DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of donations. Page rules include a strict word limit, and teachers submit projects to be manually scored each submission before it is approved to be posted on the DonorsChoose.org website.

This data set includes every project submitted to DonorsChoose.org in the 2009-2014 school years, or in other words, there are more than one million projects in this data set.

- Kick to start: Kaggle kernel processes and resources to convert 500-600 projects on the fly into text vectors and save them as pickle files
- Kick to start: Kaggle kernel processes and resources to engineer coding and other additional features for the modeling or machine learning
- Kick to finish: Kaggle kernel processes and resources to run machine learning models and make predictions

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 the teacher's project proposal as well as other features of the project, the teacher, and school. DonorsChoose.org uses the results from this competition to improve their ability to match teachers with donors who are most likely to fund their projects.

About the DonorsChoose Data Set

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

Feature	Description
`teacher_acctid`	A unique identifier for the proposal's teacher. **Example:** `c90749f5d961ff974ada2ff5a7ba6bd7`
`schoolid`	Yes it is a school. **Example:** `53690f1b5d0bc5d2a553d4a649f60673`
`school_ncesid`	This is the NCES ID for a school. **Example:** `000015000872`

`school_latitude`
`school_longitude`
	Good news! A student's for which the project is targeted. Data of the following anonymized:
	`school_state`
	`school_metro`
	`school_district`
	`school_county`
	`school_charter`
	`school_magnet`
	`school_nlns`
	`school_kipp`
	`school_charter_num`
	`school_mail_to`
	`school_agreeing`
	`school_for_profit`

`school_zip`	Zip or more precise approximate school location. **Example:** `92126`
`school_city`	Data where school is located (Wikipedia's city page). **Example:** `San Diego`
`school_state`	Data where school is located (Wikipedia's state page). **Example:** `CA`
`school_metro_type`	Yes, it contains: rural, suburban, urban, town. **Example:** `suburban`
`school_percentage_free_lunch`	Yes, it contains: lower, lowest, mid, missing. **Example:** `lowest`
`school_nces_subtype`	Yes, it contains: Agam, Elem, Elem-Sec, Hi, Kindergarten, Mid, Mid-Hi, Other, Preschool, Sec. **Example:** `Elem`
`school_nces_status`	Yes, it contains: closed, inactive, open, pending, prospective. **Example:** `closed`

`school_year_round`
`school_ninth_grade`
`school_eighth_grade`
`school_seventh_grade`
`school_sixth_grade`
`school_grade_prek`
`school_kindergarten`
`school_first_grade`
`school_second_grade`
`school_third_grade`
`school_fourth_grade`
`school_fifth_grade`
	Yes, it contains: year round, ninth grade, eighth grade, seventh grade, sixth grade, grade prek, kindergarten, first grade, second grade, third grade, fourth grade, fifth grade. **Example:** `ninth grade`

`school_university_prep`
`school_high_school`
`school_middle_school`
`school_k8`
`school_elementary`
`school_primary`
`school_jr_high`
`school_all_girls`
`school_all_boys`
`school_c_scores`
`school_mid_scores`
`school_hi_scores`
`school_sped_scores`
	Yes, it contains: high school, k8, middle school, university prep. **Example:** `high school`
	Yes, it contains: elementary, primary. **Example:** `elementary`
	Yes, it contains: all girls, all boys, jr high. **Example:** `all girls`
	Yes, it contains: c scores, mid scores, hi scores, sped scores. **Example:** `c scores`

`teacher_prefix`	Teacher's prefix (Mr., Ms., etc.). **Example:** `Ms.`
`teacher_number_of_previously_posted_projects`	Number of school applicants previously submitted by the same teacher. **Example:** `0`

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

Additionally, the `train.csv` data set provides more data about the resources required for each project. Each line in the file represents a resource, such that there may be more than one resource per project.

Feature	Description
`id`	A project id. **Example:** `p036502`
`description`	Desc of what project plans to use the resources for. **Example:** `Nearpod subscription 36mo, 36 units`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	The price of the resource required. **Example:** `$297.00`

Note: Some projects require multiple resources. The `id` value corresponds to a `project_id` in `train.csv`, so you can join the two files together to find out more about the requested resources.

The files ending in `.zip` contain the same data, but with fewer decimal places.

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

Notes on the Essay Data

- Prior to May 17, 2016, the prompts for the essays were as follows:
  - `__ preparation` asked for "Project Proposal Details"
  - `__ project_title` asked for "Project Title"
  - `__ project_essay_1` asked for "Project Essay" (or "Project Proposal")
  - `__ project_essay_2` asked for "Project Essay"
  - `__ project_essay_3` asked for "Project Essay"
  - `__ project_essay_4` asked for "Project Essay"
  - `__ project_resource_summary` asked for "Resource Summary"
- On May 17, 2016, the wording of essays was changed to ask:
  - `__ project_title` asked for "Project Title"
  - `__ project_essay_1` asked for "Describe your students: What makes your students special? Specific details about their background, interests, and aspirations (optional)"
  - `__ project_essay_2` asked for "How will use these resources to help achieve this project and your students' learning goals?"
  - `__ project_essay_3` asked for "Describe how your students will use the materials you're asking for."
  - `__ project_essay_4` asked for "Describe how your students will use the materials you're asking for."
  - `__ project_resource_summary` asked for "Resource Summary"

For all projects with post_submission_date of 2016-05-17 and later, the values of `project_essay_1` and `project_essay_2` will be blank.

```
In [1]: %matplotlib inline
        %load_ext autoreload
        %autoreload 2
        import numpy as np
        import pandas as pd
        import sqlite3
        import pickle as pickl
        import gensim
        import gensim.models.word2vec as w2v
        import gensim.models.phrases
        import gensim.models.phrases.scoring_npmi_sanitized
        import gensim.models.phrases.scoring_npmi
        from urlparse import urlparse
        from urlparse.urlparse import SplitResult
        from gensim.models.phrases import Phrases, Phraser
        from gensim.models.word2vec import Word2Vec
        from gensim.models.word2vec.corpuslinkedsentences import CorpusLinkedSentences
        from os.path import isfile, isdir, getsize
        import os, sys, re, gc, zipfile, tarfile, shutil, fnmatch
        import matplotlib, matplotlib.pyplot as plt
        import seaborn as sns
        import pandas as pd, numpy as np
        from IPYTHON import get_ipython
        from IPYTHON import load_ext
        get_ipython().load_ext('autoreload')
        get_ipython().run_line_magic('autoreload', '2')
        get_ipython().run_line_magic('matplotlib', 'inline')
        from IPYTHON import display
        from IPYTHON.display import display
        from IPYTHON import IPYTHON
        from IPYTHON import get_ipython
        from IPYTHON.core.interactiveshell import InteractiveShell
        from IPYTHON.core.interactiveshell.ExecutionResult import ExecutionResult
        from IPYTHON.utils import io
        from IPYTHON.utils import rootdirs, rmfile, mkdirs
        from IPYTHON.utils import path, pycolorize, text, tokenize2, py3compat
        from IPYTHON.utils import encoding, strdispatch, localipyenv, ipunitest
        from IPYTHON.utils import ipstruct, tempdir, chainmap, collections
        from IPYTHON.utils import coloratorium, warnings_and_exceptions, PyColorize
        from IPYTHON.utils import generics, group_by_kind, frozenjson, jsonutil
        from IPYTHON.utils import sysinfo, frame, dtemplate, decorators, data
        from IPYTHON.utils import dir2, globalipapi, strdispatch, rlineimpl, text
        from IPYTHON.utils import PickleShareDB, process, py3compat, path, terminal
        from IPYTHON import IPYTHON, get_ipython, embed, shell, start_ipythonx
```

1.1 Reading Data

```
In [2]: sql_conn = sqlite3.connect('./input/sqlite.db')
        print("Connection established!")
        print("Type conn.close() to close this session.")
        print("Type %lsmagic for a list of available commands.")
        print("Number of data points in train data: {:num:,}".\
              format(pd.read_sql('select count(*) from train', sql_conn).iloc[0,0]))
        print("Number of data points in resource data: {:num:,}".\
              format(pd.read_sql('select count(*) from resources', sql_conn).iloc[0,0]))
        print("Type conn.execute('SQL query string') to make a new query.")
        print("Type df = pd.read_sql('SQL query string', conn) to load the query into a data frame.")
        print("Type df.head() or df.describe() to get a quick look at the data.")
        print("Type df.columns to see a list of available features.")

        Connection established!
        Type conn.close() to close this session.
        Type %lsmagic for a list of available commands.
        Number of data points in train data: 1,110,017
        Number of data points in resource data: 1,541,272
        Type conn.execute('SQL query string') to make a new query.
        Type df = pd.read_sql('SQL query string', conn) to load the query into a data frame.
        Type df.head() or df.describe() to get a quick look at the data.
        Type df.columns to see a list of available features.
```

1.2 Data Analysis

```
In [3]: pd.read_sql('select * from sqlite_master where type="table"', sql_conn)
        # Source: https://stackoverflow.com/questions/1861351/how-to-list-all-tables-in-an-sqlite-database
        pd.read_sql('select * from sqlite_master where type="table"', sql_conn).\
            apply(lambda s: s[s.tbl_name == s.name].\
                  dropna(axis=1).\
                  ftfy.fixtext(ignore_errors=True).\
                  str.replace('\n', ' ').\
                  str.replace('\t', ' ').\
                  str.strip(), axis=1).\
            ftfy.fixtext(ignore_errors=True).\
            to_csv('./output/sqlite_master.tsv', index=False, encoding='utf-8')
        !head -n 2 ./output/sqlite_master.tsv
        !tail -n 2 ./output/sqlite_master.tsv
        !wc -l ./output/sqlite_master.tsv

        wc: stdin: read: Bad file descriptor
        wc: stdin: read: Bad file descriptor
        wc: stdin: read: Bad file descriptor
```

```
In [20]: [Image Description: number of work in writing in Datafiner: https://kaggle.com/code/ryan0967/0000-0000-0000-0000/edit]
[
```

```
In [20]: project_data[project_grade_category].value_counts()
```

```
Out[20]: grades_prek_2    44670
         grades_3_5       37137
         grades_6_8       16923
         grades_9_12      10963
         Name: project_grade_category, dtype: int64
```

```
In [21]: # replacing spaces with _ character
         project_data['project_grade_category'] = project_data['project_grade_category'].str.replace(' ', '_')
         project_data['project_grade_category'] = project_data['project_grade_category'].str.replace('-', '_')
         print("After replacing spaces with _ character and replacing - with _ character")
         project_data['project_grade_category'].value_counts()
```

```
After replacing spaces with _ character and replacing - with _ character
```

```
Out[21]: grades_prek_2    44670
         grades_3_5       37137
         grades_6_8       16923
         grades_9_12      10963
         Name: project_grade_category, dtype: int64
```

Preprocessing categorical features : teacher_prefix

```
In [22]: project_data[teacher_prefix].value_counts()
```

```
Out[22]: Mrs.    57269
         Ms.     38955
         Mr.     10648
         Teacher  2360
         Dr.        13
         Name: teacher_prefix, dtype: int64
```

```
In [23]: print("After replacing nan with Teacher")
         project_data[teacher_prefix] = project_data[teacher_prefix].fillna('Teacher')
         print("After replacing nan with Teacher and replacing . with _ character")
         project_data[teacher_prefix] = project_data[teacher_prefix].str.replace('.', '')
         project_data[teacher_prefix].value_counts()
```

```
After replacing nan with Teacher
After replacing nan with Teacher and replacing . with _ character
```

```
Out[23]: Mrs    57269
         Ms     38955
         Mr     10648
         Teacher  2360
         Dr        13
         Name: teacher_prefix, dtype: int64
```

```
In [24]: project_data[teacher_prefix].isnull().sum()
```

```
Out[24]: 0
```

Preprocessing categorical features : school_state

```
In [25]: project_data[school_state].value_counts()
```

```
Out[25]: CA    15388
         NY     8828
         TX     7396
         FL     6924
         NC     6091
         IL     5091
         MD     4350
         GA     4026
         IN     3918
         MI     3807
         AZ     3372
         PA     3355
         WA     3350
         OH     3078
         MA     2733
         NJ     2620
         VA     2477
         WI     2405
         CO     2390
         SC     2309
         CT     2267
         MN     2192
         MO     2082
         TN     1871
         LA     1746
         OR     1742
         AL     1396
         KY     1390
         OK     1317
         NV     1011
         AR     1000
         UT      915
         KS      894
         HI      876
         IA      802
         DC      767
         ME      704
         NM      697
         ID      608
         NE      561
         MS      541
         WV      536
         DE      523
         AK      510
         NH      494
         RI      285
         MT      245
         SD      226
         ND      107
         VT      105
         WY      098
         Name: school_state, dtype: int64
```

1.4 Preparing data for models

```
In [26]: project_data[project_essay_1].value_counts()
```

```
Out[26]: ...
```

```
         ...
         Name: project_essay_1, dtype: int64
```

```
In [27]: project_data[project_essay_2].value_counts()
```

```
Out[27]: ...
```

```
         ...
         Name: project_essay_2, dtype: int64
```

```
In [28]: project_data[project_essay_3].value_counts()
```

```
Out[28]: ...
```

```
         ...
         Name: project_essay_3, dtype: int64
```

```
In [29]: project_data[project_essay_4].value_counts()
```

```
Out[29]: ...
```

```
         ...
         Name: project_essay_4, dtype: int64
```

```
In [30]: project_data[project_resource_summary].value_counts()
```

```
Out[30]: ...
```

```
         ...
         Name: project_resource_summary, dtype: int64
```

```
In [31]: project_data[project_title].value_counts()
```

```
Out[31]: ...
```

```
         ...
         Name: project_title, dtype: int64
```

```
In [32]: project_data[teacher_prefix].value_counts()
```

```
Out[32]: ...
```

```
         ...
         Name: teacher_prefix, dtype: int64
```

```
In [33]: project_data[school_state].value_counts()
```

```
Out[33]: ...
```

```
         ...
         Name: school_state, dtype: int64
```

1.4.1 Preparing data for models

```
In [34]: project_data[project_grade_category].value_counts()
```

```
Out[34]: ...
```

```
         ...
         Name: project_grade_category, dtype: int64
```

```
In [35]: project_data[project_subject_subcategories].value_counts()
```

```
Out[35]: ...
```

```
         ...
         Name: project_subject_subcategories, dtype: int64
```

```
In [36]: project_data[project_subject_subcategories].value_counts()
```

```
Out[36]: ...
```

```
         ...
         Name: project_subject_subcategories, dtype: int64
```

```
In [37]: project_data[project_title].value_counts()
```

```
Out[37]: ...
```

```
         ...
         Name: project_title, dtype: int64
```

```
In [38]: project_data[teacher_prefix].value_counts()
```

```
Out[38]: ...
```

```
         ...
         Name: teacher_prefix, dtype: int64
```

```
In [39]: project_data[school_state].value_counts()
```

```
Out[39]: ...
```

```
         ...
         Name: school_state, dtype: int64
```

1.4.2 Preparing data for models

- https://www.ordinalencoder.com/ordinalencoder-example-scikitlearn-pandas-and-keras-tensorflow

```
In [40]: # We use count vectorizer to convert the values into one hot encoded features
         vocabulary_1, idf_1 = CountVectorizer(lowercase=False, binary=True,).fit(project_data['clean_categories'].values).get_vocabulary(), CountVectorizer(lowercase=False, binary=True,).fit(project_data['clean_categories'].values).idf_
         print("Our vocabulary", vocabulary_1)
         print("Our idf", dict(zip(vocabulary_1, idf_1)))
         categories_one_hot = CountVectorizer(vocabulary=vocabulary_1, lowercase=False, binary=True,).fit_transform(project_data['clean_categories'].values)
         print("Shape of matrix after one hot encoding ",categories_one_hot.shape)
```

```
Our vocabulary ['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Our idf {'Warmth': 1.2712477396654737, 'Care_Hunger': 1.7773259718730718, 'History_Civics': 1.5484185314619035, 'Music_Arts': 1.8826430493618411, 'AppliedLearning': 1.6959489051780165, 'SpecialNeeds': 1.6776672860419791, 'Health_Sports': 1.4843112752430321, 'Math_Science': 1.2548561964725925, 'Literacy_Language': 1.0296822953784301}
Shape of matrix after one hot encoding (109248, 9)
```

```
In [41]: # We use count vectorizer to convert the values into one hot encoded features
         vocabulary_2, idf_2 = CountVectorizer(lowercase=False, binary=True,).fit(project_data['clean_subcategories'].values).get_vocabulary(), CountVectorizer(lowercase=False, binary=True,).fit(project_data['clean_subcategories'].values).idf_
         print("Our vocabulary", vocabulary_2)
         print("Our idf", dict(zip(vocabulary_2, idf_2)))
         sub_categories_one_hot = CountVectorizer(vocabulary=vocabulary_2, lowercase=False, binary=True,).fit_transform(project_data['clean_subcategories'].values)
         print("Shape of matrix after one hot encoding ",sub_categories_one_hot.shape)
```

```
Our vocabulary ['Warm', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Our idf {'Warm': 1.7773259718730718, 'Care_Hunger': 1.7773259718730718, 'History_Civics': 1.5484185314619035, 'Music_Arts': 1.8826430493618411, 'AppliedLearning': 1.6959489051780165, 'SpecialNeeds': 1.6776672860419791, 'Health_Sports': 1.4843112752430321, 'Math_Science': 1.2548561964725925, 'Literacy_Language': 1.0296822953784301}
Shape of matrix after one hot encoding (109248, 9)
```

1.4.3 Vectorizing Categorical data

- https://www.ordinalencoder.com/ordinalencoder-example-scikitlearn-pandas-and-keras-tensorflow

```
In [42]: # We use count vectorizer to convert the values into one hot encoded features
         vocabulary_3, idf_3 = CountVectorizer(lowercase=False, binary=True,).fit(project_data['teacher_prefix'].values).get_vocabulary(), CountVectorizer(lowercase=False, binary=True,).fit(project_data['teacher_prefix'].values).idf_
         print("Our vocabulary", vocabulary_3)
         print("Our idf", dict(zip(vocabulary_3, idf_3)))
         teacher_prefix_one_hot = CountVectorizer(vocabulary=vocabulary_3, lowercase=False, binary=True,).fit_transform(project_data['teacher_prefix'].values)
         print("Shape of matrix after one hot encoding ",teacher_prefix_one_hot.shape)
```

```
Our vocabulary ['Teacher', 'Mrs', 'Ms', 'Mr', 'Dr']
Our idf {'Teacher': 1.4843112752430321, 'Mrs': 1.0296822953784301, 'Ms': 1.2548561964725925, 'Mr': 1.8826430493618411, 'Dr': 9.210340371976184}
Shape of matrix after one hot encoding (109248, 5)
```

```
In [43]: # We use count vectorizer to convert the values into one hot encoded features
         vocabulary_4, idf_4 = CountVectorizer(lowercase=False, binary=True,).fit(project_data['school_state'].values).get_vocabulary(), CountVectorizer(lowercase=False, binary=True,).fit(project_data['school_state'].values).idf_
         print("Our vocabulary", vocabulary_4)
         print("Our idf", dict(zip(vocabulary_4, idf_4)))
         school_state_one_hot = CountVectorizer(vocabulary=vocabulary_4, lowercase=False, binary=True,).fit_transform(project_data['school_state'].values)
         print("Shape of matrix after one hot encoding ",school_state_one_hot.shape)
```

```
Our vocabulary ['CA', 'NY', 'TX', 'FL', 'NC', 'IL', 'MD', 'GA', 'IN', 'MI', 'AZ', 'PA', 'WA', 'OH', 'MA', 'NJ', 'VA', 'WI', 'CO', 'SC', 'CT', 'MN', 'MO', 'TN', 'LA', 'OR', 'AL', 'KY', 'OK', 'NV', 'AR', 'UT', 'KS', 'HI', 'IA', 'DC', 'ME', 'NM', 'ID', 'NE', 'MS', 'WV', 'DE', 'AK', 'NH', 'RI', 'MT', 'SD', 'ND', 'VT', 'WY']
Our idf {'CA': 0.7000400448683771, 'NY': 0.9432989322643689, 'TX': 0.9748074602899414, 'FL': 1.0000250156164392, 'NC': 1.0186621731966702, 'IL': 1.0371517027863777, 'MD': 1.1517323674474095, 'GA': 1.1577788392388453, 'IN': 1.1592260765241603, 'MI': 1.1641532182193779, 'AZ': 1.1831867990657817, 'PA': 1.1846914308175989, 'WA': 1.1851850111501665, 'OH': 1.1965784284662087, 'MA': 1.2190476190476191, 'NJ': 1.2258992919143966, 'VA': 1.2376726703600951, 'WI': 1.2441372941176471, 'CO': 1.2458158277047924, 'SC': 1.2481889070467241, 'CT': 1.2496163682864450, 'MN': 1.2513035381750465, 'MO': 1.2584801813392070, 'TN': 1.2672025723472668, 'LA': 1.2771490750816104, 'OR': 1.2779453850315968, 'AL': 1.2939866368653421, 'KY': 1.2945571067411923, 'OK': 1.2977871437821609, 'NV': 1.3337726138865305, 'AR': 1.3362068965517241, 'UT': 1.3571835205992509, 'KS': 1.3604953559958289, 'HI': 1.3616557734204793, 'IA': 1.3773632930132388, 'DC': 1.3896457765667574, 'ME': 1.3926162790697674, 'NM': 1.3940774487471526, 'ID': 1.4073881373569198, 'NE': 1.4258760107816711, 'MS': 1.4261992619926199, 'WV': 1.4277736318407960, 'DE': 1.4309021113243761, 'AK': 1.4331210191082802, 'NH': 1.4452854914196567, 'RI': 1.5375618375558867, 'MT': 1.5518394648829431, 'SD': 1.5677966101694915, 'ND': 1.6985888992574263, 'VT': 1.7000796812749003, 'WY': 1.7000796812749003}
Shape of matrix after one hot encoding (109248, 51)
```

1.4.4 Vectorizing Text data

1.4.5.1 Bag of words

```
In [44]: # We use count vectorizer to convert the values into one hot encoded features
         vocabulary_5, idf_5 = CountVectorizer(lowercase=False, min_df=10,).fit(project_data['clean_essays'].values).get_vocabulary(), CountVectorizer(lowercase=False, min_df=10,).fit(project_data['clean_essays'].values).idf_
         print("Our vocabulary", len(vocabulary_5))
         print("Our idf", dict(zip(vocabulary_5, idf_5)))
         essay_bow = CountVectorizer(lowercase=False, min_df=10,).fit_transform(project_data['clean_essays'].values)
         print("Shape of matrix after one hot encoding ",essay_bow.shape)
```

```
Our vocabulary 16623
Our idf {'zoo': 7.9248174665860235, 'zones': 5.7011607687449765, 'zones.': 8.217674125765325, 'zoom': 4.7095302013123339, 'zoo.': 8.217674125765325, 'zoo.com': 8.217674125765325, 'zoo.com/animal-adoption': 8.217674125765325, 'zoo.com/animal-adoption-and-care': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/cages': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/cleaning-supplies': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/food': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/climbing-playstands': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/climbing-playstands/ladders': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings/small-bird-swings-with-bells': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings/small-bird-swings-with-bells/swings-with-bells': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings/small-bird-swings-with-bells/swings-with-bells/swings-with-bells-1': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings/small-bird-swings-with-bells/swings-with-bells/swings-with-bells-1/swings-with-bells-1': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings/small-bird-swings-with-bells/swings-with-bells/swings-with-bells-1/swings-with-bells-1-1': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings/small-bird-swings-with-bells/swings-with-bells/swings-with-bells-1/swings-with-bells-1-1/swings-with-bells-1-1': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings/small-bird-swings-with-bells/swings-with-bells/swings-with-bells-1/swings-with-bells-1-1/swings-with-bells-1-1/swings-with-bells-1': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings/small-bird-swings-with-bells/swings-with-bells/swings-with-bells-1/swings-with-bells-1-1/swings-with-bells-1-1/swings-with-bells-1-1': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings/small-bird-swings-with-bells/swings-with-bells/swings-with-bells-1/swings-with-bells-1-1/swings-with-bells-1-1/swings-with-bells-1-1/swings-with-bells-1': 8.217674125765325, 'zoo.com/animal-adoption-and-care/bird-care-supplies/toys/hanging-toys/swings/bird-toys/swings/small-bird-swings/small-bird-swings-with-bells/swings-with-bells/swings-with-bells-1/swings-with-bells-1-1/swings-with-bells-1-1/swings-with-bells-1-1/swings-with-bells-1
```

In [100]: # please write all the code with proper documentation and proper titles for each subsection
when you plot any graph make sure you use
a. proper title with font size 25
b. proper labels of axis with font size 15
c. legend if needed
d. proper line color
e. grid of a plot
f. grid line width

```
import TfIdfVectorizer as TV
from sklearn.model_selection import TSNE
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

Text_title_1 = preprocessed_titles[0:5000]
Text_title_2 = preprocessed_titles[5000:10000]

print(len(Text_title_1))
print(len(Text_title_2))

# Taking only top 5000 words
vectorizer = TfIdfVectorizer(ngram_range=(1,1))
vectorizer.fit(Text_title_1 + Text_title_2)

print("the shape of out text TfIdf vocabulary after top 5000 words",vectorizer.shape)

feature = vectorizer.get_feature_names()
print("some of the features",feature[0:10])
print("some of the features",feature[5000:5010])
```

```
5000
5000
the shape of out text TfIdf vocabulary after top 5000 words (5000,)
some of the features ['aaron', 'abandoned', 'abc', 'ability', 'able', 'abortion', 'about', 'above', 'abraham', 'abroad']
some of the features ['ebola', 'ecology', 'economics', 'edge', 'education', 'effect', 'effects', 'efficacy', 'efficiency', 'egg']
```

```
# Preparing data frame for top 5000 words
df_title = pd.DataFrame(vectorizer.transform(Text_title_1 + Text_title_2).toarray(), columns=feature)
df_title.shape
```

```
(10000, 5000)
```

```
# Adding a column with the text length of each sentence in the data preprocessed
df_title['text_length'] = df_title[df_title.columns].sum(axis=1)
df_title.shape
```

```
(10000, 5001)
```

```
# Sorting data according to text length in descending order
df_title = df_title.sort_values('text_length', ascending=False)
df_title
```

	aaron	abandoned	abc	ability	able	abortion	about	above	abraham	abroad	...	zeitgeist	zero	zika	zimbabwe	zinc	zinedine	zionism	zoo	zoos	text_length
3000	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9
4000	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9
1000	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9
2000	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9
7000	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9000	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9001	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9002	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9003	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

10000 rows × 5001 columns

2.2 TSNE with 'TFIDF' encoding of 'project_title' feature

In [101]: # please write all the code with proper documentation, and proper titles for each subsection
when you plot any graph make sure you use
a. proper title with font size 25
b. proper labels of axis with font size 15
c. legend if needed
d. proper line color
e. grid of a plot
f. grid line width

```
import TSNE as TS
from sklearn.model_selection import TSNE
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.cluster as cl
```

```
Text_title_df = df_title[df_title.columns]

print(Text_title_df.shape)
```

```
(10000, 5001)
```

```
# Taking 5000 data points
Text_title_df = Text_title_df.sample(5000)
Text_title_df.shape
```

```
(5000, 5001)
```

```
# Standardizing the data
from sklearn.preprocessing import StandardScaler
Text_title_df = StandardScaler().fit_transform(Text_title_df)
Text_title_df = Text_title_df - np.abs(Text_title_df.min())
Text_title_df.shape
```

```
(5000, 5001)
```

```
# Applying TSNE
model = TSNE(n_components=2, perplexity=30, learning_rate=200)
tsne_data = model.fit_transform(Text_title_df)

# Creating a new data frame which contain the cordinates of each point along with their labels
tsne_data = np.vstack((tsne_data.T, df_title['text_length'].sample(5000).values.T,df_title['text_length'].sample(5000).values.T,df_title['text_length'].sample(5000).values.T,
```