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). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy`

Feature	Description
	• 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
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_4:__ "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."

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
```

OBSERVATION: The above block is used to load all the libraries and module that are used in the assignment.

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv', nrows=50000)
resource_data = pd.read_csv('resources.csv')
```

OBSERVATION: The above block takes all records from the data to perform analysis.

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 (50000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
```

```
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

OBSERVATION: There are 109248 rows and 17 columns in the 'project_data' data frame.

In [4]:

```
print("Number of data points in resources data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in resources 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

OBSERVATION: The above block shows the number of columns(4) and rows(1541272) in the resources file.

1.2 Data Analysis

In [5]:

```
# PROVIDE CITATIONS TO YOUR CODE IF YOU TAKE IT FROM ANOTHER WEBSITE.
# https://matplotlib.org/gallery/pie_and_polar_charts/pie_and_donut_labels.html#sphx-glr-gallery-pie-and-polar-charts-pie-and-donut-labels-py


y_value_counts = project_data['project_is_approved'].value_counts()
print("Number of projects thar are approved for funding ", y_value_counts[1], ", (",
(y_value_counts[1]/(y_value_counts[1]+y_value_counts[0]))*100,"%)")
print("Number of projects thar are not approved for funding ", y_value_counts[0], ", (",
(y_value_counts[0]/(y_value_counts[1]+y_value_counts[0]))*100,"%)")

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(aspect="equal"))
recipe = ["Accepted", "Not Accepted"]

data = [y_value_counts[1], y_value_counts[0]]

wedges, texts = ax.pie(data, wedgeprops=dict(width=0.6), startangle=40)

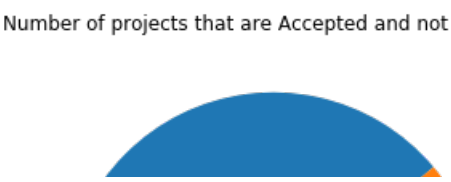
bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
kw = dict(xycoords='data', textcoords='data', arrowprops=dict(arrowstyle="-"),
          bbox=bbox_props, zorder=0, va="center")

for i, p in enumerate(wedges):
    ang = (p.theta2 - p.theta1)/2. + p.theta1
    y = np.sin(np.deg2rad(ang))
    x = np.cos(np.deg2rad(ang))
    horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
    connectionstyle = "angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle": connectionstyle})
    ax.annotate(recipe[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y),
                horizontalalignment=horizontalalignment, **kw)

ax.set_title("Number of projects that are Accepted and not accepted")

plt.show()
```

```
Number of projects thar are approved for funding  42286 , ( 84.572 %)
Number of projects thar are not approved for funding  7714 , ( 15.428 %)
```



OBSERVATION: The above block gives the percentage of projects that are approved(85%) and not approved(15%) for funding, in the form of statements and pie chart.

1.2.1 Univariate Analysis: School State

In [6]:

```
# Pandas dataframe groupby count, mean: https://stackoverflow.com/a/19385591/4084039

temp = pd.DataFrame(project_data.groupby("school_state")
["project_is_approved"].apply(np.mean)).reset_index()
#print(temp.head())
# if you have data which contain only 0 and 1, then the mean = percentage (think about it)
temp.columns = ['state_code', 'num_proposals']

'''# How to plot US state heatmap: https://datascience.stackexchange.com/a/9620

scl = [[0.0, 'rgb(242,240,247)'],[0.2, 'rgb(218,218,235)'],[0.4, 'rgb(188,189,220)'],\
            [0.6, 'rgb(158,154,200)'],[0.8, 'rgb(117,107,177)'],[1.0, 'rgb(84,39,143)']]

data = [ dict(
        type='choropleth',
        colorscale = scl,
        autocolorscale = False,
        locations = temp['state_code'],
        z = temp['num_proposals'].astype(float),
        locationmode = 'USA-states',
        text = temp['state_code'],
        marker = dict(line = dict (color = 'rgb(255,255,255)',width = 2)),
        colorbar = dict(title = "% of pro")
    ) ]

layout = dict(
        title = 'Project Proposals % of Acceptance Rate by US States',
        geo = dict(
            scope='usa',
            projection=dict( type='albers usa' ),
            showlakes = True,
            lakecolor = 'rgb(255, 255, 255)',
        ),
    )

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='us-map-heat-map')
'''
```

Out[6]:

```
'# How to plot US state heatmap: https://datascience.stackexchange.com/a/9620\n\nscl = [[0.0, \'rg
b(242,240,247)\'],[0.2, \'rgb(218,218,235)\'],[0.4, \'rgb(188,189,220)\'],            [0.6, \'rgb(1
58,154,200)\'],[0.8, \'rgb(117,107,177)\'],[1.0, \'rgb(84,39,143)\']]\n\ndata = [ dict(\n        ty
pe=\'choropleth\',\n        colorscale = scl,\n        autocolorscale = False,\n        locations =
temp[\'state_code\'],\n        z = temp[\'num_proposals\'].astype(float),\n        locationmode = \
'USA-states\',\n        text = temp[\'state_code\'],\n        marker = dict(line = dict (color = \'
rgb(255,255,255)\',width = 2)),\n        colorbar = dict(title = "% of pro")\n    ) ]\n\nlayout = d
ict(\n        title = \'Project Proposals % of Acceptance Rate by US States\',\n        geo = dict(
\n            scope=\'usa\',\n            projection=dict( type=\'albers usa\' ),\n            show
akes = True,\n            lakecolor = \'rgb(255, 255, 255)\',\n        ),\n    )\n\nfig =
go.Figure(data=data, layout=layout)\noffline.iplot(fig, filename=\'us-map-heat-map\')\n'
```

OBSERVATION: The above block groups the 'project_data' data frame on 'school_state' and calculates the mean of the total projects approved per state to store in temp.

In [7]:

```
# https://www.csi.cuny.edu/sites/default/files/pdf/administration/ops/2letterstabbrev.pdf
temp.sort_values(by=['num_proposals'], inplace=True)
print("States with lowest % approvals")
print(temp.head(5))
print('='*50)
print("States with highest % approvals")
print(temp.tail(5))
```

```
States with lowest % approvals
   state_code  num_proposals
26         MT       0.764151
46         VT       0.781250
7          DC       0.785425
43         TX       0.803916
13         ID       0.804636
==================================================
States with highest % approvals
   state_code  num_proposals
35         OH       0.877966
30         NH       0.879433
6          CT       0.882429
47         WA       0.888486
28         ND       0.904762
```

OBSERVATION: Above block lists out the states with their project approval percentage. The state VT has the lowest approval percentage(80%) and the highest value(89%) belongs to DE.

In [8]:

```
#stacked bar plots matplotlib:
https://matplotlib.org/gallery/lines_bars_and_markers/bar_stacked.html
def stack_plot(data, xtick, col2='project_is_approved', col3='total'):
    ind = np.arange(data.shape[0])

    plt.figure(figsize=(20,5))
    p1 = plt.bar(ind, data[col3].values)
    p2 = plt.bar(ind, data[col2].values)

    plt.ylabel('Projects')
    plt.title('Number of projects aproved vs rejected')
    plt.xticks(ind, list(data[xtick].values))
    plt.legend((p1[0], p2[0]), ('total', 'accepted'))
    plt.show()
```

In [9]:

```
def univariate_barplots(data, col1, col2='project_is_approved', top=False):
    # Count number of zeros in dataframe python: https://stackoverflow.com/a/51540521/4084039
    temp = pd.DataFrame(project_data.groupby(col1)[col2].agg(lambda x: x.eq(1).sum())).reset_index(
)

    # Pandas dataframe grouby count: https://stackoverflow.com/a/19385591/4084039
    temp['total'] = pd.DataFrame(project_data.groupby(col1)
[col2].agg({'total':'count'})).reset_index()['total']
    temp['Avg'] = pd.DataFrame(project_data.groupby(col1)[col2].agg({'Avg':'mean'})).reset_index()[
'Avg']

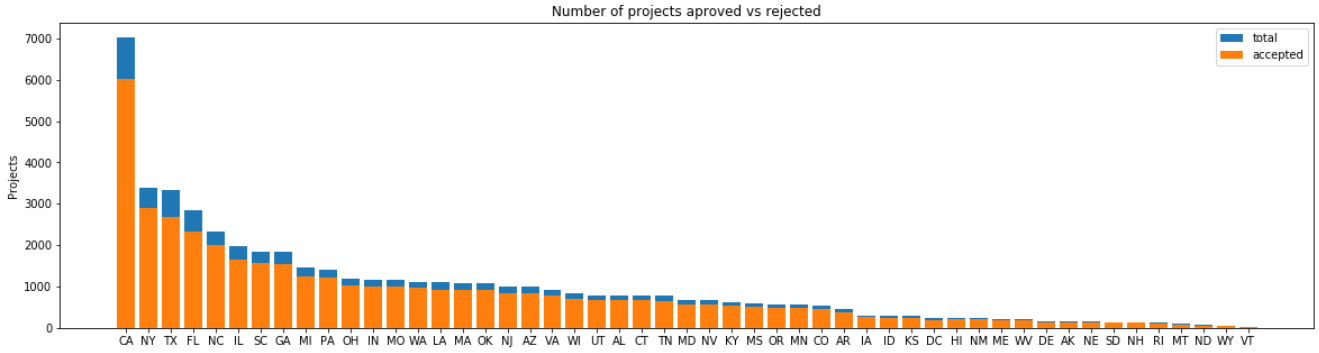
    temp.sort_values(by=['total'],inplace=True, ascending=False)

    if top:
        temp = temp[0:top]

    stack_plot(temp, xtick=col1, col2=col2, col3='total')
    print(temp.head(5))
    print("="*50)
    print(temp.tail(5))
```

In [10]:

```
univariate_barplots(project_data, 'school_state', 'project_is_approved', False)
```




```
   school_state  project_is_approved  total       Avg
4            CA                 6013   7024  0.856065
34           NY                 2903   3393  0.855585
43           TX                 2669   3320  0.803916
9            FL                 2328   2839  0.820007
27           NC                 2006   2340  0.857265
==================================================
   school_state  project_is_approved  total       Avg
39           RI                  107    126  0.849206
26           MT                   81    106  0.764151
28           ND                   57     63  0.904762
50           WY                   42     51  0.823529
46           VT                   25     32  0.781250
```

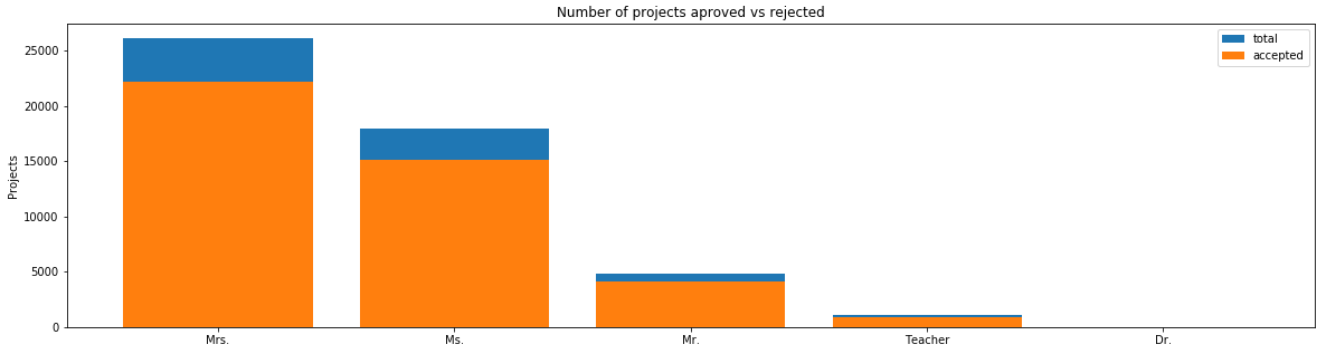
SUMMARY: Every state has greater than 80% success rate in approval

OBSERVATION: The above plot and the output show that the approval percentage of all the states remain almost same, regardless of the state and the number of projects proposed by those states. The state CA has proposed most number of projects and VT proposed least number of projects with approval percentage of 85.81 and 80 respectively.

1.2.2 Univariate Analysis: teacher_prefix

In [11]:

```
univariate_barplots(project_data, 'teacher_prefix', 'project_is_approved' , top=False)
```



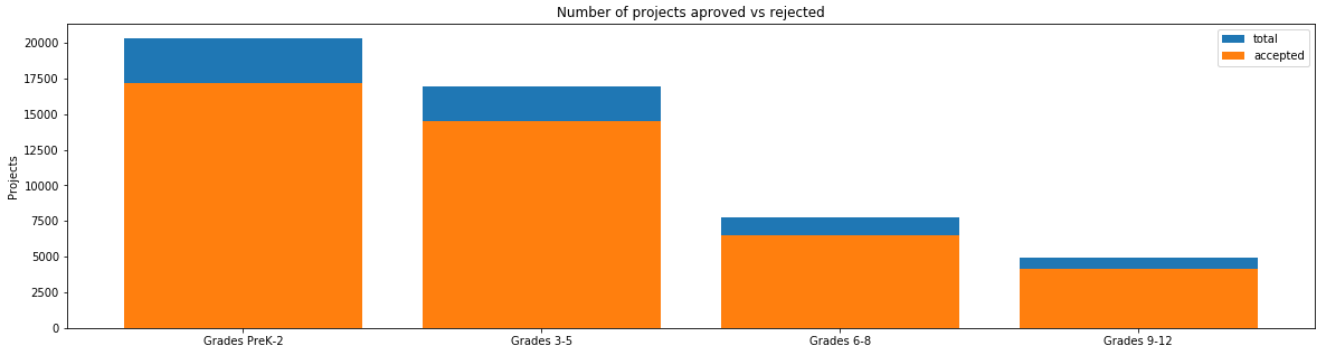

```
  teacher_prefix  project_is_approved  total       Avg
2           Mrs.                22239  26140  0.850765
3            Ms.                15124  17936  0.843220
1            Mr.                 4073   4859  0.838238
4        Teacher                  847   1061  0.798303
0            Dr.                    1      2  0.500000
==================================================
  teacher_prefix  project_is_approved  total       Avg
2           Mrs.                22239  26140  0.850765
3            Ms.                15124  17936  0.843220
1            Mr.                 4073   4859  0.838238
4        Teacher                  847   1061  0.798303
0            Dr.                    1      2  0.500000
```

OBSERVATION: The above info shows that there are more projects proposed by teachers with 'Mrs.' prefix followed by 'Ms.' and 'Mr.'. There are very few projects proposed by teachers having a doctorate. This may be because, there may be more number of lady teachers and very few techers with a doctorate.

1.2.3 Univariate Analysis: project_grade_category

In [12]:

```
univariate_barplots(project_data, 'project_grade_category', 'project_is_approved', top=False)
```




```
  project_grade_category  project_is_approved  total       Avg
3          Grades PreK-2                17185  20316  0.845885
0             Grades 3-5                14466  16968  0.852546
1             Grades 6-8                 6503   7750  0.839097
2            Grades 9-12                 4132   4966  0.832058
==================================================
  project_grade_category  project_is_approved  total       Avg
3          Grades PreK-2                17185  20316  0.845885
0             Grades 3-5                14466  16968  0.852546
1             Grades 6-8                 6503   7750  0.839097
2            Grades 9-12                 4132   4966  0.832058
```

OBSERVATION: Above plots show that approval percentage of projects remain almost same for all the grades. But the number of projects sent for approval reduces at higher grades.

1.2.4 Univariate Analysis: project_subject_categories

In [13]:

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

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

In [14]:

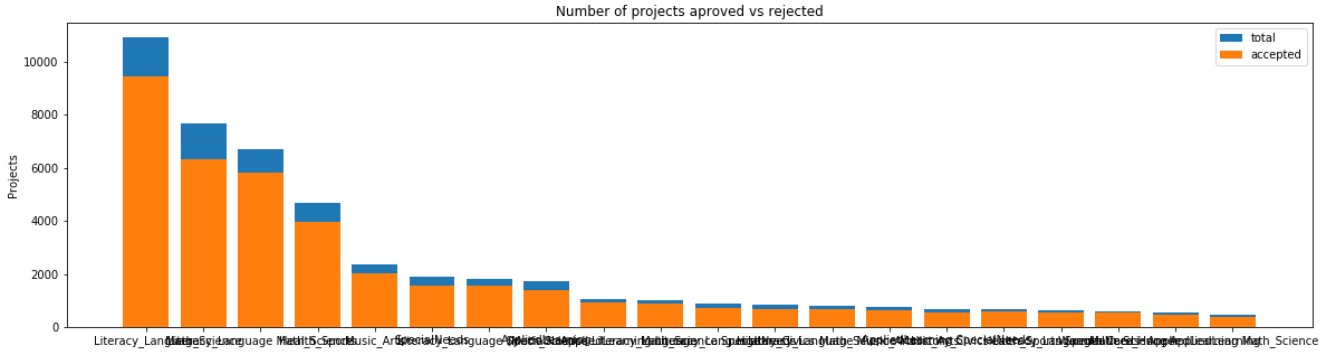
```
project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
project_data.head(2)
```

Out[14]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra

In [15]:

```
univariate_barplots(project_data, 'clean_categories', 'project_is_approved', top=20)
```




```
                clean_categories  project_is_approved  total       Avg
23               Literacy_Language                 9444  10927  0.864281
31                    Math_Science                 6307   7695  0.819623
27  Literacy_Language Math_Science                 5805   6705  0.865772
8                    Health_Sports                 3976   4700  0.845957
39                      Music_Arts                 2021   2358  0.857082
==================================================
                   clean_categories  project_is_approved  total       Avg
19  History_Civics Literacy_Language                  595    651  0.913978
14        Health_Sports SpecialNeeds                  547    633  0.864139
49                Warmth Care_Hunger                  561    606  0.925743
32      Math_Science AppliedLearning                  474    565  0.838938
4       AppliedLearning Math_Science                  386    477  0.809224
```

OBSERVATION: More number of projects are suggested for the 'Literacy_Language' category and less number of categories for 'AppliedLearning Math_Science'. Even though we don't see a major change in approval percentage for any category, 'Warmth Care_Hunger' has high approval percentage. May be these projects are related to some of the basic amenities required by the students.

In [16]:

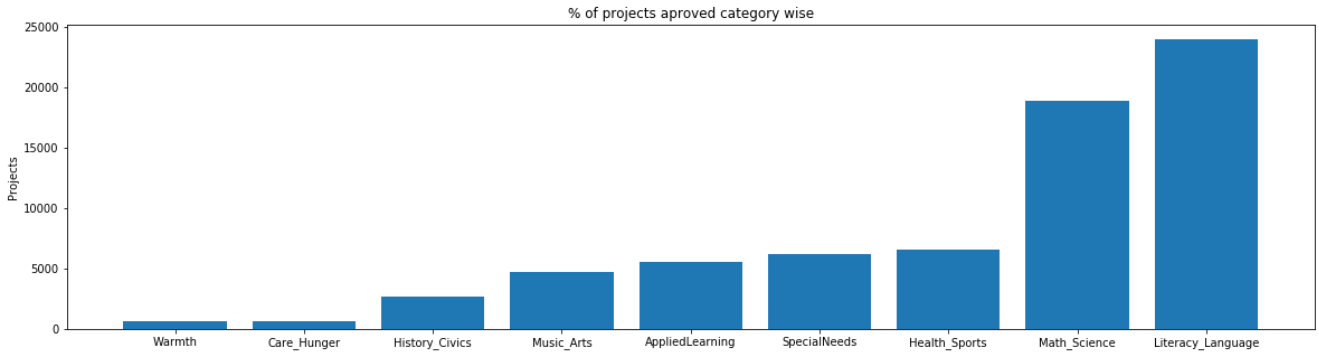
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())
```

In [17]:

```
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))


ind = np.arange(len(sorted_cat_dict))
plt.figure(figsize=(20,5))
p1 = plt.bar(ind, list(sorted_cat_dict.values()))
```

```
plt.ylabel('Projects')
plt.title('% of projects aproved category wise')
plt.xticks(ind, list(sorted_cat_dict.keys()))
plt.show()
```



In [18]:

```
for i, j in sorted_cat_dict.items():
    print("{:20} :{:10}".format(i,j))
```

```
Warmth               :       643
Care_Hunger          :       643
History_Civics       :      2689
Music_Arts           :      4699
AppliedLearning      :      5569
SpecialNeeds         :      6233
Health_Sports        :      6538
Math_Science         :     18874
Literacy_Language    :     23998
```

OBSERVATION: When projects with multiple categories are split and the number of projects for each category are taken, the counts will be as shown above.

1.2.5 Univariate Analysis: project_subject_subcategories

In [19]:

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

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

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

In [20]:

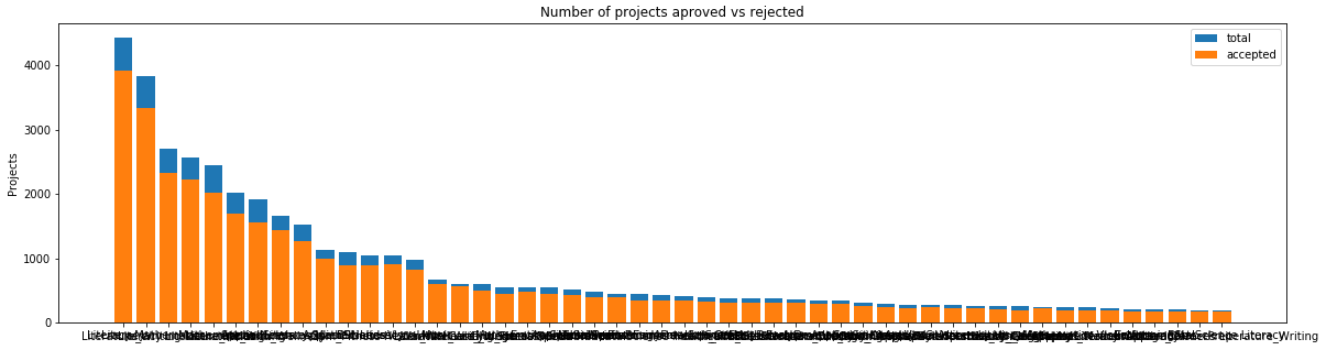
```
project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
project_data.head(2)
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra

In [21]:

```
univariate_barplots(project_data, 'clean_subcategories', 'project_is_approved', top=50)
```




```
                clean_subcategories  project_is_approved  total       Avg
303                        Literacy                 3919   4434  0.883852
305             Literacy Mathematics                3336   3833  0.870337
317  Literature_Writing Mathematics                 2331   2705  0.861738
304     Literacy Literature_Writing                 2217   2570  0.862646
327                     Mathematics                 2013   2441  0.824662
==================================================
                   clean_subcategories  project_is_approved  total       Avg
3    AppliedSciences College_CareerPrep                  169    201  0.840796
367                      PerformingArts                  177    200  0.885000
125                                 ESL                  171    199  0.859296
188       EnvironmentalScience Literacy                  169    195  0.866667
17   AppliedSciences Literature_Writing                  165    188  0.877660
```

OBSERVATION: There is not much of a difference in the approval percentage of subcategories when there is change in the number of projects sent for approval.

In [22]:

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

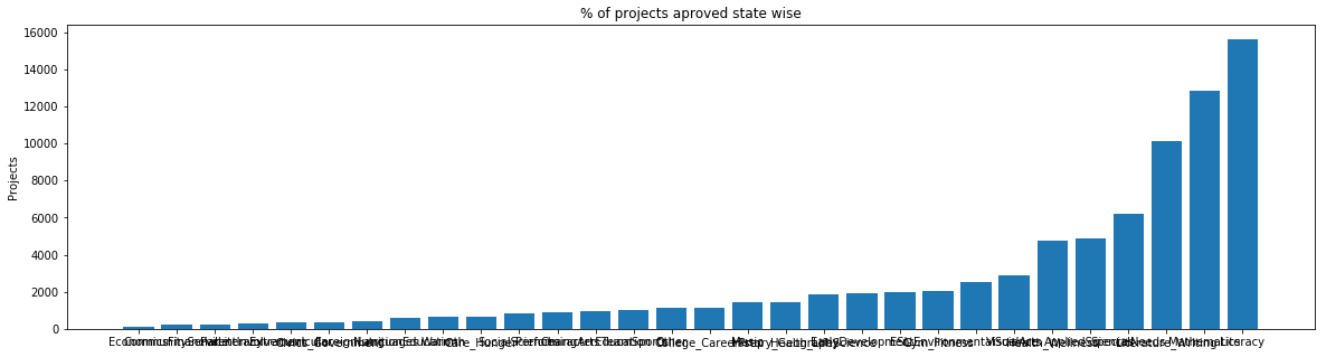
In [23]:

```
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))


ind = np.arange(len(sorted_sub_cat_dict))
plt.figure(figsize=(20,5))
p1 = plt.bar(ind, list(sorted_sub_cat_dict.values()))

plt.ylabel('Projects')
plt.title('% of projects aproved state wise')
```

```
plt.xticks(ind, list(sorted_sub_cat_dict.keys()))
plt.show()
```



In [24]:

```
for i, j in sorted_sub_cat_dict.items():
    print("{:20} :{:10}".format(i,j))
```

```
Economics            :       127
CommunityService     :       214
FinancialLiteracy    :       253
ParentInvolvement    :       302
Extracurricular      :       373
Civics_Government    :       380
ForeignLanguages     :       388
NutritionEducation   :       617
Warmth               :       643
Care_Hunger          :       643
SocialSciences       :       864
PerformingArts       :       910
CharacterEducation   :       958
TeamSports           :       995
Other                :      1128
College_CareerPrep   :      1168
Music                :      1432
History_Geography    :      1433
Health_LifeScience   :      1876
EarlyDevelopment     :      1937
ESL                  :      1999
Gym_Fitness          :      2068
EnvironmentalScience :      2533
VisualArts           :      2865
Health_Wellness      :      4732
AppliedSciences      :      4901
SpecialNeeds         :      6233
Literature_Writing   :     10127
Mathematics          :     12832
Literacy             :     15611
```

OBSERVATION: When projects with multiple subcategories are split and the number of projects for each subcategory are taken, the counts will be as shown above.

1.2.6 Univariate Analysis: Text features (Title)

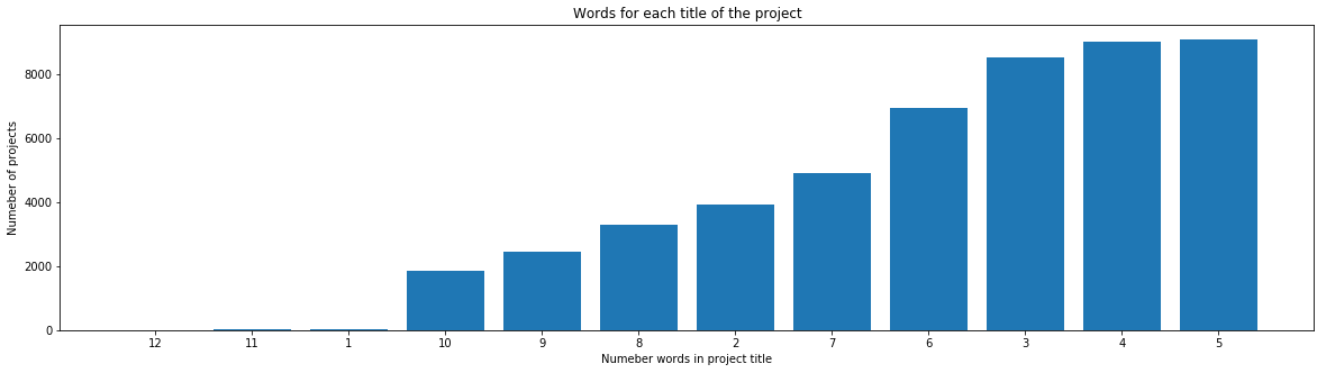
In [25]:

```
#How to calculate number of words in a string in DataFrame:
https://stackoverflow.com/a/37483537/4084039
word_count = project_data['project_title'].str.split().apply(len).value_counts()
word_dict = dict(word_count)
word_dict = dict(sorted(word_dict.items(), key=lambda kv: kv[1]))


ind = np.arange(len(word_dict))
plt.figure(figsize=(20,5))
p1 = plt.bar(ind, list(word_dict.values()))

plt.ylabel('Numeber of projects')
plt.xlabel('Numeber words in project title')
```

```
plt.title('Words for each title of the project')
plt.xticks(ind, list(word_dict.keys()))
plt.show()
```



OBSERVATION: The above plot shows that the majority of the projects contains 3,4,5 words. Where as there are very less number of projects with more number of words(11,12,13).

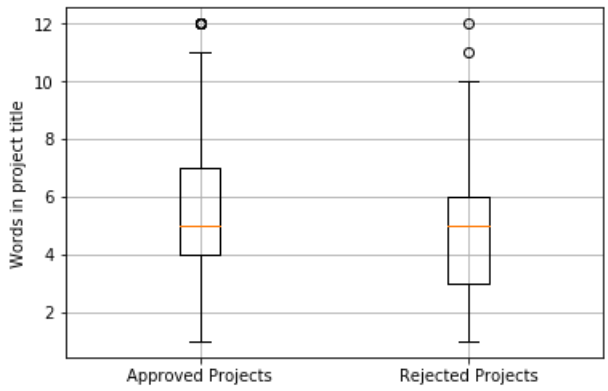
In [26]:

```
approved_title_word_count = project_data[project_data['project_is_approved']==1]['project_title'].
str.split().apply(len)
approved_title_word_count = approved_title_word_count.values

rejected_title_word_count = project_data[project_data['project_is_approved']==0]['project_title'].
str.split().apply(len)
rejected_title_word_count = rejected_title_word_count.values
```

In [27]:

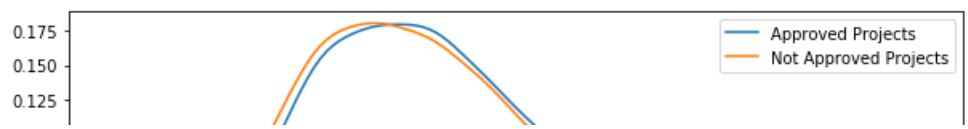
```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([approved_title_word_count, rejected_title_word_count])
plt.xticks([1,2],('Approved Projects','Rejected Projects'))
plt.ylabel('Words in project title')
plt.grid()
plt.show()
```

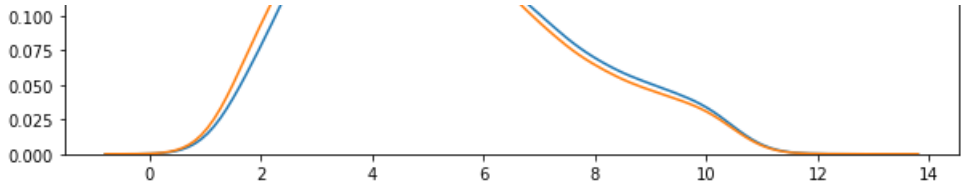


OBSERVATION: The above box plots show that the majority of approved projects contain 4-7 words in the title. And the majority un-approved projects contain 3-6 words in title. Since more number of words imply better explanation of detail, more words in title allows the projects to get approved.

In [28]:

```
plt.figure(figsize=(10,3))
sns.kdeplot(approved_title_word_count,label="Approved Projects", bw=0.6)
sns.kdeplot(rejected_title_word_count,label="Not Approved Projects", bw=0.6)
plt.legend()
plt.show()
```





OBSERVATION: The kernel density estimate plot of both approved and un-approved projects are similar to each other and we can't infer much from this.

1.2.7 Univariate Analysis: Text features (Project Essay's)

In [29]:

```
# 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)
```

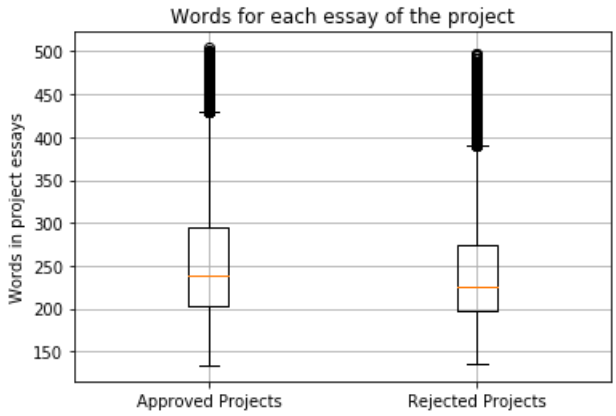
In [30]:

```
approved_word_count = project_data[project_data['project_is_approved']==1]['essay'].str.split().apply(len)
approved_word_count = approved_word_count.values

rejected_word_count = project_data[project_data['project_is_approved']==0]['essay'].str.split().apply(len)
rejected_word_count = rejected_word_count.values
```

In [31]:

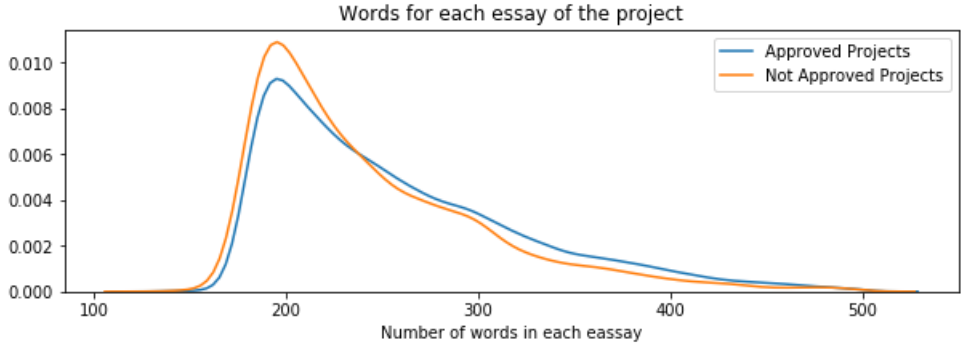
```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([approved_word_count, rejected_word_count])
plt.title('Words for each essay of the project')
plt.xticks([1,2],('Approved Projects','Rejected Projects'))
plt.ylabel('Words in project essays')
plt.grid()
plt.show()
```



OBSERVATION: The above box plots show that the projects submitted with more number of words in the essays had better chance of approval than the projects with less words in their essays. This may be due to better explanation of projects which may be an implication due to more number of words.

In [32]:

```
plt.figure(figsize=(10,3))
sns.distplot(approved_word_count, hist=False, label="Approved Projects")
sns.distplot(rejected_word_count, hist=False, label="Not Approved Projects")
plt.title('Words for each essay of the project')
plt.xlabel('Number of words in each eassay')
plt.legend()
plt.show()
```



OBSERVATION: The above plot shows that the projects with less number of words in the essays have more chance of getting rejected than the projects with more words in their essays.

1.2.8 Univariate Analysis: Cost per project

In [33]:

```
# we get the cost of the project using resource.csv file
resource_data.head(2)
```

Out[33]:

	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

In [34]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
```

Out[34]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [35]:

```
# join two dataframes in python:
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

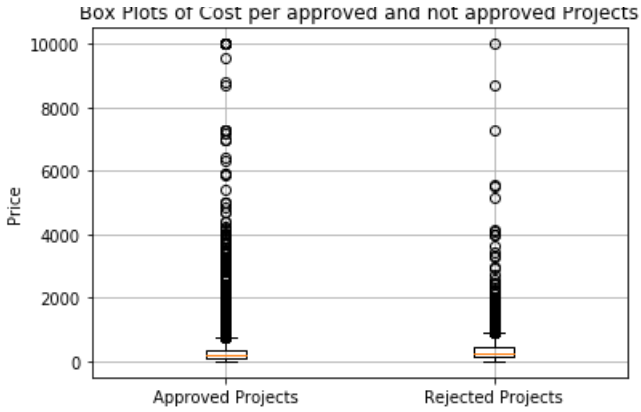
In [36]:

```
approved_price = project_data[project_data['project_is_approved']==1]['price'].values

rejected_price = project_data[project_data['project_is_approved']==0]['price'].values
```

In [37]:

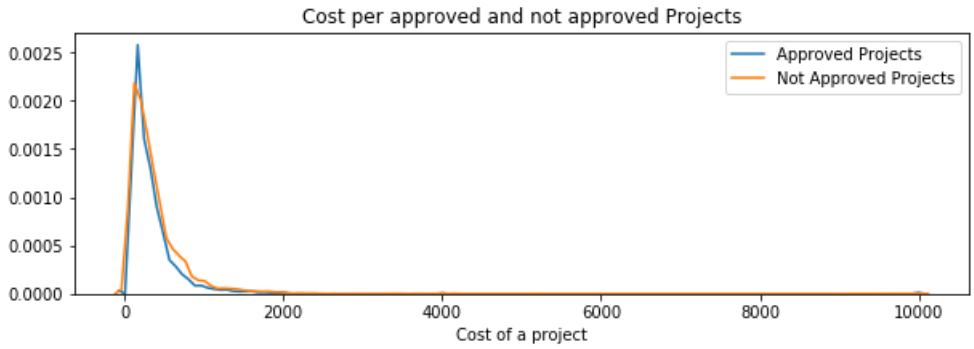
```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([approved_price, rejected_price])
plt.title('Box Plots of Cost per approved and not approved Projects')
plt.xticks([1,2],('Approved Projects','Rejected Projects'))
plt.ylabel('Price')
plt.grid()
plt.show()
```



OBSERVATION: The above box plots does not reveal much info on the role played by price of project, in order for the project to be selected.

In [38]:

```
plt.figure(figsize=(10,3))
sns.distplot(approved_price, hist=False, label="Approved Projects")
sns.distplot(rejected_price, hist=False, label="Not Approved Projects")
plt.title('Cost per approved and not approved Projects')
plt.xlabel('Cost of a project')
plt.legend()
plt.show()
```



OBSERVATION: The distribution plot also doesn't reveal much about the part played by the price of project, in order for the project to be selected.

In [39]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Percentile", "Approved Projects", "Not Approved Projects"]

for i in range(0,101,5):
    x.add_row([i,np.round(np.percentile(approved_price,i), 3), np.round(np.percentile(rejected_pric
e,i), 3)])
print(x)
```

```
+------------+------------------+------------------------+
| Percentile | Approved Projects | Not Approved Projects |
+------------+------------------+------------------------+
|      0     |        0.66      |          1.97          |
|      5     |       13.772     |         41.429         |
|     10     |       33.99      |         73.18          |
|     15     |       58.57      |         98.548         |
|     20     |       78.64      |        118.006         |
|     25     |       99.99      |        140.785         |
|     30     |      117.035     |        161.972         |
|     35     |       137.08     |        183.825         |
|     40     |       157.0      |        208.726         |
|     45     |      177.802     |        234.921         |
|     50     |      198.085     |        261.255         |
```

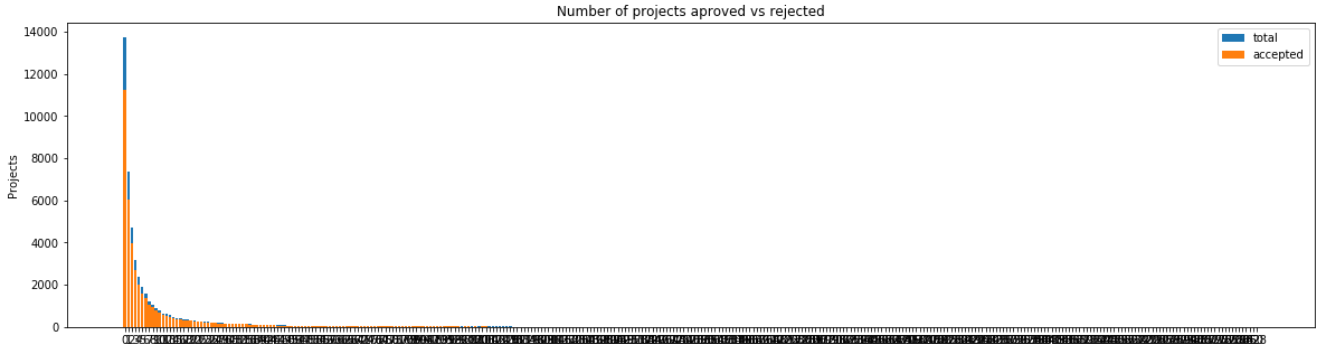
```
|      55      |       221.808       |         292.798         |
|      60      |        254.0        |         322.998         |
|      65      |       283.812       |         359.934         |
|      70      |       319.975       |          399.99         |
|      75      |       364.265       |         446.437         |
|      80      |        409.98       |         517.156         |
|      85      |        479.0        |         615.586         |
|      90      |        592.99       |         742.624         |
|      95      |        805.89       |         991.616         |
|     100      |        9999.0       |          9999.0         |
+--------------+---------------------+-------------------------+
```

OBSERVATION: The percentiles in above table reveals that, the price quoted for the approved projects are comparitively less than that of the non-approved projects.

1.2.9 Univariate Analysis: teacher_number_of_previously_posted_projects

In [40]:

```
univariate_barplots(project_data, 'teacher_number_of_previously_posted_projects',
'project_is_approved' , top=False)
```




```
   teacher_number_of_previously_posted_projects  project_is_approved  total  \
0                                             0                11260  13735
1                                             1                 6055   7352
2                                             2                 3944   4709
3                                             3                 2690   3199
4                                             4                 2032   2399

        Avg
0  0.819803
1  0.823585
2  0.837545
3  0.840888
4  0.847020
==================================================
     teacher_number_of_previously_posted_projects  project_is_approved  total  \
250                                           266                   1      1
249                                           265                   1      1
247                                           262                   1      1
245                                           260                   1      1
326                                           428                   1      1

     Avg
250  1.0
249  1.0
247  1.0
245  1.0
326  1.0
```

OBSERVATION: The above analysis show that there is an approval percentage of above 80% regardless of the number of projects proposed by the teacher. However the approval percentage is more for teachers who have less number previously proposed projects.

1.2.10 Univariate Analysis: project_resource_summary

In [41]:

```
project_data.columns.values
project_data.shape
```

```
project_data.shape
```

Out[41]:

```
(50000, 20)
```

In [42]:

```
res_sum_num=project_data[project_data['project_resource_summary'].str.match('.*[0-9*].*')]#https://davidhamann.de/2017/06/26/pandas-select-elements-by-string/
```

In [43]:

```
res_sum_num.shape
```

Out[43]:

```
(7083, 20)
```

In [44]:

```
def univariate_barplots_res_sum_num(data, col1, col2='project_is_approved', top=False):
    # Count number of zeros in dataframe python: https://stackoverflow.com/a/51540521/4084039
    temp = pd.DataFrame(res_sum_num.groupby(col1)[col2].agg(lambda x: x.eq(1).sum())).reset_index()

    # Pandas dataframe grouby count: https://stackoverflow.com/a/19385591/4084039
    temp['total'] = pd.DataFrame(res_sum_num.groupby(col1)[col2].agg({'total':'count'})).reset_index()['total']
    temp['Avg'] = pd.DataFrame(res_sum_num.groupby(col1)[col2].agg({'Avg':'mean'})).reset_index()['Avg']

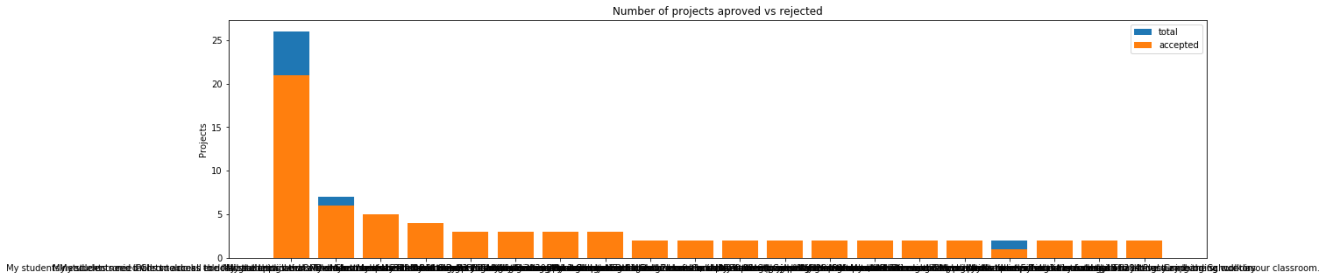
    temp.sort_values(by=['total'],inplace=True, ascending=False)

    if top:
        temp = temp[0:top]

    stack_plot(temp, xtick=col1, col2=col2, col3='total')
    print(temp.head(5))
    print("="*50)
    print(temp.tail(5))
```

In [45]:

```
univariate_barplots_res_sum_num(res_sum_num, 'project_resource_summary', 'project_is_approved', top=20)
```




```
                              project_resource_summary  project_is_approved  \
5580  My students need electronic tablets to do all ...                   21
3785  My students need Chromebooks to do all the thi...                    6
1650  My students need 3 Texas Instruments TI - 84 P...                    5
4354  My students need a Dell Chromebook 3120 and Go...                    4
5813  My students need iPads to access educational a...                    3

      total       Avg
5580     26  0.807692
3785      7  0.857143
1650      5  1.000000
4354      4  1.000000
5813      3  1.000000
==================================================
                              project resource summary  project is approved  \
```

```
2360  My students need 4 Samsung Chromebooks to rese...                  2
3807  My students need Chromebooks to prepare and en...                  1
3485  My students need 7 Hokki stools to help them s...                  2
2912  My students need 5 Texas Instruments TI - 84 P...                  2
1682  My students need 3 chromebooks for literacy an...                  2

      total  Avg
2360      2  1.0
3807      2  0.5
3485      2  1.0
2912      2  1.0
1682      2  1.0
```

OBSERVATION: The above analysis show the approval percentage of projects with numeric values in the resource summary. Based on the bar plots we can see that there is almost 90% approval rate on proposals with numbers in the resource summary.

In [46]:

```
res_sum_without_num=project_data[project_data['project_resource_summary'].str.contains('.*[0-9].*'
)==False]#https://davidhamann.de/2017/06/26/pandas-select-elements-by-string/
```

In [47]:

```
res_sum_without_num.shape
```

Out[47]:

```
(42918, 20)
```

In [48]:

```
def univariate_barplots_res_sum_without_num(data, col1, col2='project_is_approved', top=False):
    # Count number of zeros in dataframe python: https://stackoverflow.com/a/51540521/4084039
    temp = pd.DataFrame(res_sum_without_num.groupby(col1)[col2].agg(lambda x: x.eq(1).sum())).reset
_index()

    # Pandas dataframe grouby count: https://stackoverflow.com/a/19385591/4084039
    temp['total'] = pd.DataFrame(res_sum_without_num.groupby(col1)[col2].agg({'total':'count'})).re
set_index()['total']
    temp['Avg'] = pd.DataFrame(res_sum_without_num.groupby(col1)[col2].agg({'Avg':'mean'})).reset_i
ndex()['Avg']

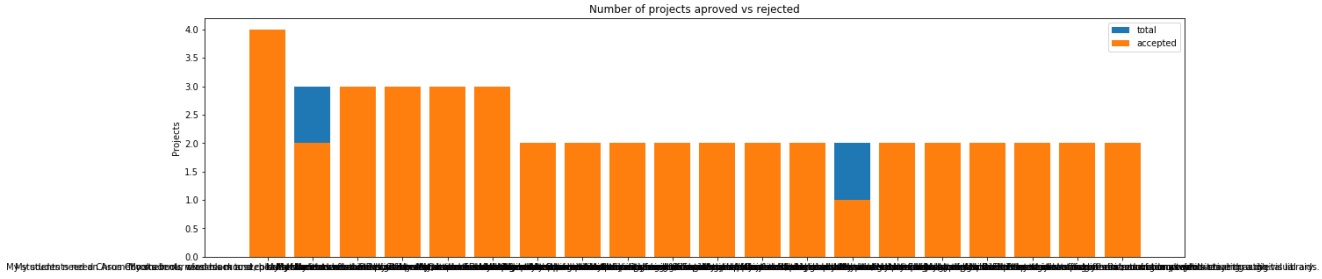
    temp.sort_values(by=['total'],inplace=True, ascending=False)

    if top:
        temp = temp[0:top]

    stack_plot(temp, xtick=col1, col2=col2, col3='total')
    print(temp.head(5))
    print("="*50)
    print(temp.tail(5))
```

In [49]:

```
univariate_barplots_res_sum_without_num(res_sum_without_num, 'project_resource_summary',
'project_is_approved', top=20)
```




```
                                project_resource_summary  project_is_approved  \
16102  My students need black and color ink for our c...                    4
13064  My students need an Asus Chromebook, wireless ...                    2
668    My students need Chromebooks in our classroom ...                    3
```

```
10639  My students need a variety of books for our cl...                    3
4327   My students need a Dell Chromebook and EDU Man...                    3

       total       Avg
16102      4  1.000000
13064      3  0.666667
668        3  1.000000
10639      3  1.000000
4327       3  1.000000
==================================================
                              project_resource_summary  project_is_approved  \
22933  My students need graphic novels to spark their...                    2
3515   My students need Sphero robotic balls, Nubby C...                    2
12994  My students need an Apple MacBook Pro to conne...                    2
25164  My students need iPads to maximize the potenti...                    2
18579  My students need colored ink to make their lea...                    2

       total  Avg
22933      2  1.0
3515       2  1.0
12994      2  1.0
25164      2  1.0
18579      2  1.0
```

OBSERVATION: The above analysis show the approval percentage of projects without numeric values in the resource summary. Based on the bar plots we can see that there is almost 100% approval rate on proposals without numbers in the resource summary. So the presence of numeric values in the resource summary does not seem to be playing a important role in project approval. But there is slight edge for the summaries with numeric values over those with numeric values.

1.3 Text preprocessing

1.3.1 Essay Text

In [50]:

```
project_data.head(2)
```

Out[50]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra

In [51]:

```
# printing some random essays.
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[5000])
print("="*50)
```

```
print(project_data['essay'].values[5999])
print("="*50)
```

```
My students are English learners that are working on English as their second or third languages. W
e are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of langua
ge to our school. \r\n\r\n We have over 24 languages represented in our English Learner program wi
th students at every level of mastery.  We also have over 40 countries represented with the
families within our school.  Each student brings a wealth of knowledge and experiences to us that
open our eyes to new cultures, beliefs, and respect.\"The limits of your language are the limits o
f your world.\"-Ludwig Wittgenstein  Our English learner's have a strong support system at home th
at begs for more resources.  Many times our parents are learning to read and speak English along s
ide of their children.  Sometimes this creates barriers for parents to be able to help their child
learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy providing these dvd's and
players, students are able to continue their mastery of the English language even if no one at hom
e is able to assist.  All families with students within the Level 1 proficiency status, will be a
offered to be a part of this program.  These educational videos will be specially chosen by the En
glish Learner Teacher and will be sent home regularly to watch.  The videos are to help the child
develop early reading skills.\r\n\r\nParents that do not have access to a dvd player will have the
opportunity to check out a dvd player to use for the year.  The plan is to use these videos and ed
ucational dvd's for the years to come for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year all love learning, at l
east most of the time. At our school, 97.3% of the students receive free or reduced price lunch. O
f the 560 students, 97.3% are minority students. \r\nThe school has a vibrant community that loves
to get together and celebrate. Around Halloween there is a whole school parade to show off the bea
utiful costumes that students wear. On Cinco de Mayo we put on a big festival with crafts made by
the students, dances, and games. At the end of the year the school hosts a carnival to celebrate t
he hard work put in during the school year, with a dunk tank being the most popular activity.My st
udents will use these five brightly colored Hokki stools in place of regular, stationary, 4-legged
chairs. As I will only have a total of ten in the classroom and not enough for each student to hav
e an individual one, they will be used in a variety of ways. During independent reading time they
will be used as special chairs students will each use on occasion. I will utilize them in place of
chairs at my small group tables during math and reading times. The rest of the day they will be us
ed by the students who need the highest amount of movement in their life in order to stay focused
on school.\r\n\r\nWhenever asked what the classroom is missing, my students always say more Hokki
Stools. They can't get their fill of the 5 stools we already have. When the students are sitting i
n group with me on the Hokki Stools, they are always moving, but at the same time doing their
work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be ta
ken. There are always students who head over to the kidney table to get one of the stools who are
disappointed as there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hours
a day. The Hokki stools will be a compromise that allow my students to do desk work and move at th
e same time. These stools will help students to meet their 60 minutes a day of movement by
allowing them to activate their core muscles for balance while they sit. For many of my students,
these chairs will take away the barrier that exists in schools for a child who can't sit
still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment with plain walls, rows of
desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work
hard to create a warm inviting themed room for my students look forward to coming to each day.\r\n
\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in
Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free a
nd reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very uniq
ue as there are no walls separating the classrooms. These 9 and 10 year-old students are very eage
r learners; they are like sponges, absorbing all the information and experiences and keep on wanti
ng more.With these resources such as the comfy red throw pillows and the whimsical nautical hangin
g decor and the blue fish nets, I will be able to help create the mood in our classroom setting to
be one of a themed nautical environment. Creating a classroom environment is very important in the
success in each and every child's education. The nautical photo props will be used with each child
as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pic
tures of each child with them, have them developed, and then hung in our classroom ready for their
first day of 4th grade.  This kind gesture will set the tone before even the first day of school!
The nautical thank you cards will be used throughout the year by the students as they create thank
you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our
classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of
my own pocket on resources to get our classroom ready. Please consider helping with this project t
o make our new school year a very successful one. Thank you!nannan
==================================================
My class is made up of students from various grade levels. We  work hard in filling learning gaps
and have students reach grade level. My students are dealing with emotional issues that make it ha
rd for them to handle frustration with tasks and need a lot of individual attention. By learning t
o work independently, my students will have the chance to mainstream into other classrooms with th
eir peer groups. Our biggest goal with my students is for them to learn not only to control their
emotions but to learn how to be students. Many of them have spent a large amount of time absent fr
om school for different reasons and need to get into the routine of being in class and on task all
day. Modeling good classroom routines and task is important for them to master and move back into
general education classrooms. Being apart of a Title 1 school means resources that students need a
```

re massive and a lot of supplies are shared with parents to make sure homework is completed.Bouncy Bands will give my students a way to get rid of anxiety, tension, and energy all while staying at their desk and working independently. Students will use the bands at either their desk or at a whole group table with a chair and avoid having to get up or be asked to stop moving. Movement is the key to keeping students with ADHD and other disabilities focused and finishing up their assignments or staying on task while the teacher is teaching. \r\n   My goal is to help my students learn helpful strategies that will allow them to join their peers in the general education setting.   By learning how to maintain focus by getting their wiggles and extra energy and grow academically.nannan
==
My students are bright eyed and starving for knowledge. My Pre-K -1 children are like a sponge, they want to learn everything and they learn so much by using their senses and intuition. Although my students are on the Autistic spectrum they are just as curious and want what all children want which is to have fun, be excepted by their peers and mentors, feel safe and comforted, explore the world around them and they want to experience and learn new things.My students have language and speech delays and verbal communication is very difficult. My students use a picture exchange communication system and are learning to read and write. \r\n\r\nThe Boogie Boards will be a great tool for my students to use. They will be able to write and use these boards to communicate. It will also motivate them to learn. The size and different textures of the stylus will make it easy for my students to hold the stylus. My students's fine motor skills are also low. \r\n\r\nThe Magnetic board, Magnetic Letters and stamps will be a great addition to help facilitate language.\r\nnannan
==

In [52]:

```
# 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 [53]:

```
sent = decontracted(project_data['essay'].values[1000])
print(sent)
print("="*50)
```

How do you remember your days of school? Was it in a sterile environment with plain walls, rows of desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work hard to create a warm inviting themed room for my students look forward to coming to each day.\r\n\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free and reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very unique as there are no walls separating the classrooms. These 9 and 10 year-old students are very eager learners; they are like sponges, absorbing all the information and experiences and keep on wanting more.With these resources such as the comfy red throw pillows and the whimsical nautical hanging decor and the blue fish nets, I will be able to help create the mood in our classroom setting to be one of a themed nautical environment. Creating a classroom environment is very important in the success in each and every child is education. The nautical photo props will be used with each child as they step foot into our classroom for the first time on Meet the Teacher evening. I will take pictures of each child with them, have them developed, and then hung in our classroom ready for their first day of 4th grade. This kind gesture will set the tone before even the first day of school! The nautical thank you cards will be used throughout the year by the students as they create thank you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of my own pocket on resources to get our classroom ready. Please consider helping with this project to make our new school year a very successful one. Thank you!nannan
==

In [54]:

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

```
How do you remember your days of school? Was it in a sterile environment with plain walls, rows of
desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work
hard to create a warm inviting themed room for my students look forward to coming to each day.
My class is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas.  They
attend a Title I school, which means there is a high enough percentage of free and reduced-price l
unch to qualify. Our school is an  open classroom  concept, which is very unique as there are no w
alls separating the classrooms. These 9 and 10 year-old students are very eager learners; they are
like sponges, absorbing all the information and experiences and keep on wanting more.With these re
sources such as the comfy red throw pillows and the whimsical nautical hanging decor and the blue
fish nets, I will be able to help create the mood in our classroom setting to be one of a themed n
autical environment. Creating a classroom environment is very important in the success in each and
every child is education. The nautical photo props will be used with each child as they step foot
into our classroom for the first time on Meet the Teacher evening. I will take pictures of each ch
ild with them, have them developed, and then hung in our classroom ready for their first day of 4t
h grade.  This kind gesture will set the tone before even the first day of school! The nautical th
ank you cards will be used throughout the year by the students as they create thank you cards to t
heir team groups.    Your generous donations will help me to help make our classroom a fun,
inviting, learning environment from day one.    It costs lost of money out of my own pocket on res
ources to get our classroom ready. Please consider helping with this project to make our new schoo
l year a very successful one. Thank you!nannan
```

In [55]:

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

```
How do you remember your days of school Was it in a sterile environment with plain walls rows of d
esks and a teacher in front of the room A typical day in our room is nothing like that I work hard
to create a warm inviting themed room for my students look forward to coming to each day My class
is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas They attend a Title
I school which means there is a high enough percentage of free and reduced price lunch to qualify
Our school is an open classroom concept which is very unique as there are no walls separating the
classrooms These 9 and 10 year old students are very eager learners they are like sponges absorbin
g all the information and experiences and keep on wanting more With these resources such as the co
mfy red throw pillows and the whimsical nautical hanging decor and the blue fish nets I will be ab
le to help create the mood in our classroom setting to be one of a themed nautical environment Cre
ating a classroom environment is very important in the success in each and every child is
education The nautical photo props will be used with each child as they step foot into our
classroom for the first time on Meet the Teacher evening I will take pictures of each child with t
hem have them developed and then hung in our classroom ready for their first day of 4th grade This
kind gesture will set the tone before even the first day of school The nautical thank you cards wi
ll be used throughout the year by the students as they create thank you cards to their team groups
Your generous donations will help me to help make our classroom a fun inviting learning environmen
t from day one It costs lost of money out of my own pocket on resources to get our classroom ready
Please consider helping with this project to make our new school year a very successful one Thank
you nannan
```

In [56]:

```
# 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', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
```

```
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [57]:

```
# Combining all the above statemennts
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%|██████████████████████████████████████████████████████████████████████████| 50000/50000 [00:
55<00:00, 893.82it/s]
```

In [58]:

```
project_data['preprocessed_essays'] = preprocessed_essays
```

1.3.2 Project title Text

In [59]:

```
# printing some random essays.
print(project_data['project_title'].values[7])
print("="*50)
print(project_data['project_title'].values[5])
print("="*50)
```

```
It's the 21st Century
==================================================
Flexible Seating for Mrs. Jarvis' Terrific Third Graders!!
==================================================
```

In [60]:

```
sent_title = decontracted(project_data['project_title'].values[7])
print(sent_title)
print("="*50)
sent_title1 = decontracted(project_data['project_title'].values[9])
print(sent_title1)
```

```
It is the 21st Century
==================================================
Just For the Love of Reading--\r\nPure Pleasure
```

In [61]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent_title1 = sent_title1.replace('\\r', ' ')
sent_title1 = sent_title1.replace('\\"', ' ')
sent_title1 = sent_title1.replace('\\n', ' ')
```

```
print(sent_title1)
```

```
Just For the Love of Reading--  Pure Pleasure
```

In [62]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_title1 = re.sub('[^A-Za-z0-9]+', ' ', sent_title1)
print(sent_title1)
```

```
Just For the Love of Reading Pure Pleasure
```

In [63]:

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

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:02<00:00, 22451.28it/s]
```

In [64]:

```
# after preprocesing
print(preprocessed_project_title[5])
print(preprocessed_project_title[7])
```

```
flexible seating mrs jarvis terrific third graders
it 21st century
```

In [65]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
title_list = []
for i in title:
    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 & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    title_list.append(temp.strip())
```

In [66]:

```
project_data['preprocessed_project_title'] = title_list
#project_data.drop(['project_title'], axis=1, inplace=True)
project_data.head(2)
```

Out[66]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra

2 rows × 22 columns

1. 4 Preparing data for models

In [67]:

```
project_data.columns
```

Out[67]:

```
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', 'price', 'quantity',
       'preprocessed_essays', 'preprocessed_project_title'],
      dtype='object')
```

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

       - quantity : numerical
       - teacher_number_of_previously_posted_projects : numerical
       - price : numerical
```

1.4.1 Vectorizing Categorical data

- https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

In [68]:

```
list(project_data.columns.values)
```

Out[68]:

```
['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',
 'price',
 'quantity',
 'preprocessed_essays',
 'preprocessed_project_title']
```

In [69]:

```
project_data['teacher_prefix'].values
```

Out[69]:

```
array(['Mrs.', 'Mr.', 'Ms.', ..., 'Mrs.', 'Mrs.', 'Mrs.'], dtype=object)
```

In [70]:

```
project_data['school_state'].values
```

Out[70]:

```
array(['IN', 'FL', 'AZ', ..., 'SD', 'CT', 'KY'], dtype=object)
```

In [71]:

```
project_data['project_grade_category'].values
```

Out[71]:

```
array(['Grades PreK-2', 'Grades 6-8', 'Grades 6-8', ..., 'Grades 3-5',
       'Grades PreK-2', 'Grades PreK-2'], dtype=object)
```

In [72]:

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
vectorizer.fit(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())


categories_one_hot = vectorizer.transform(project_data['clean_categories'].values)
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (50000, 9)
```

In [73]:

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

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(project_data['clean_subcategories'].values)
print(vectorizer.get_feature_names())


sub_categories_one_hot = vectorizer.transform(project_data['clean_subcategories'].values)
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (50000, 30)
```

In [74]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
state_dict = dict(my_counter)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))
```

In [75]:

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())


states_one_hot = vectorizer.transform(project_data['school_state'].values)
print("Shape of matrix after one hot encodig ",states_one_hot.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'NH', 'SD', 'NE', 'AK', 'DE', 'WV', 'ME', 'NM', 'HI', 'DC', 'KS', 'I
D', 'IA', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD', 'CT', 'TN', 'AL', 'UT', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'MA', 'LA', 'WA', 'MO', 'IN', 'OH', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
Shape of matrix after one hot encodig  (50000, 51)
```

In [76]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    #print(word)
    my_counter.update(word.split('  '))
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grades_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grades_dict.items(), key=lambda kv: kv[1]))
```

In [77]:

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(project_data['project_grade_category'].values)
print(vectorizer.get_feature_names())


grades_one_hot = vectorizer.transform(project_data['project_grade_category'].values)
print("Shape of matrix after one hot encodig ",grades_one_hot.shape)
```

```
['Grades 9-12', 'Grades 6-8', 'Grades 3-5', 'Grades PreK-2']
```

```
Shape of matrix after one hot encodig  (50000, 4)
```

In [78]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('')
for word in project_data['teacher_prefix'].values:
    #print(word)
    my_counter.update(word.split())
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacherprefix_dict = dict(my_counter)
sorted_teacherprefix_dict = dict(sorted(teacherprefix_dict.items(), key=lambda kv: kv[1]))
```

In [79]:

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_teacherprefix_dict.keys()), lowercase=False, b
inary=True)
vectorizer.fit(project_data['teacher_prefix'].values)
print(vectorizer.get_feature_names())


teacherprefix_one_hot = vectorizer.transform(project_data['teacher_prefix'].values)
print("Shape of matrix after one hot encodig ",teacherprefix_one_hot.shape)
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encodig  (50000, 5)
```

1.4.2 Vectorizing Text data

1.4.2.1 Bag of words

In [80]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (50000, 12211)
```

In [81]:

```
type(text_bow)
```

Out[81]:

```
scipy.sparse.csr.csr_matrix
```

1.4.2.2 Bag of Words on `project_title`

In [82]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
title_bow = vectorizer.fit_transform(preprocessed_project_title)
print("Shape of matrix after one hot encodig ",title_bow.shape)
```

```
Shape of matrix after one hot encodig  (50000, 2135)
```

In [83]:

```
type(title_bow)
```

Out[83]:

```
scipy.sparse.csr.csr_matrix
```

1.4.2.3 TFIDF vectorizer

In [84]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (50000, 12211)
```

1.4.2.4 TFIDF Vectorizer on `project_title`

In [85]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
title_tfidf = vectorizer.fit_transform(preprocessed_project_title)
print("Shape of matrix after one hot encodig ",title_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (50000, 2135)
```

1.4.2.5 Using Pretrained Models: Avg W2V

In [86]:

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

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
```

```
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/

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


'''
```

Out[86]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\n
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [87]:

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

In [88]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors.append(vector)

print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))
#print(avg_w2v_vectors[0])
```

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:28<00:00, 1745.18it/s]
```

```
50000
300
```

1.4.2.6 Using Pretrained Models: AVG W2V on `project_title`

In [89]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_project_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_title.append(vector)

print(len(avg_w2v_title))
print(len(avg_w2v_title[0]))
#print(avg_w2v_title[0])
```

```
100%|████████████████████████████████████████████████████████████████████████████| 50000/50000
[00:01<00:00, 44238.11it/s]
```

```
50000
300
```

1.4.2.7 Using Pretrained Models: TFIDF weighted W2V

In [90]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [91]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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.append(vector)

print(len(tfidf_w2v_vectors))
print(len(tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 50000/50000 [03:
40<00:00, 226.61it/s]
```

```
50000
300
```

1.4.2.9 Using Pretrained Models: TFIDF weighted W2V on `project_title`

In [92]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model1 = TfidfVectorizer()
tfidf_model1.fit(preprocessed_project_title)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model1.get_feature_names(), list(tfidf_model1.idf_)))
tfidf_title = set(tfidf_model1.get_feature_names())
```

average Word2Vec

compute average word2vec for each review.

tfidf_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list for sentence in tqdm(preprocessed_project_title): # 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_w2v_title): vec = model[word] # getting the vector for each word

```
        # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.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_title.append(vector)
```

print(len(tfidf_w2v_title)) print(len(tfidf_w2v_title[0]))

1.4.3 Vectorizing Numerical features

In [93]:

```
# the cost feature is already in numerical values,  we are going to represent the money, as numeri
cal values within the range 0-1
# normalization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_normalized = 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)

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

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

```
Mean : 299.33367619999996, Standard deviation : 378.20927190421384
```

In [94]:

```
price_normalized
```

Out[94]:

```
array([[-0.38268146],
       [-0.00088225],
```

```
[ 0.57512161],
...,
[-0.65382764],
[-0.52109689],
[ 0.54492668]])
```

1.4.4 Merging all the above features

- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors

In [95]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_normalized.shape)
```

```
(50000, 9)
(50000, 30)
(50000, 12211)
(50000, 1)
```

In [96]:

```
# Assignment 3: Apply KNN# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_normalized))
X.shape
```

Out[96]:

```
(50000, 12251)
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning to find best K**
   - Find the best hyper parameter which results in the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
   - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
4. **[Task-2]**
   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features
     - ```
       from sklearn.datasets import load_digits
       from sklearn.feature_selection import SelectKBest, chi2
       X, y = load_digits(return_X_y=True)
       ```
       ```

```
X.shape
X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
X_new.shape
========
output:
(1797, 64)
(1797, 20)
```

- Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

In [97]:

```
Y = project_data['project_is_approved'].values
del project_data['project_is_approved']
project_data.columns
#project_data.loc[:, project_data.columns != 'project_is_approved'].values
#X1 = np.array(preprocessed_essays)
#X2 = project_data['preprocessed_project_title'].values
```

Out[97]:

```
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', 'clean_categories',
       'clean_subcategories', 'essay', 'price', 'quantity',
       'preprocessed_essays', 'preprocessed_project_title'],
      dtype='object')
```

In [98]:

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

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)# this i
s for time series split
X_train, X_test, y_train, y_test = train_test_split(project_data, Y, test_size=0.33) # this is rand
om splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random
splitting


print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)
```

```
(22445, 21) (22445,)
(11055, 21) (11055,)
(16500, 21) (16500,)
====================================================================================================
```

In [99]:

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
vectorizer.fit(X_train['clean_categories'].values)

X_train_categories_one_hot = vectorizer.transform(X_train['clean_categories'].values)
```

```
X_cv_categories_one_hot = vectorizer.transform(X_cv['clean_categories'].values)
X_test_categories_one_hot = vectorizer.transform(X_test['clean_categories'].values)
```

In [100]:

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(X_train['clean_subcategories'].values)

X_train_sub_categories_one_hot = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_sub_categories_one_hot = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_sub_categories_one_hot = vectorizer.transform(X_test['clean_subcategories'].values)
```

In [101]:

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(X_train['school_state'].values)

X_train_states_one_hot = vectorizer.transform(X_train['school_state'].values)
X_cv_states_one_hot = vectorizer.transform(X_cv['school_state'].values)
X_test_states_one_hot = vectorizer.transform(X_test['school_state'].values)
```

In [102]:

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(X_train['project_grade_category'].values)

X_train_grades_one_hot = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grades_one_hot = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grades_one_hot = vectorizer.transform(X_test['project_grade_category'].values)
```

In [103]:

```
# we use count vectorizer to convert the values into one hot encoded features
vectorizer = CountVectorizer(vocabulary=list(sorted_teacherprefix_dict.keys()), lowercase=False, b
inary=True)
vectorizer.fit(X_train['teacher_prefix'].values)

X_train_teacherprefix_one_hot = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacherprefix_one_hot = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacherprefix_one_hot = vectorizer.transform(X_test['teacher_prefix'].values)
```

In [104]:

```
# the cost feature is already in numerical values,  we are going to represent the money, as numeri
cal values within the range 0-1
# normalization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

price_scalar = StandardScaler()
price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation
of this data

# Now standardize the data with above maen and variance.
X_train_price_normalized = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
X_cv_price_normalized = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
X_test_price_normalized = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

Essay Pre-processing

In [105]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_essays'])
```

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

In [106]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_essays'])
X_train_essay_tfid = vectorizer.transform(X_train['preprocessed_essays'])
X_cv_essay_tfid= vectorizer.transform(X_cv['preprocessed_essays'])
X_test_essay_tfid = vectorizer.transform(X_test['preprocessed_essays'])
```

In [107]:

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

In [108]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_essay_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_essay_avg_w2v_vectors.append(vector)

X_cv_essay_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_cv_essay_avg_w2v_vectors.append(vector)

X_test_essay_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_essay_avg_w2v_vectors.append(vector)
```

```
100%|████████████████████████████████████████████████████████████| 22445/22445
[00:11<00:00, 1978.56it/s]
100%|████████████████████████████████████████████████████████████| 11055/11055
[00:05<00:00, 1936.90it/s]
100%|████████████████████████████████████████████████████████████| 16500/16500
[00:08<00:00, 1947.49it/s]
```

In [109]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_essays'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [110]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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
    X_train_essay_tfidf_w2v_vectors.append(vector)

X_cv_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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
    X_cv_essay_tfidf_w2v_vectors.append(vector)

X_test_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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
    X_test_essay_tfidf_w2v_vectors.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████| 22445/22445 [01:
33<00:00, 241.07it/s]
100%|████████████████████████████████████████████████████████████████| 11055/11055 [00:
50<00:00, 220.32it/s]
100%|████████████████████████████████████████████████████████████████| 16500/16500 [01:
14<00:00, 232.32it/s]
```

Text Pre-processing

In [111]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_project_title'])
# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['preprocessed_project_title'])
X_cv_title_bow = vectorizer.transform(X_cv['preprocessed_project_title'])
X_test_title_bow = vectorizer.transform(X_test['preprocessed_project_title'])
```

In [112]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['preprocessed_project_title'])
X_train_title_tfid = vectorizer.transform(X_train['preprocessed_project_title'])
X_cv_title_tfid= vectorizer.transform(X_cv['preprocessed_project_title'])
X_test_title_tfid = vectorizer.transform(X_test['preprocessed_project_title'])
```

In [113]:

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

In [114]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_title_avg_w2v_vectors.append(vector)

X_cv_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_cv_title_avg_w2v_vectors.append(vector)

X_test_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_title_avg_w2v_vectors.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████| 22445/22445
[00:00<00:00, 244809.89it/s]
100%|████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 197293.14it/s]
100%|████████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 213213.85it/s]
```

In [115]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_project_title'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [116]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_train['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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
    X_train_title_tfidf_w2v_vectors.append(vector)

X_cv_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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
    X_cv_title_tfidf_w2v_vectors.append(vector)

X_test_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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
    X_test_title_tfidf_w2v_vectors.append(vector)
```

```
100%|████████████████████████████████████████████████████████████| 22445/22445
[00:00<00:00, 192509.82it/s]
100%|████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 215044.13it/s]
100%|████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 239448.13it/s]
```

2.4.1 Applying KNN brute force on BOW, SET 1

2.4.1 Applying KNN brute force on BOW, SET 1

1.1 Method 1: Simple for loop

In [117]:

```
X_train_bow=hstack((X_train_categories_one_hot,X_train_sub_categories_one_hot,X_train_states_one_ho
t,X_train_grades_one_hot,X_train_teacherprefix_one_hot,X_train_price_normalized,X_train_essay_bow,
X_train_title_bow)).tocsr()

X_cv_bow=hstack((X_cv_categories_one_hot,X_cv_sub_categories_one_hot,X_cv_states_one_hot,X_cv_grade
s_one_hot,X_cv_teacherprefix_one_hot,X_cv_price_normalized,X_cv_essay_bow,X_cv_title_bow)).tocsr()

X_test_bow=hstack((X_test_categories_one_hot,X_test_sub_categories_one_hot,X_test_states_one_hot,X
_test_grades_one_hot,X_test_teacherprefix_one_hot,X_test_price_normalized,X_test_essay_bow,X_test_t
itle_bow)).tocsr()
```

In [118]:

```
X_train_bow
```

Out[118]:

```
<22445x9081 sparse matrix of type '<class 'numpy.float64'>'
 with 2486483 stored elements in Compressed Sparse Row format>
```

In [119]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

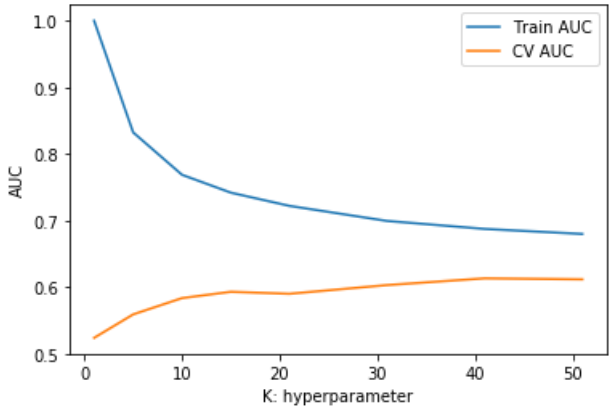
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    #y_train_pred =  neigh.predict_proba(X_train_bow)[:,1]
    y_train_pred = []
    for i in range(0, X_train_bow.shape[0], 1000):
        y_train_pred.extend(neigh.predict_proba(X_train_bow[i:i+1000])[:,1])
    #y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]
    y_cv_pred = []
    for i in range(0, X_cv_bow.shape[0], 1000):
        y_cv_pred.extend(neigh.predict_proba(X_cv_bow[i:i+1000])[:,1])

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



1.2 Method 2: GridSearch or randomsearch

In [120]:

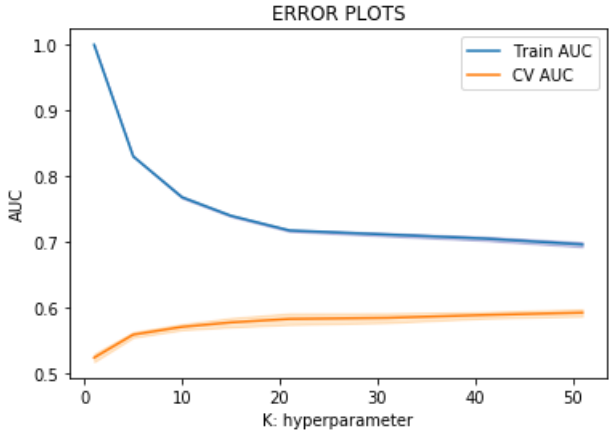
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 23, 26, 31]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkb
lue')

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [121]:

```
print(clf.best_score_)
print(clf.best_estimator_)
print(clf.best_params_)
```

```
0.5920381740370688
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
```

```
            metric_params=None, n_jobs=1, n_neighbors=31, p=2,
            weights='uniform')
{'n_neighbors': 31}
```

In [122]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = clf.best_params_['n_neighbors']
```

2. Testing with Test data

In [123]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

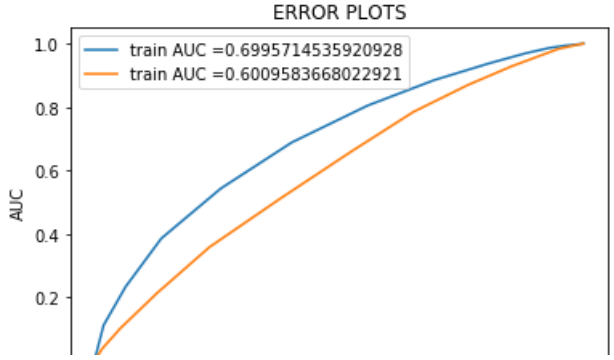

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = []
for i in range(0, X_train_bow.shape[0], 1000):
    y_train_pred.extend(neigh.predict_proba(X_train_bow[i:i+1000])[:,1])
#y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]
y_test_pred = []
for i in range(0, X_test_bow.shape[0], 1000):
        y_test_pred.extend(neigh.predict_proba(X_test_bow[i:i+1000])[:,1])

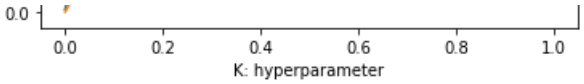
#train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_bow)[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_pred)
#test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,  y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_bow)))
```





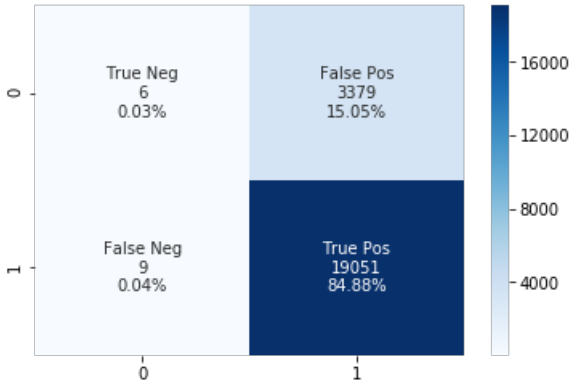
==========

```
Train confusion matrix
[[    6  3379]
 [    9 19051]]
Test confusion matrix
[[    5  2600]
 [    3 13892]]
```

In [124]:

```
train_conf_matrix_BOW=confusion_matrix(y_train, neigh.predict(X_train_bow))
#https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
                train_conf_matrix_BOW.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                     train_conf_matrix_BOW.flatten()/np.sum(train_conf_matrix_BOW)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
sns.heatmap(train_conf_matrix_BOW, annot=labels, fmt='', cmap='Blues');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Train confusion matrix');
```

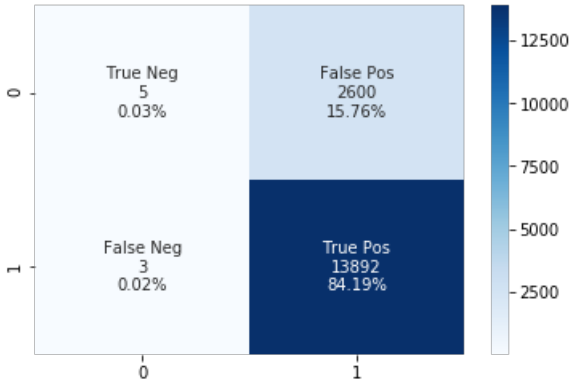
Train confusion matrix



In [125]:

```
test_conf_matrix_BOW=confusion_matrix(y_test, neigh.predict(X_test_bow))
#https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
                test_conf_matrix_BOW.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                     test_conf_matrix_BOW.flatten()/np.sum(test_conf_matrix_BOW)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
sns.heatmap(test_conf_matrix_BOW, annot=labels, fmt='', cmap='Blues');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Test confusion matrix');
```

Test confusion matrix



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [126]:

```
X_train_tfid=hstack((X_train_categories_one_hot,X_train_sub_categories_one_hot,X_train_states_one_h
ot,X_train_grades_one_hot,X_train_teacherprefix_one_hot,X_train_price_normalized,X_train_essay_tfid
,X_train_title_tfid)).tocsr()

X_cv_tfid=hstack((X_cv_categories_one_hot,X_cv_sub_categories_one_hot,X_cv_states_one_hot,X_cv_grad
es_one_hot,X_cv_teacherprefix_one_hot,X_cv_price_normalized,X_cv_essay_tfid,X_cv_title_tfid)).tocs
r()

X_test_tfid=hstack((X_test_categories_one_hot,X_test_sub_categories_one_hot,X_test_states_one_hot,
X_test_grades_one_hot,X_test_teacherprefix_one_hot,X_test_price_normalized,X_test_essay_tfid,X_tes
t_title_tfid)).tocsr()
```

In [127]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

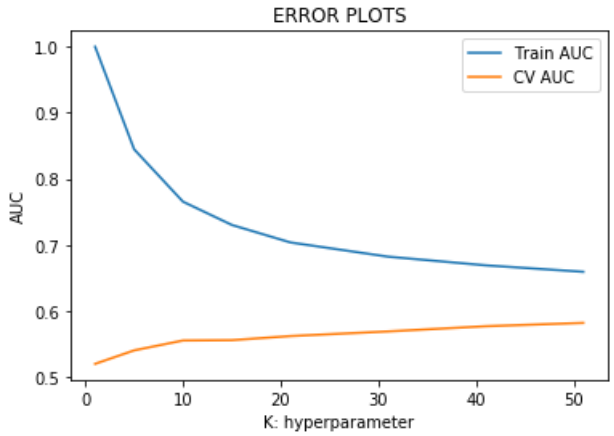
"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_tfid, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    #y_train_pred =  neigh.predict_proba(X_train_bow)[:,1]
    y_train_pred = []
    for i in range(0, X_train_tfid.shape[0], 1000):
        y_train_pred.extend(neigh.predict_proba(X_train_tfid[i:i+1000])[:,1])
    #y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]
    y_cv_pred = []
    for i in range(0, X_cv_tfid.shape[0], 1000):
        y_cv_pred.extend(neigh.predict_proba(X_cv_tfid[i:i+1000])[:,1])

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
```

```
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [128]:

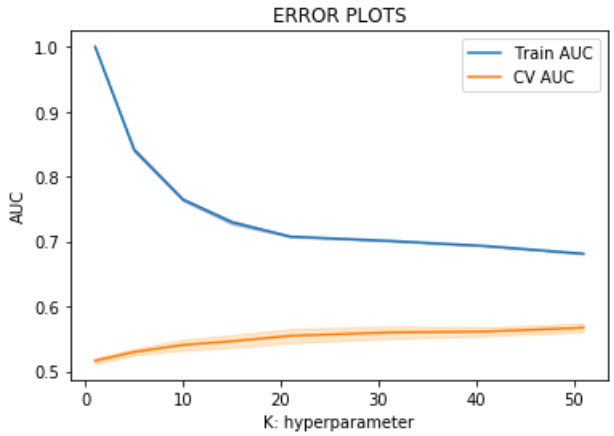
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 23, 26, 31]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_tfid, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkb
lue')

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [129]:

```
print(clf.best_score_)
print(clf.best_estimator_)
print(clf.best_params_)
```

```
0.5671187975970479
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=31, p=2,
           weights='uniform')
{'n_neighbors': 31}
```

In [130]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = clf.best_params_['n_neighbors']
```

2. Testing with Test data

In [131]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

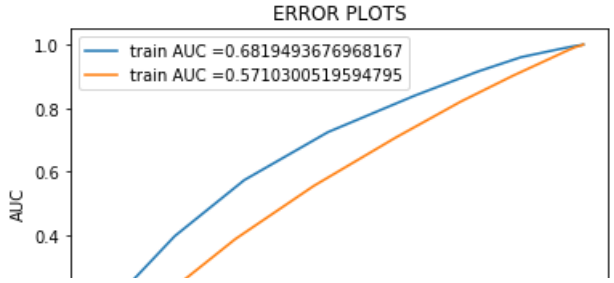

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_tfid, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = []
for i in range(0, X_train_tfid.shape[0], 1000):
    y_train_pred.extend(neigh.predict_proba(X_train_tfid[i:i+1000])[:,1])
#y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]
y_test_pred = []
for i in range(0, X_test_tfid.shape[0], 1000):
        y_test_pred.extend(neigh.predict_proba(X_test_tfid[i:i+1000])[:,1])

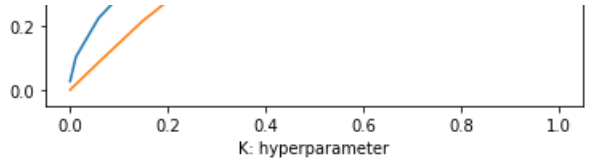
#train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_bow)[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_pred)
#test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,  y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tfid)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfid)))
```





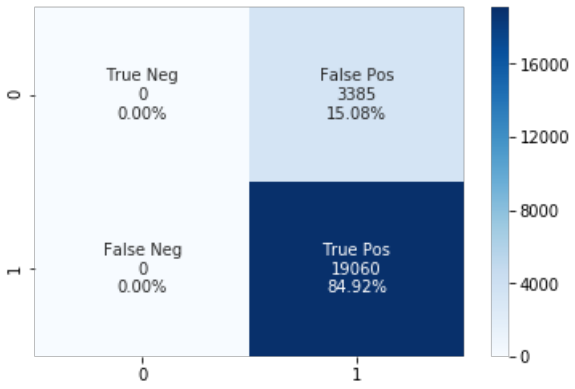
==========

```
Train confusion matrix
[[    0  3385]
 [    0 19060]]
Test confusion matrix
[[    0  2605]
 [    1 13894]]
```

In [132]:

```
train_conf_matrix_tfid=confusion_matrix(y_train, neigh.predict(X_train_tfid))
#https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
                train_conf_matrix_tfid.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                     train_conf_matrix_tfid.flatten()/np.sum(train_conf_matrix_tfid)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
sns.heatmap(train_conf_matrix_tfid, annot=labels, fmt='', cmap='Blues');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Train confusion matrix');
```

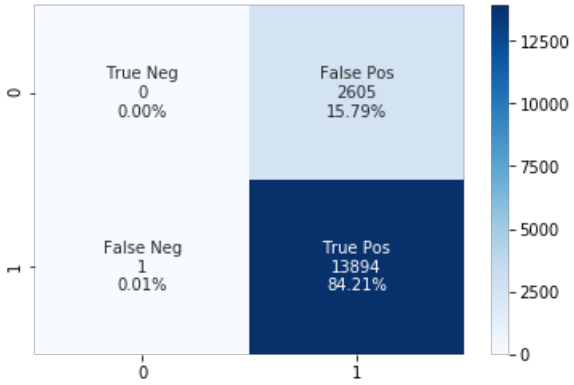
```
Train confusion matrix
```



In [133]:

```
test_conf_matrix_tfid=confusion_matrix(y_test, neigh.predict(X_test_tfid))
#https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
                test_conf_matrix_tfid.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                     test_conf_matrix_tfid.flatten()/np.sum(test_conf_matrix_tfid)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
sns.heatmap(test_conf_matrix_tfid, annot=labels, fmt='', cmap='Blues');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Test confusion matrix');
```

Test confusion matrix



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [134]:

```
X_train_avg_w2v=hstack((X_train_categories_one_hot,X_train_sub_categories_one_hot,X_train_states_or
e_hot,X_train_grades_one_hot,X_train_teacherprefix_one_hot,X_train_price_normalized,X_train_essay_a
vg_w2v_vectors,X_train_title_avg_w2v_vectors)).tocsr()

X_cv_avg_w2v=hstack((X_cv_categories_one_hot,X_cv_sub_categories_one_hot,X_cv_states_one_hot,X_cv_
grades_one_hot,X_cv_teacherprefix_one_hot,X_cv_price_normalized,X_cv_essay_avg_w2v_vectors,X_cv_tit
le_avg_w2v_vectors)).tocsr()

X_test_avg_w2v=hstack((X_test_categories_one_hot,X_test_sub_categories_one_hot,X_test_states_one_ho
t,X_test_grades_one_hot,X_test_teacherprefix_one_hot,X_test_price_normalized,X_test_essay_avg_w2v_v
ectors,X_test_title_avg_w2v_vectors)).tocsr()
```

In [135]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

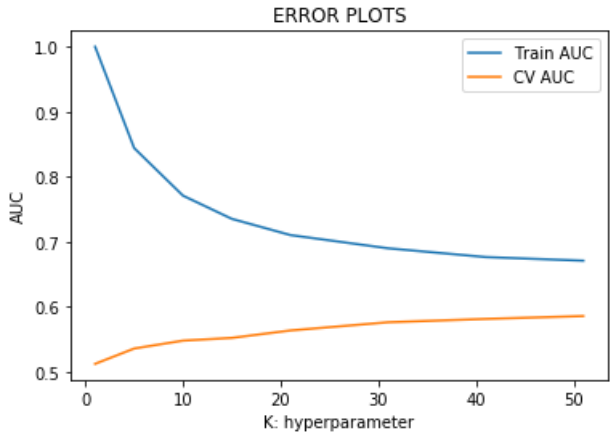
"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_avg_w2v, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    #y_train_pred =  neigh.predict_proba(X_train_bow)[:,1]
    y_train_pred = []
    for i in range(0, X_train_avg_w2v.shape[0], 1000):
        y_train_pred.extend(neigh.predict_proba(X_train_avg_w2v[i:i+1000])[:,1])
    #y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]
    y_cv_pred = []
    for i in range(0, X_cv_avg_w2v.shape[0], 1000):
        y_cv_pred.extend(neigh.predict_proba(X_cv_avg_w2v[i:i+1000])[:,1])

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
```

```
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [136]:

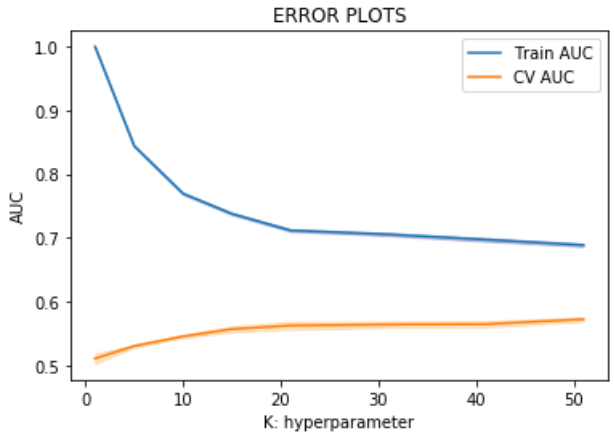
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 23, 26, 31]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_avg_w2v, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkb
lue')

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [137]:

```
print(clf.best_score_)
```

```
print(clf.best_score_)
print(clf.best_estimator_)
print(clf.best_params_)
best_k = clf.best_params_['n_neighbors']
```

```
0.5720248437125569
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=31, p=2,
           weights='uniform')
{'n_neighbors': 31}
```

2. Testing with Test data

In [138]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

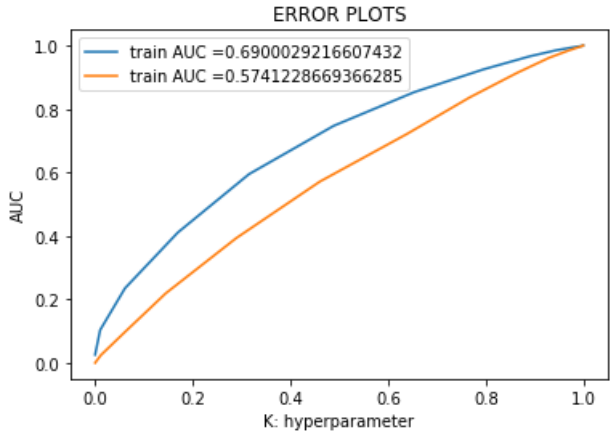

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_avg_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = []
for i in range(0, X_train_avg_w2v.shape[0], 1000):
    y_train_pred.extend(neigh.predict_proba(X_train_avg_w2v[i:i+1000])[:,1])
#y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]
y_test_pred = []
for i in range(0, X_test_avg_w2v.shape[0], 1000):
        y_test_pred.extend(neigh.predict_proba(X_test_avg_w2v[i:i+1000])[:,1])

#train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_bow)[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_pred)
#test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,  y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_avg_w2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_avg_w2v)))
```



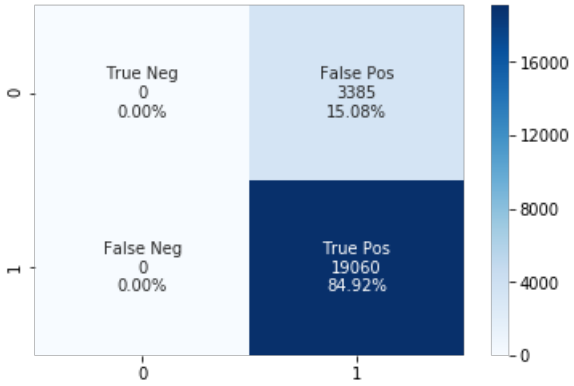
==========

```
Train confusion matrix
[[    0  3385]
 [    0 19060]]
Test confusion matrix
[[    0  2605]
 [    1 13894]]
```

In [139]:

```
train_conf_matrix_avg_w2v=confusion_matrix(y_train, neigh.predict(X_train_avg_w2v))
#https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
                train_conf_matrix_avg_w2v.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                     train_conf_matrix_avg_w2v.flatten()/np.sum(train_conf_matrix_avg_w2v)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
sns.heatmap(train_conf_matrix_avg_w2v, annot=labels, fmt='', cmap='Blues');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Train confusion matrix');
```

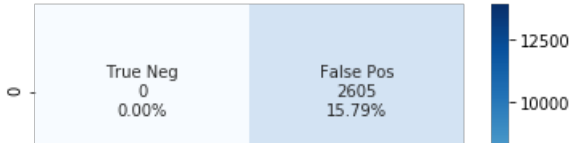
```
Train confusion matrix
```

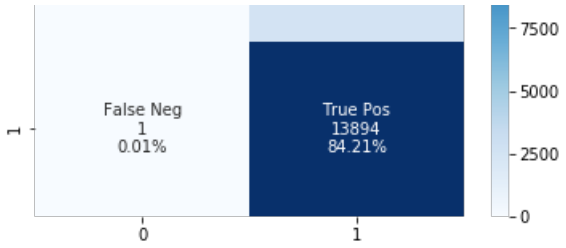


In [140]:

```
test_conf_matrix_avg_w2v=confusion_matrix(y_test, neigh.predict(X_test_avg_w2v))
#https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
                test_conf_matrix_avg_w2v.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                     test_conf_matrix_avg_w2v.flatten()/np.sum(test_conf_matrix_avg_w2v)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
sns.heatmap(test_conf_matrix_avg_w2v, annot=labels, fmt='', cmap='Blues');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Test confusion matrix');
```

```
Test confusion matrix
```





2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [141]:

```
X_train_tfidf_w2v_vectors=hstack((X_train_categories_one_hot,X_train_sub_categories_one_hot,X_trai
n_states_one_hot,X_train_grades_one_hot,X_train_teacherprefix_one_hot,X_train_price_normalized,X_t
rain_essay_tfidf_w2v_vectors,X_train_title_tfidf_w2v_vectors)).tocsr()

X_cv_tfidf_w2v_vectors=hstack((X_cv_categories_one_hot,X_cv_sub_categories_one_hot,X_cv_states_one_
hot,X_cv_grades_one_hot,X_cv_teacherprefix_one_hot,X_cv_price_normalized,X_cv_essay_tfidf_w2v_vectc
s,X_cv_title_tfidf_w2v_vectors)).tocsr()

X_test_tfidf_w2v_vectors=hstack((X_test_categories_one_hot,X_test_sub_categories_one_hot,X_test_sta
tes_one_hot,X_test_grades_one_hot,X_test_teacherprefix_one_hot,X_test_price_normalized,X_test_essay
_tfidf_w2v_vectors,X_test_title_tfidf_w2v_vectors)).tocsr()
```

In [142]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

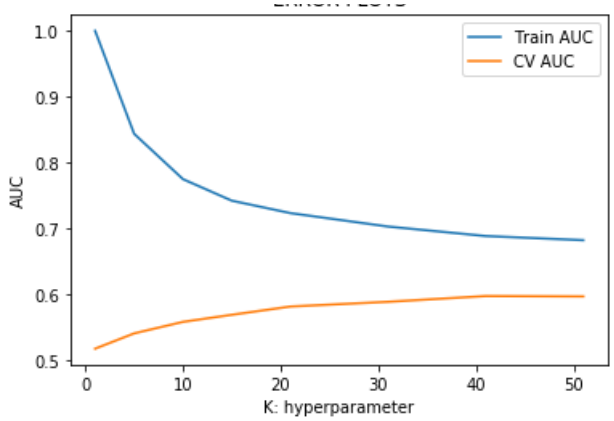
"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_tfidf_w2v_vectors, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    #y_train_pred =  neigh.predict_proba(X_train_bow)[:,1]
    y_train_pred = []
    for i in range(0, X_train_tfidf_w2v_vectors.shape[0], 1000):
        y_train_pred.extend(neigh.predict_proba(X_train_tfidf_w2v_vectors[i:i+1000])[:,1])
    #y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]
    y_cv_pred = []
    for i in range(0, X_cv_tfidf_w2v_vectors.shape[0], 1000):
        y_cv_pred.extend(neigh.predict_proba(X_cv_tfidf_w2v_vectors[i:i+1000])[:,1])

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

ERROR PLOTS



In [143]:

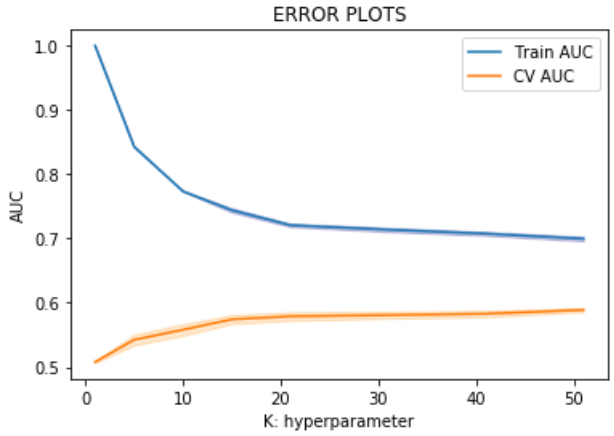
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 23, 26, 31]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_tfidf_w2v_vectors, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkb
lue')

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(K,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [144]:

```
print(clf.best_score_)
print(clf.best_estimator_)
print(clf.best_params_)
best_k = clf.best_params_['n_neighbors']
```

```
0.588921915855745
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=31, p=2,
           weights='uniform')
```

```
{'n_neighbors': 31}
```

2. Testing with Test data

In [145]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

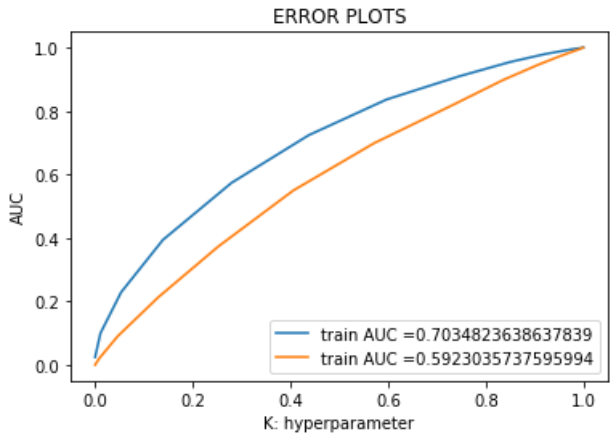

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_tfidf_w2v_vectors, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = []
for i in range(0, X_train_avg_w2v.shape[0], 1000):
    y_train_pred.extend(neigh.predict_proba(X_train_tfidf_w2v_vectors[i:i+1000])[:,1])
#y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]
y_test_pred = []
for i in range(0, X_test_avg_w2v.shape[0], 1000):
        y_test_pred.extend(neigh.predict_proba(X_test_tfidf_w2v_vectors[i:i+1000])[:,1])

#train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_bow)[:,1])
train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_pred)
#test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test,  y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tfidf_w2v_vectors)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfidf_w2v_vectors)))
```




```
====================================================================================================

Train confusion matrix
[[    1  3384]
 [    1 19059]]
Test confusion matrix
[[    0  2605]
 [    0 13895]]
```

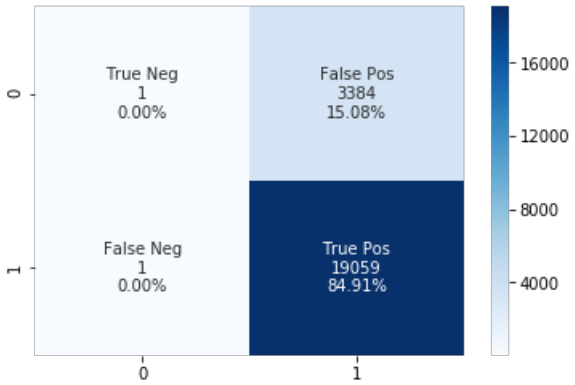
In [146]:

In [146]:

```
train_conf_matrix_tfidf_w2v_vectors=confusion_matrix(y_train,
neigh.predict(X_train_tfidf_w2v_vectors))
#https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
                train_conf_matrix_tfidf_w2v_vectors.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in

train_conf_matrix_tfidf_w2v_vectors.flatten()/np.sum(train_conf_matrix_tfidf_w2v_vectors)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
sns.heatmap(train_conf_matrix_tfidf_w2v_vectors, annot=labels, fmt='', cmap='Blues');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Train confusion matrix');
```

Train confusion matrix

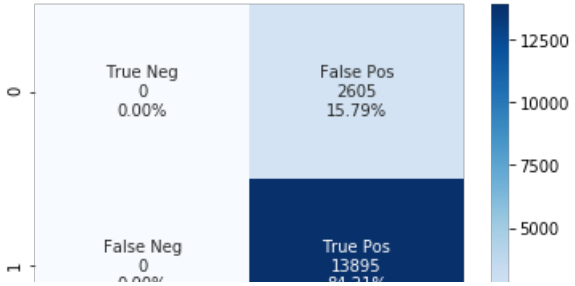


In [147]:

```
test_conf_matrix_tfidf_w2v_vectors=confusion_matrix(y_test, neigh.predict(X_test_tfidf_w2v_vectors
))
#https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ['{0:0.0f}'.format(value) for value in
                test_conf_matrix_tfidf_w2v_vectors.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in

test_conf_matrix_tfidf_w2v_vectors.flatten()/np.sum(test_conf_matrix_tfidf_w2v_vectors)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
sns.heatmap(test_conf_matrix_tfidf_w2v_vectors, annot=labels, fmt='', cmap='Blues');
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Test confusion matrix');
```

Test confusion matrix



In [ ]:

```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr (False Positive Rate)
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```