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	De
project_id	A unique identifier for the proposed project. **Example:**
project_title	Title of the project. **E** • Art Will Make You • First Gr
project_grade_category	Grade level of students for which the project is targeted. One of the enumerate • Grades • Gra • Gra • Grad
project_subject_categories	One or more (comma-separated) subject categories for the projec following enumerated list • Applied L • Care & • Health & • History & • Literacy & L • Math & • Music & T • Specia • **Ex** • Music & T • Literacy & Language, Math &
school_state	State where school is located (Two-letter U.S. p (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Posta **Exar**
project_subject_subcategories	One or more (comma-separated) subject subcategories for th **E** • L • Literature & Writing, Social S
project_resource_summary	An explanation of the resources needed for the project. **E** • My students need hands on literacy materials to sensory
project_essay_1	First applicat
project_essay_2	Second applicat
project_essay_3	Third applicat
project_essay_4	Fourth applicat
project_submitted_datetime	Datetime when project application was submitted. **Example:** 201 12:43
teacher_id	A unique identifier for the teacher of the proposed project. bdf8baa8fedef6bfeec7ae4ff

Feature	De
	Teacher's title. One of the following enumerate
`teacher_prefix`	• • • • • • T
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the sam **Exa**

* 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."
- __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 plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

```
C:\Users\RASHU TYAGI\Anaconda3\lib\site-packages\smart_open\ssh.py:34: Use
rWarning: paramiko missing, opening SSH/SCP/SFTP paths will be disabled.
`pip install paramiko` to suppress
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disa
bled.  `pip install paramiko` to suppress')
```

Reading Data

In [2]:

```
project_data = pd.read_csv(r'D:\Rashu Studies\AppliedAICourse\Assignments\Mandatory Ass
ignments\Mandatory Assignment 2 DonorsChoose\train_data.csv')
resource_data = pd.read_csv(r'D:\Rashu Studies\AppliedAICourse\Assignments\Mandatory As
signments\Mandatory Assignment 2 DonorsChoose\resources.csv')
```

In [3]:

```
print("Number of datapoints in our train data are",project_data.shape)
print("--" *50)
print("The attributes of our train data are",project_data.columns.values)
```

```
Number of datapoints in our train data are (109248, 17)
--------------------------------------------------------------------------------
-------------------------
The attributes of our train data are ['Unnamed: 0' 'id' 'teacher_id' 'teac
her_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']
```

In [4]:

```
project_data.head(5)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	

In [5]:

```
print("Number of datapoints in our resource data are",resource_data.shape)
print("--" *50)
print("The attributes of our resource data are",resource_data.columns.values)
```

```
Number of datapoints in our resource data are (1541272, 4)
--------------------------------------------------------------------------------
-------------------------
The attributes of our resource data are ['id' 'description' 'quantity' 'pr
ice']
```

In [6]:

```
resource_data.head(5)
```

Out[6]:

	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
2	p069063	Cory Stories: A Kid's Book About Living With Adhd	1	8.45
3	p069063	Dixon Ticonderoga Wood-Cased #2 HB Pencils, Bo...	2	13.59
4	p069063	EDUCATIONAL INSIGHTS FLUORESCENT LIGHT FILTERS...	3	24.95

Here note that the number of data points in resources is way too large because there will be many projects which will be demanding for more than one resource at a time.

DATA ANALYSIS

In [7]:

```
y_value_counts = project_data['project_is_approved'].value_counts()
print(y_value_counts)
```

```
1    92706
0    16542
Name: project_is_approved, dtype: int64
```

In [8]:

```
print("Number of projects that 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 that are not approved for funding ", y_value_counts[0], ", ("
, (y_value_counts[0]/(y_value_counts[1]+y_value_counts[0]))*100,"%)")
```

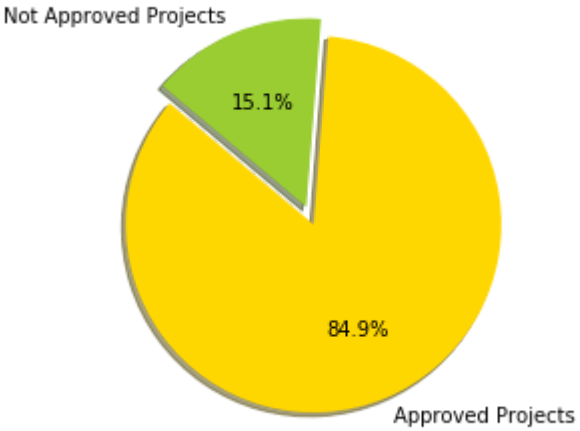
```
Number of projects that are approved for funding  92706 , ( 84.85830404217
927 %)
Number of projects that are not approved for funding  16542 , ( 15.1416959
57820739 %)
```

So we can say that around 85% of the projects have been approved and around 15% of the projects have been rejected

Now lets visualise the approved and not approved projects using pie chart

In [9]:

```
# drawing a pie chart for the above data received
# source for the code :- https://pythonspot.com/matplotlib-pie-chart/
# Data to plot
labels = 'Approved Projects', 'Not Approved Projects'
sizes = [y_value_counts[1],y_value_counts[0]]
colors = ['gold', 'yellowgreen']
explode = (0.1, 0)  # explode 1st slice which means the project approved one

# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```



From now on keep in mind that if I am calculating the mean of the project is approved data then it automatically corresponds to the percentages of project approved data only because this data has 0 and 1 values only.

1.2.1 Univariate Analysis : School State

In [10]:

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


temp = pd.DataFrame(project_data.groupby("school_state")["project_is_approved"].apply(n
p.mean)).reset_index()

temp.columns = ['state_codes' , 'percentage_proposals_approved']

temp.head(30)
```

Out[10]:

	state_codes	percentage_proposals_approved
0	AK	0.840580
1	AL	0.854711
2	AR	0.831268
3	AZ	0.838379
4	CA	0.858136
5	CO	0.841584
6	CT	0.868912
7	DC	0.802326
8	DE	0.897959
9	FL	0.831690
10	GA	0.840020
11	HI	0.856016
12	IA	0.852853
13	ID	0.835498
14	IL	0.852874
15	IN	0.845038
16	KS	0.839117
17	KY	0.863497
18	LA	0.831245
19	MA	0.860193
20	MD	0.838838
21	ME	0.847525
22	MI	0.845302
23	MN	0.857616
24	MO	0.854814
25	MS	0.845049
26	MT	0.816327
27	NC	0.855038
28	ND	0.888112
29	NE	0.841424

In [11]:

```
# https://stackoverflow.com/questions/43893457/python-pandas-understanding-inplace-true

#When inplace=True is passed, the data is renamed in place (it returns nothing), so yo
u'd use:

#df.an_operation(inplace=True)

#When inplace=False is passed (this is the default value, so isn't necessary), performs
the operation and returns a copy of the object, so you'd use:
#df = df.an_operation(inplace=False)


temp.sort_values(by = ['percentage_proposals_approved'],inplace = True)

temp.head(10)
```

Out[11]:

	state_codes	percentage_proposals_approved
46	VT	0.800000
7	DC	0.802326
43	TX	0.813142
26	MT	0.816327
18	LA	0.831245
2	AR	0.831268
9	FL	0.831690
36	OK	0.834798
13	ID	0.835498
44	UT	0.836511

From above we can say that for every state 80% is the minimum project approved percentage

In [12]:

```
temp.tail(10)
```

Out[12]:

	state_codes	percentage_proposals_approved
32	NM	0.859964
40	SC	0.860010
19	MA	0.860193
17	KY	0.863497
6	CT	0.868912
30	NH	0.873563
35	OH	0.875152
47	WA	0.876178
28	ND	0.888112
8	DE	0.897959

From above we can say that 89% is the maximum project approval percentage for any state

In [13]:

```
print("Top 5 states with least approval of projects percentage")

temp.head(5)
```

```
Top 5 states with least approval of projects percentage
```

Out[13]:

	state_codes	percentage_proposals_approved
46	VT	0.800000
7	DC	0.802326
43	TX	0.813142
26	MT	0.816327
18	LA	0.831245

In [14]:

```
print("Top 5 states with highest approval of projects percentage")
temp.tail(5)
```

```
Top 5 states with highest approval of projects percentage
```

Out[14]:

	state_codes	percentage_proposals_approved
30	NH	0.873563
35	OH	0.875152
47	WA	0.876178
28	ND	0.888112
8	DE	0.897959

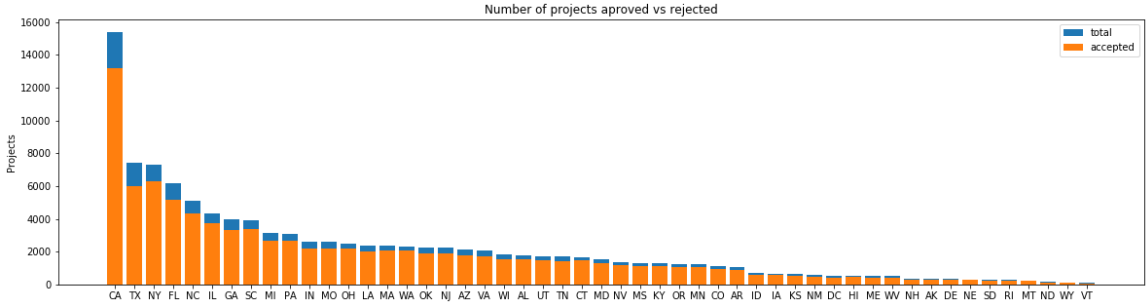
This is a generic funtion we are defining for plotting stacked bar plots for any data we have

In [15]:

```
#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()

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("These are the top 5 results")
    print(temp.head(5))
    print("="*50)
    print("These are the bottom 5 results")
    print(temp.tail(5))
```

In [16]:

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




```
These are the top 5 results
   school_state  project_is_approved  total       Avg
4            CA                13205  15388  0.858136
43           TX                 6014   7396  0.813142
34           NY                 6291   7318  0.859661
9            FL                 5144   6185  0.831690
27           NC                 4353   5091  0.855038
==================================================
These are the bottom 5 results
   school_state  project_is_approved  total       Avg
39           RI                  243    285  0.852632
26           MT                  200    245  0.816327
28           ND                  127    143  0.888112
50           WY                   82     98  0.836735
46           VT                   64     80  0.800000
```

Conclusion

We can say from the above plot that every state has atleast 80% project approval rate.

The company performs well by providng at least 80% project approval rate.

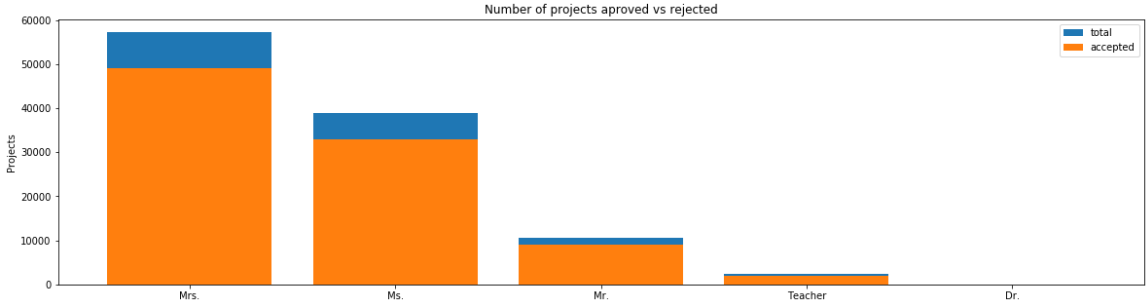
CA(Caifornia) has the highest number of project proposals out of which 85% are approved also

VT(Vermont) has the least number of project proposals

1.2.2 Univariate Analysis: teacher_prefix

In [17]:

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




```
These are the top 5 results
  teacher_prefix  project_is_approved  total       Avg
2           Mrs.                48997  57269  0.855559
3            Ms.                32860  38955  0.843537
1            Mr.                 8960  10648  0.841473
4        Teacher                 1877   2360  0.795339
0            Dr.                    9     13  0.692308
==================================================
These are the bottom 5 results
  teacher_prefix  project_is_approved  total       Avg
2           Mrs.                48997  57269  0.855559
3            Ms.                32860  38955  0.843537
1            Mr.                 8960  10648  0.841473
4        Teacher                 1877   2360  0.795339
0            Dr.                    9     13  0.692308
```

Conclusion

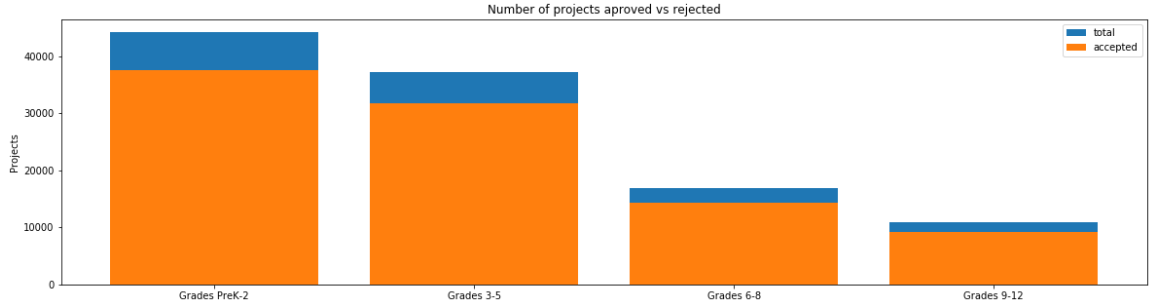
Highest amount of proects approval are received by the married ladies which basically means ladies with older age.

Least amount of proect approval are received by the Doctors or the PHD holders it is because they submitted the less amount of projects

1.2.3 Univariate Analysis: project_grade_category

In [18]:

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




```
These are the top 5 results
  project_grade_category  project_is_approved  total       Avg
3          Grades PreK-2                37536  44225  0.848751
0             Grades 3-5                31729  37137  0.854377
1             Grades 6-8                14258  16923  0.842522
2            Grades 9-12                 9183  10963  0.837636
==================================================
These are the bottom 5 results
  project_grade_category  project_is_approved  total       Avg
3          Grades PreK-2                37536  44225  0.848751
0             Grades 3-5                31729  37137  0.854377
1             Grades 6-8                14258  16923  0.842522
2            Grades 9-12                 9183  10963  0.837636
```

Conclusion

Project Approval rate of grade 3-5 is the highest.

There are very less project requests from grade 9-12

The highest project requests came from the grade of the category PreK -2

1.2.4 Univariate Analysis: project_subject_categories

In [19]:

```
catogories = list(project_data['project_subject_categories'].values)

# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-stri
ng

# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyth
on


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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & Science"=> "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:"M
ath & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spa
ces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())
```

Now we are dropping the project subject categories and placing the cleaned categories into it

In [20]:

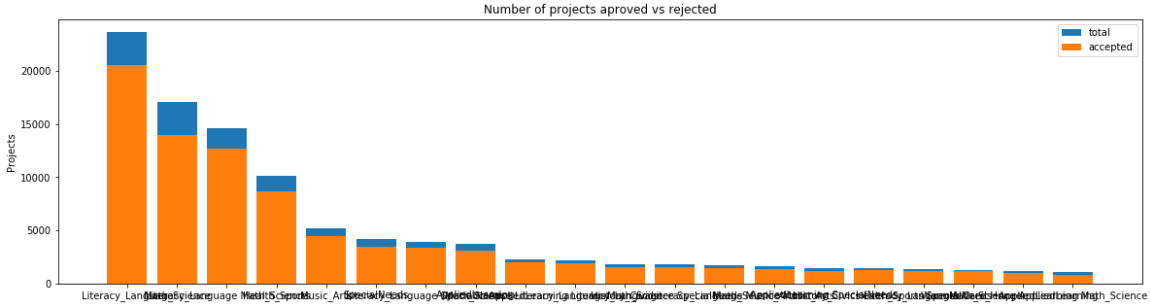
```
project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
project_data.head(5)
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	

In [21]:

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




```
These are the top 5 results
                  clean_categories  project_is_approved  total       Avg
24                Literacy_Language                20520  23655  0.867470
32                     Math_Science                13991  17072  0.819529
28  Literacy_Language Math_Science                12725  14636  0.869432
8                     Health_Sports                 8640  10177  0.848973
40                       Music_Arts                 4429   5180  0.855019
==================================================
These are the bottom 5 results
                    clean_categories  project_is_approved  total       Avg
19  History_Civics Literacy_Language                 1271   1421  0.894441
14         Health_Sports SpecialNeeds                 1215   1391  0.873472
50                 Warmth Care_Hunger                 1212   1309  0.925898
33       Math_Science AppliedLearning                 1019   1220  0.835246
4        AppliedLearning Math_Science                  855   1052  0.812738
```

Conclusion

The company has given almost equal importance to every subject and field in case of approving the projects or rejecting them.This is because the approved projects rate when seen category of subject wise dont show much of differences in the average value.

The highest project approval rate is for the subject category of warmth,care and hunger with approval rate of 92.5% also this category is a sensitive issue also

Literacy Language has received the highest number of project requests with approval rate of around 87%

Now we are trying to seperate out the cases where there are more than one subject categories

In [22]:

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

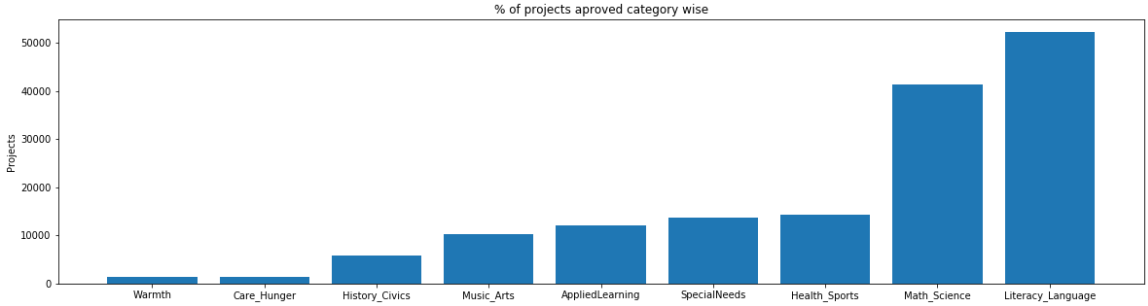
```
Counter({'Literacy_Language': 52239, 'Math_Science': 41421, 'Health_Sport
s': 14223, 'SpecialNeeds': 13642, 'AppliedLearning': 12135, 'Music_Arts':
10293, 'History_Civics': 5914, 'Warmth': 1388, 'Care_Hunger': 1388})
```

Getting the indivdual categories now

In [23]:

```
# 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]))

for p,q in sorted_cat_dict.items():
    print(p,q)

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()
```

```
Warmth 1388
Care_Hunger 1388
History_Civics 5914
Music_Arts 10293
AppliedLearning 12135
SpecialNeeds 13642
Health_Sports 14223
Math_Science 41421
Literacy_Language 52239
```



In [24]:

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

```
Warmth               :      1388
Care_Hunger          :      1388
History_Civics       :      5914
Music_Arts           :     10293
AppliedLearning      :     12135
SpecialNeeds         :     13642
Health_Sports        :     14223
Math_Science         :     41421
Literacy_Language    :     52239
```

Conclusion

When seen individual category wise we find that highest number of project requests came from Literacy and Language and from Math and science background and we get very less project requests from warmth care and hunger category that is only 1388 projects

1.2.5 Univariate Analysis: project_subject_subcategories

In [25]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & Science"=> "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:"M
ath & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spa
ces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())
```

Removing the project_subject_Subcategory and replacing it with clean_subcategories column

In [26]:

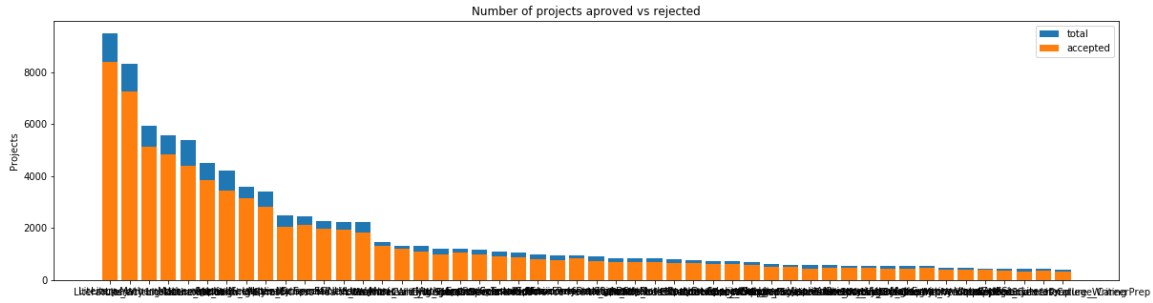
```
project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
project_data.head(5)
```

Out[26]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	

In [27]:

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




```
These are the top 5 results
               clean_subcategories  project_is_approved  total       Avg
317                        Literacy                 8371   9486  0.882458
319            Literacy Mathematics                 7260   8325  0.872072
331  Literature_Writing Mathematics                 5140   5923  0.867803
318     Literacy Literature_Writing                 4823   5571  0.865733
342                     Mathematics                 4385   5379  0.815207
==================================================
These are the bottom 5 results
                   clean_subcategories  project_is_approved  total
Avg
196       EnvironmentalScience Literacy                  389    444  0.876
126
127                                ESL                  349    421  0.828
979
79                  College_CareerPrep                  343    421  0.814
727
17   AppliedSciences Literature_Writing                 361    420  0.859
524
3    AppliedSciences College_CareerPrep                 330    405  0.814
815
```

Getting the indivdual categories now

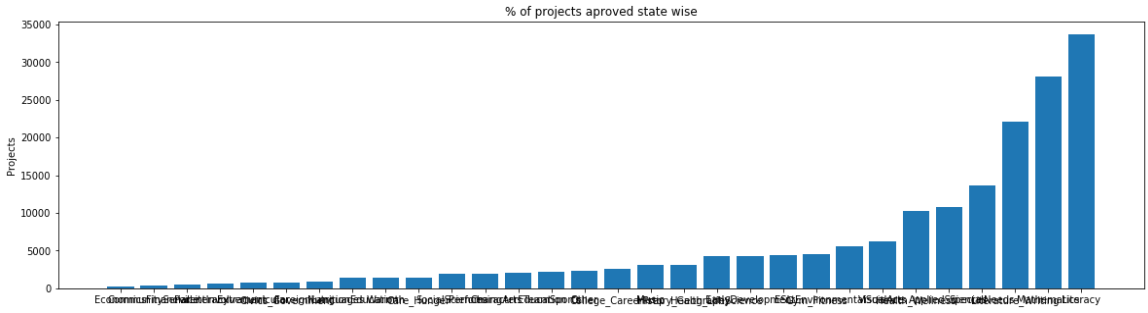
In [28]:

```
# 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())
print(my_counter)
```

```
Counter({'Literacy': 33700, 'Mathematics': 28074, 'Literature_Writing': 22
179, 'SpecialNeeds': 13642, 'AppliedSciences': 10816, 'Health_Wellness': 1
0234, 'VisualArts': 6278, 'EnvironmentalScience': 5591, 'Gym_Fitness': 450
9, 'ESL': 4367, 'EarlyDevelopment': 4254, 'Health_LifeScience': 4235, 'His
tory_Geography': 3171, 'Music': 3145, 'College_CareerPrep': 2568, 'Other':
2372, 'TeamSports': 2192, 'CharacterEducation': 2065, 'PerformingArts': 19
61, 'SocialSciences': 1920, 'Warmth': 1388, 'Care_Hunger': 1388, 'Nutritio
nEducation': 1355, 'ForeignLanguages': 890, 'Civics_Government': 815, 'Ext
racurricular': 810, 'ParentInvolvement': 677, 'FinancialLiteracy': 568, 'C
ommunityService': 441, 'Economics': 269})
```

In [29]:

```
# 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 [30]:

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

```
Economics            :       269
CommunityService     :       441
FinancialLiteracy    :       568
ParentInvolvement    :       677
Extracurricular      :       810
Civics_Government    :       815
ForeignLanguages     :       890
NutritionEducation   :      1355
Warmth               :      1388
Care_Hunger          :      1388
SocialSciences       :      1920
PerformingArts       :      1961
CharacterEducation   :      2065
TeamSports           :      2192
Other                :      2372
College_CareerPrep   :      2568
Music                :      3145
History_Geography    :      3171
Health_LifeScience   :      4235
EarlyDevelopment     :      4254
ESL                  :      4367
Gym_Fitness          :      4509
EnvironmentalScience :      5591
VisualArts           :      6278
Health_Wellness      :     10234
AppliedSciences      :     10816
SpecialNeeds         :     13642
Literature_Writing   :     22179
Mathematics          :     28074
Literacy             :     33700
```

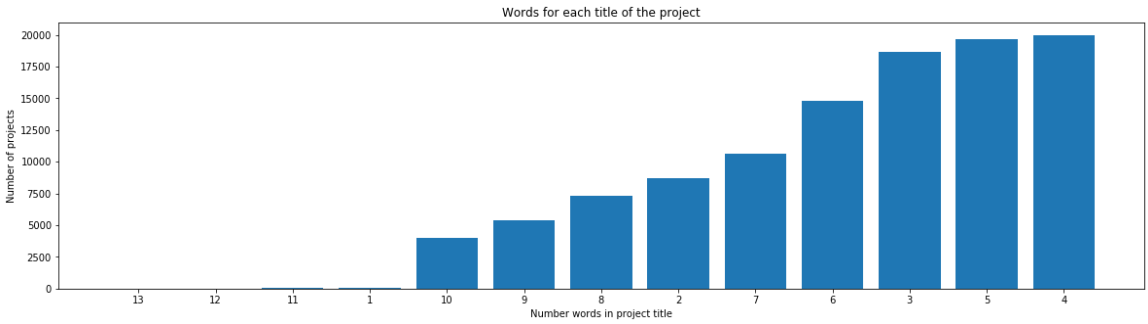
From individual categories also we find that literacy has the highest number of project requests followed by mathematics and the least number of project requests being made by the economics category

1.2.6 Univariate Analysis: Text features (Title)

Analysing the number of words in the title of the project

In [31]:

```
#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('Number of projects')
plt.xlabel('Number words in project title')
plt.title('Words for each title of the project')
plt.xticks(ind, list(word_dict.keys()))
plt.show()
```



From above plot we can see that the maximum number of projects received contains 4 and 5 number of words in their title

There are no projects having number of words more than 10 in their project title

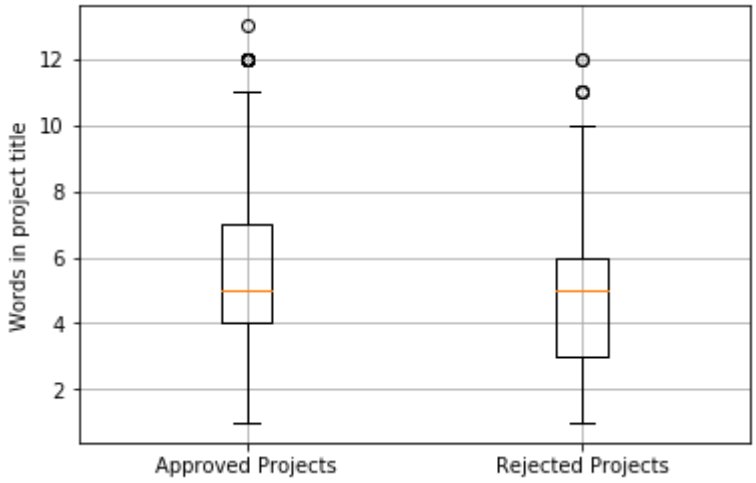
In [32]:

```
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

print(approved_title_word_count)
```

```
[5 2 3 ... 6 5 7]
```

In [33]:

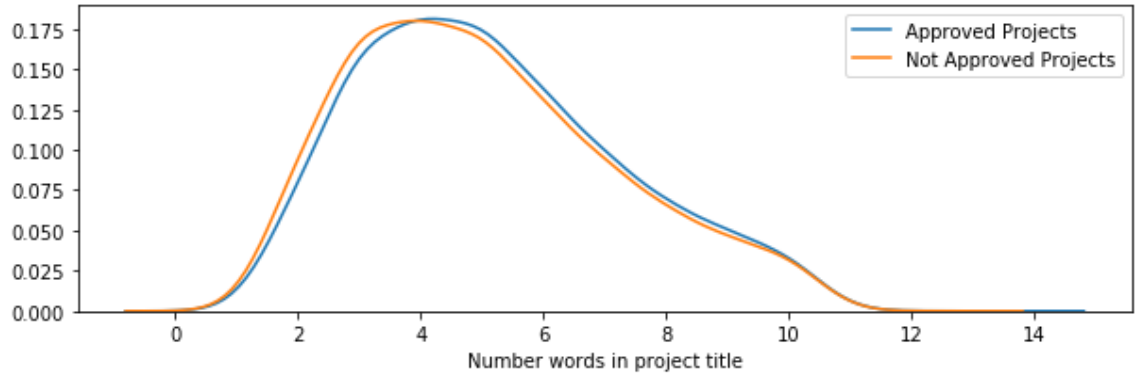
```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([approved_title_word_count, rejected_title_word_count])
plt.xticks([1,2],('Approved Projects','Rejected Projects'))
plt.ylabel('Words in project title')
plt.grid()
plt.show()
```



From above plot we can say that most of the approved projects have number of words near 5 also the maximum number of rejected projects have number of words in their title equal to 5 only hence not much can be said from the above plot.

In [34]:

```
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.xlabel('Number words in project title')
plt.show()
```



From above plot we can see that the highest probabiity of having the approved projects lies for those projects whose titles have words between 4 to 6

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

In [35]:

```
# for some projects we have 2 essays submitted while for many other projects we have 4
 essays submitted based on their date of
# submission hence in order to deal with this fact we are merging all the essays of one
project into one essay and do analysis on it.

# 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)
project_data["essay"].head(2)
```

Out[35]:

```
0    My students are English learners that are work...
1    Our students arrive to our school eager to lea...
Name: essay, dtype: object
```

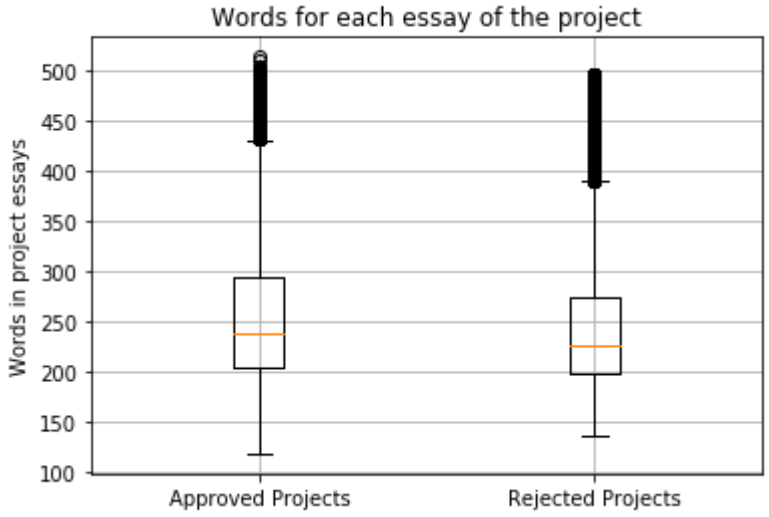
In [36]:

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

print(approved_word_count)

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

```
[221 213 234 ... 181 254 263]
```

In [37]:

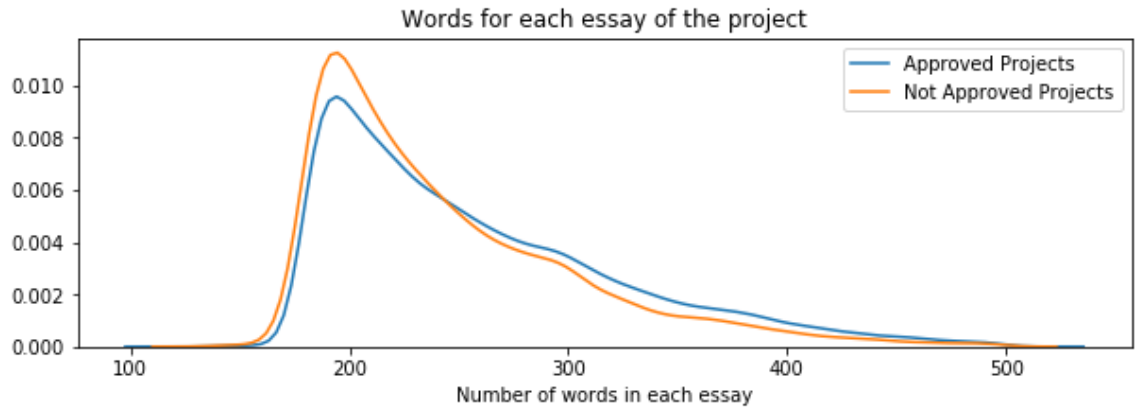
```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([approved_word_count, rejected_word_count])
plt.title('Words for each essay of the project')
plt.xticks([1,2],('Approved Projects','Rejected Projects'))
plt.ylabel('Words in project essays')
plt.grid()
plt.show()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



From above distribution we can say that approved projects have slightly more number of words in their essay as compared to the non approved ones.

In [38]:

```
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 essay')
plt.legend()
plt.show()
```



From the above plot we can say that the projects whose essays' had the number of words greater than 250 at that point the approval probability started increasing slightly.

Hence we can say that the approved projects have slighlty more number of words in their essays compared to the one which are not approved

1.2.8 Univariate Analysis: Cost per project

In [39]:

```
# we get the cost of the project using resource.csv file
resource_data.head(5)
```

Out[39]:

	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
2	p069063	Cory Stories: A Kid's Book About Living With Adhd	1	8.45
3	p069063	Dixon Ticonderoga Wood-Cased #2 HB Pencils, Bo...	2	13.59
4	p069063	EDUCATIONAL INSIGHTS FLUORESCENT LIGHT FILTERS...	3	24.95

In [40]:

```
# 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(5)
```

Out[40]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21
2	p000003	298.97	4
3	p000004	1113.69	98
4	p000005	485.99	8

In [41]:

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

project_data.head(5)
```

Out[41]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	

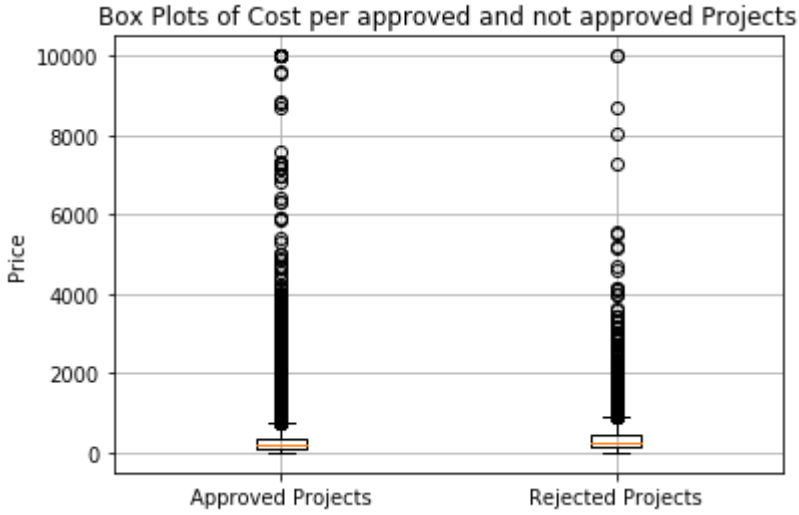
In [42]:

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

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

print(approved_price)
```

```
[299.   232.9   67.98 ... 239.96  73.05 109.9 ]
```

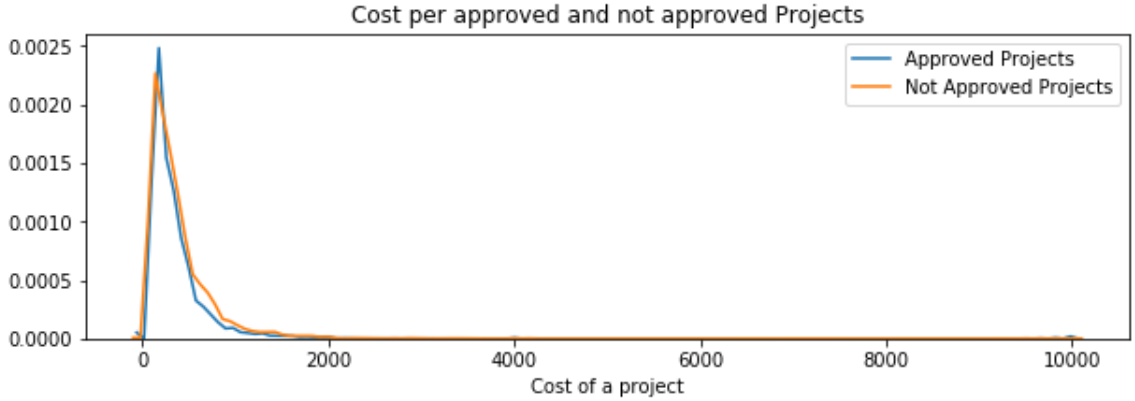
In [43]:

```
# 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()
```



In [44]:

```
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()
```



From the box plot it is very hard to infer anything but from the distribution plots we can say that costlier projects are mostly not approved

In [45]:

```
from prettytable import PrettyTable
```

In [46]:

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

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

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(r
ejected_price,i), 3)])
print(x)
```

```
+------------+-------------------+-----------------------+
| Percentile | Approved Projects | Not Approved Projects |
+------------+-------------------+-----------------------+
|     0      |        0.66       |          1.97         |
|     5      |       13.59       |          41.9         |
|     10     |       33.88       |         73.67         |
|     15     |        58.0       |         99.109        |
|     20     |       77.38       |         118.56        |
|     25     |       99.95       |        140.892        |
|     30     |       116.68      |         162.23        |
|     35     |      137.232      |        184.014        |
|     40     |       157.0       |        208.632        |
|     45     |      178.265      |        235.106        |
|     50     |       198.99      |        263.145        |
|     55     |       223.99      |         292.61        |
|     60     |       255.63      |        325.144        |
|     65     |      285.412      |         362.39        |
|     70     |      321.225      |         399.99        |
|     75     |      366.075      |        449.945        |
|     80     |       411.67      |        519.282        |
|     85     |       479.0       |        618.276        |
|     90     |       593.11      |        739.356        |
|     95     |      801.598      |        992.486        |
|    100     |       9999.0      |         9999.0        |
+------------+-------------------+-----------------------+
```

From above table we can clearly say that in every percentile value or region the approved project prices are always less than than that of unapproved project prices

They have accepted the maximum cost for a project as 9999 dollars which is quite a high value and far from the other costs

Throughout every percentile value it can be seen that approved projects cost less than the not approved projects

1.2.9 Univariate Analysis: teacher_number_of_previously_posted_projects

In [47]:

```
teacher_number_of_previously_posted_projects_temp = project_data.groupby("teacher_id")[
'teacher_id',"teacher_number_of_previously_posted_projects"]
```

In [48]:

```
teacher_number_of_previously_posted_projects_temp.head()
```

Out[48]:

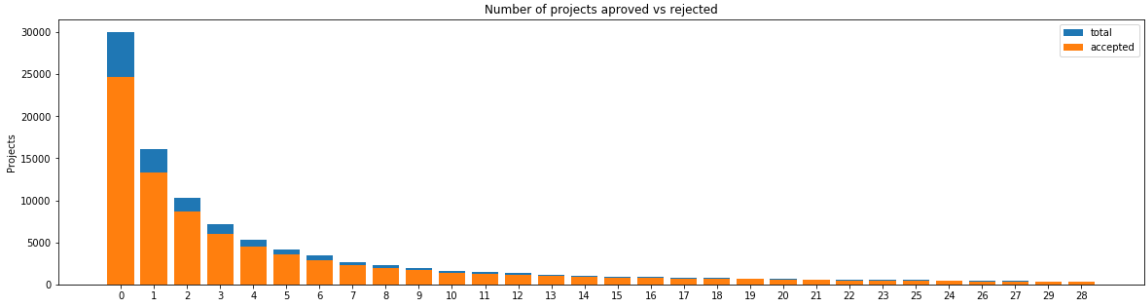
	teacher_id	teacher_number_of_previously_posted_projects
0	c90749f5d961ff158d4b4d1e7dc665fc	0
1	897464ce9ddc600bced1151f324dd63a	7
2	3465aaf82da834c0582ebd0ef8040ca0	1
3	f3cb9bffbba169bef1a77b243e620b60	4
4	be1f7507a41f8479dc06f047086a39ec	1
5	a50a390e8327a95b77b9e495b58b9a6e	1
6	9b40170bfa65e399981717ee8731efc3	1
7	5bfd3d12fae3d2fe88684bbac570c9d2	7
8	487448f5226005d08d36bdd75f095b31	28
9	140eeac1885c820ad5592a409a3a8994	36
10	363788b51d40d978fe276bcb1f8a2b35	37
11	4ba7c721133ef651ca54a03551746708	32
12	5e52c92b7e3c472aad247a239d345543	5
13	178f6ae765cd4e0fb143a77c47fd65e2	30
14	424819801de22a60bba7d0f4354d0258	15
15	bb6d6d054824fa01576ab38dfa2be160	3
16	4ad7e280fddff889e1355cc9f29c3b89	1
17	e39abda057354c979c5b075cffbe5f88	0
18	fcd9b003fc1891383f340a89da02a1a6	0
19	8e07a98deb1bc74c75b97521e05b1691	9
20	e0c1aad1f71badeff703fadc15f57680	23
21	2d4a4d2d774e5c2fdd25b2ba0e7341f8	0
22	30f08fbe02eba5453c4ce2e857e88eb4	0
23	258ef2e6ab5ce007ac6764ce15d261ba	2
24	74f8690562c44fc88f65f845b9fe61d0	0
25	b8bf3507cee960d5fedcb27719df2d59	11
26	7a0a5de5ed94e7036946b1ac3eaa99d0	2
27	efdc3cf14d136473c9f62becc00d4cec	2
28	22c8184c4660f1c589bea061d14b7f35	5
29	45f16a103f1e00b7439861d4e0728a59	0
...	...	...
109212	c6baf09cc7f9b45eab75bb73e07f4940	0
109213	3d272057fd2628412405625886828ef0	0
109214	93a4d06b3f2000ee5971d0f1073de313	0
109215	b40866089ad918919693aaf8f358f37d	3
109216	bed506f4ff578dc0948c02338266fa2c	9
109217	d6af99a696ce9c46b321a44d0bc4a28c	0

	teacher_id	teacher_number_of_previously_posted_projects
109218	91f5c69bf72c82edb9bc1f55596d8d95	4
109219	9a6784108c76576565f46446594f99c4	0
109220	19c622a38a0cd76c2e9dbcc40541fabd	3
109221	031e299278ac511616b2950fc1312a55	1
109224	651866d8215616f65934aafcbee21bf5	7
109225	c628dff071aa8028b08a5d4972bef2a1	1
109227	e15cd063caa1ce11a45f2179535105f2	0
109229	523f95270c6aec82bee90e3931ceeeca	0
109230	ee59900af64d9244487e7ed87d0bc423	0
109231	9d7a4dae637d1a170778e2db1515e574	7
109232	c116af7435274872bea9ff123a69cf6a	0
109233	b90258ab009b84e0dc11a7186d597141	1
109234	dd68d9fbae85933c0173c13f66291cbe	9
109235	9636fcacbf65eb393133a94c83c4a0d4	1
109236	2950019dd34581dbcddcae683e74207a	6
109237	07fd2c09f8dfcc74dbb161e1ec3df1fe	4
109240	58c112dcb2f1634a4d4236bf0dcdcb31	10
109241	3e5c98480f4f39d465837b2955df6ae0	0
109242	fe10e79b7aeb570dfac87eeea7e9a8f1	26
109243	fadf72d6cd83ce6074f9be78a6fcd374	0
109244	1984d915cc8b91aa16b4d1e6e39296c6	0
109245	cdbfd04aa041dc6739e9e576b1fb1478	3
109246	6d5675dbfafa1371f0e2f6f1b716fe2d	0
109247	ca25d5573f2bd2660f7850a886395927	0

102889 rows × 2 columns

In [49]:

```
univariate_barplots(teacher_number_of_previously_posted_projects_temp, 'teacher_number_
of_previously_posted_projects', 'project_is_approved', top=30)
```




```
These are the top 5 results
   teacher_number_of_previously_posted_projects  project_is_approved  tota
l  \
0                                             0                24652  3001
4
1                                             1                13329  1605
8
2                                             2                 8705  1035
0
3                                             3                 5997   711
0
4                                             4                 4452   526
6

        Avg
0  0.821350
1  0.830054
2  0.841063
3  0.843460
4  0.845423
==================================================
These are the bottom 5 results
    teacher_number_of_previously_posted_projects  project_is_approved  tot
al  \
24                                           24                 405    4
49
26                                           26                 378    4
45
27                                           27                 352    3
94
29                                           29                 336    3
70
28                                           28                 313    3
52

        Avg
24  0.902004
26  0.849438
27  0.893401
29  0.908108
28  0.889205
```

The number of projects previously submitted varies from 0 to even 30 and the approval percetages also varies highly

There is seen that around 82 % approval is there for even the teachers who have earlier not submitted even a single project from this we can convey that prior submission of project is not that much required for the project to get approved

Project Resource Summary Analysis

In [50]:

```
# First let us get the project resource summary in seperate variable

summary = []

for i in project_data["project_resource_summary"]:
    summary.append(i)

summary[0:5:1]
```

Out[50]:

```
['My students need opportunities to practice beginning reading skills in E
nglish at home.',
 'My students need a projector to help with viewing educational programs',
 'My students need shine guards, athletic socks, Soccer Balls, goalie glov
es, and training materials for the upcoming Soccer season.',
 'My students need to engage in Reading and Math in a way that will inspir
e them with these Mini iPads!',
 'My students need hands on practice in mathematics. Having fun and person
alized journals and charts will help them be more involved in our daily Ma
th routines.']
```

In [51]:

```
print(len(summary))

print(project_data['project_resource_summary'].count())

# We should get both the values same as both mean the same thing only
```

```
109248
109248
```

In [52]:

```
# Now we will check if the project resource summary consist of a numerical digit and st
ore that numerical digit
# with the corresponding index for which project resource summary it is contained in th
e form of a dictionary


numeric_value = {}

for i in tqdm(range(0,len(summary),1)):
    for j in summary[i].split(" "):
        #how to check if a string has a digit is seen from here https://stackoverflow.c
om/questions/19859282/check-if-a-string-contains-a-number
        if j.isdigit():
            numeric_value[i] = int(j)
```

```
100%|██████████████████████████████████████████████████████████████████████████████
██| 109248/109248 [00:00<00:00, 347310.03it/s]
```

In [53]:

```
# now numeric_value is a dictionary which consist of which digit is present for which w
hich resource summary text

# Also note that here we have key value pairs of only those resource summary texts whic
h had a numeric digit in their summary for all others we dont even have their key also
```

In [54]:

```
print("The number of project resource summaries which have numeric digits in their summ
ary are ",len(numeric_value))

print("Percentage of resource summaries which have a numeric digit in them", 100.0*(len
(numeric_value)/len(summary)))
```

```
The number of project resource summaries which have numeric digits in thei
r summary are  11237
Percentage of resource summaries which have a numeric digit in them 10.285
771821909783
```

Only 10% of the resource summaries have a numeric digit in them which basically means only 10% of the people have actually stated for the amount of the resources they need exactly

In [55]:

```
# Adding 0 for those resource summaries in the value column which have no numeric value
in their resource summary

numeric_value_total = {}

for y in tqdm(range(0,len(summary),1)):
    if y in numeric_value.keys():
        numeric_value_total[y] = numeric_value[y]
    else:
        numeric_value_total[y] = 0

print(len(numeric_value_total))
```

```
100%|██████████████████████████████████████████████████████████████████████████████
| 109248/109248 [00:00<00:00, 2028012.67it/s]

109248
```

In [56]:

```
# Now we dont have any use of what numerical digit was there in front of which resource
summary what we are
# concerned about is only that whether the resource summary had a numerical digit or no
t hence we will convert all the numeric_values
# to 0 and 1 type where 0 will stand for not containing  the numerical digit and 1 will
stand for containing the numerical digit


numeric_value_binary = []

for m in numeric_value_total.values():
    if m > 0:
        numeric_value_binary.append(1)
    else:
        numeric_value_binary.append(0)

print(numeric_value_binary[0:40:1])

print(len(numeric_value_binary))
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
109248
```

In [57]:

```
# Now we are adding a new column in our data which will tell if our resource data summa
ry consist of a digit or not

project_data['digit_in_resource_summary'] = numeric_value_binary

project_data.head(5)
```

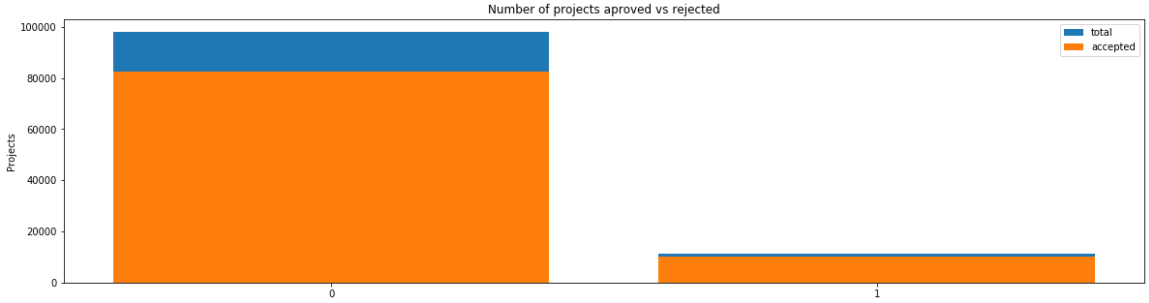
Out[57]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	

5 rows × 21 columns

In [58]:

```
univariate_barplots(teacher_number_of_previously_posted_projects_temp, 'digit_in_resour
ce_summary', 'project_is_approved', top=30)
```




```
These are the top 5 results
   digit_in_resource_summary  project_is_approved  total       Avg
0                          0                82563  98012  0.842376
1                          1                10143  11236  0.902723
==================================================
These are the bottom 5 results
   digit_in_resource_summary  project_is_approved  total       Avg
0                          0                82563  98012  0.842376
1                          1                10143  11236  0.902723
```

There are only 11236 projects whose resource summary had a numeric digit out of which 10143 are approved also which is around 90% approval rate which in the other case is only 84% hence a numeric digit expressing how much of resources the company needs helps in easily approving it

1.3 Text preprocessing

1.3.1 Essay Text

In [59]:

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

Out[59]:

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

2 rows × 21 columns

In [60]:

```
# 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[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

My students are English learners that are working on English as their seco nd or third languages. We are a melting pot of refugees, immigrants, and n ative-born Americans bringing the gift of language to our school. \r\n\r\n We have over 24 languages represented in our English Learner program with students at every level of mastery. We also have over 40 countries repres ented with the families within our school. Each student brings a wealth o f knowledge and experiences to us that open our eyes to new cultures, beli efs, and respect.\"The limits of your language are the limits of your worl d.\"-Ludwig Wittgenstein Our English learner's have a strong support syst em at home that begs for more resources. Many times our parents are learn ing to read and speak English along side of their children. Sometimes thi s creates barriers for parents to be able to help their child learn phonet ics, letter recognition, and other reading skills.\r\n\r\nBy providing the se dvd's and players, students are able to continue their mastery of the E nglish language even if no one at home is able to assist. All families wi th 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 English Learner Teacher and will be sent home regularly to watch. The videos are to help the child develop early reading skills.\r\n\r\nPare nts that do not have access to a dvd player will have the opportunity to c heck out a dvd player to use for the year. The plan is to use these video s and educational dvd's for the years to come for other EL students.\r\nna nnan

==========

The 51 fifth grade students that will cycle through my classroom this year all love learning, at least most of the time. At our school, 97.3% of the students receive free or reduced price lunch. Of the 560 students, 97.3% a re minority students. \r\nThe school has a vibrant community that loves to get together and celebrate. Around Halloween there is a whole school parad e to show off the beautiful costumes that students wear. On Cinco de Mayo we put on a big festival with crafts made by the students, dances, and gam es. At the end of the year the school hosts a carnival to celebrate the ha rd work put in during the school year, with a dunk tank being the most pop ular activity.My students will use these five brightly colored Hokki stool s 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 have an i ndividual 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 used by th e 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 mis sing, 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 in group wi th me on the Hokki Stools, they are always moving, but at the same time do ing their work. Anytime the students get to pick where they can sit, the H okki Stools are the first to be taken. 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 student s to do desk work and move at the same time. These stools will help studen ts 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, th ese chairs will take away the barrier that exists in schools for a child w ho can't sit still.nannan

==========

How do you remember your days of school? Was it in a sterile environment w ith plain walls, rows of desks, and a teacher in front of the room? A typi cal day in our room is nothing like that. I work hard to create a warm inv iting 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 r

aces 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'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 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

==

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

==

The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy school has 803 students which is makeup is 97.6% African-American, making up the largest segment of the student body. A typical school in Dallas is made up of 23.2% African-American students. Most of the students are on free or reduced lunch. We aren't receiving doctors, lawyers, or engineers children from rich backgrounds or neighborhoods. As an educator I am inspiring minds of young children and we focus not only on academics but one smart, effective, efficient, and disciplined students with good character.In our classroom we can utilize the Bluetooth for swift transitions during class. I use a speaker which doesn't amplify the sound enough to receive the message. Due to the volume of my speaker my students can't hear videos or books clearly and it isn't making the lessons as meaningful. But with the blue tooth speaker my students will be able to hear and I can stop, pause and replay it at any time.\r\nThe cart will allow me to have more room for storage of things that are needed for the day and has an extra part to it I can use. The table top chart has all of the letter, words and pictures for students to learn about different letters and it is more accessible.nannan

==

In [61]:

```
# 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 [62]:

```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations. \r\n\r\nThe materials we have are the ones I seek out fo
r my students. I teach in a Title I school where most of the students rece
ive free or reduced price lunch.  Despite their disabilities and limitatio
ns, my students love coming to school and come eager to learn and explore.
Have you ever felt like you had ants in your pants and you needed to groov
e and move as you were in a meeting? This is how my kids feel all the tim
e. The want to be able to move as they learn or so they say.Wobble chairs
are the answer and I love then because they develop their core, which enha
nces gross motor and in Turn fine motor skills. \r\nThey also want to lear
n through games, my kids do not want to sit and do worksheets. They want t
o learn to count by jumping and playing. Physical engagement is the key to
our success. The number toss and color and shape mats can make that happe
n. My students will forget they are doing work and just have the fun a 6 y
ear old deserves.nannan
==================================================
```

In [63]:

```
# \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)
```

```
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations.     The materials we have are the ones I seek out for my
students. I teach in a Title I school where most of the students receive f
ree or reduced price lunch.  Despite their disabilities and limitations, m
y students love coming to school and come eager to learn and explore.Have
you ever felt like you had ants in your pants and you needed to groove and
move as you were in a meeting? This is how my kids feel all the time. The
want to be able to move as they learn or so they say.Wobble chairs are the
answer and I love then because they develop their core, which enhances gro
ss motor and in Turn fine motor skills.   They also want to learn through
games, my kids do not want to sit and do worksheets. They want to learn to
count by jumping and playing. Physical engagement is the key to our succes
s. The number toss and color and shape mats can make that happen. My stude
nts will forget they are doing work and just have the fun a 6 year old des
erves.nannan
```

In [64]:

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

```
My kindergarten students have varied disabilities ranging from speech and
language delays cognitive delays gross fine motor delays to autism They ar
e eager beavers and always strive to work their hardest working past their
limitations The materials we have are the ones I seek out for my students
I teach in a Title I school where most of the students receive free or red
uced price lunch Despite their disabilities and limitations my students lo
ve coming to school and come eager to learn and explore Have you ever felt
like you had ants in your pants and you needed to groove and move as you w
ere in a meeting This is how my kids feel all the time The want to be able
to move as they learn or so they say Wobble chairs are the answer and I lo
ve then because they develop their core which enhances gross motor and in
Turn fine motor skills They also want to learn through games my kids do no
t want to sit and do worksheets They want to learn to count by jumping and
playing Physical engagement is the key to our success The number toss and
color and shape mats can make that happen My students will forget they are
doing work and just have the fun a 6 year old deserves nannan
```

In [65]:

```
# 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', 'each', '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', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [66]:

```
# 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%|████████████████████████████████████████████████████████████████████████████████| 109248/109248 [00:49<00:00, 2239.79it/s]
```

In [67]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[67]:

```
'my kindergarten students varied disabilities ranging speech language dela
ys cognitive delays gross fine motor delays autism they eager beavers alwa
ys strive work hardest working past limitations the materials ones i seek
students i teach title i school students receive free reduced price lunch
despite disabilities limitations students love coming school come eager le
arn explore have ever felt like ants pants needed groove move meeting this
kids feel time the want able move learn say wobble chairs answer i love de
velop core enhances gross motor turn fine motor skills they also want lear
n games kids not want sit worksheets they want learn count jumping playing
physical engagement key success the number toss color shape mats make happ
en my students forget work fun 6 year old deserves nannan'
```

Project title Text

Now we will simply apply the above preprocessing steps on the project title as well as it is also a text feature

In [68]:

```
# printing some random titles.
print(project_data['project_title'].values[0])
print("="*50)
print(project_data['project_title'].values[150])
print("="*50)
print(project_data['project_title'].values[1000])
print("="*50)
print(project_data['project_title'].values[20000])
print("="*50)
print(project_data['project_title'].values[99999])
print("="*50)
```

```
Educational Support for English Learners at Home
==================================================
More Movement with Hokki Stools
==================================================
Sailing Into a Super 4th Grade Year
==================================================
We Need To Move It While We Input It!
==================================================
Inspiring Minds by Enhancing the Educational Experience
==================================================
```

We have already written the preprocessing codes for different preprocessing approaches now we will simply use those codes on the project titles

In [69]:

```
preprocessed_project_titles = []

for t in tqdm(project_data["project_title"]):
    title = decontracted(t)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    title = ' '.join(f for f in title.split() if f not in stopwords)
    preprocessed_project_titles.append(title.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████
██| 109248/109248 [00:02<00:00, 47460.69it/s]
```

In [70]:

```
project_data['preprocessed_project_titles'] = preprocessed_project_titles
```

In [71]:

```
# printing some random titles after preprocessing

print(project_data['preprocessed_project_titles'].values[0])
print("="*50)
print(project_data['preprocessed_project_titles'].values[150])
print("="*50)
print(project_data['preprocessed_project_titles'].values[1000])
print("="*50)
print(project_data['preprocessed_project_titles'].values[20000])
print("="*50)
print(project_data['preprocessed_project_titles'].values[99999])
print("="*50)
```

```
educational support english learners home
==================================================
more movement hokki stools
==================================================
sailing into super 4th grade year
==================================================
we need to move it while we input it
==================================================
inspiring minds enhancing educational experience
==================================================
```

Preparing data for models

In [72]:

```
project_data.columns
```

Out[72]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_ti
tle',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'clean_categories', 'clean_subcategories', 'essay', 'price', 'quant
ity',
       'digit_in_resource_summary', 'preprocessed_project_titles'],
      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

In [73]:

```
# we use count vectorizer to convert the values into one hot encoded features
# we are working on the individual category data here

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', 'AppliedLearnin
g', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (109248, 9)
```

In [74]:

```
type(categories_one_hot)
```

Out[74]:

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

Every column in the sparse matrix categories_one_hot will refer to each individual category

In [75]:

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

# we are working on project subject sub categories now

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


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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvemen
t', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'Nutrition
Education', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts',
'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Musi
c', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'A
ppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Lit
eracy']
Shape of matrix after one hot encodig  (109248, 30)
```

In [76]:

```
type(sub_categories_one_hot)
```

Out[76]:

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

Every column in the sparse matrix sub_categories_one_hot will refer to each individual category

One Hot encoding of the School States As they are also categorical features only

In [77]:

```
type(list(project_data['school_state']))
```

Out[77]:

```
list
```

In [78]:

```
mylist = list(project_data['school_state'])
```

In [79]:

```
# We are removing the duplicate values from our list of the state codes

# Source :- https://www.w3schools.com/python/python_howto_remove_duplicates.asp

mylist_actual = list(dict.fromkeys(mylist))
```

In [80]:

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

# now we are working on school state data

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=mylist_actual, lowercase=False, binary=True)
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())


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

```
['IN', 'FL', 'AZ', 'KY', 'TX', 'CT', 'GA', 'SC', 'NC', 'CA', 'NY', 'OK',
'MA', 'NV', 'OH', 'PA', 'AL', 'LA', 'VA', 'AR', 'WA', 'WV', 'ID', 'TN', 'M
S', 'CO', 'UT', 'IL', 'MI', 'HI', 'IA', 'RI', 'NJ', 'MO', 'DE', 'MN', 'M
E', 'WY', 'ND', 'OR', 'AK', 'MD', 'WI', 'SD', 'NE', 'NM', 'DC', 'KS', 'M
T', 'NH', 'VT']
Shape of matrix after one hot encoding  (109248, 51)
```

One hot encoding of teacher prefix category

In [139]:

```
#print(list(project_data['teacher_prefix']))
```

As we can see that there are lot of duplicates above hence we need to remove them first

In [82]:

```
mylist_teacher_prefix = list(project_data['teacher_prefix'])
```

In [83]:

```
# We are removing the duplicate values from our list of the teacher prefix

# Source :- https://www.w3schools.com/python/python_howto_remove_duplicates.asp

mylist_teacher_prefix_actual = list(dict.fromkeys(mylist_teacher_prefix))
```

In [84]:

```
mylist_teacher_prefix_actual
```

Out[84]:

```
['Mrs.', 'Mr.', 'Ms.', 'Teacher', nan, 'Dr.']
```

In [89]:

```
# removing the nan from the teacher prefix category as there is no such category of tea
cher which exists

mylist_teacher_prefix_actual = [p for p in mylist_teacher_prefix_actual if str(p) != 'n
an']
mylist_teacher_prefix_actual
```

Out[89]:

```
['Mrs.', 'Mr.', 'Ms.', 'Teacher', 'Dr.']
```

In [90]:

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

# now we are working on teacher prefix data

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=mylist_teacher_prefix_actual, lowercase=False,
binary=True)

# I was getting an error like "np.nan is an invalid document, expected byte or unicode
 string."
# below id the solution

# https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerr
or-np-nan-is-an-invalid-document


vectorizer.fit(project_data['teacher_prefix'].values.astype('U'))
print(vectorizer.get_feature_names())


teacher_prefix_one_hot = vectorizer.transform(project_data['teacher_prefix'].values.ast
ype('U'))
print("Shape of matrix after one hot encoding ",teacher_prefix_one_hot.shape)
```

```
['Mrs.', 'Mr.', 'Ms.', 'Teacher', 'Dr.']
Shape of matrix after one hot encoding  (109248, 5)
```

One Hot encoding of project grade category

In [91]:

```
#print(list(project_data['project_grade_category']))
```

As we can see that there are lot of duplicates above hence we need to remove them first

In [92]:

```
mylist_project_grade_category = list(project_data['project_grade_category'])
```

In [93]:

```
# We are removing the duplicate values from our list of the project grade category

# Source :- https://www.w3schools.com/python/python_howto_remove_duplicates.asp

mylist_project_grade_category_actual = list(dict.fromkeys(mylist_project_grade_category
))
```

In [94]:

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

# now we are working on project grade category data

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=mylist_project_grade_category_actual, lowercase
=False, binary=True)


vectorizer.fit(project_data['project_grade_category'].values)
print(vectorizer.get_feature_names())


project_grade_one_hot = vectorizer.transform(project_data['project_grade_category'].val
ues)
print("Shape of matrix after one hot encoding ",project_grade_one_hot.shape)
```

```
['Grades PreK-2', 'Grades 6-8', 'Grades 3-5', 'Grades 9-12']
Shape of matrix after one hot encoding  (109248, 4)
```

Vectorizing Text data

Bag of words

We are first applying BOW on the preprocessed essays data

In [95]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
# min_df stands for that purpose only

vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encoding ",text_bow.shape)
```

```
Shape of matrix after one hot encoding  (109248, 16623)
```

Here 16623 corresponds to the fact that there are 16623 different words in our essays vocabulary

Now we are applying BAG of words on project title

In [96]:

```
# We are considering only the words which appeared in at least 3 documents(rows or proj
ects).
# min_df stands for that purpose only

vectorizer = CountVectorizer(min_df=3)
title_text_bow = vectorizer.fit_transform(preprocessed_project_titles)
print("Shape of matrix after one hot encoding ",title_text_bow.shape)
```

```
Shape of matrix after one hot encoding  (109248, 7013)
```

Here 7013 corresponds to the fact that there are 7013 different words in our project title vocabulary

TFIDF

We are first applying TFIDF on the preprocessed essays data

In [97]:

```
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 encoding ",text_tfidf.shape)
```

```
Shape of matrix after one hot encoding  (109248, 16623)
```

We are first applying TFIDF on the preprocessed project title

In [98]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=3)
title_text_tfidf = vectorizer.fit_transform(preprocessed_project_titles)
print("Shape of matrix after one hot encoding ",title_text_tfidf.shape)
```

```
Shape of matrix after one hot encoding  (109248, 7013)
```

Using Pretrained Models: Avg W2V

In [99]:

```
'''
# 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-p
ickle-to-save-and-load-variables-in-python/

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

'''
```

Out[99]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/3823034
9/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove Mode
l")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    f
or line in tqdm(f):\n        splitLine = line.split()\n        word = spli
tLine[0]\n        embedding = np.array([float(val) for val in splitLine
[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," w
ords loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.
txt\')\n\n# ==========================\n#Output:\n    \n#Loading Glove M
odel\n#1917495it [06:32, 4879.69it/s]\n#Done. 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.exte
nd(i.split(\' \'))\nprint("all the words in the coupus", len(words))\nword
s = set(words)\nprint("the unique words in the coupus", len(words))\n\nint
er_words = set(model.keys()).intersection(words)\nprint("The number of wor
ds that are present in both glove vectors and our coupus",       len(inter
_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_cou
rpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i in w
ords_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec lengt
h", len(words_courpus))\n\n\n# stronging variables into pickle files pytho
n: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables
-in-python/\n\nimport pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n
pickle.dump(words_courpus, f)\n\n'
```

In [100]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
ickle-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 [101]:

```
# average Word2Vec
# compute average word2vec for each preprocessed essay.

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]))
```

```
100%|██████████████████████████████████████████████████████████████████████
███| 109248/109248 [00:26<00:00, 4189.84it/s]

109248
300
```

Now using this pre trained model on Project_title

In [102]:

```
# average Word2Vec
# compute average word2vec for each preprocessed project title.

avg_w2v_vectors_title = []; # the avg-w2v for each sentence/review is stored in this li
st
for sentence in tqdm(preprocessed_project_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_title.append(vector)

print(len(avg_w2v_vectors_title))
print(len(avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██| 109248/109248 [00:01<00:00, 81866.79it/s]

109248
300
```

Now training for TFIDF W2V

In [103]:

```
# Similarly you can vectorize for title also

tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_project_titles)
# 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 [104]:

```
# tfidf Word2Vec
# computing average word2vec for each Project Title is stored in this list

tfidf_w2v_vectors_title = []; # the tfidf-w2v for each title
for sentence in tqdm(preprocessed_project_titles): # for each title
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the title
    for word in sentence.split(): # for each word in a title
        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((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title.append(vector)

print(len(tfidf_w2v_vectors_title))
print(len(tfidf_w2v_vectors_title[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██| 109248/109248 [00:02<00:00, 36994.38it/s]

109248
300
```

Vectorizing Numerical features

In [105]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.pr
eprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

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

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_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1
))
```

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

Seeing the mean value average cost of a project is around 299 dollars and standard deviation is around 368 dollars

In [106]:

```
price_standardized
```

Out[106]:

```
array([[-0.3905327 ],
       [ 0.00239637],
       [ 0.59519138],
       ...,
       [-0.15825829],
       [-0.61243967],
       [-0.51216657]])
```

Vectorizing the numerical feature - Teacher number of previously posted projects.

In [107]:

```
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

from sklearn.preprocessing import StandardScaler

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

previous_posted_projects_scalar = StandardScaler()

previous_posted_projects_scalar.fit(project_data['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

# finding the mean and standard deviation of this data

print(f"Mean : {previous_posted_projects_scalar.mean_[0]}, Standard deviation : {np.sqrt(previous_posted_projects_scalar.var_[0])}")

# Now standardize the data with above maen and variance.

previous_posted_projects_standardized = previous_posted_projects_scalar.transform(project_data['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
```

```
C:\Users\RASHU TYAGI\Anaconda3\lib\site-packages\sklearn\utils\validation.
py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.


Mean : 11.153165275336848, Standard deviation : 27.77702641477403

C:\Users\RASHU TYAGI\Anaconda3\lib\site-packages\sklearn\utils\validation.
py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

On a average there are 11 previously submitted projects by teachers with sandard deviation of 27

Now an important step is to merge all the data we calculated above so that we can work for visualisation on it

Merging all the above features

In [108]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(school_state_one_hot.shape)
print(teacher_prefix_one_hot.shape)
print(project_grade_one_hot.shape)
print(text_bow.shape)
print(title_text_bow.shape)
print(text_tfidf.shape)
print(title_text_tfidf.shape)
print(len(avg_w2v_vectors_title))
print(len(tfidf_w2v_vectors_title))
print(price_standardized.shape)
print(previous_posted_projects_standardized.shape)
```

```
(109248, 9)
(109248, 30)
(109248, 51)
(109248, 5)
(109248, 4)
(109248, 16623)
(109248, 7013)
(109248, 16623)
(109248, 7013)
109248
109248
(109248, 1)
(109248, 1)
```

Now from above we can see that we have sparse matrices as well as dense matrices and we need to combine them

TSNE with `BOW` encoding of `project_title` feature

Combining all the other features into one data matrix

In [109]:

```
# 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,school_state_one_hot , project_grade_one_hot, teacher_prefix_one_hot, price_standardized, previous_posted_projects_standardized, title_text_bow))

X.shape
```

Out[109]:

```
(109248, 7114)
```

In [110]:

```
# we are compressing the row matrix in order to count duplicate values as only one

X = X.tocsr()
X_actual = X[0:5000,:]
```

In [111]:

```
X_actual.shape
```

Out[111]:

```
(5000, 7114)
```

In [112]:

```
from sklearn.manifold import TSNE
import seaborn as sn
```

In [113]:

```
X_actual = X_actual.toarray()
model = TSNE(n_components = 2, perplexity = 50,n_iter = 5000, random_state = 0,verbose
= 5)
tsne_bow = model.fit_transform(X_actual)
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 5000 samples in 1.425s...
[t-SNE] Computed neighbors for 5000 samples in 254.876s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.906721
[t-SNE] Computed conditional probabilities in 0.241s
[t-SNE] Iteration 50: error = 80.4645920, gradient norm = 0.0641310 (50 it
erations in 4.774s)
[t-SNE] Iteration 100: error = 79.4814987, gradient norm = 0.0781803 (50 i
terations in 3.266s)
[t-SNE] Iteration 150: error = 79.0817261, gradient norm = 0.0832111 (50 i
terations in 2.882s)
[t-SNE] Iteration 200: error = 79.4602432, gradient norm = 0.0625924 (50 i
terations in 2.715s)
[t-SNE] Iteration 250: error = 79.2712936, gradient norm = 0.0682343 (50 i
terations in 2.601s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 79.271
294
[t-SNE] Iteration 300: error = 2.2918966, gradient norm = 0.0011459 (50 it
erations in 2.752s)
[t-SNE] Iteration 350: error = 2.0759370, gradient norm = 0.0004228 (50 it
erations in 2.340s)
[t-SNE] Iteration 400: error = 1.9976540, gradient norm = 0.0002232 (50 it
erations in 2.323s)
[t-SNE] Iteration 450: error = 1.9561896, gradient norm = 0.0001870 (50 it
erations in 2.333s)
[t-SNE] Iteration 500: error = 1.9310809, gradient norm = 0.0001373 (50 it
erations in 2.352s)
[t-SNE] Iteration 550: error = 1.9158256, gradient norm = 0.0001012 (50 it
erations in 2.353s)
[t-SNE] Iteration 600: error = 1.9054778, gradient norm = 0.0000887 (50 it
erations in 2.344s)
[t-SNE] Iteration 650: error = 1.8992432, gradient norm = 0.0000776 (50 it
erations in 2.317s)
[t-SNE] Iteration 700: error = 1.8950548, gradient norm = 0.0000610 (50 it
erations in 2.302s)
[t-SNE] Iteration 750: error = 1.8912011, gradient norm = 0.0000577 (50 it
erations in 2.318s)
[t-SNE] Iteration 800: error = 1.8881503, gradient norm = 0.0000518 (50 it
erations in 2.320s)
[t-SNE] Iteration 850: error = 1.8855035, gradient norm = 0.0000578 (50 it
erations in 2.302s)
[t-SNE] Iteration 900: error = 1.8835144, gradient norm = 0.0000480 (50 it
erations in 2.293s)
[t-SNE] Iteration 950: error = 1.8821071, gradient norm = 0.0000440 (50 it
erations in 2.333s)
[t-SNE] Iteration 1000: error = 1.8809301, gradient norm = 0.0000394 (50 i
terations in 2.325s)
[t-SNE] Iteration 1050: error = 1.8797413, gradient norm = 0.0000377 (50 i
terations in 2.306s)
[t-SNE] Iteration 1100: error = 1.8785573, gradient norm = 0.0000396 (50 i
terations in 2.305s)
[t-SNE] Iteration 1150: error = 1.8777356, gradient norm = 0.0000410 (50 i
terations in 2.308s)
[t-SNE] Iteration 1200: error = 1.8770760, gradient norm = 0.0000362 (50 i
terations in 2.295s)
[t-SNE] Iteration 1250: error = 1.8765290, gradient norm = 0.0000304 (50 i
```

```
terations in 2.306s)
[t-SNE] Iteration 1300: error = 1.8756980, gradient norm = 0.0000330 (50 i
terations in 2.307s)
[t-SNE] Iteration 1350: error = 1.8750217, gradient norm = 0.0000322 (50 i
terations in 2.300s)
[t-SNE] Iteration 1400: error = 1.8744754, gradient norm = 0.0000308 (50 i
terations in 2.280s)
[t-SNE] Iteration 1450: error = 1.8740635, gradient norm = 0.0000286 (50 i
terations in 2.284s)
[t-SNE] Iteration 1500: error = 1.8736203, gradient norm = 0.0000286 (50 i
terations in 2.287s)
[t-SNE] Iteration 1550: error = 1.8732057, gradient norm = 0.0000269 (50 i
terations in 2.291s)
[t-SNE] Iteration 1600: error = 1.8727623, gradient norm = 0.0000274 (50 i
terations in 2.280s)
[t-SNE] Iteration 1650: error = 1.8723146, gradient norm = 0.0000267 (50 i
terations in 2.302s)
[t-SNE] Iteration 1700: error = 1.8718575, gradient norm = 0.0000299 (50 i
terations in 2.298s)
[t-SNE] Iteration 1750: error = 1.8715829, gradient norm = 0.0000259 (50 i
terations in 2.300s)
[t-SNE] Iteration 1800: error = 1.8712797, gradient norm = 0.0000250 (50 i
terations in 2.284s)
[t-SNE] Iteration 1850: error = 1.8709583, gradient norm = 0.0000261 (50 i
terations in 2.278s)
[t-SNE] Iteration 1900: error = 1.8705487, gradient norm = 0.0000241 (50 i
terations in 2.294s)
[t-SNE] Iteration 1950: error = 1.8702501, gradient norm = 0.0000265 (50 i
terations in 2.327s)
[t-SNE] Iteration 2000: error = 1.8700128, gradient norm = 0.0000242 (50 i
terations in 2.306s)
[t-SNE] Iteration 2050: error = 1.8697006, gradient norm = 0.0000273 (50 i
terations in 2.281s)
[t-SNE] Iteration 2100: error = 1.8693939, gradient norm = 0.0000249 (50 i
terations in 2.306s)
[t-SNE] Iteration 2150: error = 1.8692200, gradient norm = 0.0000240 (50 i
terations in 2.303s)
[t-SNE] Iteration 2200: error = 1.8690413, gradient norm = 0.0000235 (50 i
terations in 2.308s)
[t-SNE] Iteration 2250: error = 1.8687720, gradient norm = 0.0000205 (50 i
terations in 2.301s)
[t-SNE] Iteration 2300: error = 1.8685969, gradient norm = 0.0000224 (50 i
terations in 2.307s)
[t-SNE] Iteration 2350: error = 1.8684056, gradient norm = 0.0000211 (50 i
terations in 2.278s)
[t-SNE] Iteration 2400: error = 1.8682102, gradient norm = 0.0000223 (50 i
terations in 2.298s)
[t-SNE] Iteration 2450: error = 1.8680613, gradient norm = 0.0000203 (50 i
terations in 2.287s)
[t-SNE] Iteration 2500: error = 1.8679342, gradient norm = 0.0000206 (50 i
terations in 2.326s)
[t-SNE] Iteration 2550: error = 1.8677845, gradient norm = 0.0000221 (50 i
terations in 2.315s)
[t-SNE] Iteration 2600: error = 1.8675785, gradient norm = 0.0000205 (50 i
terations in 2.290s)
[t-SNE] Iteration 2650: error = 1.8675058, gradient norm = 0.0000210 (50 i
terations in 2.283s)
[t-SNE] Iteration 2700: error = 1.8673322, gradient norm = 0.0000198 (50 i
terations in 2.283s)
[t-SNE] Iteration 2750: error = 1.8672557, gradient norm = 0.0000204 (50 i
terations in 2.297s)
```

```
[t-SNE] Iteration 2800: error = 1.8670983, gradient norm = 0.0000195 (50 i
terations in 2.292s)
[t-SNE] Iteration 2850: error = 1.8669156, gradient norm = 0.0000201 (50 i
terations in 2.275s)
[t-SNE] Iteration 2900: error = 1.8667446, gradient norm = 0.0000188 (50 i
terations in 2.296s)
[t-SNE] Iteration 2950: error = 1.8666143, gradient norm = 0.0000200 (50 i
terations in 2.294s)
[t-SNE] Iteration 3000: error = 1.8664774, gradient norm = 0.0000191 (50 i
terations in 2.285s)
[t-SNE] Iteration 3050: error = 1.8664017, gradient norm = 0.0000170 (50 i
terations in 2.290s)
[t-SNE] Iteration 3100: error = 1.8662851, gradient norm = 0.0000189 (50 i
terations in 2.296s)
[t-SNE] Iteration 3150: error = 1.8661783, gradient norm = 0.0000199 (50 i
terations in 2.296s)
[t-SNE] Iteration 3200: error = 1.8660978, gradient norm = 0.0000190 (50 i
terations in 2.321s)
[t-SNE] Iteration 3250: error = 1.8659953, gradient norm = 0.0000187 (50 i
terations in 2.333s)
[t-SNE] Iteration 3300: error = 1.8658500, gradient norm = 0.0000190 (50 i
terations in 2.296s)
[t-SNE] Iteration 3350: error = 1.8657905, gradient norm = 0.0000170 (50 i
terations in 2.295s)
[t-SNE] Iteration 3400: error = 1.8657027, gradient norm = 0.0000186 (50 i
terations in 2.297s)
[t-SNE] Iteration 3450: error = 1.8655680, gradient norm = 0.0000174 (50 i
terations in 2.279s)
[t-SNE] Iteration 3500: error = 1.8654783, gradient norm = 0.0000194 (50 i
terations in 2.296s)
[t-SNE] Iteration 3550: error = 1.8654369, gradient norm = 0.0000177 (50 i
terations in 2.290s)
[t-SNE] Iteration 3600: error = 1.8653647, gradient norm = 0.0000188 (50 i
terations in 2.300s)
[t-SNE] Iteration 3650: error = 1.8652779, gradient norm = 0.0000158 (50 i
terations in 2.289s)
[t-SNE] Iteration 3700: error = 1.8652166, gradient norm = 0.0000159 (50 i
terations in 2.298s)
[t-SNE] Iteration 3750: error = 1.8651512, gradient norm = 0.0000171 (50 i
terations in 2.288s)
[t-SNE] Iteration 3800: error = 1.8650323, gradient norm = 0.0000176 (50 i
terations in 2.291s)
[t-SNE] Iteration 3850: error = 1.8648993, gradient norm = 0.0000195 (50 i
terations in 2.304s)
[t-SNE] Iteration 3900: error = 1.8648492, gradient norm = 0.0000184 (50 i
terations in 2.302s)
[t-SNE] Iteration 3950: error = 1.8647705, gradient norm = 0.0000188 (50 i
terations in 2.291s)
[t-SNE] Iteration 4000: error = 1.8647325, gradient norm = 0.0000162 (50 i
terations in 2.277s)
[t-SNE] Iteration 4050: error = 1.8647032, gradient norm = 0.0000178 (50 i
terations in 2.281s)
[t-SNE] Iteration 4100: error = 1.8646023, gradient norm = 0.0000197 (50 i
terations in 2.319s)
[t-SNE] Iteration 4150: error = 1.8645638, gradient norm = 0.0000168 (50 i
terations in 2.313s)
[t-SNE] Iteration 4200: error = 1.8644994, gradient norm = 0.0000169 (50 i
terations in 2.287s)
[t-SNE] Iteration 4250: error = 1.8644712, gradient norm = 0.0000169 (50 i
terations in 2.322s)
[t-SNE] Iteration 4300: error = 1.8643674, gradient norm = 0.0000170 (50 i
```

```
terations in 2.311s)
[t-SNE] Iteration 4350: error = 1.8642981, gradient norm = 0.0000209 (50 i
terations in 2.301s)
[t-SNE] Iteration 4400: error = 1.8642493, gradient norm = 0.0000158 (50 i
terations in 2.296s)
[t-SNE] Iteration 4450: error = 1.8641571, gradient norm = 0.0000251 (50 i
terations in 2.293s)
[t-SNE] Iteration 4500: error = 1.8640599, gradient norm = 0.0000197 (50 i
terations in 2.297s)
[t-SNE] Iteration 4550: error = 1.8639449, gradient norm = 0.0000153 (50 i
terations in 2.304s)
[t-SNE] Iteration 4600: error = 1.8638388, gradient norm = 0.0000176 (50 i
terations in 2.312s)
[t-SNE] Iteration 4650: error = 1.8637356, gradient norm = 0.0000167 (50 i
terations in 2.301s)
[t-SNE] Iteration 4700: error = 1.8637193, gradient norm = 0.0000170 (50 i
terations in 2.313s)
[t-SNE] Iteration 4750: error = 1.8635790, gradient norm = 0.0000180 (50 i
terations in 2.300s)
[t-SNE] Iteration 4800: error = 1.8635428, gradient norm = 0.0000171 (50 i
terations in 2.318s)
[t-SNE] Iteration 4850: error = 1.8634509, gradient norm = 0.0000162 (50 i
terations in 2.295s)
[t-SNE] Iteration 4900: error = 1.8633739, gradient norm = 0.0000174 (50 i
terations in 2.296s)
[t-SNE] Iteration 4950: error = 1.8633103, gradient norm = 0.0000156 (50 i
terations in 2.301s)
[t-SNE] Iteration 5000: error = 1.8632411, gradient norm = 0.0000161 (50 i
terations in 2.292s)
[t-SNE] KL divergence after 5000 iterations: 1.863241
```

In [114]:

```
tsne_bow.shape
```

Out[114]:

```
(5000, 2)
```

In [115]:

```
# getting the column which states whether the project is approved or not

approval = project_data["project_is_approved"]
approval_actual = approval[0: 5000]
len(approval_actual)
```

Out[115]:

```
5000
```

In [116]:

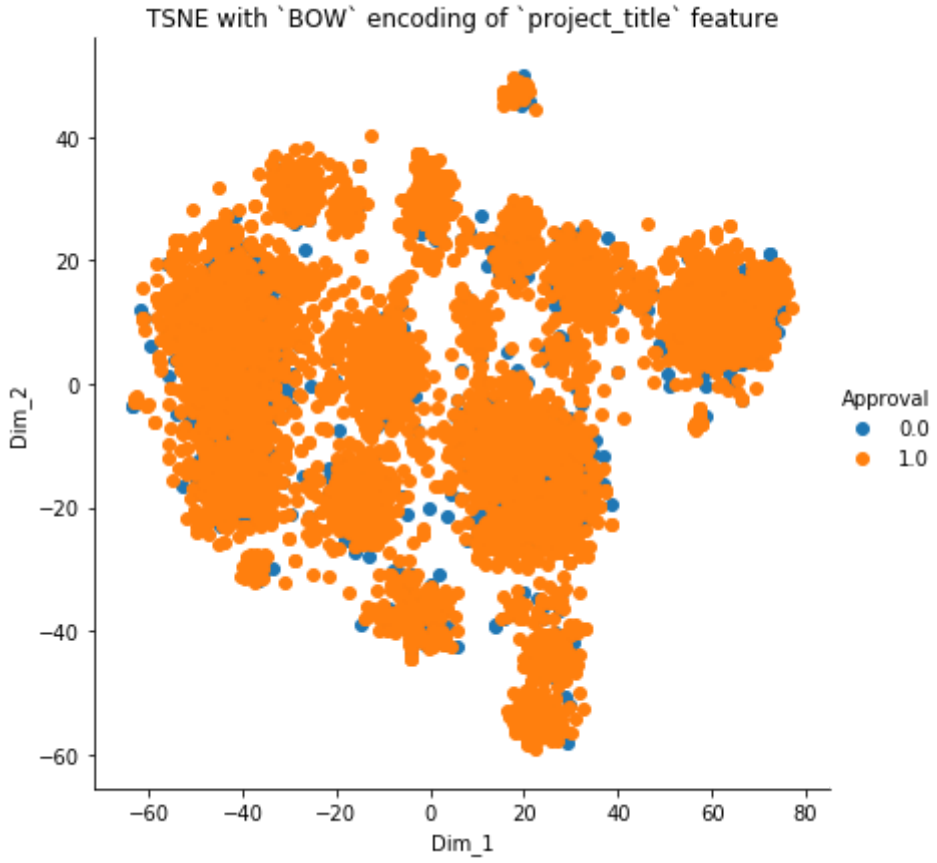
```
tsne_bow = np.column_stack((tsne_bow,approval_actual))
tsne_bow.shape
```

Out[116]:

```
(5000, 3)
```

In [117]:

```
# now creating the dataframe which we will be giving for the visualisation purpose


tsne_df_BOW = pd.DataFrame(data=tsne_bow,columns=("Dim_1", "Dim_2" , "Approval"))
sn.FacetGrid(tsne_df_BOW,hue ="Approval" , height=6).map(plt.scatter, 'Dim_1', 'Dim_2')
.add_legend()
plt.title("TSNE with `BOW` encoding of `project_title` feature")
plt.show()
```



The data points are very much well scattered thus not much of proper conclusion can be drawn from here.

There are also a lot of overlapping regions

TSNE with `TFIDF` encoding of `project_title` feature

Combining all the other features into one data matrix

In [118]:

```
# 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,school_state_one_hot , project_grade_one_hot, teacher_prefix_one_hot, price_standardized, previous_posted_projects_standardized,title_text_tfidf ))

X.shape
```

Out[118]:

```
(109248, 7114)
```

In [119]:

```
# we are compressing the row matrix in order to count duplicate values as only one

X = X.tocsr()
X_actual = X[0:5000,:]
```

In [120]:

```
X_actual.shape
```

Out[120]:

```
(5000, 7114)
```

In [121]:

```
X_actual = X_actual.toarray()
model = TSNE(n_components = 2, perplexity = 50,n_iter = 5000, random_state = 0,verbose
= 5)
tsne_tfidf = model.fit_transform(X_actual)
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 5000 samples in 1.427s...
[t-SNE] Computed neighbors for 5000 samples in 250.630s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.756508
[t-SNE] Computed conditional probabilities in 0.238s
[t-SNE] Iteration 50: error = 83.1399689, gradient norm = 0.0014676 (50 it
erations in 2.744s)
[t-SNE] Iteration 100: error = 75.4885483, gradient norm = 0.0018130 (50 i
terations in 3.031s)
[t-SNE] Iteration 150: error = 75.1433029, gradient norm = 0.0004171 (50 i
terations in 2.554s)
[t-SNE] Iteration 200: error = 75.1195068, gradient norm = 0.0000537 (50 i
terations in 2.582s)
[t-SNE] Iteration 250: error = 75.1173935, gradient norm = 0.0000347 (50 i
terations in 2.631s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 75.117
393
[t-SNE] Iteration 300: error = 1.8846128, gradient norm = 0.0011436 (50 it
erations in 2.099s)
[t-SNE] Iteration 350: error = 1.5400053, gradient norm = 0.0004603 (50 it
erations in 1.984s)
[t-SNE] Iteration 400: error = 1.3969320, gradient norm = 0.0002670 (50 it
erations in 1.950s)
[t-SNE] Iteration 450: error = 1.3212646, gradient norm = 0.0001757 (50 it
erations in 2.007s)
[t-SNE] Iteration 500: error = 1.2765913, gradient norm = 0.0001313 (50 it
erations in 1.973s)
[t-SNE] Iteration 550: error = 1.2493484, gradient norm = 0.0001049 (50 it
erations in 1.983s)
[t-SNE] Iteration 600: error = 1.2309437, gradient norm = 0.0000888 (50 it
erations in 1.994s)
[t-SNE] Iteration 650: error = 1.2189980, gradient norm = 0.0000785 (50 it
erations in 2.035s)
[t-SNE] Iteration 700: error = 1.2106904, gradient norm = 0.0000700 (50 it
erations in 2.015s)
[t-SNE] Iteration 750: error = 1.2044965, gradient norm = 0.0000669 (50 it
erations in 2.009s)
[t-SNE] Iteration 800: error = 1.1999072, gradient norm = 0.0000581 (50 it
erations in 2.004s)
[t-SNE] Iteration 850: error = 1.1959416, gradient norm = 0.0000539 (50 it
erations in 2.051s)
[t-SNE] Iteration 900: error = 1.1929940, gradient norm = 0.0000532 (50 it
erations in 2.024s)
[t-SNE] Iteration 950: error = 1.1905043, gradient norm = 0.0000496 (50 it
erations in 2.026s)
[t-SNE] Iteration 1000: error = 1.1882298, gradient norm = 0.0000472 (50 i
terations in 2.106s)
[t-SNE] Iteration 1050: error = 1.1862153, gradient norm = 0.0000482 (50 i
terations in 2.028s)
[t-SNE] Iteration 1100: error = 1.1845264, gradient norm = 0.0000484 (50 i
terations in 2.064s)
[t-SNE] Iteration 1150: error = 1.1832837, gradient norm = 0.0000471 (50 i
terations in 2.042s)
[t-SNE] Iteration 1200: error = 1.1821579, gradient norm = 0.0000425 (50 i
terations in 2.099s)
[t-SNE] Iteration 1250: error = 1.1812468, gradient norm = 0.0000432 (50 i
```

```
terations in 2.031s)
[t-SNE] Iteration 1300: error = 1.1803256, gradient norm = 0.0000391 (50 i
terations in 2.039s)
[t-SNE] Iteration 1350: error = 1.1794428, gradient norm = 0.0000419 (50 i
terations in 2.098s)
[t-SNE] Iteration 1400: error = 1.1786114, gradient norm = 0.0000415 (50 i
terations in 2.043s)
[t-SNE] Iteration 1450: error = 1.1778871, gradient norm = 0.0000388 (50 i
terations in 2.039s)
[t-SNE] Iteration 1500: error = 1.1772658, gradient norm = 0.0000343 (50 i
terations in 2.069s)
[t-SNE] Iteration 1550: error = 1.1765245, gradient norm = 0.0000341 (50 i
terations in 2.063s)
[t-SNE] Iteration 1600: error = 1.1753474, gradient norm = 0.0000368 (50 i
terations in 2.038s)
[t-SNE] Iteration 1650: error = 1.1746188, gradient norm = 0.0000344 (50 i
terations in 2.063s)
[t-SNE] Iteration 1700: error = 1.1739235, gradient norm = 0.0000314 (50 i
terations in 2.112s)
[t-SNE] Iteration 1750: error = 1.1732821, gradient norm = 0.0000301 (50 i
terations in 2.052s)
[t-SNE] Iteration 1800: error = 1.1726784, gradient norm = 0.0000301 (50 i
terations in 2.061s)
[t-SNE] Iteration 1850: error = 1.1721454, gradient norm = 0.0000304 (50 i
terations in 2.124s)
[t-SNE] Iteration 1900: error = 1.1717062, gradient norm = 0.0000291 (50 i
terations in 2.067s)
[t-SNE] Iteration 1950: error = 1.1712290, gradient norm = 0.0000312 (50 i
terations in 2.063s)
[t-SNE] Iteration 2000: error = 1.1708410, gradient norm = 0.0000302 (50 i
terations in 2.105s)
[t-SNE] Iteration 2050: error = 1.1704495, gradient norm = 0.0000294 (50 i
terations in 2.083s)
[t-SNE] Iteration 2100: error = 1.1700774, gradient norm = 0.0000297 (50 i
terations in 2.051s)
[t-SNE] Iteration 2150: error = 1.1697233, gradient norm = 0.0000323 (50 i
terations in 2.088s)
[t-SNE] Iteration 2200: error = 1.1694200, gradient norm = 0.0000269 (50 i
terations in 2.076s)
[t-SNE] Iteration 2250: error = 1.1691750, gradient norm = 0.0000292 (50 i
terations in 2.061s)
[t-SNE] Iteration 2300: error = 1.1688747, gradient norm = 0.0000267 (50 i
terations in 2.063s)
[t-SNE] Iteration 2350: error = 1.1685508, gradient norm = 0.0000289 (50 i
terations in 2.080s)
[t-SNE] Iteration 2400: error = 1.1683463, gradient norm = 0.0000281 (50 i
terations in 2.088s)
[t-SNE] Iteration 2450: error = 1.1681114, gradient norm = 0.0000281 (50 i
terations in 2.072s)
[t-SNE] Iteration 2500: error = 1.1678818, gradient norm = 0.0000256 (50 i
terations in 2.113s)
[t-SNE] Iteration 2550: error = 1.1677654, gradient norm = 0.0000247 (50 i
terations in 2.064s)
[t-SNE] Iteration 2600: error = 1.1675262, gradient norm = 0.0000262 (50 i
terations in 2.073s)
[t-SNE] Iteration 2650: error = 1.1672890, gradient norm = 0.0000248 (50 i
terations in 2.104s)
[t-SNE] Iteration 2700: error = 1.1670705, gradient norm = 0.0000274 (50 i
terations in 2.079s)
[t-SNE] Iteration 2750: error = 1.1668401, gradient norm = 0.0000255 (50 i
terations in 2.062s)
```

```
[t-SNE] Iteration 2800: error = 1.1666988, gradient norm = 0.0000271 (50 i
terations in 2.090s)
[t-SNE] Iteration 2850: error = 1.1665366, gradient norm = 0.0000252 (50 i
terations in 2.144s)
[t-SNE] Iteration 2900: error = 1.1663481, gradient norm = 0.0000272 (50 i
terations in 2.136s)
[t-SNE] Iteration 2950: error = 1.1661259, gradient norm = 0.0000231 (50 i
terations in 2.081s)
[t-SNE] Iteration 3000: error = 1.1659666, gradient norm = 0.0000238 (50 i
terations in 2.087s)
[t-SNE] Iteration 3050: error = 1.1658012, gradient norm = 0.0000238 (50 i
terations in 2.023s)
[t-SNE] Iteration 3100: error = 1.1656317, gradient norm = 0.0000231 (50 i
terations in 2.016s)
[t-SNE] Iteration 3150: error = 1.1655467, gradient norm = 0.0000223 (50 i
terations in 2.038s)
[t-SNE] Iteration 3200: error = 1.1653621, gradient norm = 0.0000233 (50 i
terations in 2.015s)
[t-SNE] Iteration 3250: error = 1.1652739, gradient norm = 0.0000236 (50 i
terations in 2.035s)
[t-SNE] Iteration 3300: error = 1.1651140, gradient norm = 0.0000216 (50 i
terations in 2.084s)
[t-SNE] Iteration 3350: error = 1.1649662, gradient norm = 0.0000191 (50 i
terations in 2.046s)
[t-SNE] Iteration 3400: error = 1.1648253, gradient norm = 0.0000224 (50 i
terations in 2.014s)
[t-SNE] Iteration 3450: error = 1.1646043, gradient norm = 0.0000242 (50 i
terations in 2.007s)
[t-SNE] Iteration 3500: error = 1.1644939, gradient norm = 0.0000239 (50 i
terations in 2.033s)
[t-SNE] Iteration 3550: error = 1.1643434, gradient norm = 0.0000225 (50 i
terations in 1.991s)
[t-SNE] Iteration 3600: error = 1.1642371, gradient norm = 0.0000246 (50 i
terations in 2.000s)
[t-SNE] Iteration 3650: error = 1.1641399, gradient norm = 0.0000204 (50 i
terations in 2.030s)
[t-SNE] Iteration 3700: error = 1.1640038, gradient norm = 0.0000246 (50 i
terations in 1.999s)
[t-SNE] Iteration 3750: error = 1.1639613, gradient norm = 0.0000200 (50 i
terations in 2.014s)
[t-SNE] Iteration 3800: error = 1.1638137, gradient norm = 0.0000238 (50 i
terations in 2.028s)
[t-SNE] Iteration 3850: error = 1.1637572, gradient norm = 0.0000258 (50 i
terations in 2.008s)
[t-SNE] Iteration 3900: error = 1.1636591, gradient norm = 0.0000200 (50 i
terations in 2.012s)
[t-SNE] Iteration 3950: error = 1.1635907, gradient norm = 0.0000196 (50 i
terations in 2.000s)
[t-SNE] Iteration 4000: error = 1.1634473, gradient norm = 0.0000228 (50 i
terations in 2.039s)
[t-SNE] Iteration 4050: error = 1.1633145, gradient norm = 0.0000200 (50 i
terations in 2.000s)
[t-SNE] Iteration 4100: error = 1.1632321, gradient norm = 0.0000201 (50 i
terations in 2.016s)
[t-SNE] Iteration 4150: error = 1.1631643, gradient norm = 0.0000194 (50 i
terations in 2.056s)
[t-SNE] Iteration 4200: error = 1.1630181, gradient norm = 0.0000213 (50 i
terations in 2.020s)
[t-SNE] Iteration 4250: error = 1.1629357, gradient norm = 0.0000208 (50 i
terations in 2.027s)
[t-SNE] Iteration 4300: error = 1.1628813, gradient norm = 0.0000229 (50 i
```

```
terations in 2.057s)
[t-SNE] Iteration 4350: error = 1.1627896, gradient norm = 0.0000196 (50 i
terations in 2.027s)
[t-SNE] Iteration 4400: error = 1.1627333, gradient norm = 0.0000204 (50 i
terations in 2.007s)
[t-SNE] Iteration 4450: error = 1.1626606, gradient norm = 0.0000190 (50 i
terations in 2.021s)
[t-SNE] Iteration 4500: error = 1.1625243, gradient norm = 0.0000206 (50 i
terations in 2.051s)
[t-SNE] Iteration 4550: error = 1.1625377, gradient norm = 0.0000203 (50 i
terations in 2.038s)
[t-SNE] Iteration 4600: error = 1.1624794, gradient norm = 0.0000216 (50 i
terations in 2.028s)
[t-SNE] Iteration 4650: error = 1.1624082, gradient norm = 0.0000213 (50 i
terations in 2.072s)
[t-SNE] Iteration 4700: error = 1.1623236, gradient norm = 0.0000192 (50 i
terations in 2.059s)
[t-SNE] Iteration 4750: error = 1.1622064, gradient norm = 0.0000173 (50 i
terations in 2.027s)
[t-SNE] Iteration 4800: error = 1.1621385, gradient norm = 0.0000193 (50 i
terations in 2.058s)
[t-SNE] Iteration 4850: error = 1.1620328, gradient norm = 0.0000197 (50 i
terations in 2.024s)
[t-SNE] Iteration 4900: error = 1.1619436, gradient norm = 0.0000220 (50 i
terations in 2.019s)
[t-SNE] Iteration 4950: error = 1.1618651, gradient norm = 0.0000201 (50 i
terations in 2.022s)
[t-SNE] Iteration 5000: error = 1.1618139, gradient norm = 0.0000183 (50 i
terations in 2.051s)
[t-SNE] KL divergence after 5000 iterations: 1.161814
```

In [122]:

```
tsne_tfidf.shape
```

Out[122]:

```
(5000, 2)
```

In [123]:

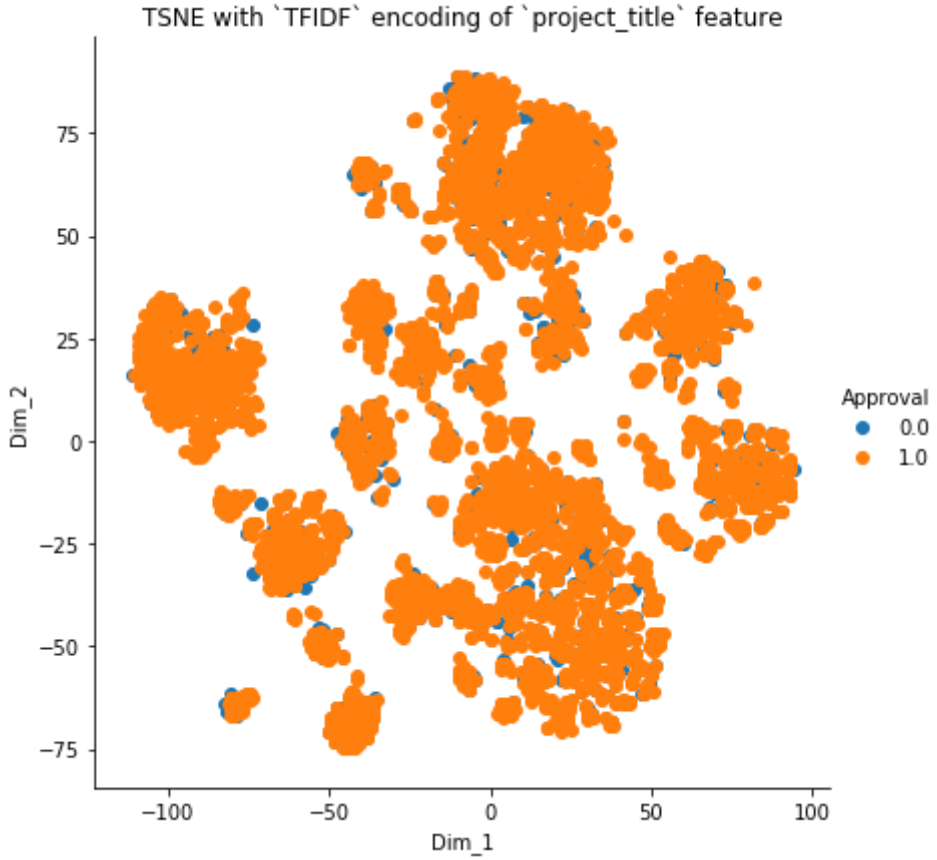
```
tsne_tfidf = np.column_stack((tsne_tfidf,approval_actual))
tsne_tfidf.shape
```

Out[123]:

```
(5000, 3)
```

In [124]:

```
# now creating the dataframe which we will be giving for the visualisation purpose


tsne_df_tfidf = pd.DataFrame(data=tsne_tfidf,columns=("Dim_1", "Dim_2" , "Approval"))
sn.FacetGrid(tsne_df_tfidf,hue ="Approval" , height=6).map(plt.scatter, 'Dim_1', 'Dim_
2').add_legend()
plt.title("TSNE with `TFIDF` encoding of `project_title` feature")
plt.show()
```



Clusters can be seen seperated but there is overlapping of both approved and unapproved projects there hence nothing can be said

TSNE with `AVG W2V` encoding of `project_title` feature

Combining all the other features into one data matrix

In [125]:

```
# 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,school_state_one_hot , project_grade_one_hot, teacher_prefix_one_hot, price_standardized, previous_posted_projects_standardized, avg_w2v_vectors_title))

X.shape
```

Out[125]:

```
(109248, 401)
```

In [126]:

```
# we are compressing the row matrix in order to count duplicate values as only one

X = X.tocsr()
X_actual = X[0:5000,:]
```

In [127]:

```
X_actual.shape
```

Out[127]:

```
(5000, 401)
```

In [128]:

```
X_actual = X_actual.toarray()
model = TSNE(n_components = 2, perplexity = 50,n_iter = 5000, random_state = 0,verbose
= 5)
tsne_avgw2v = model.fit_transform(X_actual)
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.082s...
[t-SNE] Computed neighbors for 5000 samples in 17.328s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 1.189318
[t-SNE] Computed conditional probabilities in 0.245s
[t-SNE] Iteration 50: error = 80.4043655, gradient norm = 0.0265963 (50 it
erations in 4.731s)
[t-SNE] Iteration 100: error = 80.4017944, gradient norm = 0.0170390 (50 i
terations in 4.473s)
[t-SNE] Iteration 150: error = 80.4213943, gradient norm = 0.0444283 (50 i
terations in 4.904s)
[t-SNE] Iteration 200: error = 80.6309662, gradient norm = 0.0194031 (50 i
terations in 5.667s)
[t-SNE] Iteration 250: error = 80.5810089, gradient norm = 0.0131933 (50 i
terations in 6.501s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.581
009
[t-SNE] Iteration 300: error = 2.7544475, gradient norm = 0.0013649 (50 it
erations in 3.704s)
[t-SNE] Iteration 350: error = 2.5280411, gradient norm = 0.0004467 (50 it
erations in 2.174s)
[t-SNE] Iteration 400: error = 2.4397306, gradient norm = 0.0002887 (50 it
erations in 2.186s)
[t-SNE] Iteration 450: error = 2.3938961, gradient norm = 0.0001607 (50 it
erations in 2.192s)
[t-SNE] Iteration 500: error = 2.3657420, gradient norm = 0.0001174 (50 it
erations in 2.214s)
[t-SNE] Iteration 550: error = 2.3473382, gradient norm = 0.0000912 (50 it
erations in 2.227s)
[t-SNE] Iteration 600: error = 2.3345108, gradient norm = 0.0000829 (50 it
erations in 2.238s)
[t-SNE] Iteration 650: error = 2.3255937, gradient norm = 0.0000617 (50 it
erations in 2.274s)
[t-SNE] Iteration 700: error = 2.3190370, gradient norm = 0.0000534 (50 it
erations in 2.298s)
[t-SNE] Iteration 750: error = 2.3140721, gradient norm = 0.0000481 (50 it
erations in 2.283s)
[t-SNE] Iteration 800: error = 2.3104684, gradient norm = 0.0000435 (50 it
erations in 2.291s)
[t-SNE] Iteration 850: error = 2.3076441, gradient norm = 0.0000396 (50 it
erations in 2.289s)
[t-SNE] Iteration 900: error = 2.3052759, gradient norm = 0.0000367 (50 it
erations in 2.288s)
[t-SNE] Iteration 950: error = 2.3033197, gradient norm = 0.0000390 (50 it
erations in 2.286s)
[t-SNE] Iteration 1000: error = 2.3016000, gradient norm = 0.0000369 (50 i
terations in 2.289s)
[t-SNE] Iteration 1050: error = 2.3000896, gradient norm = 0.0000326 (50 i
terations in 2.283s)
[t-SNE] Iteration 1100: error = 2.2987301, gradient norm = 0.0000359 (50 i
terations in 2.303s)
[t-SNE] Iteration 1150: error = 2.2976203, gradient norm = 0.0000358 (50 i
terations in 2.291s)
[t-SNE] Iteration 1200: error = 2.2966726, gradient norm = 0.0000349 (50 i
terations in 2.277s)
[t-SNE] Iteration 1250: error = 2.2959723, gradient norm = 0.0000321 (50 i
```

```
terations in 2.286s)
[t-SNE] Iteration 1300: error = 2.2953272, gradient norm = 0.0000344 (50 i
terations in 2.272s)
[t-SNE] Iteration 1350: error = 2.2947693, gradient norm = 0.0000267 (50 i
terations in 2.300s)
[t-SNE] Iteration 1400: error = 2.2942197, gradient norm = 0.0000275 (50 i
terations in 2.291s)
[t-SNE] Iteration 1450: error = 2.2935364, gradient norm = 0.0000281 (50 i
terations in 2.293s)
[t-SNE] Iteration 1500: error = 2.2930298, gradient norm = 0.0000270 (50 i
terations in 2.249s)
[t-SNE] Iteration 1550: error = 2.2925787, gradient norm = 0.0000359 (50 i
terations in 2.243s)
[t-SNE] Iteration 1600: error = 2.2923491, gradient norm = 0.0000321 (50 i
terations in 2.293s)
[t-SNE] Iteration 1650: error = 2.2920678, gradient norm = 0.0000275 (50 i
terations in 2.258s)
[t-SNE] Iteration 1700: error = 2.2917550, gradient norm = 0.0000345 (50 i
terations in 2.245s)
[t-SNE] Iteration 1750: error = 2.2915840, gradient norm = 0.0000245 (50 i
terations in 2.253s)
[t-SNE] Iteration 1800: error = 2.2912617, gradient norm = 0.0000330 (50 i
terations in 2.320s)
[t-SNE] Iteration 1850: error = 2.2909763, gradient norm = 0.0000296 (50 i
terations in 2.266s)
[t-SNE] Iteration 1900: error = 2.2905169, gradient norm = 0.0000278 (50 i
terations in 2.266s)
[t-SNE] Iteration 1950: error = 2.2900920, gradient norm = 0.0000286 (50 i
terations in 2.278s)
[t-SNE] Iteration 2000: error = 2.2897220, gradient norm = 0.0000258 (50 i
terations in 2.269s)
[t-SNE] Iteration 2050: error = 2.2894528, gradient norm = 0.0000235 (50 i
terations in 2.278s)
[t-SNE] Iteration 2100: error = 2.2892728, gradient norm = 0.0000238 (50 i
terations in 2.279s)
[t-SNE] Iteration 2150: error = 2.2890286, gradient norm = 0.0000220 (50 i
terations in 2.295s)
[t-SNE] Iteration 2200: error = 2.2887452, gradient norm = 0.0000210 (50 i
terations in 2.284s)
[t-SNE] Iteration 2250: error = 2.2885082, gradient norm = 0.0000226 (50 i
terations in 2.276s)
[t-SNE] Iteration 2300: error = 2.2882171, gradient norm = 0.0000249 (50 i
terations in 2.285s)
[t-SNE] Iteration 2350: error = 2.2879696, gradient norm = 0.0000230 (50 i
terations in 2.276s)
[t-SNE] Iteration 2400: error = 2.2878046, gradient norm = 0.0000250 (50 i
terations in 2.267s)
[t-SNE] Iteration 2450: error = 2.2876425, gradient norm = 0.0000270 (50 i
terations in 2.282s)
[t-SNE] Iteration 2500: error = 2.2875898, gradient norm = 0.0000221 (50 i
terations in 2.262s)
[t-SNE] Iteration 2550: error = 2.2873952, gradient norm = 0.0000242 (50 i
terations in 2.269s)
[t-SNE] Iteration 2600: error = 2.2871761, gradient norm = 0.0000185 (50 i
terations in 2.297s)
[t-SNE] Iteration 2650: error = 2.2869344, gradient norm = 0.0000212 (50 i
terations in 2.269s)
[t-SNE] Iteration 2700: error = 2.2867253, gradient norm = 0.0000223 (50 i
terations in 2.275s)
[t-SNE] Iteration 2750: error = 2.2864785, gradient norm = 0.0000228 (50 i
terations in 2.284s)
```

```
[t-SNE] Iteration 2800: error = 2.2862260, gradient norm = 0.0000290 (50 i
terations in 2.272s)
[t-SNE] Iteration 2850: error = 2.2861354, gradient norm = 0.0000233 (50 i
terations in 2.277s)
[t-SNE] Iteration 2900: error = 2.2860785, gradient norm = 0.0000285 (50 i
terations in 2.275s)
[t-SNE] Iteration 2950: error = 2.2859123, gradient norm = 0.0000335 (50 i
terations in 2.284s)
[t-SNE] Iteration 3000: error = 2.2857935, gradient norm = 0.0000261 (50 i
terations in 2.264s)
[t-SNE] Iteration 3050: error = 2.2857592, gradient norm = 0.0000269 (50 i
terations in 2.270s)
[t-SNE] Iteration 3100: error = 2.2856386, gradient norm = 0.0000221 (50 i
terations in 2.265s)
[t-SNE] Iteration 3150: error = 2.2855616, gradient norm = 0.0000171 (50 i
terations in 2.259s)
[t-SNE] Iteration 3200: error = 2.2853873, gradient norm = 0.0000242 (50 i
terations in 2.274s)
[t-SNE] Iteration 3250: error = 2.2852802, gradient norm = 0.0000257 (50 i
terations in 2.261s)
[t-SNE] Iteration 3300: error = 2.2850530, gradient norm = 0.0000265 (50 i
terations in 2.269s)
[t-SNE] Iteration 3350: error = 2.2849188, gradient norm = 0.0000267 (50 i
terations in 2.252s)
[t-SNE] Iteration 3400: error = 2.2847848, gradient norm = 0.0000175 (50 i
terations in 2.248s)
[t-SNE] Iteration 3450: error = 2.2847385, gradient norm = 0.0000210 (50 i
terations in 2.247s)
[t-SNE] Iteration 3500: error = 2.2846503, gradient norm = 0.0000193 (50 i
terations in 2.267s)
[t-SNE] Iteration 3550: error = 2.2845247, gradient norm = 0.0000235 (50 i
terations in 2.248s)
[t-SNE] Iteration 3600: error = 2.2843723, gradient norm = 0.0000176 (50 i
terations in 2.251s)
[t-SNE] Iteration 3650: error = 2.2843344, gradient norm = 0.0000222 (50 i
terations in 2.255s)
[t-SNE] Iteration 3700: error = 2.2842169, gradient norm = 0.0000259 (50 i
terations in 2.253s)
[t-SNE] Iteration 3750: error = 2.2841043, gradient norm = 0.0000237 (50 i
terations in 2.247s)
[t-SNE] Iteration 3800: error = 2.2840867, gradient norm = 0.0000203 (50 i
terations in 2.258s)
[t-SNE] Iteration 3850: error = 2.2839806, gradient norm = 0.0000177 (50 i
terations in 2.233s)
[t-SNE] Iteration 3900: error = 2.2838962, gradient norm = 0.0000231 (50 i
terations in 2.234s)
[t-SNE] Iteration 3950: error = 2.2838476, gradient norm = 0.0000365 (50 i
terations in 2.249s)
[t-SNE] Iteration 4000: error = 2.2838106, gradient norm = 0.0000256 (50 i
terations in 2.259s)
[t-SNE] Iteration 4050: error = 2.2837045, gradient norm = 0.0000235 (50 i
terations in 2.247s)
[t-SNE] Iteration 4100: error = 2.2836256, gradient norm = 0.0000235 (50 i
terations in 2.255s)
[t-SNE] Iteration 4150: error = 2.2835190, gradient norm = 0.0000231 (50 i
terations in 2.239s)
[t-SNE] Iteration 4200: error = 2.2834883, gradient norm = 0.0000199 (50 i
terations in 2.244s)
[t-SNE] Iteration 4250: error = 2.2833598, gradient norm = 0.0000275 (50 i
terations in 2.247s)
[t-SNE] Iteration 4300: error = 2.2834754, gradient norm = 0.0000341 (50 i
```

```
terations in 2.256s)
[t-SNE] Iteration 4350: error = 2.2834597, gradient norm = 0.0000237 (50 i
terations in 2.271s)
[t-SNE] Iteration 4400: error = 2.2834377, gradient norm = 0.0000203 (50 i
terations in 2.278s)
[t-SNE] Iteration 4450: error = 2.2833509, gradient norm = 0.0000237 (50 i
terations in 2.273s)
[t-SNE] Iteration 4500: error = 2.2833097, gradient norm = 0.0000173 (50 i
terations in 2.268s)
[t-SNE] Iteration 4550: error = 2.2831566, gradient norm = 0.0000155 (50 i
terations in 2.266s)
[t-SNE] Iteration 4600: error = 2.2830451, gradient norm = 0.0000164 (50 i
terations in 2.284s)
[t-SNE] Iteration 4650: error = 2.2828336, gradient norm = 0.0000180 (50 i
terations in 2.265s)
[t-SNE] Iteration 4700: error = 2.2827742, gradient norm = 0.0000163 (50 i
terations in 2.268s)
[t-SNE] Iteration 4750: error = 2.2827017, gradient norm = 0.0000156 (50 i
terations in 2.265s)
[t-SNE] Iteration 4800: error = 2.2825472, gradient norm = 0.0000171 (50 i
terations in 2.278s)
[t-SNE] Iteration 4850: error = 2.2824028, gradient norm = 0.0000208 (50 i
terations in 2.264s)
[t-SNE] Iteration 4900: error = 2.2823880, gradient norm = 0.0000166 (50 i
terations in 2.266s)
[t-SNE] Iteration 4950: error = 2.2822635, gradient norm = 0.0000154 (50 i
terations in 2.263s)
[t-SNE] Iteration 5000: error = 2.2821434, gradient norm = 0.0000158 (50 i
terations in 2.260s)
[t-SNE] KL divergence after 5000 iterations: 2.282143
```

In [129]:

```
tsne_avgw2v.shape
```

Out[129]:

```
(5000, 2)
```

In [130]:

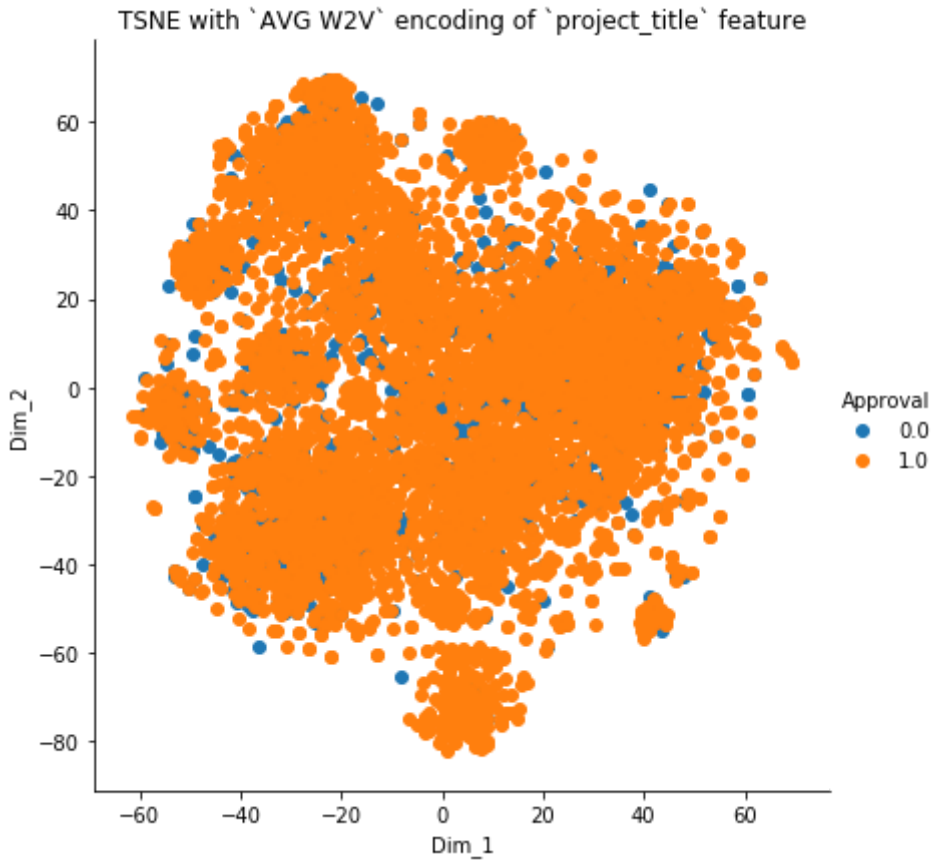
```
tsne_avgw2v = np.column_stack((tsne_avgw2v,approval_actual))
tsne_avgw2v.shape
```

Out[130]:

```
(5000, 3)
```

In [131]:

```
# now creating the dataframe which we will be giving for the visualisation purpose


tsne_df_avgw2v = pd.DataFrame(data=tsne_avgw2v,columns=("Dim_1", "Dim_2" , "Approval"))
sn.FacetGrid(tsne_df_avgw2v,hue ="Approval" , height=6).map(plt.scatter, 'Dim_1', 'Dim_
2').add_legend()
plt.title("TSNE with `AVG W2V` encoding of `project_title` feature")
plt.show()
```



No clusters are observable hence nothing can be said from this data neither

TSNE with `TFIDF Weighted W2V` encoding of `project_title` feature

Combining all the other features into one data matrix

In [132]:

```
# 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,school_state_one_hot , project_grade_one_hot, teacher_prefix_one_hot, price_standardized, previous_posted_projects_standardized, avg_w2v_vectors_title))

X.shape
```

Out[132]:

```
(109248, 401)
```

In [133]:

```
# we are compressing the row matrix in order to count duplicate values as only one

X = X.tocsr()
X_actual = X[0:5000,:]
```

In [134]:

```
X_actual.shape
```

Out[134]:

```
(5000, 401)
```

In [135]:

```
X_actual = X_actual.toarray()
model = TSNE(n_components = 2, perplexity = 50,n_iter = 5000, random_state = 0,verbose
= 5)
tsne_tfidfw2v = model.fit_transform(X_actual)
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.084s...
[t-SNE] Computed neighbors for 5000 samples in 17.205s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 1.189318
[t-SNE] Computed conditional probabilities in 0.241s
[t-SNE] Iteration 50: error = 80.4043655, gradient norm = 0.0265963 (50 it
erations in 4.788s)
[t-SNE] Iteration 100: error = 80.4017944, gradient norm = 0.0170390 (50 i
terations in 4.383s)
[t-SNE] Iteration 150: error = 80.4213943, gradient norm = 0.0444283 (50 i
terations in 5.000s)
[t-SNE] Iteration 200: error = 80.6309662, gradient norm = 0.0194031 (50 i
terations in 5.666s)
[t-SNE] Iteration 250: error = 80.5810089, gradient norm = 0.0131933 (50 i
terations in 6.650s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.581
009
[t-SNE] Iteration 300: error = 2.7544475, gradient norm = 0.0013649 (50 it
erations in 3.705s)
[t-SNE] Iteration 350: error = 2.5280411, gradient norm = 0.0004467 (50 it
erations in 2.271s)
[t-SNE] Iteration 400: error = 2.4397306, gradient norm = 0.0002887 (50 it
erations in 2.175s)
[t-SNE] Iteration 450: error = 2.3938961, gradient norm = 0.0001607 (50 it
erations in 2.197s)
[t-SNE] Iteration 500: error = 2.3657420, gradient norm = 0.0001174 (50 it
erations in 2.225s)
[t-SNE] Iteration 550: error = 2.3473382, gradient norm = 0.0000912 (50 it
erations in 2.244s)
[t-SNE] Iteration 600: error = 2.3345108, gradient norm = 0.0000829 (50 it
erations in 2.240s)
[t-SNE] Iteration 650: error = 2.3255937, gradient norm = 0.0000617 (50 it
erations in 2.257s)
[t-SNE] Iteration 700: error = 2.3190370, gradient norm = 0.0000534 (50 it
erations in 2.254s)
[t-SNE] Iteration 750: error = 2.3140721, gradient norm = 0.0000481 (50 it
erations in 2.251s)
[t-SNE] Iteration 800: error = 2.3104684, gradient norm = 0.0000435 (50 it
erations in 2.257s)
[t-SNE] Iteration 850: error = 2.3076441, gradient norm = 0.0000396 (50 it
erations in 2.261s)
[t-SNE] Iteration 900: error = 2.3052759, gradient norm = 0.0000367 (50 it
erations in 2.284s)
[t-SNE] Iteration 950: error = 2.3033197, gradient norm = 0.0000390 (50 it
erations in 2.267s)
[t-SNE] Iteration 1000: error = 2.3016000, gradient norm = 0.0000369 (50 i
terations in 2.256s)
[t-SNE] Iteration 1050: error = 2.3000896, gradient norm = 0.0000326 (50 i
terations in 2.260s)
[t-SNE] Iteration 1100: error = 2.2987301, gradient norm = 0.0000359 (50 i
terations in 2.271s)
[t-SNE] Iteration 1150: error = 2.2976203, gradient norm = 0.0000358 (50 i
terations in 2.269s)
[t-SNE] Iteration 1200: error = 2.2966726, gradient norm = 0.0000349 (50 i
terations in 2.263s)
[t-SNE] Iteration 1250: error = 2.2959723, gradient norm = 0.0000321 (50 i
```

```
terations in 2.263s)
[t-SNE] Iteration 1300: error = 2.2953272, gradient norm = 0.0000344 (50 i
terations in 2.271s)
[t-SNE] Iteration 1350: error = 2.2947693, gradient norm = 0.0000267 (50 i
terations in 2.320s)
[t-SNE] Iteration 1400: error = 2.2942197, gradient norm = 0.0000275 (50 i
terations in 2.327s)
[t-SNE] Iteration 1450: error = 2.2935364, gradient norm = 0.0000281 (50 i
terations in 2.332s)
[t-SNE] Iteration 1500: error = 2.2930298, gradient norm = 0.0000270 (50 i
terations in 2.409s)
[t-SNE] Iteration 1550: error = 2.2925787, gradient norm = 0.0000359 (50 i
terations in 2.507s)
[t-SNE] Iteration 1600: error = 2.2923491, gradient norm = 0.0000321 (50 i
terations in 2.604s)
[t-SNE] Iteration 1650: error = 2.2920678, gradient norm = 0.0000275 (50 i
terations in 2.317s)
[t-SNE] Iteration 1700: error = 2.2917550, gradient norm = 0.0000345 (50 i
terations in 2.324s)
[t-SNE] Iteration 1750: error = 2.2915840, gradient norm = 0.0000245 (50 i
terations in 2.340s)
[t-SNE] Iteration 1800: error = 2.2912617, gradient norm = 0.0000330 (50 i
terations in 2.289s)
[t-SNE] Iteration 1850: error = 2.2909763, gradient norm = 0.0000296 (50 i
terations in 2.294s)
[t-SNE] Iteration 1900: error = 2.2905169, gradient norm = 0.0000278 (50 i
terations in 2.301s)
[t-SNE] Iteration 1950: error = 2.2900920, gradient norm = 0.0000286 (50 i
terations in 2.912s)
[t-SNE] Iteration 2000: error = 2.2897220, gradient norm = 0.0000258 (50 i
terations in 2.392s)
[t-SNE] Iteration 2050: error = 2.2894528, gradient norm = 0.0000235 (50 i
terations in 2.308s)
[t-SNE] Iteration 2100: error = 2.2892728, gradient norm = 0.0000238 (50 i
terations in 2.305s)
[t-SNE] Iteration 2150: error = 2.2890286, gradient norm = 0.0000220 (50 i
terations in 2.385s)
[t-SNE] Iteration 2200: error = 2.2887452, gradient norm = 0.0000210 (50 i
terations in 2.359s)
[t-SNE] Iteration 2250: error = 2.2885082, gradient norm = 0.0000226 (50 i
terations in 2.357s)
[t-SNE] Iteration 2300: error = 2.2882171, gradient norm = 0.0000249 (50 i
terations in 2.324s)
[t-SNE] Iteration 2350: error = 2.2879696, gradient norm = 0.0000230 (50 i
terations in 2.421s)
[t-SNE] Iteration 2400: error = 2.2878046, gradient norm = 0.0000250 (50 i
terations in 2.348s)
[t-SNE] Iteration 2450: error = 2.2876425, gradient norm = 0.0000270 (50 i
terations in 2.343s)
[t-SNE] Iteration 2500: error = 2.2875898, gradient norm = 0.0000221 (50 i
terations in 2.306s)
[t-SNE] Iteration 2550: error = 2.2873952, gradient norm = 0.0000242 (50 i
terations in 2.331s)
[t-SNE] Iteration 2600: error = 2.2871761, gradient norm = 0.0000185 (50 i
terations in 2.307s)
[t-SNE] Iteration 2650: error = 2.2869344, gradient norm = 0.0000212 (50 i
terations in 2.277s)
[t-SNE] Iteration 2700: error = 2.2867253, gradient norm = 0.0000223 (50 i
terations in 2.295s)
[t-SNE] Iteration 2750: error = 2.2864785, gradient norm = 0.0000228 (50 i
terations in 2.282s)
```

```
[t-SNE] Iteration 2800: error = 2.2862260, gradient norm = 0.0000290 (50 i
terations in 2.301s)
[t-SNE] Iteration 2850: error = 2.2861354, gradient norm = 0.0000233 (50 i
terations in 2.293s)
[t-SNE] Iteration 2900: error = 2.2860785, gradient norm = 0.0000285 (50 i
terations in 2.268s)
[t-SNE] Iteration 2950: error = 2.2859123, gradient norm = 0.0000335 (50 i
terations in 2.346s)
[t-SNE] Iteration 3000: error = 2.2857935, gradient norm = 0.0000261 (50 i
terations in 2.319s)
[t-SNE] Iteration 3050: error = 2.2857592, gradient norm = 0.0000269 (50 i
terations in 2.282s)
[t-SNE] Iteration 3100: error = 2.2856386, gradient norm = 0.0000221 (50 i
terations in 2.294s)
[t-SNE] Iteration 3150: error = 2.2855616, gradient norm = 0.0000171 (50 i
terations in 2.288s)
[t-SNE] Iteration 3200: error = 2.2853873, gradient norm = 0.0000242 (50 i
terations in 2.286s)
[t-SNE] Iteration 3250: error = 2.2852802, gradient norm = 0.0000257 (50 i
terations in 2.336s)
[t-SNE] Iteration 3300: error = 2.2850530, gradient norm = 0.0000265 (50 i
terations in 2.258s)
[t-SNE] Iteration 3350: error = 2.2849188, gradient norm = 0.0000267 (50 i
terations in 2.252s)
[t-SNE] Iteration 3400: error = 2.2847848, gradient norm = 0.0000175 (50 i
terations in 2.277s)
[t-SNE] Iteration 3450: error = 2.2847385, gradient norm = 0.0000210 (50 i
terations in 2.355s)
[t-SNE] Iteration 3500: error = 2.2846503, gradient norm = 0.0000193 (50 i
terations in 2.606s)
[t-SNE] Iteration 3550: error = 2.2845247, gradient norm = 0.0000235 (50 i
terations in 2.461s)
[t-SNE] Iteration 3600: error = 2.2843723, gradient norm = 0.0000176 (50 i
terations in 2.394s)
[t-SNE] Iteration 3650: error = 2.2843344, gradient norm = 0.0000222 (50 i
terations in 2.372s)
[t-SNE] Iteration 3700: error = 2.2842169, gradient norm = 0.0000259 (50 i
terations in 2.374s)
[t-SNE] Iteration 3750: error = 2.2841043, gradient norm = 0.0000237 (50 i
terations in 2.469s)
[t-SNE] Iteration 3800: error = 2.2840867, gradient norm = 0.0000203 (50 i
terations in 2.377s)
[t-SNE] Iteration 3850: error = 2.2839806, gradient norm = 0.0000177 (50 i
terations in 2.359s)
[t-SNE] Iteration 3900: error = 2.2838962, gradient norm = 0.0000231 (50 i
terations in 2.361s)
[t-SNE] Iteration 3950: error = 2.2838476, gradient norm = 0.0000365 (50 i
terations in 2.369s)
[t-SNE] Iteration 4000: error = 2.2838106, gradient norm = 0.0000256 (50 i
terations in 2.379s)
[t-SNE] Iteration 4050: error = 2.2837045, gradient norm = 0.0000235 (50 i
terations in 2.360s)
[t-SNE] Iteration 4100: error = 2.2836256, gradient norm = 0.0000235 (50 i
terations in 2.407s)
[t-SNE] Iteration 4150: error = 2.2835190, gradient norm = 0.0000231 (50 i
terations in 2.358s)
[t-SNE] Iteration 4200: error = 2.2834883, gradient norm = 0.0000199 (50 i
terations in 2.342s)
[t-SNE] Iteration 4250: error = 2.2833598, gradient norm = 0.0000275 (50 i
terations in 2.354s)
[t-SNE] Iteration 4300: error = 2.2834754, gradient norm = 0.0000341 (50 i
```

```
terations in 2.362s)
[t-SNE] Iteration 4350: error = 2.2834597, gradient norm = 0.0000237 (50 i
terations in 2.397s)
[t-SNE] Iteration 4400: error = 2.2834377, gradient norm = 0.0000203 (50 i
terations in 2.377s)
[t-SNE] Iteration 4450: error = 2.2833509, gradient norm = 0.0000237 (50 i
terations in 2.395s)
[t-SNE] Iteration 4500: error = 2.2833097, gradient norm = 0.0000173 (50 i
terations in 2.367s)
[t-SNE] Iteration 4550: error = 2.2831566, gradient norm = 0.0000155 (50 i
terations in 2.386s)
[t-SNE] Iteration 4600: error = 2.2830451, gradient norm = 0.0000164 (50 i
terations in 2.379s)
[t-SNE] Iteration 4650: error = 2.2828336, gradient norm = 0.0000180 (50 i
terations in 2.386s)
[t-SNE] Iteration 4700: error = 2.2827742, gradient norm = 0.0000163 (50 i
terations in 2.375s)
[t-SNE] Iteration 4750: error = 2.2827017, gradient norm = 0.0000156 (50 i
terations in 2.369s)
[t-SNE] Iteration 4800: error = 2.2825472, gradient norm = 0.0000171 (50 i
terations in 2.370s)
[t-SNE] Iteration 4850: error = 2.2824028, gradient norm = 0.0000208 (50 i
terations in 2.368s)
[t-SNE] Iteration 4900: error = 2.2823880, gradient norm = 0.0000166 (50 i
terations in 2.367s)
[t-SNE] Iteration 4950: error = 2.2822635, gradient norm = 0.0000154 (50 i
terations in 2.375s)
[t-SNE] Iteration 5000: error = 2.2821434, gradient norm = 0.0000158 (50 i
terations in 2.362s)
[t-SNE] KL divergence after 5000 iterations: 2.282143
```

In [136]:

```
tsne_tfidfw2v.shape
```

Out[136]:

```
(5000, 2)
```

In [137]:

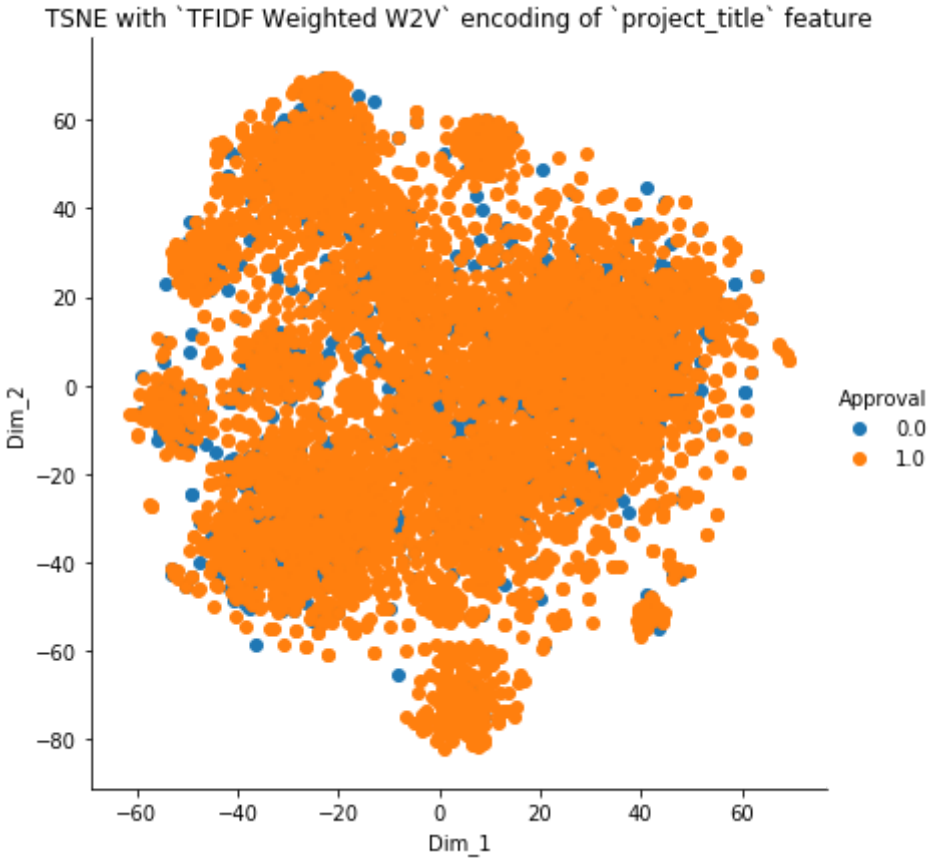
```
tsne_tfidfw2v = np.column_stack((tsne_tfidfw2v,approval_actual))
tsne_tfidfw2v.shape
```

Out[137]:

```
(5000, 3)
```

In [138]:

```
# now creating the dataframe which we will be giving for the visualisation purpose


tsne_df_tfidfw2v = pd.DataFrame(data=tsne_tfidfw2v,columns=("Dim_1", "Dim_2" , "Approva
l"))
sn.FacetGrid(tsne_df_tfidfw2v,hue ="Approval" , height=6).map(plt.scatter, 'Dim_1', 'Di
m_2').add_legend()
plt.title("TSNE with `TFIDF Weighted W2V` encoding of `project_title` feature")
plt.show()
```



Very much overlapping hence unable to form clusters for similar type of data here hence nothing useful can be achieved

In [ ]:

In [ ]: