqdm(project_data['essay'].values):
             sent = decontracted(sentance)
             sent = sent.replace('\\r', ' ')
             sent = sent.replace('\\"', ' ')
             sent = sent.replace('\\n', ' ')
             sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
             # https://gist.github.com/sebleier/554280
             sent = ' '.join(e for e in sent.split() if e not in stopwords)
             preprocessed_essays.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████| 1
09248/109248 [01:14<00:00, 1464.42it/s]
```

```
In [15]: # after preprocesing

         print( len(preprocessed_essays[20000].split()))
```

```
126
```

```
In [16]: # count words in each combined essay
         word_count_essays = []
         for sentance in tqdm(preprocessed_essays):
              count = len(sentance.split())
              word_count_essays.append(count)

         print(word_count_essays[20000])
```

```
100%|███████████████████████████████████████████████████████████████████| 10
9248/109248 [00:01<00:00, 62320.01it/s]

126
```

1.4 Preprocessing of `project_title`

```
In [17]: # similarly you can preprocess the titles also


         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 = 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%|████████████████████████████████████████████████████████████████████████| 10
9248/109248 [00:04<00:00, 27206.93it/s]
```

```
In [18]: print(preprocessed_titles[20000])
```

```
we need to move it while we input it
```

```
In [19]: #count words in title

         word_count_titles = []
         for sentance in tqdm(preprocessed_titles):
              count = len(sentance.split())
              word_count_titles.append(count)

         print(word_count_titles[20000])
```

```
100%|███████████████████████████████████████████████████████████████████████| 109
248/109248 [00:00<00:00, 650286.70it/s]

9
```

1.4.1 Add preprocessed data to dataframe

```
In [20]: preprocessed_title = pd.DataFrame({'preprocessed_titles': preprocessed_titles})
         preprocessed_essay = pd.DataFrame({'preprocessed_essays': preprocessed_essays})
         word_count_title = pd.DataFrame({'word_count_titles': word_count_titles})
         word_count_essay = pd.DataFrame({'word_count_essays': word_count_essays})
```

```
In [21]: project_data = pd.concat([project_data, preprocessed_title,preprocessed_essay, \
                                   word_count_title, word_count_essay], axis=1)

         project_data.head(1)
```

Out[21]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_dat
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:

1 rows × 22 columns

Calculate sentiment score of each essay

In [22]:
```
'''

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

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

sentiment_score_essays_neg = []
sentiment_score_essays_neu = []
sentiment_score_essays_pos = []
sentiment_score_essays_com = []

for sentance in tqdm(preprocessed_essays):
     for_sentiment = sentance
     ss = sid.polarity_scores(for_sentiment)
     sentiment_score_essays_neg.append(ss['neg'])
     sentiment_score_essays_neu.append(ss['neu'])
     sentiment_score_essays_pos.append(ss['pos'])
     sentiment_score_essays_com.append(ss['compound'])



for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

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


'''
```

Out[22]:
```
"\n\nimport nltk\nnltk.downloader.download('vader_lexicon')\nfrom nltk.sentimen
t.vader import SentimentIntensityAnalyzer\n\n# import nltk\n# nltk.download('vad
er_lexicon')\n\nsid = SentimentIntensityAnalyzer()\n\nsentiment_score_essays_neg
= []\nsentiment_score_essays_neu = []\nsentiment_score_essays_pos = []\nsentimen
t_score_essays_com = []\n\nfor sentance in tqdm(preprocessed_essays):\n     for_
sentiment = sentance\n     ss = sid.polarity_scores(for_sentiment)\n     sentime
nt_score_essays_neg.append(ss['neg'])\n     sentiment_score_essays_neu.append(ss
['neu'])\n     sentiment_score_essays_pos.append(ss['pos'])\n     sentiment_scor
e_essays_com.append(ss['compound'])\n\n    \n    \nfor k in ss:\n    print('{0}:
{1}, '.format(k, ss[k]), end='')\n\n# we can use these 4 things as features/attr
ibutes (neg, neu, pos, compound)\n# neg: 0.0, neu: 0.753, pos: 0.247, compound:
0.93\n\n\n"
```

```
In [23]: '''

         print(sentiment_score_essays_neg[10])
         print(sentiment_score_essays_neu[10])
         print(sentiment_score_essays_pos[10])
         print(sentiment_score_essays_com[10])
         '''
```

```
Out[23]: '\n\nprint(sentiment_score_essays_neg[10])\nprint(sentiment_score_essays_neu[1
         0])\nprint(sentiment_score_essays_pos[10])\nprint(sentiment_score_essays_com[1
         0])\n'
```

```
In [24]: '''
         ss_neg = pd.DataFrame({'sentiment_score_essays_neg': sentiment_score_essays_neg})
         ss_neu = pd.DataFrame({'sentiment_score_essays_neu': sentiment_score_essays_neu})
         ss_pos = pd.DataFrame({'sentiment_score_essays_pos': sentiment_score_essays_pos})
         ss_com = pd.DataFrame({'sentiment_score_essays_com': sentiment_score_essays_com})
         '''
```

```
Out[24]: "\nss_neg = pd.DataFrame({'sentiment_score_essays_neg': sentiment_score_essays_n
         eg})\nss_neu = pd.DataFrame({'sentiment_score_essays_neu': sentiment_score_essay
         s_neu})\nss_pos = pd.DataFrame({'sentiment_score_essays_pos': sentiment_score_es
         says_pos})\nss_com = pd.DataFrame({'sentiment_score_essays_com': sentiment_score
         _essays_com})\n"
```

```
In [25]: '''

         project_data = pd.concat([project_data,ss_neg, ss_neu, ss_pos, ss_com], axis=1)

         project_data.head(1)
         '''
```

```
Out[25]: '\n\nproject_data = pd.concat([project_data,ss_neg, ss_neu, ss_pos, ss_com], axi
         s=1)\n\nproject_data.head(1)\n'
```

```
In [ ]:
```

1.5 Preparing data for models

1.5.1 Merge project data with resource data

```
In [26]: price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).r
         eset_index()
         project_data = pd.merge(project_data, price_data, on='id', how='left')
```

```
In [27]: project_data.columns
```

```
Out[27]: Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
                'project_submitted_datetime', 'project_grade_category', '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',
                'clean_categories', 'clean_subcategories', 'essay',
                'preprocessed_titles', 'preprocessed_essays', 'word_count_titles',
                'word_count_essays', 'price', 'quantity'],
               dtype='object')
```

```
In [28]: project_data.head(1)
```

Out[28]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_dat
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:

1 rows × 24 columns

we are going to consider

```
- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category : categorical data
- teacher_prefix : categorical data

- project_title : text data
- text : text data
- project_resource_summary: text data (optinal)

- quantity : numerical (optinal)
- teacher_number_of_previously_posted_projects : numerical
- price : numerical
```

Assignment 10: Clustering

- step 1: Choose any vectorizer (data matrix) that you have worked in any of the assignments, and got the best AUC value.
- step 2: Choose any of the feature selection (https://scikit-learn.org/stable/modules/feature_selection.html)/reduction algorithms (https://scikit-learn.org/stable/modules/decomposition.html) ex: selectkbest features, pretrained word vectors, model based feature selection etc and reduce the number of features to 5k features.
- step 3: Apply all three kmeans, Agglomerative clustering, DBSCAN
  - **K-Means Clustering:**
    ● Find the best ‘k’ using the elbow-knee method (plot k vs inertia_)
  - **Agglomerative Clustering:**
    ● Apply agglomerative algorithm (https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/) and try a different number of clusters like 2,5 etc.
    ● As this is very computationally expensive, take **5k** datapoints only to perform hierarchical clustering because they do take a considerable amount of time to run.
  - **DBSCAN Clustering:**
    ● Find the best ‘eps’ using the elbow-knee method (https://stackoverflow.com/a/48558030/4084039).
    ● Take **5k** datapoints only.
- step 4: Summarize each cluster by manually observing few points from each cluster.
- step 5: You need to plot the word cloud with essay text for each cluster for each of algorithms mentioned in step 3.

In [ ]:

In [ ]:

2.1 Choose the best data matrix on which you got the best AUC

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

In [29]:

```
#Stratify vs ramdom sampling. oversampling for imbalanced data
#https://stats.stackexchange.com/questions/250273/benefits-of-stratified-vs-rando
m-sampling-for-generating-training-data-in-classi




from sklearn.model_selection import train_test_split

# train = project_data.drop(['project_is_approved'], axis=1, inplace=True)  # thi
s will drop in raw data so would not work

X_train, X_test, y_train, y_test = train_test_split(project_data, project_data['p
roject_is_approved'],
                                                    test_size=0.33, stratify = pr
oject_data['project_is_approved'])

#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.3
3, stratify=y_train)


X_train.drop(['project_is_approved'], axis=1, inplace=True)
X_test.drop(['project_is_approved'], axis=1, inplace=True)
#X_cv.drop(['project_is_approved'], axis=1, inplace=True)
```

In [30]:

```
print(X_test.shape)
print(y_test.shape)
print(X_train.shape)
print(y_train.shape)
```

```
(36052, 23)
(36052,)
(73196, 23)
(73196,)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [31]:
```
# Encoding of Categorical Features:

# Category:
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=F
alse, binary=True)
categories_one_hot_train = vectorizer.fit_transform(X_train['clean_categories'].v
alues)
#categories_one_hot_cv = vectorizer.transform(X_cv['clean_categories'].values)
categories_one_hot_test = vectorizer.transform(X_test['clean_categories'].values)

print(vectorizer.get_feature_names())
print("category Shape of matrix after one hot encodig ",categories_one_hot_train.
shape)



# Subcategory
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowerca
se=False, binary=True)
sub_categories_one_hot_train = vectorizer.fit_transform(X_train['clean_subcategor
ies'].values)
#sub_categories_one_hot_cv = vectorizer.transform(X_cv['clean_subcategories'].val
ues)
sub_categories_one_hot_test = vectorizer.transform(X_test['clean_subcategories'].
values)


print(vectorizer.get_feature_names())
print("subctg Shape of matrix after one hot encodig ",sub_categories_one_hot_trai
n.shape)


#you can do the similar thing with state, teacher_prefix and project_grade_catego
ry also

vectorizer = CountVectorizer(lowercase=False, binary=True)
state_one_hot_train = vectorizer.fit_transform(X_train['school_state'].values)
#state_one_hot_cv = vectorizer.transform(X_cv['school_state'].values)
state_one_hot_test = vectorizer.transform(X_test['school_state'].values)


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



vectorizer = CountVectorizer(lowercase=False, binary=True)


tp_one_hot_train = vectorizer.fit_transform(X_train['teacher_prefix'].apply(lambd
a x: np.str_(x)))
#tp_one_hot_cv = vectorizer.transform(X_cv['teacher_prefix'].apply(lambda x: np.s
tr_(x)))
tp_one_hot_test = vectorizer.transform(X_test['teacher_prefix'].apply(lambda x: n
p.str_(x)))


print("tp Shape of matrix after one hot encodig ",tp_one_hot_train.shape)
```

```python
# Project Grade List
from collections import Counter
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.splitlines())


grade_list = dict(my_counter)
print(grade_list)

sorted_grade_list = dict(sorted(grade_list.items(), key=lambda kv: kv[1]))
print(sorted_grade_list)

# If not generating the above list and put into vocabulary, the vector will some
 mess up results ['12', 'Grades', 'PreK']

# This is because of space and new lines. Otherwise no need for vocabulary

vectorizer = CountVectorizer(vocabulary=list(sorted_grade_list.keys()),lowercase=
False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values)
print(vectorizer.get_feature_names())

pg_one_hot_train = vectorizer.transform(X_train['project_grade_category'].values)
#pg_one_hot_cv = vectorizer.transform(X_cv['project_grade_category'].values)
pg_one_hot_test = vectorizer.transform(X_test['project_grade_category'].values)


print("pg Shape of matrix after one hot encodig ",pg_one_hot_train.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'Sp
ecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
category Shape of matrix after one hot encodig  (73196, 9)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Ext
racurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'W
armth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterEducation',
'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geography', 'Heal
th_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalScienc
e', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literat
ure_Writing', 'Mathematics', 'Literacy']
subctg Shape of matrix after one hot encodig  (73196, 30)
state Shape of matrix after one hot encodig  (73196, 51)
tp Shape of matrix after one hot encodig  (73196, 6)
{'Grades PreK-2': 44225, 'Grades 6-8': 16923, 'Grades 3-5': 37137, 'Grades 9-1
2': 10963}
{'Grades 9-12': 10963, 'Grades 6-8': 16923, 'Grades 3-5': 37137, 'Grades PreK-
2': 44225}
['Grades 9-12', 'Grades 6-8', 'Grades 3-5', 'Grades PreK-2']
pg Shape of matrix after one hot encodig  (73196, 4)
```

In [32]:
```
print("teacher prefix of matrix after one hot encoding ",tp_one_hot_train[0:5])
print("project grade matrix after one hot encoding ", pg_one_hot_train.shape)
print(X_train['project_grade_category'].values)
```

```
teacher prefix of matrix after one hot encoding    (0, 1)       1
  (1, 3)        1
  (2, 3)        1
  (3, 3)        1
  (4, 3)        1
project grade matrix after one hot encoding  (73196, 4)
['Grades PreK-2' 'Grades PreK-2' 'Grades 6-8' ... 'Grades 3-5'
 'Grades 3-5' 'Grades PreK-2']
```

In [33]:
```
# Numerical Data


from sklearn import preprocessing

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.
... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)


#instead of standardize, try normalization since chi2 requires non-negative


#price_scalar = Normalizer()
price_scalar = preprocessing.StandardScaler()
price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and st
andard deviation of this data
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scal
ar.var_[0])}")

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

#price_standardized_cv = price_scalar.transform(X_cv['price'].values.reshape(-1,
 1))
price_standardized_test = price_scalar.transform(X_test['price'].values.reshape(-
1, 1))


print(price_standardized_train.mean())
print(price_standardized_train.std())

print(len(price_standardized_train))
```

```
1.2833180729477737e-16
1.0
73196
```

In [34]:
```
previous_scalar = preprocessing.StandardScaler()
previous_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].value
s.reshape(-1,1))

# finding the mean and standard deviation of this data
#print(f"Mean : {previous_scalar.mean_[0]}, Standard deviation : {np.sqrt(previou
s_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
previous_standardized_train = previous_scalar.transform(X_train['teacher_number_o
f_previously_posted_projects'].values.reshape(-1, 1))



#previous_standardized_cv = previous_scalar.transform(X_cv['teacher_number_of_pre
viously_posted_projects'].values.reshape(-1, 1))


previous_standardized_test = previous_scalar.transform(X_test['teacher_number_of_
previously_posted_projects'].values.reshape(-1, 1))


print(previous_standardized_train.mean())
print(previous_standardized_train.std())

print(previous_standardized_train[100])
```

```
2.2327016397729798e-17
1.0
[-0.36463408]
```

In [35]:
```
wc_scalar = preprocessing.StandardScaler()
wc_scalar.fit(X_train['word_count_essays'].values.reshape(-1,1))

# finding the mean and standard deviation of this data
#print(f"Mean : {previous_scalar.mean_[0]}, Standard deviation : {np.sqrt(previou
s_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
wc_standardized_train = wc_scalar.transform(X_train['word_count_essays'].values.r
eshape(-1, 1))
#wc_standardized_cv = wc_scalar.transform(X_cv['word_count_essays'].values.reshap
e(-1, 1))
wc_standardized_test = wc_scalar.transform(X_test['word_count_essays'].values.res
hape(-1, 1))


print(wc_standardized_train.mean())
print(wc_standardized_train.std())

print(wc_standardized_train[100])
```

```
-3.130635994899O7e-17
0.9999999999999999
[0.84123827]
```

In [36]:
```
wc_title_scalar = preprocessing.StandardScaler()
wc_title_scalar.fit(X_train['word_count_titles'].values.reshape(-1,1))

# finding the mean and standard deviation of this data
#print(f"Mean : {previous_scalar.mean_[0]}, Standard deviation : {np.sqrt(previou
s_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
wc_title_standardized_train = wc_title_scalar.transform(X_train['word_count_title
s'].values.reshape(-1, 1))
#wc_title_standardized_cv = wc_title_scalar.transform(X_cv['word_count_titles'].v
alues.reshape(-1, 1))
wc_title_standardized_test = wc_title_scalar.transform(X_test['word_count_titles'
].values.reshape(-1, 1))


print(wc_title_standardized_train.mean())
print(wc_title_standardized_train.std())

print(wc_title_standardized_train[100])
```

```
-9.086124934032649e-17
0.9999999999999999
[-0.184783]
```

In [37]:
```
quantity_scalar = preprocessing.StandardScaler()
quantity_scalar.fit(X_train['quantity'].values.reshape(-1,1))

# finding the mean and standard deviation of this data
#print(f"Mean : {previous_scalar.mean_[0]}, Standard deviation : {np.sqrt(previou
s_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
quantity_standardized_train = quantity_scalar.transform(X_train['quantity'].value
s.reshape(-1, 1))
#quantity_standardized_cv = quantity_scalar.transform(X_cv['quantity'].values.res
hape(-1, 1))
quantity_standardized_test = quantity_scalar.transform(X_test['quantity'].values.
reshape(-1, 1))


print(quantity_standardized_train.mean())
print(quantity_standardized_train.std())

print(quantity_standardized_train[100])
#print(quantity_standardized_train)
```

```
-3.572322623636768e-17
1.0
[0.62082931]
```

In [38]:
```
'''

ss_neg_scalar = preprocessing.StandardScaler()
ss_neg_scalar.fit(X_train['sentiment_score_essays_neg'].values.reshape(-1,1))

# finding the mean and standard deviation of this data
#print(f"Mean : {previous_scalar.mean_[0]}, Standard deviation : {np.sqrt(previou
s_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
ss_neg_standardized_train = ss_neg_scalar.transform(X_train['sentiment_score_essa
ys_neg'].values.reshape(-1, 1))
#ss_neg_standardized_cv = ss_neg_scalar.transform(X_cv['sentiment_score_essays_ne
g'].values.reshape(-1, 1))
ss_neg_standardized_test = ss_neg_scalar.transform(X_test['sentiment_score_essays
_neg'].values.reshape(-1, 1))


print(ss_neg_standardized_train.mean())
print(ss_neg_standardized_train.std())

print(ss_neg_standardized_train[100])
#print(quantity_standardized_train)
'''
```

Out[38]:
```
'\n\nss_neg_scalar = preprocessing.StandardScaler()\nss_neg_scalar.fit(X_train
[\'sentiment_score_essays_neg\'].values.reshape(-1,1)) \n\n# finding the mean an
d standard deviation of this data\n#print(f"Mean : {previous_scalar.mean_[0]}, S
tandard deviation : {np.sqrt(previous_scalar.var_[0])}")\n\n# Now standardize th
e data with above maen and variance.\nss_neg_standardized_train = ss_neg_scalar.
transform(X_train[\'sentiment_score_essays_neg\'].values.reshape(-1, 1))\n#ss_ne
g_standardized_cv = ss_neg_scalar.transform(X_cv[\'sentiment_score_essays_neg
\'].values.reshape(-1, 1))\nss_neg_standardized_test = ss_neg_scalar.transform(X
_test[\'sentiment_score_essays_neg\'].values.reshape(-1, 1))\n\n\nprint(ss_neg_s
tandardized_train.mean())\nprint(ss_neg_standardized_train.std())\n\nprint(ss_ne
g_standardized_train[100])\n#print(quantity_standardized_train)\n'
```

In [39]:
```
'''


ss_neu_scalar = preprocessing.StandardScaler()
ss_neu_scalar.fit(X_train['sentiment_score_essays_neu'].values.reshape(-1,1))

# finding the mean and standard deviation of this data
#print(f"Mean : {previous_scalar.mean_[0]}, Standard deviation : {np.sqrt(previou
s_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
ss_neu_standardized_train = ss_neu_scalar.transform(X_train['sentiment_score_essa
ys_neu'].values.reshape(-1, 1))
#ss_neu_standardized_cv = ss_neu_scalar.transform(X_cv['sentiment_score_essays_ne
u'].values.reshape(-1, 1))
ss_neu_standardized_test = ss_neu_scalar.transform(X_test['sentiment_score_essays
_neu'].values.reshape(-1, 1))


print(ss_neu_standardized_train.mean())
print(ss_neu_standardized_train.std())

print(ss_neu_standardized_train[100])
#print(quantity_standardized_train)
```

```
  File "<ipython-input-39-f22aa3f395ba>", line 20
    #print(quantity_standardized_train)

^
SyntaxError: EOF while scanning triple-quoted string literal
```

In [ ]:
```
'''

ss_pos_scalar = preprocessing.StandardScaler()
ss_pos_scalar.fit(X_train['sentiment_score_essays_pos'].values.reshape(-1,1))

# finding the mean and standard deviation of this data
#print(f"Mean : {previous_scalar.mean_[0]}, Standard deviation : {np.sqrt(previou
s_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
ss_pos_standardized_train = ss_pos_scalar.transform(X_train['sentiment_score_essa
ys_pos'].values.reshape(-1, 1))
#ss_pos_standardized_cv = ss_pos_scalar.transform(X_cv['sentiment_score_essays_po
s'].values.reshape(-1, 1))
ss_pos_standardized_test = ss_pos_scalar.transform(X_test['sentiment_score_essays
_pos'].values.reshape(-1, 1))


print(ss_pos_standardized_train.mean())
print(ss_pos_standardized_train.std())

print(ss_pos_standardized_train[100])
#print(quantity_standardized_train)
```

```
In [ ]: '''

        ss_com_scalar = preprocessing.StandardScaler()
        ss_com_scalar.fit(X_train['sentiment_score_essays_com'].values.reshape(-1,1))

        # finding the mean and standard deviation of this data
        #print(f"Mean : {previous_scalar.mean_[0]}, Standard deviation : {np.sqrt(previou
        s_scalar.var_[0])}")

        # Now standardize the data with above maen and variance.
        ss_com_standardized_train = ss_com_scalar.transform(X_train['sentiment_score_essa
        ys_com'].values.reshape(-1, 1))
        #ss_com_standardized_cv = ss_com_scalar.transform(X_cv['sentiment_score_essays_co
        m'].values.reshape(-1, 1))
        ss_com_standardized_test = ss_com_scalar.transform(X_test['sentiment_score_essays
        _com'].values.reshape(-1, 1))


        print(ss_com_standardized_train.mean())
        print(ss_com_standardized_train.std())

        print(ss_com_standardized_train[100])
        #print(quantity_standardized_train)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

2.3.1 Bag of words

```
In [ ]: '''

        # We are considering only the words which appeared in at least 10 documents(rows
         or projects).
        vectorizer = CountVectorizer(ngram_range = (2,2),min_df=10,max_features = 5000)
        text_train_bow = vectorizer.fit_transform(X_train['preprocessed_essays'].values)


        # should fit_transferm only on train data . Transform on test data
        #text_cv_bow = vectorizer.transform(X_cv['preprocessed_essays'].values)
        text_test_bow = vectorizer.transform(X_test['preprocessed_essays'].values)

        print("Shape of matrix after one hot encodig ",text_train_bow.shape)
        print("Shape of matrix after one hot encodig ",text_test_bow.shape)
        #print("Shape of matrix after one hot encodig ",text_cv_bow.shape)
        print(text_train_bow[1])
```

```
In [ ]: '''

        # you can vectorize the title also
        # before you vectorize the title make sure you preprocess it


        vectorizer = CountVectorizer(ngram_range = (2,2),min_df=10,max_features = 5000)
        title_train_bow = vectorizer.fit_transform(X_train['preprocessed_titles'].values)
        #title_cv_bow = vectorizer.transform(X_cv['preprocessed_titles'].values)
        title_test_bow = vectorizer.transform(X_test['preprocessed_titles'].values)

        print("Shape of matrix after one hot encodig ",title_train_bow.shape)
        #print("Shape of matrix after one hot encodig ",title_cv_bow.shape)
        print("Shape of matrix after one hot encodig ",title_test_bow.shape)
```

2.3.2 TFIDF

```
In [42]: from sklearn.feature_extraction.text import TfidfVectorizer
         vectorizer = TfidfVectorizer(ngram_range=(2,2), min_df=10, max_features = 5000)
         text_train_tfidf = vectorizer.fit_transform(X_train['preprocessed_essays'].values
         )
         #text_cv_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].values)
         text_test_tfidf = vectorizer.transform(X_test['preprocessed_essays'].values)

         print("Shape of matrix after one hot encodig ",text_train_tfidf.shape)
         #print("Shape of matrix after one hot encodig ",text_cv_tfidf.shape)
         print("Shape of matrix after one hot encodig ",text_test_tfidf.shape)

         Shape of matrix after one hot encodig  (73196, 5000)
         Shape of matrix after one hot encodig  (36052, 5000)
```

```
In [43]: # Similarly you can vectorize for title also
         from sklearn.feature_extraction.text import TfidfVectorizer
         vectorizer = TfidfVectorizer(ngram_range = (2,2), min_df=10)
         title_train_tfidf = vectorizer.fit_transform(X_train['preprocessed_titles'].value
         s)
         #title_cv_tfidf = vectorizer.transform(X_cv['preprocessed_titles'].values)
         title_test_tfidf = vectorizer.transform(X_test['preprocessed_titles'].values)

         print("Shape of matrix after one hot encodig ",title_train_tfidf.shape)
         #print("Shape of matrix after one hot encodig ",title_cv_tfidf.shape)
         print("Shape of matrix after one hot encodig ",title_test_tfidf.shape)

         Shape of matrix after one hot encodig  (73196, 2675)
         Shape of matrix after one hot encodig  (36052, 2675)
```

2.3.3 AVG W2V

```
In [44]: '''

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

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



         # Word2Vec does not provide good result if only vectorize by letter, not words
         # Need to split training to words first


         '''



         # Step 1: Getting each word from the sentence
         def list_of_words(Sentence):
             return Sentence.split()

         list_of_Sentence=list(X_train['preprocessed_essays'].values)
         print(len(list_of_Sentence))
         words_list_of_each_Sentence=list(map(list_of_words,list_of_Sentence))

         Step 2: Apply word2vec

         from gensim.models import word2vec
         w2v_model = word2vec.Word2Vec(words_list_of_each_Sentence, size=100,workers=2, mi
         n_count=0)
         #this line of code trains your w2v model on the give list of sentances. Instead o
         f train on X_train.values,
         need to train individual words in it


         glove_words = list(w2v_model.wv.vocab)
         print("number of words that occured minimum 5 times ",len(glove_words))
         print("sample words ", glove_words)

         '''

  File "<ipython-input-44-d81c83be075c>", line 37
    Step 2: Apply word2vec
         ^
SyntaxError: invalid syntax
```

```
In [ ]: with open('glove_vectors', 'rb') as f:
            w2v_model = pickle.load(f)
            glove_words =  set(model.keys())
```

```
In [ ]: '''

        # average Word2Vec
        # compute average word2vec for each review.
        avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in t
        his list
        for sentence in tqdm(X_train['preprocessed_essays'].values): # for each review/se
        ntence
            vector = np.zeros(300) # as word vectors are of zero length, if word2vec then
        use 50
            cnt_words =0; # num of words with a valid vector in the sentence/review
            for word in sentence.split(): # for each word in a review/sentence
                if word in glove_words:
                       #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec
        function
                    vector += w2v_model[word]
                    cnt_words += 1
            if cnt_words != 0:
                vector /= cnt_words
            avg_w2v_vectors_train.append(vector)

        print(len(avg_w2v_vectors_train))
```

```
In [ ]: #this line of code trains your w2v model on the give list of sentances
        '''

        w2v_model=Word2Vec(X_cv['preprocessed_essays'].values,min_count=5,size=50, worker
        s=4)


        glove_words = list(w2v_model.wv.vocab)


        print("number of words that occured minimum 5 times ",len(glove_words))
        print("sample words ", glove_vector[0:50])
        '''
```

In [ ]:
```
'''
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this
list
for sentence in tqdm(X_cv['preprocessed_essays'].values): # for each review/sente
nce
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec f
unction
            vector += w2v_model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)

print(len(avg_w2v_vectors_cv))
'''
```

In [ ]:
```
'''

avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in th
is list
for sentence in tqdm(X_test['preprocessed_essays'].values): # for each review/sen
tence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
             #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec
 function
            vector += w2v_model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)

print(len(avg_w2v_vectors_test))
```

In [ ]:
```
'''

# Similarly you can vectorize for title also

#w2v_model=Word2Vec(X_train['preprocessed_titles'].values,min_count=5,size=50, wo
rkers=4)


#glove_words = list(w2v_model.wv.vocab)
#print("number of words that occured minimum 5 times ",len(glove_words))
#print("sample words ", glove_words[0:50])

avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sentence in tqdm(X_train['preprocessed_titles'].values): # for each review/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
             #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec
 function
            vector += w2v_model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_train.append(vector)

print(len(avg_w2v_vectors_titles_train))
```

In [ ]:
```
# Similarly you can vectorize for title also

#w2v_model=Word2Vec(X_cv['preprocessed_titles'],min_count=5,size=50, workers=4)


#glove_words = list(w2v_model.wv.vocab)
#print("number of words that occured minimum 5 times ",len(glove_words))
#print("sample words ", glove_words[0:50])
'''
avg_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is stored
 in this list
for sentence in tqdm(X_cv['preprocessed_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
             #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec
 function
            vector += w2v_model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_cv.append(vector)

print(len(avg_w2v_vectors_titles_cv))
'''
```

```
In [ ]: '''

        # Similarly you can vectorize for title also

        #w2v_model=Word2Vec(X_test['preprocessed_titles'].values,min_count=5,size=50, wor
        kers=4)


        #glove_words = list(w2v_model.wv.vocab)
        #print("number of words that occured minimum 5 times ",len(glove_words))
        #print("sample words ", glove_words[0:50])

        avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is store
        d in this list
        for sentence in tqdm(X_test['preprocessed_titles'].values): # for each review/sen
        tence
            vector = np.zeros(300) # as word vectors are of zero length
            cnt_words =0; # num of words with a valid vector in the sentence/review
            for word in sentence.split(): # for each word in a review/sentence
                if word in glove_words:
                     #vector += w2v_model.wv[word] # this is for w2v_model from Word2Vec
         function
                    vector += w2v_model[word]
                    cnt_words += 1
            if cnt_words != 0:
                vector /= cnt_words
            avg_w2v_vectors_titles_test.append(vector)

        print(len(avg_w2v_vectors_titles_test))
```

2.3.4 TFIDF WEIGHTED W2V

```
In [ ]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        #w2v_model=Word2Vec(X_train['preprocessed_essays'].values,min_count=5,size=50, wo
        rkers=4)
        #glove_words = list(w2v_model.wv.vocab)

        # Test and train should use same tfidf model
        tfidf_model_train= TfidfVectorizer()
        tfidf_model_train.fit(X_train['preprocessed_essays'].values)
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_tra
        in.idf_)))
        tfidf_words_train = set(tfidf_model_train.get_feature_names())
```

In [ ]:
```
'''

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in
this list
for sentence in tqdm(X_train['preprocessed_essays'].values): # for each review/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_train):
            vec = w2v_model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
e((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split
())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
```

```
In [ ]: #w2v_model=Word2Vec(X_cv['preprocessed_essays'].values,min_count=5,size=50, worke
        rs=4)
        #glove_words = list(w2v_model.wv.vocab)

        '''
        tfidf_model_cv= TfidfVectorizer()
        tfidf_model_cv.fit(X_cv['preprocessed_essays'].values)
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(tfidf_model_cv.get_feature_names(), list(tfidf_model_cv.idf
        _)))
        tfidf_words_cv = set(tfidf_model_cv.get_feature_names())



        tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in th
        is list

        for sentence in tqdm(X_cv['preprocessed_essays'].values): # for each review/sente
        nce
            vector = np.zeros(300) # as word vectors are of zero length
            tf_idf_weight =0; # num of words with a valid vector in the sentence/review
            for word in sentence.split(): # for each word in a review/sentence
                if (word in glove_words) and (word in tfidf_words_cv):
                    vec = w2v_model[word] # getting the vector for each word
                    # here we are multiplying idf value(dictionary[word]) and the tf valu
        e((sentence.count(word)/len(sentence.split())))
                    tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split
        ())) # getting the tfidf value for each word
                    vector += (vec * tf_idf) # calculating tfidf weighted w2v
                    tf_idf_weight += tf_idf
            if tf_idf_weight != 0:
                vector /= tf_idf_weight
            tfidf_w2v_vectors_cv.append(vector)

        print(len(tfidf_w2v_vectors_cv))

        '''
```

In [ ]:
```
'''


#w2v_model=Word2Vec(X_test['preprocessed_essays'].values,min_count=5,size=50, wor
kers=4)
#glove_words = list(w2v_model.wv.vocab)

# test and train should use same tfidf w2v model
#tfidf_model_test= TfidfVectorizer()
#tfidf_model_test.fit(X_test['preprocessed_essays'].values)
## we are converting a dictionary with word as a key, and the idf as a value
#dictionary = dict(zip(tfidf_model_test.get_feature_names(), list(tfidf_model_tes
t.idf_)))
tfidf_words_test = set(tfidf_model_train.get_feature_names())


tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in
 this list


for sentence in tqdm(X_test['preprocessed_essays'].values): # for each review/sen
tence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_test):
            vec = w2v_model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
e((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split
())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
```

similarly convert title into tfidf w2v

```
In [ ]: '''

    # average Word2Vec
# compute average word2vec for each review.

#w2v_model=Word2Vec(X_train['preprocessed_titles'].values,min_count=5,size=50, wo
rkers=4)
#glove_words = list(w2v_model.wv.vocab)




tfidf_model_title_train= TfidfVectorizer()
tfidf_model_title_train.fit(X_train['preprocessed_titles'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_title_train.get_feature_names(), list(tfidf_mod
el_title_train.idf_)))
tfidf_words_title_train = set(tfidf_model_title_train.get_feature_names())




tfidf_w2v_vectors_title_train = []; # the avg-w2v for each sentence/review is sto
red in this list
for sentence in tqdm(X_train['preprocessed_titles'].values): # for each review/se
ntence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_title_train):
            vec = w2v_model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
e((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split
())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title_train.append(vector)

print(len(tfidf_w2v_vectors_title_train))
print(len(tfidf_w2v_vectors_title_train[0]))
```

```
In [ ]:     # average Word2Vec
        # compute average word2vec for each review.

        #w2v_model=Word2Vec(X_cv['preprocessed_titles'],min_count=5,size=50, workers=4)
        #glove_words = list(w2v_model.wv.vocab)


        '''


        tfidf_model_title_cv= TfidfVectorizer()
        tfidf_model_title_cv.fit(X_cv['preprocessed_titles'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(tfidf_model_title_cv.get_feature_names(), list(tfidf_model_
        title_cv.idf_)))
        tfidf_words_title_cv = set(tfidf_model_title_cv.get_feature_names())




        tfidf_w2v_vectors_title_cv = []; # the avg-w2v for each sentence/review is stored
        in this list
        for sentence in tqdm(X_cv['preprocessed_titles']): # for each review/sentence
            vector = np.zeros(300) # as word vectors are of zero length
            tf_idf_weight =0; # num of words with a valid vector in the sentence/review
            for word in sentence.split(): # for each word in a review/sentence
                if (word in glove_words) and (word in tfidf_words_title_cv):
                    vec = w2v_model[word] # getting the vector for each word
                    # here we are multiplying idf value(dictionary[word]) and the tf valu
        e((sentence.count(word)/len(sentence.split())))
                    tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split
        ())) # getting the tfidf value for each word
                    vector += (vec * tf_idf) # calculating tfidf weighted w2v
                    tf_idf_weight += tf_idf
            if tf_idf_weight != 0:
                vector /= tf_idf_weight
            tfidf_w2v_vectors_title_cv.append(vector)

        print(len(tfidf_w2v_vectors_title_cv))
        print(len(tfidf_w2v_vectors_title_cv[0]))


        '''
```

```
In [ ]: '''

            # average Word2Vec
        # compute average word2vec for each review.

        #w2v_model=Word2Vec(X_test['preprocessed_titles'].values,min_count=5,size=50, wor
        kers=4)
        #glove_words = list(w2v_model.wv.vocab)





        #tfidf_model_title_test= TfidfVectorizer()
        #tfidf_model_title_test.fit(X_test['preprocessed_titles'].values)
        ## we are converting a dictionary with word as a key, and the idf as a value
        #dictionary = dict(zip(tfidf_model_title_test.get_feature_names(), list(tfidf_mod
        el_title_test.idf_)))
        tfidf_words_title_test = set(tfidf_model_title_train.get_feature_names())





        tfidf_w2v_vectors_title_test = []; # the avg-w2v for each sentence/review is stor
        ed in this list
        for sentence in tqdm(X_test['preprocessed_titles'].values): # for each review/sen
        tence
            vector = np.zeros(300) # as word vectors are of zero length
            tf_idf_weight =0; # num of words with a valid vector in the sentence/review
            for word in sentence.split(): # for each word in a review/sentence
                if (word in glove_words) and (word in tfidf_words_title_test):
                    vec = w2v_model[word] # getting the vector for each word
                    # here we are multiplying idf value(dictionary[word]) and the tf valu
        e((sentence.count(word)/len(sentence.split())))
                    tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split
        ())) # getting the tfidf value for each word
                    vector += (vec * tf_idf) # calculating tfidf weighted w2v
                    tf_idf_weight += tf_idf
            if tf_idf_weight != 0:
                vector /= tf_idf_weight
            tfidf_w2v_vectors_title_test.append(vector)

        print(len(tfidf_w2v_vectors_title_test))
        print(len(tfidf_w2v_vectors_title_test[0]))
```

2.4 Dimensionality Reduction on the selected features

```
In [46]: # Please write all the code with proper documentation


         from scipy.sparse import hstack

         X_train_tfidf = hstack((categories_one_hot_train, sub_categories_one_hot_train, s
         tate_one_hot_train, pg_one_hot_train, tp_one_hot_train, price_standardized_train,
         previous_standardized_train, \
                              quantity_standardized_train, wc_standardized_train, wc_titl
         e_standardized_train,\
                                                 title_train_tfidf, text_train_tfidf)).
         tocsr()

         X_test_tfidf= hstack((categories_one_hot_test, sub_categories_one_hot_test, state
         _one_hot_test, pg_one_hot_test, tp_one_hot_test, price_standardized_test,  previo
         us_standardized_test, \
                            quantity_standardized_test, wc_standardized_test, wc_title_st
         andardized_test,\
                                               title_test_tfidf, text_test_tfidf)).tocsr
         ()

         #X_cv_tfidf = hstack((categories_one_hot_cv, sub_categories_one_hot_cv, state_one
         _hot_cv, pg_one_hot_cv, tp_one_hot_cv, price_standardized_cv, previous_standardiz
         ed_cv,\
          #                  quantity_standardized_cv, wc_standardized_cv, wc_title_standa
         rdized_cv,\
           #                 ss_neg_standardized_cv,ss_neu_standardized_cv,ss_pos_standard
         ized_cv, ss_com_standardized_cv,\
            #                title_cv_tfidf, text_cv_tfidf)).tocsr()
```

```
In [ ]:
```

```
In [47]: print(X_test_tfidf.shape)
         print(y_test.shape)

         print(X_train_tfidf.shape)
         print(y_train.shape)

         #print(X_cv_tfidf.shape)
         #print(y_cv.shape)

         (36052, 7780)
         (36052,)
         (73196, 7780)
         (73196,)
```

Since model takes very long time to run, decided to only use Top 2000 features

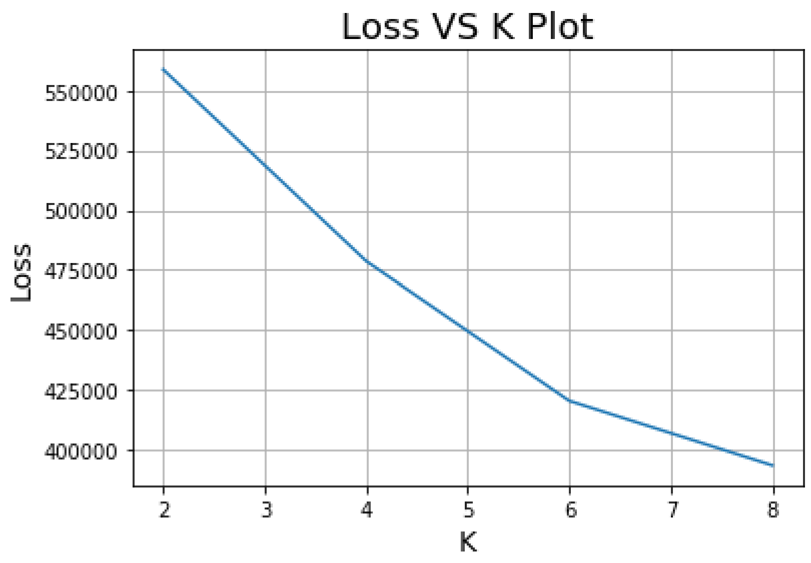
In [48]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.Sele
ctKBest.html
from sklearn.feature_selection import SelectKBest, f_classif
s = SelectKBest(f_classif,k=2000).fit(X_train_tfidf, y_train)
X_train_s = s.transform(X_train_tfidf)
X_test_s = s.transform(X_test_tfidf)
###########################################################################
print("Final Data matrix on TFIDF")
print(X_train_s.shape, y_train.shape)
# print(X_cr.shape, y_cv.shape)
print(X_test_s.shape, y_test.shape)
print("="*100)
```

```
C:\Users\wwang26\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\fea
ture_selection\univariate_selection.py:114: UserWarning:

Features [0 0 0 0] are constant.


Final Data matrix on TFIDF
(73196, 2000) (73196,)
(36052, 2000) (36052,)
================================================================================
====================
```

2.5 Apply Kmeans

In [51]:
```
from sklearn.cluster import KMeans

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

In [52]:
```
plt.plot(k_values, loss)
plt.xlabel('K',size=14)
plt.ylabel('Loss',size=14)
plt.title('Loss VS K Plot',size=18)
plt.grid()
plt.show()
```



In [53]:
```
best_k = 6
kmeans = KMeans(n_clusters=best_k, n_jobs=-1).fit(X_train_s)
```

In [54]:
```
essays = X_train['preprocessed_essays'].values
cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []
cluster5 = []
cluster6 = []
for i in range(kmeans.labels_.shape[0]):
    if kmeans.labels_[i] == 0:
        cluster1.append(essays[i])
    elif kmeans.labels_[i] == 1:
        cluster2.append(essays[i])
    elif kmeans.labels_[i] == 2:
        cluster3.append(essays[i])
    elif kmeans.labels_[i] == 3:
        cluster4.append(essays[i])
    elif kmeans.labels_[i] == 4:
        cluster5.append(essays[i])
    elif kmeans.labels_[i] == 5:
        cluster6.append(essays[i])
```

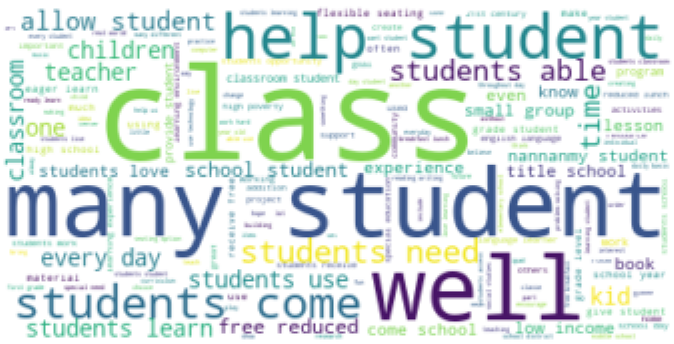
```
In [56]: for i in range(3):
             print('%s\n'%(cluster1[i]))
```

my players play public school dropped funding sports they come practice every da
y work hard become better players better young men they great kids maintain high
standards students first athletes they work hard school taken away funding athle
tics this means students go fundraising every dollar comes program not school re
quires players pay sports fees fundraise money every piece equipment even demand
s players work hard learn lot grow individuals program out town located rural pa
rt arizona arizona state lowest educational funding country many schools justifi
ably shifted money sports classroom our program supports movement working raise
money support team we requesting 3 new baseball bats season in past four years a
ble purchase bats one time these get worn time some players not afford buy bats
left using whatever remains even not work well these bats major benefit players
not afford these bats used practice games 30 players program we also play games
summer workout fall used also host youth kids camp year local youth would used w
e also requesting new bat grips try get life older bats worn by able get three n
ew bats program would give players chance newest equipment we also requesting ne
w foam rollers boxes athletic tape we trying put strong focus physical education
part baseball our players need tape not full time athletic trainer school lot ta
ping the foam rollers great addition daily stretching conditioning program nanna
n

my students amazing bunch 4th graders low income high poverty inner city public
elementary school pennsylvania the majority students receive free lunches my bun
ch includes 38 kiddies always moving asking sharing my students silly inquisitiv
e challenging importantly world my students include general education special ed
ucation students well english language learners their abilities vary greatly i a
lways seeking resources instructional strategies individualize learning opportun
ities large classroom my classroom not typical i strive create opportunities all
ow movement student led activities in i able creative design unique curriculum m
aterials meet many needs students our classroom located k 5 school innovative te
achers always go beyond despite challenges students i find classroom place come
together feel like belong not judged we bring concept school family reality i re
quested close reading comprehension center reading pen students work independent
ly guided reading these centers focus finding evidence variety texts supports id
eas close reading highly effective increase students ability go back text unders
tand structure analyze characters additionally discussion clips close reading ev
idence clips serve hands strategy students specifically show text found informat
ion also identify areas questions connections having kits help build stronger re
aders writers 4th grade classroom your donation give students access tools impro
ve understanding complex texts nannan

the library school working hard encourage students learn fun our recently redesi
gned campus currently third year girls school one district initiatives focus who
le child approach we see students young woman leaders working ensure maintain po
sitive attitude every day as creed suggests girls innovative creative recognize
importance strong mind body spirit every day pushing students reach next level l
earning journey recognize stress come the goal project provide students creative
outlets relieve stress still promoting active learning experiences this project
involves purchasing new spanish books library spanish collection at campus 50 st
udents english language learners majority spanish speakers we also spanish class
es available students learning language new spanish books build collection high
interest reads support spanish learners well support emergent english learners i
magine moving school classes taught language not understand would not enjoy esca
ping story written native language many students verbally bilingual speak englis
h spanish fluently primarily spanish home language however students may not know
read write spanish increased exposure language parents families reading also inc
rease formal spanish vocabulary knowledge spanish sentence structure on top stud
ents learn appreciate beauty spanish language they exposed figurative language w
ell many colorful illustrations help appreciate beauty stories nannan

```
In [57]: #cluster 1
         words=''
         for i in cluster1:
             words+=str(i)
         from wordcloud import WordCloud
         wordcloud = WordCloud(background_color="white").generate(words)

         # Display the generated image:
         plt.imshow(wordcloud, interpolation='bilinear')
         plt.axis("off")
         plt.show()
```



2.6 Apply AgglomerativeClustering

```
In [ ]: from sklearn.preprocessing import StandardScaler
        # dat=StandardScaler().fit_transform(X_tr.toarray())
        dat = X_train_s.toarray()
        dat
```

```
In [ ]: from sklearn.cluster import AgglomerativeClustering
        from sklearn.metrics import silhouette_score

        n = [2, 5, 8]
        sscore = []

        #print(' '*13,'- First -', ' '*17,'- Second -', ' '*16,'- Third -', ' '*16,'- Fou
        rth -')
        #print( end='          ')


        agg = AgglomerativeClustering(n_clusters = 5)
        agg.fit( dat)
        score = silhouette_score( x_train_s, agg.labels_, random_state=42)
        sscore.append(score)
        print('#'*20, end ='          ' )
```

```
In [ ]: cluster1=[]
        cluster2=[]
        essays = X_train['preprocessed_essays'].values
        for i in range(aggcl.labels_.shape[0]):
            if aggcl.labels_[i] == 0:
                cluster1.append(essays[i])
            elif aggcl.labels_[i] == 1:
                cluster2.append(essays[i])
```

```
In [ ]: for i in range(3):
            print('%s\n'%(cluster1[i]))
```

```
In [ ]: #cluster 1
        words=''
        for i in cluster1:
            words+=str(i)
        from wordcloud import WordCloud
        wordcloud = WordCloud(background_color="white").generate(words)

        # Display the generated image:
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis("off")
        plt.show()
```

2.7 Apply DBSCAN

```
In [49]: from sklearn.preprocessing import StandardScaler
         # dat=StandardScaler().fit_transform(X_tr.toarray())
         dat = X_train_s.toarray()
         dat
```

```
Out[49]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]])
```

```
In [50]: from sklearn.metrics.pairwise import euclidean_distances
         euclidean_distances(dat, dat[1].reshape(1, -1))
```

```
Out[50]: array([[1.80738901e+00],
                [4.21468485e-08],
                [3.45203506e+00],
                ...,
                [3.89612889e+00],
                [3.51115026e+00],
                [3.54682372e+00]])
```

```
In [51]: print(X_train_s.shape)

         print(X_train_s.ndim)
```

```
(73196, 2000)
2
```

In [69]:
```
min_points = 5000
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import euclidean_distances

datt=StandardScaler().fit_transform(dat)
distance=[]


from tqdm import tqdm

for point in tqdm(datt):
    temp = euclidean_distances(datt, point.reshape(1, -1))
    distance.append(temp[min_points])


sorted_distance = np.sort(np.array(distance))

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

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

  0%|
| 1/73196 [00:00<12:16:52,  1.66it/s]

  0%|
| 2/73196 [00:01<13:06:58,  1.55it/s]

  0%|
| 3/73196 [00:01<12:51:17,  1.58it/s]

  0%|
| 4/73196 [00:02<11:57:54,  1.70it/s]

  0%|
| 5/73196 [00:02<11:35:11,  1.75it/s]

  0%|
| 6/73196 [00:03<10:33:44,  1.92it/s]

  0%|
| 7/73196 [00:03<9:56:43,  2.04it/s]

  0%|
| 8/73196 [00:04<9:17:30,  2.19it/s]

  0%|
| 9/73196 [00:04<8:40:19,  2.34it/s]

  0%|
| 10/73196 [00:04<8:11:13,  2.48it/s]

  0%|
| 11/73196 [00:05<8:37:52,  2.36it/s]

  0%|
| 12/73196 [00:05<9:20:42,  2.18it/s]

  0%|
| 13/73196 [00:06<11:06:02,  1.83it/s]

  0%|
| 14/73196 [00:07<10:53:05,  1.87it/s]


  0%|
| 15/73196 [00:07<9:55:50,  2.05it/s]


  0%|
| 16/73196 [00:07<9:19:46,  2.18it/s]


  0%|
| 17/73196 [00:08<8:56:46,  2.27it/s]


  0%|
| 18/73196 [00:08<9:20:53,  2.17it/s]


  0%|
| 19/73196 [00:09<8:53:19,  2.29it/s]


  0%|
| 20/73196 [00:09<8:50:23,  2.30it/s]


  0%|
| 21/73196 [00:10<9:01:42,  2.25it/s]


  0%|
| 22/73196 [00:10<8:57:28,  2.27it/s]


  0%|
| 23/73196 [00:10<8:55:45,  2.28it/s]


  0%|
| 24/73196 [00:11<8:56:08,  2.27it/s]


  0%|
| 25/73196 [00:11<9:10:26,  2.22it/s]


  0%|
| 26/73196 [00:12<8:36:53,  2.36it/s]


  0%|
| 27/73196 [00:12<9:02:39,  2.25it/s]


  0%|
| 28/73196 [00:13<8:55:44,  2.28it/s]
```

```
  0%|
| 29/73196 [00:13<8:46:18,  2.32it/s]


  0%|
| 30/73196 [00:14<8:48:24,  2.31it/s]


  0%|
| 31/73196 [00:14<8:37:21,  2.36it/s]


  0%|
| 32/73196 [00:14<8:43:00,  2.33it/s]


  0%|
| 33/73196 [00:15<8:32:27,  2.38it/s]


  0%|
| 34/73196 [00:15<8:27:34,  2.40it/s]


  0%|
| 35/73196 [00:16<10:41:08,  1.90it/s]


  0%|
| 36/73196 [00:17<12:21:12,  1.65it/s]


  0%|
| 37/73196 [00:17<12:37:40,  1.61it/s]


  0%|
| 38/73196 [00:18<11:14:01,  1.81it/s]


  0%|
| 39/73196 [00:18<10:32:42,  1.93it/s]


  0%|
| 40/73196 [00:19<10:39:12,  1.91it/s]


  0%|
| 41/73196 [00:19<10:28:17,  1.94it/s]


  0%|
| 42/73196 [00:20<9:23:30,  2.16it/s]


  0%|
| 43/73196 [00:20<8:42:30,  2.33it/s]
```

```
  0%|
| 44/73196 [00:20<8:14:22,  2.47it/s]


  0%|
| 45/73196 [00:21<7:52:42,  2.58it/s]


  0%|
| 46/73196 [00:21<7:32:52,  2.69it/s]


  0%|
| 47/73196 [00:21<7:17:44,  2.79it/s]


  0%|
| 48/73196 [00:22<7:17:21,  2.79it/s]


  0%|
| 49/73196 [00:22<7:11:10,  2.83it/s]


  0%|
| 50/73196 [00:22<7:22:37,  2.75it/s]


  0%|
| 51/73196 [00:23<7:54:33,  2.57it/s]


  0%|
| 52/73196 [00:23<9:15:35,  2.19it/s]


  0%|
| 53/73196 [00:24<11:06:19,  1.83it/s]


  0%|
| 54/73196 [00:25<12:05:01,  1.68it/s]


  0%|
| 55/73196 [00:25<11:52:29,  1.71it/s]


  0%|
| 56/73196 [00:26<11:46:59,  1.72it/s]


  0%|
| 57/73196 [00:26<10:52:39,  1.87it/s]


  0%|
| 58/73196 [00:27<10:03:19,  2.02it/s]
```

```
  0%|
| 59/73196 [00:27<9:33:21,  2.13it/s]

  0%|
| 60/73196 [00:28<9:01:08,  2.25it/s]

  0%|
| 61/73196 [00:28<8:36:18,  2.36it/s]

  0%|
| 62/73196 [00:28<8:18:55,  2.44it/s]

  0%|
| 63/73196 [00:29<8:14:22,  2.47it/s]

  0%|
| 64/73196 [00:29<8:23:14,  2.42it/s]

  0%|
| 65/73196 [00:30<8:40:55,  2.34it/s]

  0%|
| 66/73196 [00:30<8:41:17,  2.34it/s]

  0%|
| 67/73196 [00:31<8:37:44,  2.35it/s]

  0%|
| 68/73196 [00:31<8:24:21,  2.42it/s]

  0%|
| 69/73196 [00:31<8:23:53,  2.42it/s]

  0%|
| 70/73196 [00:32<8:21:56,  2.43it/s]

  0%|
| 71/73196 [00:32<9:03:31,  2.24it/s]

  0%|
| 72/73196 [00:33<9:40:39,  2.10it/s]

  0%|
| 73/73196 [00:33<9:23:37,  2.16it/s]


  0%|
| 74/73196 [00:34<10:46:07,  1.89it/s]


  0%|
| 75/73196 [00:35<11:48:15,  1.72it/s]


  0%|
| 76/73196 [00:35<12:49:14,  1.58it/s]


  0%|
| 77/73196 [00:36<12:46:54,  1.59it/s]


  0%|
| 78/73196 [00:37<11:52:13,  1.71it/s]


  0%|
| 79/73196 [00:37<10:34:50,  1.92it/s]


  0%|
| 80/73196 [00:37<9:46:53,  2.08it/s]


  0%|
| 81/73196 [00:38<9:01:35,  2.25it/s]


  0%|
| 82/73196 [00:38<8:44:46,  2.32it/s]


  0%|
| 83/73196 [00:38<8:17:24,  2.45it/s]


  0%|
| 84/73196 [00:39<8:09:01,  2.49it/s]


  0%|
| 85/73196 [00:39<7:54:55,  2.57it/s]


  0%|
| 86/73196 [00:40<7:49:23,  2.60it/s]


  0%|
| 87/73196 [00:40<8:38:37,  2.35it/s]
```

```
  0%|
| 88/73196 [00:41<10:15:18,  1.98it/s]


  0%|
| 89/73196 [00:41<11:45:13,  1.73it/s]


  0%|
| 90/73196 [00:42<12:50:23,  1.58it/s]


  0%|
| 91/73196 [00:43<13:08:06,  1.55it/s]


  0%|
| 92/73196 [00:44<13:06:32,  1.55it/s]


  0%|
| 93/73196 [00:44<13:10:48,  1.54it/s]


  0%|
| 94/73196 [00:45<15:46:11,  1.29it/s]


  0%|
| 95/73196 [00:46<15:34:15,  1.30it/s]


  0%|
| 96/73196 [00:47<15:22:40,  1.32it/s]


  0%|
| 97/73196 [00:47<14:34:23,  1.39it/s]


  0%|
| 98/73196 [00:48<12:32:11,  1.62it/s]


  0%|
| 99/73196 [00:48<11:50:32,  1.71it/s]


  0%|
| 100/73196 [00:49<10:50:21,  1.87it/s]


  0%|
| 101/73196 [00:49<10:20:13,  1.96it/s]


  0%|
| 102/73196 [00:50<9:58:52,  2.03it/s]
```

```
  0%|
| 103/73196 [00:50<9:42:27,  2.09it/s]


  0%|
| 104/73196 [00:50<9:26:28,  2.15it/s]


  0%|
| 105/73196 [00:51<9:00:53,  2.25it/s]


  0%|
| 106/73196 [00:52<10:11:23,  1.99it/s]


  0%|
| 107/73196 [00:52<9:44:36,  2.08it/s]


  0%|
| 108/73196 [00:53<11:10:02,  1.82it/s]


  0%|
| 109/73196 [00:53<11:26:58,  1.77it/s]


  0%|
| 110/73196 [00:54<10:42:09,  1.90it/s]


  0%|
| 111/73196 [00:54<10:30:22,  1.93it/s]


  0%|
| 112/73196 [00:55<10:03:23,  2.02it/s]


  0%|
| 113/73196 [00:55<9:48:27,  2.07it/s]


  0%|
| 114/73196 [00:55<9:10:07,  2.21it/s]


  0%|
| 115/73196 [00:56<8:45:48,  2.32it/s]


  0%|
| 116/73196 [00:56<8:34:26,  2.37it/s]


  0%|
| 117/73196 [00:57<8:20:48,  2.43it/s]
```

```
  0%|
| 118/73196 [00:57<8:16:43,  2.45it/s]

  0%|
| 119/73196 [00:57<8:16:54,  2.45it/s]

  0%|
| 120/73196 [00:58<8:11:32,  2.48it/s]

  0%||
| 121/73196 [00:58<8:06:33,  2.50it/s]

  0%||
| 122/73196 [00:59<8:46:15,  2.31it/s]

  0%||
| 123/73196 [00:59<8:38:02,  2.35it/s]

  0%||
| 124/73196 [01:00<8:55:52,  2.27it/s]

  0%||
| 125/73196 [01:00<8:37:19,  2.35it/s]

  0%||
| 126/73196 [01:00<8:22:46,  2.42it/s]

  0%||
| 127/73196 [01:01<8:36:32,  2.36it/s]

  0%||
| 128/73196 [01:01<8:14:15,  2.46it/s]

  0%||
| 129/73196 [01:02<8:15:51,  2.46it/s]

  0%||
| 130/73196 [01:02<9:00:41,  2.25it/s]

  0%||
| 131/73196 [01:03<8:44:37,  2.32it/s]

  0%||
| 132/73196 [01:03<8:33:38,  2.37it/s]


  0%||
| 133/73196 [01:03<8:31:41,  2.38it/s]


  0%||
| 134/73196 [01:04<8:29:12,  2.39it/s]


  0%||
| 135/73196 [01:04<8:25:12,  2.41it/s]


  0%||
| 136/73196 [01:05<8:20:20,  2.43it/s]


  0%||
| 137/73196 [01:05<8:20:21,  2.43it/s]


  0%||
| 138/73196 [01:05<8:11:37,  2.48it/s]


  0%||
| 139/73196 [01:06<8:12:36,  2.47it/s]


  0%||
| 140/73196 [01:06<8:10:39,  2.48it/s]


  0%||
| 141/73196 [01:07<7:52:23,  2.58it/s]


  0%||
| 142/73196 [01:07<7:33:39,  2.68it/s]


  0%||
| 143/73196 [01:07<7:22:11,  2.75it/s]


  0%||
| 144/73196 [01:08<7:20:31,  2.76it/s]


  0%||
| 145/73196 [01:08<7:35:31,  2.67it/s]


  0%||
| 146/73196 [01:08<7:19:47,  2.77it/s]
```

```
  0%||
| 147/73196 [01:09<7:12:39,  2.81it/s]


  0%||
| 148/73196 [01:09<7:02:40,  2.88it/s]


  0%||
| 149/73196 [01:09<7:26:19,  2.73it/s]


  0%||
| 150/73196 [01:10<7:09:28,  2.83it/s]


  0%||
| 151/73196 [01:10<7:09:39,  2.83it/s]


  0%||
| 152/73196 [01:10<7:04:21,  2.87it/s]


  0%||
| 153/73196 [01:11<7:17:27,  2.78it/s]


  0%||
| 154/73196 [01:11<7:28:49,  2.71it/s]


  0%||
| 155/73196 [01:12<7:26:49,  2.72it/s]


  0%||
| 156/73196 [01:12<7:10:37,  2.83it/s]


  0%||
| 157/73196 [01:12<7:05:01,  2.86it/s]


  0%||
| 158/73196 [01:13<6:58:34,  2.91it/s]


  0%||
| 159/73196 [01:13<7:29:39,  2.71it/s]


  0%||
| 160/73196 [01:13<7:26:36,  2.73it/s]


  0%||
| 161/73196 [01:14<8:11:16,  2.48it/s]
```

```
  0%||
| 162/73196 [01:14<8:19:59,  2.43it/s]


  0%||
| 163/73196 [01:15<8:12:01,  2.47it/s]


  0%||
| 164/73196 [01:15<8:11:45,  2.48it/s]


  0%||
| 165/73196 [01:15<8:25:29,  2.41it/s]


  0%||
| 166/73196 [01:16<8:21:09,  2.43it/s]


  0%||
| 167/73196 [01:16<8:18:47,  2.44it/s]


  0%||
| 168/73196 [01:17<8:05:58,  2.50it/s]


  0%||
| 169/73196 [01:17<8:04:19,  2.51it/s]


  0%||
| 170/73196 [01:18<8:37:16,  2.35it/s]


  0%||
| 171/73196 [01:18<8:49:45,  2.30it/s]


  0%||
| 172/73196 [01:18<8:45:05,  2.32it/s]


  0%||
| 173/73196 [01:19<8:40:02,  2.34it/s]


  0%||
| 174/73196 [01:19<8:36:04,  2.36it/s]


  0%||
| 175/73196 [01:20<8:47:35,  2.31it/s]


  0%||
| 176/73196 [01:20<8:36:47,  2.35it/s]
```

```
  0%||
| 177/73196 [01:20<8:01:29,  2.53it/s]

  0%||
| 178/73196 [01:21<7:38:09,  2.66it/s]

  0%||
| 179/73196 [01:21<7:24:03,  2.74it/s]

  0%||
| 180/73196 [01:21<7:11:57,  2.82it/s]

  0%||
| 181/73196 [01:22<7:16:41,  2.79it/s]

  0%||
| 182/73196 [01:22<7:26:48,  2.72it/s]

  0%||
| 183/73196 [01:23<7:20:33,  2.76it/s]

  0%||
| 184/73196 [01:23<7:13:16,  2.81it/s]

  0%||
| 185/73196 [01:23<7:04:53,  2.86it/s]

  0%||
| 186/73196 [01:24<7:01:25,  2.89it/s]

  0%||
| 187/73196 [01:24<8:06:08,  2.50it/s]

  0%||
| 188/73196 [01:24<7:54:48,  2.56it/s]

  0%||
| 189/73196 [01:25<7:40:36,  2.64it/s]

  0%||
| 190/73196 [01:25<7:28:46,  2.71it/s]

  0%||
| 191/73196 [01:25<7:17:14,  2.78it/s]

  0%||
| 192/73196 [01:26<7:09:24,  2.83it/s]

  0%||
| 193/73196 [01:26<7:03:51,  2.87it/s]

  0%||
| 194/73196 [01:27<6:59:10,  2.90it/s]

  0%||
| 195/73196 [01:27<6:56:53,  2.92it/s]

  0%||
| 196/73196 [01:27<7:02:16,  2.88it/s]

  0%||
| 197/73196 [01:28<7:00:33,  2.89it/s]

  0%||
| 198/73196 [01:28<6:55:28,  2.93it/s]

  0%||
| 199/73196 [01:28<6:48:32,  2.98it/s]

  0%||
| 200/73196 [01:29<6:52:36,  2.95it/s]

  0%||
| 201/73196 [01:29<6:44:19,  3.01it/s]

  0%||
| 202/73196 [01:29<6:57:43,  2.91it/s]

  0%||
| 203/73196 [01:30<7:06:03,  2.86it/s]

  0%||
| 204/73196 [01:30<7:06:40,  2.85it/s]

  0%||
| 205/73196 [01:30<7:32:37,  2.69it/s]
```

```
  0%||
| 206/73196 [01:31<7:17:54,  2.78it/s]


  0%||
| 207/73196 [01:31<7:07:40,  2.84it/s]


  0%||
| 208/73196 [01:31<6:54:45,  2.93it/s]


  0%||
| 209/73196 [01:32<6:54:21,  2.94it/s]


  0%||
| 210/73196 [01:32<6:48:05,  2.98it/s]


  0%||
| 211/73196 [01:32<6:49:10,  2.97it/s]


  0%||
| 212/73196 [01:33<6:41:41,  3.03it/s]


  0%||
| 213/73196 [01:33<6:49:04,  2.97it/s]


  0%||
| 214/73196 [01:33<6:47:55,  2.98it/s]


  0%||
| 215/73196 [01:34<6:45:40,  3.00it/s]


  0%||
| 216/73196 [01:34<6:41:48,  3.03it/s]


  0%||
| 217/73196 [01:34<6:56:39,  2.92it/s]


  0%||
| 218/73196 [01:35<6:53:12,  2.94it/s]


  0%||
| 219/73196 [01:35<7:01:39,  2.88it/s]


  0%||
| 220/73196 [01:35<7:05:14,  2.86it/s]
```

```
  0%||
| 221/73196 [01:36<7:13:26,  2.81it/s]

  0%||
| 222/73196 [01:36<7:35:39,  2.67it/s]

  0%||
| 223/73196 [01:37<7:47:20,  2.60it/s]

  0%||
| 224/73196 [01:37<7:41:44,  2.63it/s]

  0%||
| 225/73196 [01:37<7:39:18,  2.65it/s]

  0%||
| 226/73196 [01:38<9:45:11,  2.08it/s]

  0%||
| 227/73196 [01:38<9:10:12,  2.21it/s]

  0%||
| 228/73196 [01:39<9:16:30,  2.19it/s]

  0%||
| 229/73196 [01:39<8:42:52,  2.33it/s]

  0%||
| 230/73196 [01:40<8:29:43,  2.39it/s]

  0%||
| 231/73196 [01:40<8:09:19,  2.49it/s]

  0%||
| 232/73196 [01:41<8:34:16,  2.36it/s]

  0%||
| 233/73196 [01:41<9:13:04,  2.20it/s]

  0%||
| 234/73196 [01:41<8:51:44,  2.29it/s]

  0%||
| 235/73196 [01:42<8:59:13,  2.26it/s]
```

```
  0%|▏
| 236/73196 [01:42<8:55:41,  2.27it/s]

  0%|▏
| 237/73196 [01:43<8:34:31,  2.36it/s]

  0%|▏
| 238/73196 [01:43<8:27:11,  2.40it/s]

  0%|▏
| 239/73196 [01:44<8:27:28,  2.40it/s]

  0%|▏
| 240/73196 [01:44<8:16:19,  2.45it/s]

  0%|▎
| 241/73196 [01:44<8:03:23,  2.52it/s]

  0%|▎
| 242/73196 [01:45<8:04:34,  2.51it/s]

  0%|▎
| 243/73196 [01:45<8:07:48,  2.49it/s]

  0%|▎
| 244/73196 [01:46<8:11:21,  2.47it/s]

  0%|▎
| 245/73196 [01:46<8:01:27,  2.53it/s]

  0%|▎
| 246/73196 [01:46<8:08:49,  2.49it/s]

  0%|▎
| 247/73196 [01:47<8:04:54,  2.51it/s]

  0%|▎
| 248/73196 [01:47<8:08:18,  2.49it/s]

  0%|▎
| 249/73196 [01:48<8:09:53,  2.48it/s]

  0%|▎
```

```
| 250/73196 [01:48<8:03:30,  2.51it/s]

  0%|▌
| 251/73196 [01:48<8:03:37,  2.51it/s]

  0%|▌
| 252/73196 [01:49<8:18:05,  2.44it/s]

  0%|▌
| 253/73196 [01:49<8:09:40,  2.48it/s]

  0%|▌
| 254/73196 [01:50<8:03:34,  2.51it/s]

  0%|▌
| 255/73196 [01:50<8:01:44,  2.52it/s]

  0%|▌
| 256/73196 [01:50<8:19:40,  2.43it/s]

  0%|▌
| 257/73196 [01:51<8:21:04,  2.43it/s]

  0%|▌
| 258/73196 [01:51<8:20:44,  2.43it/s]

  0%|▌
| 259/73196 [01:52<8:12:43,  2.47it/s]

  0%|▌
| 260/73196 [01:52<8:00:46,  2.53it/s]

  0%|▌
| 261/73196 [01:52<7:56:58,  2.55it/s]

  0%|▌
| 262/73196 [01:53<7:54:19,  2.56it/s]

  0%|▌
| 263/73196 [01:53<8:02:08,  2.52it/s]

  0%|▌
| 264/73196 [01:54<8:04:12,  2.51it/s]
```

```
  0%|▍
| 265/73196 [01:54<7:54:40,  2.56it/s]


  0%|▍
| 266/73196 [01:54<7:50:22,  2.58it/s]


  0%|▍
| 267/73196 [01:55<7:54:17,  2.56it/s]


  0%|▍
| 268/73196 [01:55<8:05:39,  2.50it/s]


  0%|▍
| 269/73196 [01:55<7:54:41,  2.56it/s]


  0%|▍
| 270/73196 [01:56<7:56:39,  2.55it/s]


  0%|▍
| 271/73196 [01:56<7:55:39,  2.56it/s]


  0%|▍
| 272/73196 [01:57<7:47:44,  2.60it/s]


  0%|▍
| 273/73196 [01:57<7:46:12,  2.61it/s]


  0%|▍
| 274/73196 [01:57<7:40:58,  2.64it/s]


  0%|▍
| 275/73196 [01:58<7:54:25,  2.56it/s]


  0%|▍
| 276/73196 [01:58<7:48:48,  2.59it/s]


  0%|▍
| 277/73196 [01:59<7:45:26,  2.61it/s]


  0%|▍
| 278/73196 [01:59<7:54:51,  2.56it/s]


  0%|▍
| 279/73196 [01:59<7:59:39,  2.53it/s]
```

```
  0%|▌
| 280/73196 [02:00<8:29:46,  2.38it/s]


  0%|▌
| 281/73196 [02:00<8:22:23,  2.42it/s]


  0%|▌
| 282/73196 [02:01<8:03:09,  2.52it/s]


  0%|▌
| 283/73196 [02:01<7:55:42,  2.55it/s]


  0%|▌
| 284/73196 [02:01<8:02:04,  2.52it/s]


  0%|▌
| 285/73196 [02:02<9:11:23,  2.20it/s]


  0%|▌
| 286/73196 [02:02<8:58:28,  2.26it/s]


  0%|▌
| 287/73196 [02:03<8:37:14,  2.35it/s]


  0%|▌
| 288/73196 [02:03<8:28:49,  2.39it/s]


  0%|▌
| 289/73196 [02:04<8:17:54,  2.44it/s]


  0%|▌
| 290/73196 [02:04<8:07:20,  2.49it/s]


  0%|▌
| 291/73196 [02:04<8:04:20,  2.51it/s]


  0%|▌
| 292/73196 [02:05<7:58:55,  2.54it/s]


  0%|▌
| 293/73196 [02:05<7:56:22,  2.55it/s]


  0%|▌
| 294/73196 [02:05<7:44:07,  2.62it/s]
```

```
  0%|▍
| 295/73196 [02:06<7:59:20,  2.53it/s]

  0%|▍
| 296/73196 [02:06<8:16:05,  2.45it/s]

  0%|▍
| 297/73196 [02:07<8:13:31,  2.46it/s]

  0%|▍
| 298/73196 [02:07<8:43:22,  2.32it/s]

  0%|▍
| 299/73196 [02:08<8:37:17,  2.35it/s]

  0%|▍
| 300/73196 [02:08<8:16:52,  2.45it/s]

  0%|▍
| 301/73196 [02:09<8:53:35,  2.28it/s]

  0%|▍
| 302/73196 [02:09<8:55:09,  2.27it/s]

  0%|▍
| 303/73196 [02:09<8:44:48,  2.31it/s]

  0%|▍
| 304/73196 [02:10<9:02:50,  2.24it/s]

  0%|▍
| 305/73196 [02:10<8:59:27,  2.25it/s]

  0%|▍
| 306/73196 [02:11<8:55:29,  2.27it/s]

  0%|▍
| 307/73196 [02:11<8:47:47,  2.30it/s]

  0%|▍
| 308/73196 [02:12<8:50:35,  2.29it/s]

  0%|▍
```

```
| 309/73196 [02:12<8:31:25,  2.38it/s]


  0%|▌
| 310/73196 [02:12<8:48:31,  2.30it/s]


  0%|▌
| 311/73196 [02:13<9:07:18,  2.22it/s]


  0%|▌
| 312/73196 [02:13<8:48:04,  2.30it/s]


  0%|▌
| 313/73196 [02:14<8:27:38,  2.39it/s]


  0%|▌
| 314/73196 [02:14<8:11:34,  2.47it/s]


  0%|▌
| 315/73196 [02:14<8:07:52,  2.49it/s]


  0%|▌
| 316/73196 [02:15<8:13:34,  2.46it/s]


  0%|▌
| 317/73196 [02:15<8:20:01,  2.43it/s]


  0%|▌
| 318/73196 [02:16<8:31:50,  2.37it/s]


  0%|▌
| 319/73196 [02:16<8:33:21,  2.37it/s]


  0%|▌
| 320/73196 [02:17<8:37:17,  2.35it/s]


  0%|▌
| 321/73196 [02:17<8:42:24,  2.32it/s]


  0%|▌
| 322/73196 [02:17<8:36:13,  2.35it/s]


  0%|▌
| 323/73196 [02:18<8:40:21,  2.33it/s]
```

```
  0%|▌
| 324/73196 [02:18<8:46:49,  2.31it/s]


  0%|▌
| 325/73196 [02:19<8:50:37,  2.29it/s]


  0%|▌
| 326/73196 [02:19<8:59:21,  2.25it/s]


  0%|▌
| 327/73196 [02:20<8:50:11,  2.29it/s]


  0%|▌
| 328/73196 [02:20<9:22:39,  2.16it/s]


  0%|▌
 | 329/73196 [02:21<11:27:42,  1.77it/s]


  0%|▌
 | 330/73196 [02:21<10:39:51,  1.90it/s]


  0%|▌
| 331/73196 [02:22<9:59:34,  2.03it/s]


  0%|▌
 | 332/73196 [02:22<10:30:13,  1.93it/s]


  0%|▌
| 333/73196 [02:23<9:54:39,  2.04it/s]


  0%|▌
| 334/73196 [02:23<9:22:41,  2.16it/s]


  0%|▌
| 335/73196 [02:24<8:58:30,  2.26it/s]


  0%|▌
| 336/73196 [02:24<8:21:10,  2.42it/s]


  0%|▌
| 337/73196 [02:25<9:09:33,  2.21it/s]


  0%|▌
| 338/73196 [02:25<8:58:28,  2.26it/s]
```

```
  0%|▍
| 339/73196 [02:25<8:41:49,  2.33it/s]

  0%|▍
| 340/73196 [02:26<9:05:44,  2.22it/s]

  0%|▍
| 341/73196 [02:26<8:53:31,  2.28it/s]

  0%|▍
| 342/73196 [02:27<8:28:02,  2.39it/s]

  0%|▍
| 343/73196 [02:27<8:13:28,  2.46it/s]

  0%|▍
| 344/73196 [02:27<7:48:36,  2.59it/s]

  0%|▍
| 345/73196 [02:28<7:29:07,  2.70it/s]

  0%|▍
| 346/73196 [02:28<7:34:42,  2.67it/s]

  0%|▍
| 347/73196 [02:28<7:30:31,  2.69it/s]

  0%|▍
| 348/73196 [02:29<7:22:57,  2.74it/s]

  0%|▍
| 349/73196 [02:29<7:56:53,  2.55it/s]

  0%|▍
| 350/73196 [02:30<7:57:17,  2.54it/s]

  0%|▍
| 351/73196 [02:30<7:36:19,  2.66it/s]

  0%|▍
| 352/73196 [02:30<7:38:24,  2.65it/s]

  0%|▍
| 353/73196 [02:31<9:19:42,  2.17it/s]
```

```
  0%|▌
| 354/73196 [02:31<8:32:32,  2.37it/s]

  0%|▌
| 355/73196 [02:32<8:16:35,  2.44it/s]

  0%|▌
| 356/73196 [02:32<7:48:07,  2.59it/s]

  0%|▌
| 357/73196 [02:32<7:32:40,  2.68it/s]

  0%|▌
| 358/73196 [02:33<7:13:32,  2.80it/s]

  0%|▌
| 359/73196 [02:33<7:08:07,  2.84it/s]

  0%|▌
| 360/73196 [02:33<6:58:39,  2.90it/s]

  0%|▌
| 361/73196 [02:34<6:48:35,  2.97it/s]

  0%|▌
| 362/73196 [02:34<6:52:35,  2.94it/s]

  0%|▌
| 363/73196 [02:34<6:46:15,  2.99it/s]

  0%|▌
| 364/73196 [02:35<6:50:57,  2.95it/s]

  0%|▌
| 365/73196 [02:35<6:50:53,  2.95it/s]

  1%|▌
| 366/73196 [02:35<6:50:55,  2.95it/s]

  1%|▌
| 367/73196 [02:36<7:05:18,  2.85it/s]

  1%|▌
```

```
| 368/73196 [02:36<7:02:17,  2.87it/s]

  1%|▌
| 369/73196 [02:36<7:00:46,  2.88it/s]

  1%|▌
| 370/73196 [02:37<6:58:25,  2.90it/s]

  1%|▌
| 371/73196 [02:37<6:57:27,  2.91it/s]

  1%|▌
| 372/73196 [02:37<6:56:44,  2.91it/s]

  1%|▌
| 373/73196 [02:38<6:48:52,  2.97it/s]

  1%|▌
| 374/73196 [02:38<6:51:53,  2.95it/s]

  1%|▌
| 375/73196 [02:38<6:49:59,  2.96it/s]

  1%|▌
| 376/73196 [02:39<6:53:25,  2.94it/s]

  1%|▌
| 377/73196 [02:39<6:53:19,  2.94it/s]

  1%|▌
| 378/73196 [02:40<6:58:00,  2.90it/s]

  1%|▌
| 379/73196 [02:40<7:01:49,  2.88it/s]

  1%|▌
| 380/73196 [02:40<6:58:24,  2.90it/s]

  1%|▌
| 381/73196 [02:41<6:55:26,  2.92it/s]

  1%|▌
| 382/73196 [02:41<6:44:02,  3.00it/s]
```

```
  1%|▌
| 383/73196 [02:41<6:45:40,  2.99it/s]

  1%|▌
| 384/73196 [02:42<6:51:09,  2.95it/s]

  1%|▌
| 385/73196 [02:42<6:55:49,  2.92it/s]

  1%|▌
| 386/73196 [02:42<7:03:05,  2.87it/s]

  1%|▌
| 387/73196 [02:43<6:56:54,  2.91it/s]

  1%|▌
| 388/73196 [02:43<7:05:15,  2.85it/s]

  1%|▌
| 389/73196 [02:43<7:15:39,  2.79it/s]

  1%|▌
| 390/73196 [02:44<7:23:17,  2.74it/s]

  1%|▌
| 391/73196 [02:44<7:12:29,  2.81it/s]

  1%|▌
| 392/73196 [02:44<7:08:57,  2.83it/s]

  1%|▌
| 393/73196 [02:45<7:08:12,  2.83it/s]

  1%|▌
| 394/73196 [02:45<6:54:52,  2.92it/s]

  1%|▌
| 395/73196 [02:45<6:59:12,  2.89it/s]

  1%|▌
| 396/73196 [02:46<6:54:03,  2.93it/s]

  1%|▌
| 397/73196 [02:46<6:51:54,  2.95it/s]
```

```
  1%|▌
| 398/73196 [02:46<6:49:52,  2.96it/s]

  1%|▌
| 399/73196 [02:47<6:47:31,  2.98it/s]

  1%|▌
| 400/73196 [02:47<6:47:57,  2.97it/s]

  1%|▌
| 401/73196 [02:47<6:56:16,  2.91it/s]

  1%|▌
| 402/73196 [02:48<6:48:02,  2.97it/s]

  1%|▌
| 403/73196 [02:48<6:54:27,  2.93it/s]

  1%|▌
| 404/73196 [02:48<6:58:21,  2.90it/s]

  1%|▌
| 405/73196 [02:49<6:51:49,  2.95it/s]

  1%|▌
| 406/73196 [02:49<6:51:29,  2.95it/s]

  1%|▌
| 407/73196 [02:50<6:52:36,  2.94it/s]

  1%|▌
| 408/73196 [02:50<6:53:53,  2.93it/s]

  1%|▌
| 409/73196 [02:50<6:56:14,  2.91it/s]

  1%|▌
| 410/73196 [02:51<6:57:50,  2.90it/s]

  1%|▌
| 411/73196 [02:51<7:12:11,  2.81it/s]

  1%|▌
| 412/73196 [02:51<7:12:13,  2.81it/s]
```

```
  1%|▌
| 413/73196 [02:52<7:09:51,  2.82it/s]

  1%|▌
| 414/73196 [02:52<7:01:56,  2.87it/s]

  1%|▌
| 415/73196 [02:52<6:54:47,  2.92it/s]

  1%|▌
| 416/73196 [02:53<6:54:18,  2.93it/s]

  1%|▌
| 417/73196 [02:53<6:49:24,  2.96it/s]

  1%|▌
| 418/73196 [02:53<6:50:42,  2.95it/s]

  1%|▌
| 419/73196 [02:54<6:48:08,  2.97it/s]

  1%|▌
| 420/73196 [02:54<6:46:23,  2.98it/s]

  1%|▌
| 421/73196 [02:54<6:52:06,  2.94it/s]

  1%|▌
| 422/73196 [02:55<6:59:33,  2.89it/s]

  1%|▌
| 423/73196 [02:55<7:00:29,  2.88it/s]

  1%|▌
| 424/73196 [02:55<7:03:44,  2.86it/s]

  1%|▌
| 425/73196 [02:56<7:02:33,  2.87it/s]

  1%|▌
| 426/73196 [02:56<6:55:50,  2.92it/s]

  1%|▌
```

```
| 427/73196 [02:56<6:49:51,  2.96it/s]

  1%|▌
| 428/73196 [02:57<6:50:52,  2.95it/s]

  1%|▌
| 429/73196 [02:57<6:43:35,  3.00it/s]

  1%|▌
| 430/73196 [02:57<6:43:10,  3.01it/s]

  1%|▌
| 431/73196 [02:58<6:47:01,  2.98it/s]

  1%|▌
| 432/73196 [02:58<6:51:46,  2.95it/s]

  1%|▌
| 433/73196 [02:58<6:49:08,  2.96it/s]

  1%|▌
| 434/73196 [02:59<6:48:27,  2.97it/s]

  1%|▌
| 435/73196 [02:59<6:44:53,  3.00it/s]

  1%|▌
| 436/73196 [02:59<6:39:28,  3.04it/s]

  1%|▌
| 437/73196 [03:00<6:48:12,  2.97it/s]

  1%|▌
| 438/73196 [03:00<6:44:15,  3.00it/s]

  1%|▌
| 439/73196 [03:00<6:40:56,  3.02it/s]

  1%|▌
| 440/73196 [03:01<6:53:32,  2.93it/s]

  1%|▌
| 441/73196 [03:01<6:44:38,  3.00it/s]
```

```
  1%|▌
| 442/73196 [03:01<6:50:44,  2.95it/s]


  1%|▌
| 443/73196 [03:02<6:52:27,  2.94it/s]


  1%|▌
| 444/73196 [03:02<6:51:49,  2.94it/s]


  1%|▌
| 445/73196 [03:02<6:44:55,  2.99it/s]


  1%|▌
| 446/73196 [03:03<7:09:03,  2.83it/s]


  1%|▌
| 447/73196 [03:03<7:00:59,  2.88it/s]


  1%|▌
| 448/73196 [03:03<6:55:17,  2.92it/s]


  1%|▌
| 449/73196 [03:04<6:49:14,  2.96it/s]


  1%|▌
| 450/73196 [03:04<6:47:19,  2.98it/s]


  1%|▌
| 451/73196 [03:04<6:55:50,  2.92it/s]


  1%|▌
| 452/73196 [03:05<6:59:47,  2.89it/s]


  1%|▌
| 453/73196 [03:05<6:54:25,  2.93it/s]


  1%|▌
| 454/73196 [03:06<7:07:48,  2.83it/s]


  1%|▌
| 455/73196 [03:06<7:02:17,  2.87it/s]


  1%|▌
| 456/73196 [03:06<6:59:19,  2.89it/s]
```

```
  1%|▌
| 457/73196 [03:07<7:02:39,  2.87it/s]

  1%|▌
| 458/73196 [03:07<6:53:37,  2.93it/s]

  1%|▌
| 459/73196 [03:07<6:54:10,  2.93it/s]

  1%|▌
| 460/73196 [03:08<6:55:45,  2.92it/s]

  1%|▌
| 461/73196 [03:08<7:01:56,  2.87it/s]

  1%|▌
| 462/73196 [03:08<7:07:13,  2.84it/s]

  1%|▌
| 463/73196 [03:09<7:00:21,  2.88it/s]

  1%|▌
| 464/73196 [03:09<6:54:53,  2.92it/s]

  1%|▌
| 465/73196 [03:09<6:59:09,  2.89it/s]

  1%|▌
| 466/73196 [03:10<7:03:39,  2.86it/s]

  1%|▌
| 467/73196 [03:10<7:06:49,  2.84it/s]

  1%|▌
| 468/73196 [03:10<6:59:20,  2.89it/s]

  1%|▌
| 469/73196 [03:11<7:02:20,  2.87it/s]

  1%|▌
| 470/73196 [03:11<6:56:05,  2.91it/s]

  1%|▌
| 471/73196 [03:11<6:53:49,  2.93it/s]
```

```
  1%|▌
| 472/73196 [03:12<6:47:10,  2.98it/s]

  1%|▌
| 473/73196 [03:12<6:43:39,  3.00it/s]

  1%|▌
| 474/73196 [03:12<6:40:46,  3.02it/s]

  1%|▌
| 475/73196 [03:13<6:40:01,  3.03it/s]

  1%|▌
| 476/73196 [03:13<6:39:53,  3.03it/s]

  1%|▌
| 477/73196 [03:13<6:37:15,  3.05it/s]

  1%|▌
| 478/73196 [03:14<6:39:29,  3.03it/s]

  1%|▌
| 479/73196 [03:14<6:57:48,  2.90it/s]

  1%|▌
| 480/73196 [03:14<6:48:57,  2.96it/s]

  1%|▌
| 481/73196 [03:15<6:41:10,  3.02it/s]

  1%|▌
| 482/73196 [03:15<6:56:03,  2.91it/s]

  1%|▌
| 483/73196 [03:15<6:59:55,  2.89it/s]

  1%|▌
| 484/73196 [03:16<7:15:46,  2.78it/s]

  1%|▌
| 485/73196 [03:16<7:03:30,  2.86it/s]

  1%|▌
```

```
| 486/73196 [03:16<6:53:31,  2.93it/s]

  1%|▌
| 487/73196 [03:17<6:46:48,  2.98it/s]

  1%|▌
| 488/73196 [03:17<6:50:55,  2.95it/s]

  1%|▌
| 489/73196 [03:17<6:42:37,  3.01it/s]

  1%|▌
| 490/73196 [03:18<6:44:35,  3.00it/s]

  1%|▌
| 491/73196 [03:18<6:47:14,  2.98it/s]

  1%|▌
| 492/73196 [03:19<6:50:49,  2.95it/s]

  1%|▌
| 493/73196 [03:19<7:05:10,  2.85it/s]

  1%|▌
| 494/73196 [03:19<7:31:18,  2.68it/s]

  1%|▌
| 495/73196 [03:20<7:53:31,  2.56it/s]

  1%|▌
| 496/73196 [03:20<7:45:07,  2.61it/s]

  1%|▌
| 497/73196 [03:20<7:31:44,  2.68it/s]

  1%|▌
| 498/73196 [03:21<7:36:09,  2.66it/s]

  1%|▌
| 499/73196 [03:21<7:19:50,  2.75it/s]

  1%|▌
| 500/73196 [03:22<7:18:44,  2.76it/s]
```

```
  1%|▌
| 501/73196 [03:22<7:42:40,  2.62it/s]


  1%|▌
| 502/73196 [03:22<7:49:05,  2.58it/s]


  1%|▌
| 503/73196 [03:23<7:34:33,  2.67it/s]


  1%|▌
| 504/73196 [03:23<7:21:16,  2.75it/s]


  1%|▌
| 505/73196 [03:23<7:09:31,  2.82it/s]


  1%|▌
| 506/73196 [03:24<7:02:09,  2.87it/s]


  1%|▌
| 507/73196 [03:24<7:03:41,  2.86it/s]


  1%|▌
| 508/73196 [03:24<7:02:45,  2.87it/s]


  1%|▌
| 509/73196 [03:25<7:15:13,  2.78it/s]


  1%|▌
| 510/73196 [03:25<7:10:01,  2.82it/s]


  1%|▌
| 511/73196 [03:25<7:03:09,  2.86it/s]


  1%|▌
| 512/73196 [03:26<6:59:19,  2.89it/s]


  1%|▌
| 513/73196 [03:26<6:54:58,  2.92it/s]


  1%|▌
| 514/73196 [03:26<6:57:10,  2.90it/s]


  1%|▌
| 515/73196 [03:27<6:56:10,  2.91it/s]
```

```
  1%|▌
| 516/73196 [03:27<6:50:12,  2.95it/s]

  1%|▌
| 517/73196 [03:27<6:48:36,  2.96it/s]

  1%|▌
| 518/73196 [03:28<6:44:25,  3.00it/s]

  1%|▌
| 519/73196 [03:28<6:44:32,  2.99it/s]

  1%|▌
| 520/73196 [03:29<7:00:47,  2.88it/s]

  1%|▌
| 521/73196 [03:29<7:12:25,  2.80it/s]

  1%|▌
| 522/73196 [03:29<7:18:28,  2.76it/s]

  1%|▌
| 523/73196 [03:30<7:14:40,  2.79it/s]

  1%|▌
| 524/73196 [03:30<7:12:33,  2.80it/s]

  1%|▌
| 525/73196 [03:30<7:15:43,  2.78it/s]

  1%|▌
| 526/73196 [03:31<7:06:06,  2.84it/s]

  1%|▌
| 527/73196 [03:31<7:24:28,  2.72it/s]

  1%|▌
| 528/73196 [03:31<7:22:56,  2.73it/s]

  1%|▌
| 529/73196 [03:32<7:30:12,  2.69it/s]

  1%|▌
| 530/73196 [03:32<7:24:27,  2.72it/s]
```

```
  1%|▌
| 531/73196 [03:33<7:32:26,  2.68it/s]

  1%|▌
| 532/73196 [03:33<8:36:52,  2.34it/s]

  1%|▌
| 533/73196 [03:34<8:21:37,  2.41it/s]

  1%|▌
| 534/73196 [03:34<7:55:44,  2.55it/s]

  1%|▌
| 535/73196 [03:34<7:36:14,  2.65it/s]

  1%|▌
| 536/73196 [03:35<7:19:51,  2.75it/s]

  1%|▌
| 537/73196 [03:35<7:14:33,  2.79it/s]

  1%|▌
| 538/73196 [03:35<7:16:13,  2.78it/s]

  1%|▌
| 539/73196 [03:36<7:23:47,  2.73it/s]

  1%|▌
| 540/73196 [03:36<7:48:02,  2.59it/s]

  1%|▌
| 541/73196 [03:36<7:56:30,  2.54it/s]

  1%|▌
| 542/73196 [03:37<7:48:56,  2.58it/s]

  1%|▌
| 543/73196 [03:37<7:37:18,  2.65it/s]

  1%|▌
| 544/73196 [03:38<9:43:37,  2.07it/s]

  1%|▌
```

```
 | 545/73196 [03:38<10:04:35,  2.00it/s]

  1%|▌
| 546/73196 [03:39<9:29:59,  2.12it/s]

  1%|▌
| 547/73196 [03:39<9:00:27,  2.24it/s]

  1%|▌
| 548/73196 [03:40<8:38:41,  2.33it/s]

  1%|▌
| 549/73196 [03:40<8:20:52,  2.42it/s]

  1%|▌
| 550/73196 [03:40<8:10:15,  2.47it/s]

  1%|▌
| 551/73196 [03:41<8:33:40,  2.36it/s]

  1%|▌
| 552/73196 [03:41<8:22:32,  2.41it/s]

  1%|▌
| 553/73196 [03:42<8:45:49,  2.30it/s]

  1%|▌
| 554/73196 [03:42<8:26:59,  2.39it/s]

  1%|▌
| 555/73196 [03:43<8:11:05,  2.47it/s]

  1%|▌
| 556/73196 [03:43<7:59:24,  2.53it/s]

  1%|▌
| 557/73196 [03:43<8:31:15,  2.37it/s]

  1%|▌
| 558/73196 [03:44<8:47:47,  2.29it/s]

  1%|▌
| 559/73196 [03:44<8:24:05,  2.40it/s]
```

```
  1%|▌
| 560/73196 [03:45<8:38:51,  2.33it/s]


  1%|▌
| 561/73196 [03:45<8:20:17,  2.42it/s]


  1%|▌
| 562/73196 [03:46<8:39:13,  2.33it/s]


  1%|▌
| 563/73196 [03:46<8:24:43,  2.40it/s]


  1%|▌
| 564/73196 [03:46<8:10:40,  2.47it/s]


  1%|▌
| 565/73196 [03:47<8:14:37,  2.45it/s]


  1%|▌
| 566/73196 [03:47<8:12:59,  2.46it/s]


  1%|▌
| 567/73196 [03:47<8:10:19,  2.47it/s]


  1%|▌
| 568/73196 [03:48<8:15:42,  2.44it/s]


  1%|▌
| 569/73196 [03:48<8:07:46,  2.48it/s]


  1%|▌
| 570/73196 [03:49<8:04:31,  2.50it/s]


  1%|▌
| 571/73196 [03:49<7:58:57,  2.53it/s]


  1%|▌
| 572/73196 [03:49<7:54:58,  2.55it/s]


  1%|▌
| 573/73196 [03:50<7:52:59,  2.56it/s]


  1%|▌
| 574/73196 [03:50<7:43:45,  2.61it/s]
```

```
  1%|▌
| 575/73196 [03:51<7:46:05,  2.60it/s]

  1%|▌
| 576/73196 [03:51<7:46:15,  2.60it/s]

  1%|▌
| 577/73196 [03:51<7:45:06,  2.60it/s]

  1%|▌
| 578/73196 [03:52<7:53:15,  2.56it/s]

  1%|▌
| 579/73196 [03:52<8:08:46,  2.48it/s]

  1%|▌
| 580/73196 [03:53<8:13:50,  2.45it/s]

  1%|▌
| 581/73196 [03:53<8:13:09,  2.45it/s]

  1%|▌
| 582/73196 [03:53<8:04:52,  2.50it/s]

  1%|▌
| 583/73196 [03:54<7:58:56,  2.53it/s]

  1%|▌
| 584/73196 [03:54<7:51:54,  2.56it/s]

  1%|▌
| 585/73196 [03:55<7:46:51,  2.59it/s]

  1%|▌
| 586/73196 [03:55<7:45:23,  2.60it/s]

  1%|▌
| 587/73196 [03:55<7:45:32,  2.60it/s]

  1%|▌
| 588/73196 [03:56<7:42:55,  2.61it/s]

  1%|▌
| 589/73196 [03:56<7:59:29,  2.52it/s]
```

```
  1%|▌
| 590/73196 [03:57<8:00:37,  2.52it/s]

  1%|▌
| 591/73196 [03:57<8:02:01,  2.51it/s]

  1%|▌
| 592/73196 [03:57<7:55:27,  2.55it/s]

  1%|▌
| 593/73196 [03:58<8:14:08,  2.45it/s]

  1%|▌
| 594/73196 [03:58<8:13:29,  2.45it/s]

  1%|▌
| 595/73196 [03:59<8:04:13,  2.50it/s]

  1%|▌
| 596/73196 [03:59<8:13:37,  2.45it/s]

  1%|▌
| 597/73196 [03:59<8:30:03,  2.37it/s]

  1%|▌
| 598/73196 [04:00<8:30:46,  2.37it/s]

  1%|▌
| 599/73196 [04:00<8:16:54,  2.43it/s]

  1%|▌
| 600/73196 [04:01<8:17:44,  2.43it/s]

  1%|▌
| 601/73196 [04:01<8:18:41,  2.43it/s]

  1%|▌
| 602/73196 [04:01<8:19:28,  2.42it/s]

  1%|▌
| 603/73196 [04:02<8:38:01,  2.34it/s]

  1%|▌
```

```
| 604/73196 [04:02<8:18:43,  2.43it/s]

  1%|▌
| 605/73196 [04:03<8:06:10,  2.49it/s]

  1%|▌
| 606/73196 [04:03<8:18:08,  2.43it/s]

  1%|▌
| 607/73196 [04:04<8:06:06,  2.49it/s]

  1%|▌
| 608/73196 [04:04<8:00:53,  2.52it/s]

  1%|▌
| 609/73196 [04:04<8:21:07,  2.41it/s]

  1%|▌
| 610/73196 [04:05<8:40:44,  2.32it/s]

  1%|▌
| 611/73196 [04:05<8:58:56,  2.24it/s]

  1%|▌
| 612/73196 [04:06<8:51:21,  2.28it/s]

  1%|▌
| 613/73196 [04:06<8:36:40,  2.34it/s]

  1%|▌
| 614/73196 [04:06<8:15:26,  2.44it/s]

  1%|▌
| 615/73196 [04:07<8:01:56,  2.51it/s]

  1%|▌
| 616/73196 [04:07<7:55:27,  2.54it/s]

  1%|▌
| 617/73196 [04:08<7:58:19,  2.53it/s]

  1%|▌
| 618/73196 [04:08<8:07:25,  2.48it/s]
```

```
  1%|▋
| 619/73196 [04:08<8:02:43,  2.51it/s]

  1%|▋
| 620/73196 [04:09<8:01:08,  2.51it/s]

  1%|▋
| 621/73196 [04:09<7:51:53,  2.56it/s]

  1%|▋
| 622/73196 [04:10<8:03:25,  2.50it/s]

  1%|▋
| 623/73196 [04:10<8:21:57,  2.41it/s]

  1%|▋
| 624/73196 [04:10<8:17:05,  2.43it/s]

  1%|▋
| 625/73196 [04:11<8:04:32,  2.50it/s]

  1%|▋
| 626/73196 [04:11<8:05:16,  2.49it/s]

  1%|▋
| 627/73196 [04:12<7:58:20,  2.53it/s]

  1%|▋
| 628/73196 [04:12<7:54:51,  2.55it/s]

  1%|▋
| 629/73196 [04:12<7:55:31,  2.54it/s]

  1%|▋
| 630/73196 [04:13<7:50:39,  2.57it/s]

  1%|▋
| 631/73196 [04:13<7:47:04,  2.59it/s]

  1%|▋
| 632/73196 [04:14<7:45:32,  2.60it/s]

  1%|▋
| 633/73196 [04:14<7:42:30,  2.61it/s]
```

```
  1%|▊
| 634/73196 [04:14<7:55:27,  2.54it/s]

  1%|▊
| 635/73196 [04:15<8:02:36,  2.51it/s]

  1%|▊
| 636/73196 [04:15<7:55:16,  2.54it/s]

  1%|▊
| 637/73196 [04:16<7:58:01,  2.53it/s]

  1%|▊
| 638/73196 [04:16<7:55:07,  2.55it/s]

  1%|▊
| 639/73196 [04:16<7:55:01,  2.55it/s]

  1%|▊
| 640/73196 [04:17<8:25:31,  2.39it/s]

  1%|▊
| 641/73196 [04:17<9:06:52,  2.21it/s]

  1%|▊
| 642/73196 [04:18<9:19:21,  2.16it/s]

  1%|▊
| 643/73196 [04:18<9:46:25,  2.06it/s]

  1%|▊
 | 644/73196 [04:19<10:20:47,  1.95it/s]

  1%|▊
 | 645/73196 [04:19<10:20:25,  1.95it/s]

  1%|▊
| 646/73196 [04:20<9:56:57,  2.03it/s]

  1%|▊
 | 647/73196 [04:20<10:27:08,  1.93it/s]

  1%|▊
| 648/73196 [04:21<9:35:04,  2.10it/s]
```

```
  1%|█
| 649/73196 [04:21<9:22:38,  2.15it/s]

  1%|█
| 650/73196 [04:22<8:32:03,  2.36it/s]

  1%|█
| 651/73196 [04:22<7:58:33,  2.53it/s]

  1%|█
| 652/73196 [04:22<8:17:11,  2.43it/s]

  1%|█
| 653/73196 [04:23<7:50:40,  2.57it/s]

  1%|█
| 654/73196 [04:23<7:35:21,  2.66it/s]

  1%|█
| 655/73196 [04:23<7:27:08,  2.70it/s]

  1%|█
| 656/73196 [04:24<7:13:20,  2.79it/s]

  1%|█
| 657/73196 [04:24<7:22:45,  2.73it/s]

  1%|█
| 658/73196 [04:25<7:47:18,  2.59it/s]

  1%|█
| 659/73196 [04:25<8:09:35,  2.47it/s]

  1%|█
| 660/73196 [04:25<8:05:53,  2.49it/s]

  1%|█
| 661/73196 [04:26<8:02:21,  2.51it/s]

  1%|█
| 662/73196 [04:26<8:00:49,  2.51it/s]

  1%|█
```

```
| 663/73196 [04:27<7:52:25,  2.56it/s]

  1%|▌
| 664/73196 [04:27<8:12:38,  2.45it/s]

  1%|▌
| 665/73196 [04:27<8:23:17,  2.40it/s]

  1%|▌
| 666/73196 [04:28<7:57:42,  2.53it/s]

  1%|▌
| 667/73196 [04:28<7:41:43,  2.62it/s]

  1%|▌
| 668/73196 [04:29<7:23:38,  2.72it/s]

  1%|▌
| 669/73196 [04:29<7:11:12,  2.80it/s]

  1%|▌
| 670/73196 [04:29<7:08:03,  2.82it/s]

  1%|▌
| 671/73196 [04:30<7:06:13,  2.84it/s]

  1%|▌
| 672/73196 [04:30<7:20:50,  2.74it/s]

  1%|▌
| 673/73196 [04:30<7:25:26,  2.71it/s]

  1%|▌
| 674/73196 [04:31<7:27:48,  2.70it/s]

  1%|▌
| 675/73196 [04:31<7:36:22,  2.65it/s]

  1%|▌
| 676/73196 [04:31<7:17:30,  2.76it/s]

  1%|▌
| 677/73196 [04:32<7:01:35,  2.87it/s]
```

```
  1%|▌
| 678/73196 [04:32<6:57:42,  2.89it/s]


  1%|▌
| 679/73196 [04:32<6:51:50,  2.93it/s]


  1%|▌
| 680/73196 [04:33<6:49:34,  2.95it/s]


  1%|▌
| 681/73196 [04:33<6:52:43,  2.93it/s]


  1%|▌
| 682/73196 [04:33<6:57:14,  2.90it/s]


  1%|▌
| 683/73196 [04:34<7:03:43,  2.85it/s]


  1%|▌
| 684/73196 [04:34<7:41:32,  2.62it/s]


  1%|▌
| 685/73196 [04:35<8:01:59,  2.51it/s]


  1%|▌
| 686/73196 [04:35<8:24:16,  2.40it/s]


  1%|▌
| 687/73196 [04:36<8:07:50,  2.48it/s]


  1%|▌
| 688/73196 [04:36<8:16:41,  2.43it/s]


  1%|▌
| 689/73196 [04:36<8:00:58,  2.51it/s]


  1%|▌
| 690/73196 [04:37<7:46:08,  2.59it/s]


  1%|▌
| 691/73196 [04:37<8:12:59,  2.45it/s]


  1%|▌
| 692/73196 [04:37<7:51:39,  2.56it/s]
```

```
  1%|▌
| 693/73196 [04:38<7:51:44,  2.56it/s]

  1%|▌
| 694/73196 [04:38<7:30:14,  2.68it/s]

  1%|▌
| 695/73196 [04:39<7:34:41,  2.66it/s]

  1%|▌
| 696/73196 [04:39<7:16:28,  2.77it/s]

  1%|▌
| 697/73196 [04:39<7:01:04,  2.87it/s]

  1%|▌
| 698/73196 [04:40<7:23:13,  2.73it/s]

  1%|▌
| 699/73196 [04:40<7:14:00,  2.78it/s]

  1%|▌
| 700/73196 [04:40<7:17:18,  2.76it/s]

  1%|▌
| 701/73196 [04:41<7:10:57,  2.80it/s]

  1%|▌
| 702/73196 [04:41<7:11:02,  2.80it/s]

  1%|▌
| 703/73196 [04:41<7:07:02,  2.83it/s]

  1%|▌
| 704/73196 [04:42<7:10:13,  2.81it/s]

  1%|▌
| 705/73196 [04:42<7:10:58,  2.80it/s]

  1%|▌
| 706/73196 [04:43<9:07:52,  2.21it/s]

  1%|▌
 | 707/73196 [04:44<11:18:23,  1.78it/s]
```

```
  1%|▊
 | 708/73196 [04:44<11:02:18,  1.82it/s]

  1%|▊
 | 709/73196 [04:45<10:08:20,  1.99it/s]

  1%|▊
| 710/73196 [04:45<9:42:08,  2.08it/s]

  1%|▊
| 711/73196 [04:45<8:45:31,  2.30it/s]

  1%|▊
| 712/73196 [04:46<8:18:04,  2.43it/s]

  1%|▊
| 713/73196 [04:46<7:47:04,  2.59it/s]

  1%|▊
| 714/73196 [04:46<7:31:24,  2.68it/s]

  1%|▊
 | 715/73196 [04:47<10:04:16,  2.00it/s]

  1%|▊
 | 716/73196 [04:48<10:40:12,  1.89it/s]

  1%|▊
 | 717/73196 [04:48<10:13:19,  1.97it/s]

  1%|▊
| 718/73196 [04:49<9:23:39,  2.14it/s]

  1%|▊
| 719/73196 [04:49<8:38:58,  2.33it/s]

  1%|▊
| 720/73196 [04:49<8:04:33,  2.49it/s]

  1%|▊
| 721/73196 [04:50<7:39:21,  2.63it/s]

  1%|▊
```

```
| 722/73196 [04:50<7:15:58,  2.77it/s]

  1%|█
| 723/73196 [04:50<7:07:37,  2.82it/s]

  1%|█
| 724/73196 [04:51<7:21:32,  2.74it/s]

  1%|█
| 725/73196 [04:51<7:37:20,  2.64it/s]

  1%|█
| 726/73196 [04:52<8:20:42,  2.41it/s]

  1%|█
| 727/73196 [04:52<8:00:53,  2.51it/s]

  1%|█
| 728/73196 [04:52<7:42:31,  2.61it/s]

  1%|█
| 729/73196 [04:53<7:26:08,  2.71it/s]

  1%|█
| 730/73196 [04:53<7:11:13,  2.80it/s]

  1%|█
| 731/73196 [04:53<7:11:24,  2.80it/s]

  1%|█
| 732/73196 [04:54<7:00:58,  2.87it/s]

  1%|█
| 733/73196 [04:54<6:57:14,  2.89it/s]

  1%|█
| 734/73196 [04:54<7:02:26,  2.86it/s]

  1%|█
| 735/73196 [04:55<7:02:45,  2.86it/s]

  1%|█
| 736/73196 [04:55<7:01:28,  2.87it/s]
```

```
  1%|█
| 737/73196 [04:55<6:56:55,  2.90it/s]

  1%|█
| 738/73196 [04:56<6:55:59,  2.90it/s]

  1%|█
| 739/73196 [04:56<7:24:20,  2.72it/s]

  1%|█
| 740/73196 [04:57<8:32:08,  2.36it/s]

  1%|█
| 741/73196 [04:57<9:47:34,  2.06it/s]

  1%|█
 | 742/73196 [04:58<10:44:46,  1.87it/s]

  1%|█
| 743/73196 [04:58<9:45:13,  2.06it/s]

  1%|█
| 744/73196 [04:59<9:15:14,  2.17it/s]

  1%|█
| 745/73196 [04:59<8:51:02,  2.27it/s]

  1%|█
| 746/73196 [04:59<8:12:25,  2.45it/s]

  1%|█
| 747/73196 [05:00<7:48:26,  2.58it/s]

  1%|█
| 748/73196 [05:00<7:37:06,  2.64it/s]

  1%|█
| 749/73196 [05:00<7:24:52,  2.71it/s]

  1%|█
| 750/73196 [05:01<7:15:58,  2.77it/s]

  1%|█
| 751/73196 [05:01<7:08:22,  2.82it/s]
```

```
  1%|█
| 752/73196 [05:01<7:00:25,  2.87it/s]

  1%|█
| 753/73196 [05:02<6:54:19,  2.91it/s]

  1%|█
| 754/73196 [05:02<7:17:34,  2.76it/s]

  1%|█
| 755/73196 [05:03<8:48:47,  2.28it/s]

  1%|█
| 756/73196 [05:03<9:16:54,  2.17it/s]

  1%|█
| 757/73196 [05:04<8:35:23,  2.34it/s]

  1%|█
| 758/73196 [05:04<8:35:35,  2.34it/s]

  1%|█
| 759/73196 [05:05<8:41:11,  2.32it/s]

  1%|█
| 760/73196 [05:05<9:10:22,  2.19it/s]

  1%|█
| 761/73196 [05:05<8:45:50,  2.30it/s]

  1%|█
| 762/73196 [05:06<9:16:04,  2.17it/s]

  1%|█
 | 763/73196 [05:07<10:27:38,  1.92it/s]

  1%|█
 | 764/73196 [05:07<10:36:12,  1.90it/s]

  1%|█
 | 765/73196 [05:08<13:06:39,  1.53it/s]

  1%|█
 | 766/73196 [05:09<12:27:30,  1.61it/s]
```

```
  1%|▏
| 767/73196 [05:09<11:38:25,  1.73it/s]

  1%|▏
| 768/73196 [05:10<11:21:45,  1.77it/s]

  1%|▏
| 769/73196 [05:10<10:05:04,  1.99it/s]

  1%|▏
| 770/73196 [05:11<10:08:28,  1.98it/s]

  1%|▏
| 771/73196 [05:12<15:29:26,  1.30it/s]

  1%|▏
| 772/73196 [05:13<15:20:07,  1.31it/s]

  1%|▏
| 773/73196 [05:13<14:43:11,  1.37it/s]

  1%|▏
| 774/73196 [05:14<12:27:53,  1.61it/s]

  1%|▏
| 775/73196 [05:14<11:00:01,  1.83it/s]

  1%|▏
| 776/73196 [05:14<10:11:52,  1.97it/s]

  1%|▏
| 777/73196 [05:15<10:06:15,  1.99it/s]

  1%|▏
| 778/73196 [05:16<10:34:33,  1.90it/s]

  1%|▏
| 779/73196 [05:16<10:51:25,  1.85it/s]

  1%|▏
| 780/73196 [05:17<10:45:46,  1.87it/s]

  1%|▏
  | 781/73196 [05:17<10:57:48,  1.83it/s]


   1%|█
  | 782/73196 [05:18<10:35:56,  1.90it/s]


   1%|█
  | 783/73196 [05:18<10:06:29,  1.99it/s]


   1%|█
 | 784/73196 [05:19<9:54:28,  2.03it/s]


   1%|█
 | 785/73196 [05:19<9:31:07,  2.11it/s]


   1%|█
 | 786/73196 [05:19<9:14:08,  2.18it/s]


   1%|█
 | 787/73196 [05:20<8:49:50,  2.28it/s]


   1%|█
 | 788/73196 [05:20<8:35:10,  2.34it/s]


   1%|█
 | 789/73196 [05:21<8:15:43,  2.43it/s]


   1%|█
 | 790/73196 [05:21<7:52:30,  2.55it/s]


   1%|█
 | 791/73196 [05:21<7:40:49,  2.62it/s]


   1%|█
 | 792/73196 [05:22<7:20:37,  2.74it/s]


   1%|█
 | 793/73196 [05:22<7:25:09,  2.71it/s]


   1%|█
 | 794/73196 [05:22<7:13:36,  2.78it/s]


   1%|█
 | 795/73196 [05:23<7:26:59,  2.70it/s]
```

```
  1%|▊
| 796/73196 [05:23<7:43:05,  2.61it/s]

  1%|▊
| 797/73196 [05:24<7:30:26,  2.68it/s]

  1%|▊
| 798/73196 [05:24<7:14:13,  2.78it/s]

  1%|▊
| 799/73196 [05:24<7:24:16,  2.72it/s]

  1%|▊
| 800/73196 [05:25<7:25:34,  2.71it/s]

  1%|▊
| 801/73196 [05:25<7:18:05,  2.75it/s]

  1%|▊
| 802/73196 [05:25<7:31:16,  2.67it/s]

  1%|▊
| 803/73196 [05:26<7:28:23,  2.69it/s]

  1%|▊
| 804/73196 [05:26<7:26:07,  2.70it/s]

  1%|▊
| 805/73196 [05:26<7:24:02,  2.72it/s]

  1%|▊
| 806/73196 [05:27<7:17:24,  2.76it/s]

  1%|▊
| 807/73196 [05:27<7:17:36,  2.76it/s]

  1%|▊
| 808/73196 [05:28<7:20:45,  2.74it/s]

  1%|▊
| 809/73196 [05:28<7:35:02,  2.65it/s]

  1%|▊
| 810/73196 [05:28<7:53:58,  2.55it/s]
```

```
  1%|▌
| 811/73196 [05:29<8:40:03,  2.32it/s]

  1%|▌
| 812/73196 [05:30<11:11:56,  1.80it/s]

  1%|▌
| 813/73196 [05:30<11:54:41,  1.69it/s]

  1%|▌
| 814/73196 [05:31<11:47:51,  1.70it/s]

  1%|▌
| 815/73196 [05:32<11:25:42,  1.76it/s]

  1%|▌
| 816/73196 [05:32<10:58:17,  1.83it/s]

  1%|▌
| 817/73196 [05:33<10:46:19,  1.87it/s]

  1%|▌
| 818/73196 [05:33<10:20:58,  1.94it/s]

  1%|▌
| 819/73196 [05:33<10:06:06,  1.99it/s]

  1%|▌
| 820/73196 [05:34<9:48:29,  2.05it/s]

  1%|▌
| 821/73196 [05:35<12:42:23,  1.58it/s]

  1%|▌
| 822/73196 [05:36<14:26:25,  1.39it/s]

  1%|▌
| 823/73196 [05:37<16:13:31,  1.24it/s]

  1%|▌
| 824/73196 [05:38<17:57:43,  1.12it/s]

  1%|▌
| 825/73196 [05:39<17:19:08,  1.16it/s]
```

```
  1%|█
| 826/73196 [05:39<15:24:59,  1.30it/s]

  1%|█
| 827/73196 [05:40<13:19:46,  1.51it/s]

  1%|█
| 828/73196 [05:40<11:23:42,  1.76it/s]

  1%|█
| 829/73196 [05:41<11:58:33,  1.68it/s]

  1%|█
| 830/73196 [05:41<10:54:33,  1.84it/s]

  1%|█
| 831/73196 [05:42<12:18:12,  1.63it/s]

  1%|█
| 832/73196 [05:43<15:37:08,  1.29it/s]

  1%|█
| 833/73196 [05:44<16:57:55,  1.18it/s]

  1%|█
| 834/73196 [05:45<15:20:05,  1.31it/s]

  1%|█
| 835/73196 [05:45<13:27:27,  1.49it/s]

  1%|█
| 836/73196 [05:45<11:55:19,  1.69it/s]

  1%|█
| 837/73196 [05:46<11:51:05,  1.70it/s]

  1%|█
| 838/73196 [05:46<10:23:41,  1.93it/s]

  1%|█
| 839/73196 [05:47<9:50:39,  2.04it/s]

  1%|█
```

```
| 840/73196 [05:47<9:19:56,  2.15it/s]

  1%|█
 | 841/73196 [05:48<11:43:42,  1.71it/s]

  1%|█
 | 842/73196 [05:49<13:15:39,  1.52it/s]

  1%|█
 | 843/73196 [05:50<14:40:14,  1.37it/s]

  1%|█
 | 844/73196 [05:51<14:40:44,  1.37it/s]

  1%|█
 | 845/73196 [05:51<13:00:02,  1.55it/s]

  1%|█
 | 846/73196 [05:51<11:50:03,  1.70it/s]

  1%|█
 | 847/73196 [05:52<12:18:48,  1.63it/s]

  1%|█
 | 848/73196 [05:53<11:34:07,  1.74it/s]

  1%|█
 | 849/73196 [05:53<10:13:18,  1.97it/s]

  1%|█
| 850/73196 [05:53<9:34:13,  2.10it/s]

  1%|█
| 851/73196 [05:54<9:22:43,  2.14it/s]

  1%|█
| 852/73196 [05:54<9:33:06,  2.10it/s]

  1%|█
| 853/73196 [05:55<9:40:27,  2.08it/s]

  1%|█
 | 854/73196 [05:55<10:23:05,  1.94it/s]
```

```
  1%|▌
| 855/73196 [05:56<9:53:38,  2.03it/s]

  1%|▌
| 856/73196 [05:56<9:15:36,  2.17it/s]

  1%|▌
| 857/73196 [05:57<8:58:03,  2.24it/s]

  1%|▌
| 858/73196 [05:57<9:05:15,  2.21it/s]

  1%|▌
| 859/73196 [05:57<8:28:17,  2.37it/s]

  1%|▌
| 860/73196 [05:58<8:12:58,  2.45it/s]

  1%|▌
| 861/73196 [05:58<8:15:08,  2.43it/s]

  1%|▌
| 862/73196 [05:59<8:00:51,  2.51it/s]

  1%|▌
| 863/73196 [05:59<7:43:26,  2.60it/s]

  1%|▌
| 864/73196 [05:59<7:28:11,  2.69it/s]

  1%|▌
| 865/73196 [06:00<7:15:33,  2.77it/s]

  1%|▌
| 866/73196 [06:00<7:13:29,  2.78it/s]

  1%|▌
| 867/73196 [06:01<8:29:19,  2.37it/s]

  1%|▌
| 868/73196 [06:01<7:58:03,  2.52it/s]

  1%|▌
| 869/73196 [06:01<7:37:16,  2.64it/s]
```

```
  1%|█
| 870/73196 [06:02<7:21:37,  2.73it/s]

  1%|█
| 871/73196 [06:02<7:11:02,  2.80it/s]

  1%|█
| 872/73196 [06:03<8:57:35,  2.24it/s]

  1%|█
| 873/73196 [06:03<9:21:01,  2.15it/s]

  1%|█
 | 874/73196 [06:04<10:06:34,  1.99it/s]

  1%|█
| 875/73196 [06:04<9:42:40,  2.07it/s]

  1%|█
| 876/73196 [06:04<8:59:01,  2.24it/s]

  1%|█
| 877/73196 [06:05<8:32:57,  2.35it/s]

  1%|█
| 878/73196 [06:05<8:18:16,  2.42it/s]

  1%|█
| 879/73196 [06:06<8:02:12,  2.50it/s]

  1%|█
| 880/73196 [06:06<8:47:44,  2.28it/s]

  1%|█
| 881/73196 [06:07<8:31:13,  2.36it/s]

  1%|█
| 882/73196 [06:07<8:09:10,  2.46it/s]

  1%|█
| 883/73196 [06:07<7:56:13,  2.53it/s]

  1%|█
| 884/73196 [06:08<8:32:57,  2.35it/s]
```

```
  1%|█
| 885/73196 [06:08<9:07:39,  2.20it/s]

  1%|█
| 886/73196 [06:09<8:56:27,  2.25it/s]

  1%|█
| 887/73196 [06:09<8:29:01,  2.37it/s]

  1%|█
| 888/73196 [06:09<8:21:32,  2.40it/s]

  1%|█
| 889/73196 [06:10<8:04:54,  2.49it/s]

  1%|█
| 890/73196 [06:10<8:02:03,  2.50it/s]

  1%|█
| 891/73196 [06:11<8:02:25,  2.50it/s]

  1%|█
| 892/73196 [06:11<8:32:31,  2.35it/s]

  1%|█
| 893/73196 [06:12<8:49:18,  2.28it/s]

  1%|█
| 894/73196 [06:12<8:48:19,  2.28it/s]

  1%|█
| 895/73196 [06:12<8:34:41,  2.34it/s]

  1%|█
| 896/73196 [06:13<8:30:48,  2.36it/s]

  1%|█
| 897/73196 [06:13<8:22:58,  2.40it/s]

  1%|█
| 898/73196 [06:14<8:17:25,  2.42it/s]

  1%|█
```

```
| 899/73196 [06:14<8:07:21,  2.47it/s]

  1%|█
| 900/73196 [06:14<8:11:39,  2.45it/s]

  1%|█
| 901/73196 [06:15<8:20:40,  2.41it/s]

  1%|█
| 902/73196 [06:15<8:09:16,  2.46it/s]

  1%|█
| 903/73196 [06:16<8:11:44,  2.45it/s]

  1%|█
| 904/73196 [06:16<8:06:49,  2.47it/s]

  1%|█
| 905/73196 [06:17<8:12:34,  2.45it/s]

  1%|█
| 906/73196 [06:17<8:09:43,  2.46it/s]

  1%|█
| 907/73196 [06:17<8:01:57,  2.50it/s]

  1%|█
| 908/73196 [06:18<8:01:12,  2.50it/s]

  1%|█
| 909/73196 [06:18<7:57:03,  2.53it/s]

  1%|█
| 910/73196 [06:18<7:50:26,  2.56it/s]

  1%|█
| 911/73196 [06:19<8:13:50,  2.44it/s]

  1%|█
| 912/73196 [06:19<8:10:28,  2.46it/s]

  1%|█
| 913/73196 [06:20<8:02:29,  2.50it/s]
```

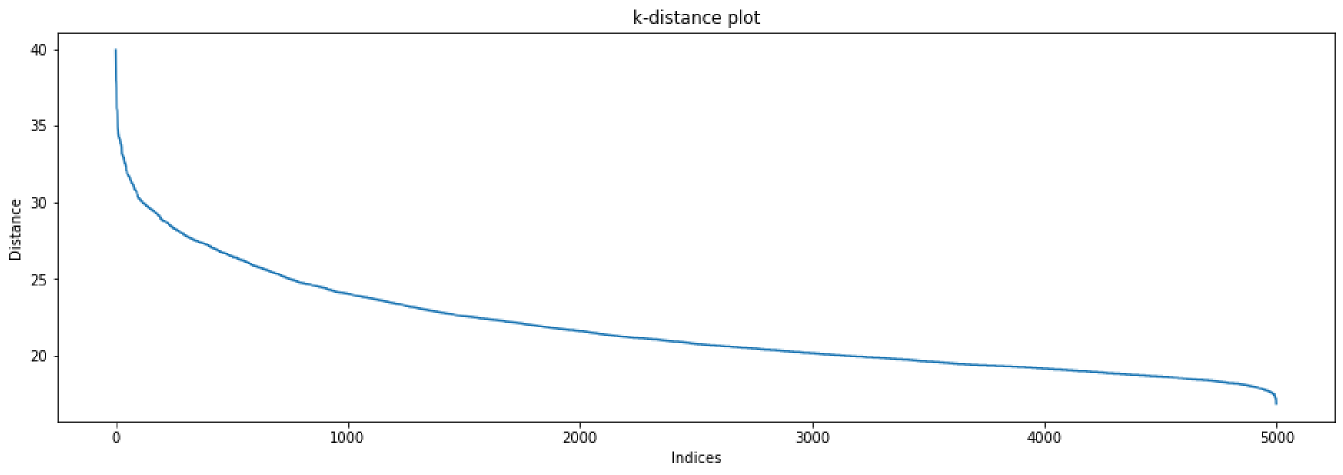
```
  1%|█
| 914/73196 [06:20<8:00:11,  2.51it/s]

  1%|█
| 915/73196 [06:21<8:08:13,  2.47it/s]

  1%|█
| 916/73196 [06:21<8:07:53,  2.47it/s]

  1%|█
| 917/73196 [06:21<8:06:46,  2.47it/s]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-69-63cf1df76908> in <module>()
     10 
     11 for point in tqdm(datt):
---> 12     temp = euclidean_distances(datt, point.reshape(1, -1))
     13     distance.append(temp[min_points])
     14 

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\metrics\pairwise.p
y in euclidean_distances(X, Y, Y_norm_squared, squared, X_norm_squared)
    230     paired_distances : distances betweens pairs of elements of X and Y.
    231     """
--> 232     X, Y = check_pairwise_arrays(X, Y)
    233 
    234     # If norms are passed as float32, they are unused. If arrays are pas
sed as

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\metrics\pairwise.p
y in check_pairwise_arrays(X, Y, precomputed, dtype)
    110     else:
    111         X = check_array(X, accept_sparse='csr', dtype=dtype,
--> 112                         estimator=estimator)
    113         Y = check_array(Y, accept_sparse='csr', dtype=dtype,
    114                         estimator=estimator)

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\utils\validation.p
y in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,
 force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features,
warn_on_dtype, estimator)
    540         if force_all_finite:
    541             _assert_all_finite(array,
--> 542                                allow_nan=force_all_finite == 'allow-na
n')
    543 
    544     if ensure_min_samples > 0:

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\utils\validation.p
y in _assert_all_finite(X, allow_nan)
     47     # safely to reduce dtype induced overflows.
     48     is_float = X.dtype.kind in 'fc'
---> 49     if is_float and (np.isfinite(_safe_accumulator_op(np.sum, X))):
     50         pass
     51     elif is_float:

~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\utils\extmath.py i
n _safe_accumulator_op(op, x, *args, **kwargs)
    686         result = op(x, *args, **kwargs, dtype=np.float64)
    687     else:
--> 688         result = op(x, *args, **kwargs)
    689     return result
    690 

<__array_function__ internals> in sum(*args, **kwargs)

~\AppData\Local\Continuum\anaconda3\lib\site-packages\numpy\core\fromnumeric.py
 in sum(a, axis, dtype, out, keepdims, initial, where)
   2180 
   2181     return _wrapreduction(a, np.add, 'sum', axis, dtype, out, keepdims=k
eepdims,
-> 2182                                initial=initial, where=where)
   2183
   2184

~\AppData\Local\Continuum\anaconda3\lib\site-packages\numpy\core\fromnumeric.py
 in _wrapreduction(obj, ufunc, method, axis, dtype, out, **kwargs)
     88                     return reduction(axis=axis, out=out, **passkwargs)
     89
---> 90     return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
     91
     92

KeyboardInterrupt:
```

In [ ]:
```python
plt.figure( figsize=(16,5))
plt.plot(points , sorted_dist )
plt.ylabel('Distance')
plt.xlabel('Indices')
plt.title('k-distance plot')
plt.show()

from IPython.display import Image
Image("~/DBSCAN.png")
```

In [45]:
```python
from IPython.display import Image
Image("DBSCAN.png")
```

Out[45]:



In [ ]:
```python

```

In [ ]:
```python
#we can see that point of inflexion is at eps=90
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=90,n_jobs=-1)
dbscan.fit(dat)
print('No of clusters: ',len(set(dbscan.labels_)))
print('Cluster are including noise i.e -1: ',set(dbscan.labels_))
```

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

In [ ]:
```
for i in range(3):
    print('%s\n'%(cluster1[i]))
```

3. Conclusion

In [56]:
```
# Please compare all your models using Prettytable library
# Please compare all your models using Prettytable library

from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

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

x.add_row(["TFIDF", "SGDClassifier",  6])
x.add_row(["TFIDF", "SGDClassifier", 5])
x.add_row(["TFIDF", "SGDClassifier", 90])

print(x)
```

```
+------------+---------------+-----------------+
| Vectorizer |     Model     | Hyper Parameter |
+------------+---------------+-----------------+
|   TFIDF    | SGDClassifier |        6        |
|   TFIDF    | SGDClassifier |        5        |
|   TFIDF    | SGDClassifier |        90       |
+------------+---------------+-----------------+
```

In [ ]: