```
NAME = "Mary Guo"
```

Lab 7: Dimensionality Reduction

Please read the following instructions very carefully

Working on the assignment / FAQs

- **Always use the seed/random_state as *42* wherever applicable** (This is to ensure repeatability in answers, across students and coding environments)
- The type of question and the points they carry are indicated in each question cell
- To avoid any ambiguity, each question also specifies what *value* must be set. Note that these are dummy values and not the answers
- If an autograded question has multiple answers (due to differences in handling NaNs, zeros etc.), all answers will be considered.
- You can delete the `raise NotImplementedError()`
- **Submitting the assignment** : Download the '.ipynb' file and the PDF file from Colab and upload it to Gradescope. Do not delete any outputs from cells before submitting.
- That's about it. Happy coding!

Available software:

- Python's Gensim module: https://radimrehurek.com/gensim/ (install using pip)
- Sklearn’s TSNE module in case you use TSNE to reduce dimension (optional)
- Python’s Matplotlib (optional)

Note: The most important hyper parameters of skip-gram/CBOW are vector size and windows size

```
!pip install gensim

import pandas as pd
import numpy as np
import gensim
import requests
import string

from IPython.display import Image
from sklearn.manifold import TSNE

# To make the visualizations
```

```
!git clone https://github.com/CAHLR/d3-scatterplot.git
from google.colab.output import eval_js
from IPython.display import Javascript
from gensim.models import KeyedVectors

# To download trained model (Google news)
import gensim.downloader as api
google_model = api.load('word2vec-google-news-300')

import nltk
from nltk import word_tokenize, tokenize
nltk.download('punkt')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dis
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.7/dist-pa
fatal: destination path 'd3-scatterplot' already exists and is not an empty dire
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

Q1 (0.25 points)

Download your text corpus. (A good place to start is the nltk corpus or the gutenberg project)

```
#your code here
url = "https://www.gutenberg.org/cache/epub/69316/pg69316.txt"



#Save the raw text that you just downloaded in this variable
raw = requests.get(url).content.decode('utf8')


#This is an autograded cell, do not edit/delete
print(raw[:1000])
```

```
The Project Gutenberg eBook of Letters of a Japanese schoolboy
("Hashimura Togo"), by Wallace Irwin

This eBook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this eBook or online at
www.gutenberg.org. If you are not located in the United States, you
will have to check the laws of the country where you are located before
using this eBook.

Title: Letters of a Japanese schoolboy ("Hashimura Togo")
```

```
Author: Wallace Irwin

Illustrator: Rollin Kirby

Release Date: November 8, 2022 [eBook #69316]

Language: English

Produced by: Peter Becker and the Online Distributed Proofreading Team
             at https://www.pgdp.net (This file was produced from images
             generously made available by The Internet Archive)

*** START OF THE PROJECT GUTENBERG EBOOK LETT
```

Q2 (0.25 points)

Tokenize your corpus. Make sure that that the result is a list of lists i.e. The top-level list (outer list) is a list of sentences, and the inner list is a list of words in a given sentence.

Consider the following text:

```
text = "I spent $15.35 on my lunch today. Food in Berkeley is very expensive!"
```

It could be tokenized as follows:

```
tok_corp = [['I', 'spent', '$', '15.35', 'on', 'my', 'lunch', 'today'],
 ['Food', 'in', 'Berkeley', 'is', 'very', 'expensive']]
```

Note: There are many different (and correct) ways of tokenizing. Your answer doesn't need to match exactly with this illustrative example.

```
#code here
pattern = r'''(?x)  # set flag to allow verbose regexps
(?:[A-Z]\.)+        # abbreviations, e.g. U.S.A.
|\w+(?:[-']\w+)*    # words with optional internal hyphens
|\$?\d+(?:\.\d+)?   # currency, e.g. $12.80
|\.\.\.             # elipses
|[.,;"'?()-_`]      # these are separate tokens
'''
tokenized_raw = " ".join(nltk.regexp_tokenize(raw, pattern))
tokenized_raw = tokenize.sent_tokenize(tokenized_raw)

nopunct = []
for sent in tokenized_raw:
  a = [w for w in sent.split() if w not in string.punctuation]
  nopunct.append(" ".join(a))
```

```
#Save the tokenized sentences as a list of list in this variable
tok_corp = [nltk.word_tokenize(sent) for sent in nopunct]
```

```
#This is an autograded cell, do not edit/delete
for sent in tok_corp[:3]:
  print(sent)
  print("\n")
```

```
    ['The', 'Project', 'Gutenberg', 'eBook', 'of', 'Letters', 'of', 'a', 'Japanese',


    ['You', 'may', 'copy', 'it', 'give', 'it', 'away', 'or', 're-use', 'it', 'under'


    ['gutenberg']
```

Q3 (0.25 points)

Train gensim using your own dataset. Name the trained model variable as `model`.

```
#code here
model = gensim.models.Word2Vec(tok_corp, min_count=1, size=16, window=5)
```

```
#This is an autograded cell, do not edit/delete
print(f'Corpus Size: {model.corpus_total_words}')
print(f'Corpus Count: {model.corpus_count}')
print(f'Training time: {model.total_train_time}')
print(f'Sample words: {list(model.wv.vocab.keys())[:10]}')
```

```
    Corpus Size: 71139
    Corpus Count: 6113
    Training time: 0.47801843299748725
    Sample words: ['The', 'Project', 'Gutenberg', 'eBook', 'of', 'Letters', 'a', 'Ja
```

Q4 (0.25 points)

Q4a

Create a list of the unique set of words from your corpus. Name the list variable as `unique_words`.

Double-click (or enter) to edit

```
#code here

unique_words = list(set([item for sublist in tok_corp for item in sublist]))
```

```
#This is an autograded cell, do not edit/delete
print(unique_words[:10])

    ['bang-bang', 'END', 'salary', 'Judge', 'Fort', 'sadden', 'fuse', 'deploy', 'Chi
```

Q4b

Extract respective vectors corresponding to the words in your corpus and store the vectors in a variable called `vector_list`.

```
#code here
vector_list = model[unique_words]

    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarni
```

```
#This is an autograded cell, do not edit/delete
print(f'Array Shape: {np.array(vector_list).shape}')
for i in range(5):
    print(unique_words[i], vector_list[i])

    Array Shape: (11278, 16)
    bang-bang [ 0.06023511 -0.04318345 -0.00479272 -0.0117695  -0.00205196  0.035713
     -0.01603615  0.02087125  0.0084474  -0.02114104 -0.0760133   0.09923886
     -0.01765154 -0.04224399 -0.04795903  0.06435612]
    END [-1.8891824e-02 -2.7848449e-02 -1.1110110e-03  2.7973875e-02
     -3.3080898e-02 -2.2050938e-02 -8.6063435e-03 -1.9078301e-02
     -1.9208135e-02 -3.5354465e-02 -3.5789870e-02  1.2150887e-02
     -1.8026620e-02 -6.3882489e-03  6.1616767e-05  2.0525901e-02]
    salary [ 0.08026363 -0.10477541 -0.04804569  0.03351863 -0.08099951 -0.00291028
     -0.00979601  0.00772638  0.0049349  -0.09710971 -0.14449555  0.16249155
      0.02996092 -0.07648869 -0.04326785  0.19868726]
    Judge [ 0.16405036 -0.19264522 -0.10041884  0.05510325 -0.09057735  0.03955089
     -0.03070532  0.0349081  -0.04200706 -0.15589476 -0.2129277   0.28126594
     -0.01709784 -0.13551208 -0.09383342  0.3043847 ]
    Fort [ 0.03130146  0.00321848 -0.02748401 -0.00369863 -0.00185576  0.02700665
      0.02525776 -0.02510339  0.00805265 -0.04022545 -0.00490281  0.03362846
     -0.02919581  0.01065883  0.00624289  0.05377763]
```

Q5 (3 points)

Based on your knowledge and understanding of the text corpus you have chosen, **form 3 hypotheses** of analogies or relationships (between words) that you expect will hold and **give a**

reason why. Experimentally validate these hypotheses using similarity of the word vectors.

Example: If using Moby Dick as the corpus, one hypothesis might be that the whale, "Moby Dick" is (cosine) more similar to "fate" than to "evil" because Moby Dick is symbolic of the nature and the universe and isn't necessarily 'bad'. Or "Moby Dick" is more similar to "opposition" than to "surrender" because Moby Dick fights for its survival.

Note: Please do NOT use the same example as in the prompt.

Note 2: It's okay if the model disproves your hypotheses.

---Your hypotheses here--- Hypothesis 1: "NICHI" is (cosine) more similar to "happy" than to "sad" because Uncle Nichi was described as a joy Japanese in the book.

Hypothesis 2: "ARTHUR" is (cosine) more similar to "good" than to "bad" because ARTHUR KICKAHAJAMA was described as a good friend of the narrator in the book.

Hypothesis 3: "HASHIMURA" is (cosine) more similar to "marry" than to "divorce" because HASHIMURA TOGO was married with Ms. Alice in the book.

```
#your code here for validating hypotheses 1
print(model.similarity('NICHI', 'happy'), model.similarity('NICHI', 'sad'))
print(model.similarity('NICHI', 'happy') > model.similarity('NICHI', 'sad'))
```

```
0.5873035 0.6452235
False
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarni

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarni
  This is separate from the ipykernel package so we can avoid doing imports unti
```

```
#your code here for validating hypotheses 2
print(model.similarity('ARTHUR', 'good'), model.similarity('ARTHUR', 'bad'))
print(model.similarity('ARTHUR', 'good') > model.similarity('ARTHUR', 'bad'))
```

```
0.97539496 0.9778442
False
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarni

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarni
  This is separate from the ipykernel package so we can avoid doing imports unti
```

```
#your code here for validating hypotheses 3
print(model.similarity('HASHIMURA', 'marry'), model.similarity('HASHIMURA', 'divorce'
print(model.similarity('HASHIMURA', 'marry') > model.similarity('HASHIMURA', 'divorce
```

```
0.98273164 0.96628183
True
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarni
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarni
  This is separate from the ipykernel package so we can avoid doing imports unti
```

Q6 Visualizing the trained vectors (1.5 points)

Q6a

Run K-means clustering on your word vectors (as you did in Q-6 of Lab-5). Use the word vectors from the model you trained in this lab.

```
#your code here

import random
from sklearn.cluster import KMeans

df = pd.DataFrame(data = vector_list, index = unique_words)
kmeans = KMeans(n_clusters=50, random_state=42)
kmeans.fit(df)
df['Cluster'] = kmeans.labels_

for i in range(50):
  print(model.similar_by_vector(kmeans.cluster_centers_[i], topn= 5))
#most_rep_cluster1 = model.similar_by_vector(kmeans.cluster_centers_[0], topn= 5)
#most_rep_cluster2 = model.similar_by_vector(kmeans.cluster_centers_[1], topn= 5)
#most_rep_cluster3 = model.similar_by_vector(kmeans.cluster_centers_[2], topn= 5)
#most_rep_cluster4 = model.similar_by_vector(kmeans.cluster_centers_[3], topn= 5)

#most_rep_cluster1, most_rep_cluster2, most_rep_cluster3, most_rep_cluster4
```

```
[('Japanese', 0.9998921155929565), ('him', 0.9998654127120972), ('following', 0.
[('for', 0.9999425411224365), ('with', 0.9999337792396545), ('they', 0.999929189
[('for', 0.9999440312385559), ('with', 0.9999428987503052), ('who', 0.9999391436
[('Sometimes', 0.937481164932251), ('22d', 0.9360834956169128), ('DAMAGES', 0.93
[('with', 0.9999467134475708), ('of', 0.9999418258666992), ('in', 0.999929249286
[('and', 0.9999873042106628), ('with', 0.999987006187439), ('of', 0.999970912933
[('about', 0.9999430179595947), ('with', 0.9999430179595947), ('for', 0.99994158
[('for', 0.9999380111694336), ('with', 0.9999146461486816), ('from', 0.999913215
[('yellow', 0.9956674575805664), ('truth', 0.9951812028884888), ('feetball', 0.9
[('by', 0.9999682307243347), ('from', 0.9999639987945557), ('of', 0.999958455562
[('and', 0.999968945980072), ('by', 0.9999637603759766), ('of', 0.99995911121368
[('pass-key', 0.9586465954780579), ('second-handed', 0.9505091309547424), ('sele
[('of', 0.9999459981918335), ('with', 0.9999343752861023), ('who', 0.99992918968
[('who', 0.9999613761901855), ('by', 0.9999489188194275), ('with', 0.99994522333
[('feetprints', 0.9540368318557739), ('Everybody', 0.9496701955795288), ('Floren
[('with', 0.9999538064002991), ('they', 0.9999423623085022), ('who', 0.999942362
[('from', 0.9999581575393677), ('of', 0.9999524354934692), ('by', 0.999951958656
[('Grasshop', 0.8794829845428467), ('tempting', 0.8627054691314697), ('bisickle'
[('disguised', 0.9892245531082153), ('Nations', 0.9888774156570435), ('yall', 0.
[('by', 0.99994957447052), ('with', 0.9999465942382812), ('who', 0.9999458193778
[('for', 0.9999529123306274), ('with', 0.9999498724937439), ('who', 0.9999467134
[('ball', 0.9976485371589661), ('AND', 0.9968286156654358), ('noise', 0.99679315
[('some', 0.9998503923416138), ('about', 0.999838650226593), ('make', 0.99982285
[('makes', 0.993000328540802), ('Come', 0.9928508400917053), ('first', 0.9927964
[('get-up', 0.9298765659332275), ('howeverly', 0.924353837966919), ('cloak', 0.9
[('for', 0.9999433159828186), ('with', 0.9999425411224365), ('about', 0.99993932
[('Meadow', 0.6888826489448547), ('caretaker', 0.6756041049957275), ('doorknob',
[('Ocean', 0.9896140098571777), ('Italian', 0.9893234372138977), ('occasional',
[('angry', 0.999695897102356), ('in', 0.9996615648269653), ('By', 0.999649405479
[('races', 0.9962745904922485), ('weekly', 0.9954196214675903), ('J.', 0.9938088
[('online', 0.9773387908935547), ('diet', 0.9735938310623169), ('objeck', 0.9708
[('Ministers', 0.9502969980239868), ('EARTH', 0.9500640034675598), ('remainder',
[('of', 0.9999527335166931), ('with', 0.9999306201934814), ('other', 0.999929666
[('somebody', 0.9682441353797913), ('peculiar', 0.9621955156326294), ('heathens'
[('other', 0.9999644756317139), ('from', 0.9999537467956543), ('in', 0.999948918
[('in', 0.999935507774353), ('are', 0.9999151229858398), ('on', 0.99990272521972
[('with', 0.9999725818634033), ('who', 0.9999516606330872), ('they', 0.999950766
[('them', 0.999944806098938), ('him', 0.999943196773529), ('from', 0.99993610382
[('one', 0.9998669624328613), ('them', 0.9998598098754883), ('all', 0.9998233318
[('who', 0.9999449253082275), ('are', 0.9999327659606934), ('of', 0.999929010868
[('exploders', 0.9352438449859619), ('column', 0.9057133197784424), ('snorty', 0
[('of', 0.999950647354126), ('from', 0.9999459981918335), ('and', 0.999935686588
[('of', 1.0), ('and', 0.9999666213989258), ('with', 0.9999494552612305), ('who',
[('for', 0.9999624490737915), ('in', 0.9999585151672363), ('by', 0.9999532699584
[('with', 0.9999440908432007), ('of', 0.9999390244483948), ('and', 0.99993610382
[('of', 0.9999585747718811), ('and', 0.9999423623085022), ('with', 0.99993193149
[('suspicious', 0.9825500249862671), ('twisted', 0.9817145466804504), ('Wallace'
[('especially', 0.976207971572876), ('_Dreadnothings_', 0.9722205400466919), ('p
[('Willy', 0.997255802154541), ('DO', 0.9970303773880005), ('among', 0.996961236
[('U.S.', 0.9868758320808411), ('fly', 0.9862155914306641), ('practice', 0.98109
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: DeprecationWarn
  if sys.path[0] == '':
```

Q6b

Reduce the dimensionality of your word vectors using TSNE

```
#your code here
from sklearn.manifold import TSNE
data_embed = TSNE(n_components=2, perplexity=50, verbose=2, method='barnes_hut').fit_
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWar
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWar
  FutureWarning,
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 11278 samples in 0.001s...
[t-SNE] Computed neighbors for 11278 samples in 3.545s...
[t-SNE] Computed conditional probabilities for sample 1000 / 11278
[t-SNE] Computed conditional probabilities for sample 2000 / 11278
[t-SNE] Computed conditional probabilities for sample 3000 / 11278
[t-SNE] Computed conditional probabilities for sample 4000 / 11278
```

```
[t-SNE] Computed conditional probabilities for sample 5000 / 11278
[t-SNE] Computed conditional probabilities for sample 6000 / 11278
[t-SNE] Computed conditional probabilities for sample 7000 / 11278
[t-SNE] Computed conditional probabilities for sample 8000 / 11278
[t-SNE] Computed conditional probabilities for sample 9000 / 11278
[t-SNE] Computed conditional probabilities for sample 10000 / 11278
[t-SNE] Computed conditional probabilities for sample 11000 / 11278
[t-SNE] Computed conditional probabilities for sample 11278 / 11278
[t-SNE] Mean sigma: 0.026312
[t-SNE] Computed conditional probabilities in 1.231s
[t-SNE] Iteration 50: error = 92.0661469, gradient norm = 0.0121664 (50 iteratic
[t-SNE] Iteration 100: error = 86.1811371, gradient norm = 0.0009317 (50 iterati
[t-SNE] Iteration 150: error = 85.9480133, gradient norm = 0.0004032 (50 iterati
[t-SNE] Iteration 200: error = 85.8849030, gradient norm = 0.0002532 (50 iterati
[t-SNE] Iteration 250: error = 85.8580780, gradient norm = 0.0001996 (50 iterati
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.858078
[t-SNE] Iteration 300: error = 3.5960159, gradient norm = 0.0012049 (50 iteratic
[t-SNE] Iteration 350: error = 3.3364768, gradient norm = 0.0004531 (50 iteratic
[t-SNE] Iteration 400: error = 3.2131424, gradient norm = 0.0002716 (50 iteratic
[t-SNE] Iteration 450: error = 3.1368361, gradient norm = 0.0001894 (50 iteratic
[t-SNE] Iteration 500: error = 3.0837140, gradient norm = 0.0001435 (50 iteratic
[t-SNE] Iteration 550: error = 3.0444088, gradient norm = 0.0001119 (50 iteratic
[t-SNE] Iteration 600: error = 3.0140224, gradient norm = 0.0000910 (50 iteratic
[t-SNE] Iteration 650: error = 2.9897637, gradient norm = 0.0000759 (50 iteratic
[t-SNE] Iteration 700: error = 2.9701347, gradient norm = 0.0000666 (50 iteratic
[t-SNE] Iteration 750: error = 2.9540040, gradient norm = 0.0000593 (50 iteratic
[t-SNE] Iteration 800: error = 2.9405499, gradient norm = 0.0000534 (50 iteratic
[t-SNE] Iteration 850: error = 2.9296770, gradient norm = 0.0000491 (50 iteratic
[t-SNE] Iteration 900: error = 2.9208298, gradient norm = 0.0000464 (50 iteratic
[t-SNE] Iteration 950: error = 2.9135993, gradient norm = 0.0000424 (50 iteratic
[t-SNE] Iteration 1000: error = 2.9078941, gradient norm = 0.0000405 (50 iterati
[t-SNE] KL divergence after 1000 iterations: 2.907894
```

Q6c

Create a dataframe with the following columns:

Column	Description
x	the first dimension of result from TSNE
y	the second dimension of result from TSNE
Feature 1	the word corresponding to the vector
Feature 2	the kmeans cluster label

Below is a sample of what the dataframe could look like:

x	y	Feature 1	Feature 2
7.154159	9.251100	lips	8
-53.254147	-13.514915	them	9
34.049191	-13.402843	topic	0
-32.515320	28.699677	sofa	24

x	y	Feature 1	Feature 2

```
#your code here
df = pd.DataFrame(data_embed[:,:2],columns=["x","y"])
df['Feature 1'] = unique_words
df['Feature 2'] = kmeans.labels_
df.head()
```

	x	y	Feature 1	Feature 2
0	-18.563307	29.503450	flannel	23
1	-2.947846	22.093597	states	27
2	-38.640141	44.129135	socialism	28
3	-50.912109	27.359970	Majesty	0
4	-17.917185	11.271204	elopement	29

Q6d: Visualization

In this question, you are required to visualize and explore the reduced dataset you created in Q6c using the d3-scatterplot library.

Note: The first code-chunk at the top in this notebook clones the library from github. Make sure that it has been executed properly before you proceed.

Save your dataset as a tsv file 'd3-scatterplot/mytext_new.tsv'

Example:

```
df.to_csv('d3-scatterplot/mytext_new.tsv', sep='\t', index=False)
```

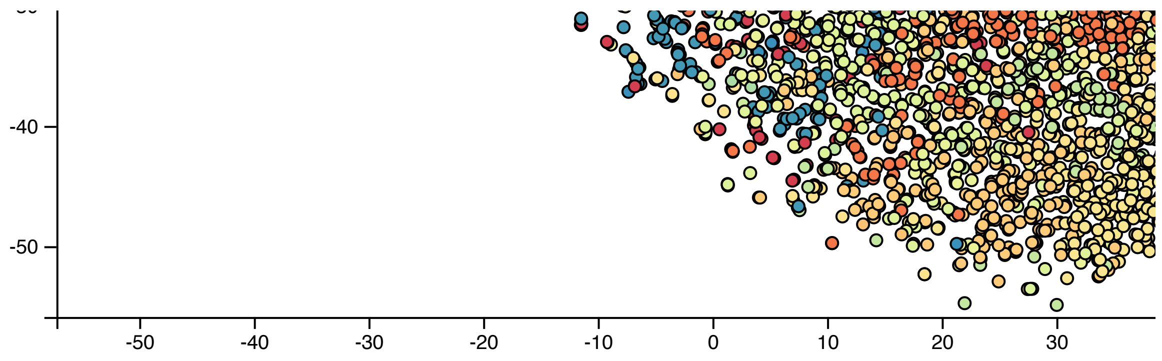
Note 1: The TSV file needs to be stored in the `d3-scatterplot` folder so that the d3-scatterplot library can access it.

Note 2: Make sure to save the TSV file WITHOUT the row index i.e. use `index=False`.

```
#your code here

df.to_csv('d3-scatterplot/mytext_new.tsv', sep='\t', index=False)
```

Visualize the reduced vectors by running the following code

```
def show_port(port, data_file, width=600, height=800):
  display(Javascript("""
  (async ()=>{
    fm = document.createElement('iframe')
    fm.src = await google.colab.kernel.proxyPort(%d) + '/index.html?dataset=%s'
    fm.width = '90%%'
    fm.height = '%d'
    fm.frameBorder = 0
    document.body.append(fm)
  })();
  """ % (port, data_file, height)))

port = 8001
data_file = 'mytext_new.tsv'
height = 1400

get_ipython().system_raw('cd d3-scatterplot && python3 -m http.server %d &' % port)
show_port(port, data_file, height)
```



Color the points by their kmeans cluster. You can do this by choosing "Feature 2" as the variable in the "Color" drop-down in the visualization.

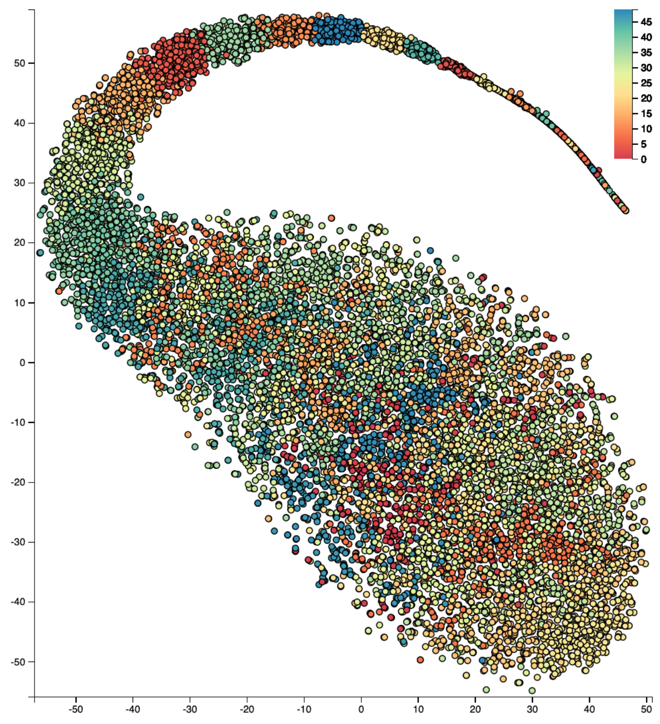
Please include a snapshot of your visualization in the notebook. Refer to the tutorial video, and/or follow the steps below:

1) Take a snapshot of the visualization and save it on your computer with the filename `trained_scatter.png`

2) Upload the `trained_scatter.png` by clicking on the 'Files' icon on the left sidebar and clicking on the upload icon (the one with an upward arrow on top left)

3) Run the code below to display the image

```
#This is an autograded cell, do not edit/delete
Image('trained_scatter.png')
```



Q7 Visualizing the PRE-TRAINED vectors (1.5 points)

In this question, you'll execute the same analysis as in Q6, but on PRE-TRAINED vectors.

Q7a

Load the google vector model

(It must be downloaded as `GoogleNews-vectors-negative300` for you if you ran the first code-chunk at the top of this notebook)

```
# google_model
google_model
```

```
    <gensim.models.keyedvectors.Word2VecKeyedVectors at 0x7f354871bd10>
```

Downsample the pre-trained google model to anywhere between 10,000 to 25,000 words.

```
#your code here
new_tok_corp = np.random.choice(tok_corp, 10000)
```

```
    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: VisibleDeprecati
```

Create a list of the unique set of words from this downsampled model

```
#your code here
new_unique_words = list(set([item for sublist in new_tok_corp for item in sublist]))
new_unique_words = [x for x in new_unique_words and google_model.vocab][:10000] #coll
```

```
len(new_unique_words)
```

```
    10000
```

Extract respective vectors corresponding to the words in the down-sampled, pre-trained model

```
#your code here

vector_list = google_model[new_unique_words]
```

Q7b

Run Kmeans clustering on the pre-trained word vectors. Make sure to use the word vectors from the pre-trained model.

```
#your code here
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=25, random_state=42)
X = np.array(vector_list)
kmeans.fit(X)

kmeans.labels_
```

```
    array([11, 22, 11, ..., 11, 14,  3], dtype=int32)
```

Q7c

Reduce the dimensionality of the word vectors from the pre-trained model using tSNE

```
#your code here
from sklearn.manifold import TSNE
data_embed = TSNE(n_components=2, perplexity=50, verbose=2, method='barnes_hut').fit_
```

```
    /usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWar
      FutureWarning,
    /usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWar
      FutureWarning,
    [t-SNE] Computing 151 nearest neighbors...
    [t-SNE] Indexed 10000 samples in 0.003s...
    [t-SNE] Computed neighbors for 10000 samples in 3.626s...
    [t-SNE] Computed conditional probabilities for sample 1000 / 10000
    [t-SNE] Computed conditional probabilities for sample 2000 / 10000
    [t-SNE] Computed conditional probabilities for sample 3000 / 10000
    [t-SNE] Computed conditional probabilities for sample 4000 / 10000
    [t-SNE] Computed conditional probabilities for sample 5000 / 10000
    [t-SNE] Computed conditional probabilities for sample 6000 / 10000
    [t-SNE] Computed conditional probabilities for sample 7000 / 10000
    [t-SNE] Computed conditional probabilities for sample 8000 / 10000
    [t-SNE] Computed conditional probabilities for sample 9000 / 10000
    [t-SNE] Computed conditional probabilities for sample 10000 / 10000
    [t-SNE] Mean sigma: 0.983152
    [t-SNE] Computed conditional probabilities in 0.770s
    [t-SNE] Iteration 50: error = 98.4306335, gradient norm = 0.0993702 (50 iteratic
    [t-SNE] Iteration 100: error = 98.1380539, gradient norm = 0.1134312 (50 iterati
    [t-SNE] Iteration 150: error = 100.5511322, gradient norm = 0.0786403 (50 iterat
    [t-SNE] Iteration 200: error = 101.1620941, gradient norm = 0.0956368 (50 iterat
    [t-SNE] Iteration 250: error = 101.0585022, gradient norm = 0.1154180 (50 iterat
    [t-SNE] KL divergence after 250 iterations with early exaggeration: 101.058502
    [t-SNE] Iteration 300: error = 3.9318995, gradient norm = 0.0042898 (50 iteratic
    [t-SNE] Iteration 350: error = 3.3794613, gradient norm = 0.0006811 (50 iteratic
    [t-SNE] Iteration 400: error = 3.1876633, gradient norm = 0.0003683 (50 iteratic
    [t-SNE] Iteration 450: error = 3.0038545, gradient norm = 0.0008544 (50 iteratic
    [t-SNE] Iteration 500: error = 2.8845005, gradient norm = 0.0005063 (50 iteratic
    [t-SNE] Iteration 550: error = 2.8224816, gradient norm = 0.0001455 (50 iteratic
    [t-SNE] Iteration 600: error = 2.7804420, gradient norm = 0.0001204 (50 iteratic
    [t-SNE] Iteration 650: error = 2.7468941, gradient norm = 0.0001025 (50 iteratic
    [t-SNE] Iteration 700: error = 2.7204587, gradient norm = 0.0000869 (50 iteratic
    [t-SNE] Iteration 750: error = 2.6990807, gradient norm = 0.0000777 (50 iteratic
    [t-SNE] Iteration 800: error = 2.6813445, gradient norm = 0.0000706 (50 iteratic
    [t-SNE] Iteration 850: error = 2.6667705, gradient norm = 0.0000631 (50 iteratic
    [t-SNE] Iteration 900: error = 2.6547108, gradient norm = 0.0000596 (50 iteratic
    [t-SNE] Iteration 950: error = 2.6442175, gradient norm = 0.0000542 (50 iteratic
    [t-SNE] Iteration 1000: error = 2.6351495, gradient norm = 0.0000494 (50 iterati
    [t-SNE] KL divergence after 1000 iterations: 2.635149
```

Q7d

Create a dataframe with the following columns using the pre-trained vectors and corpus:

Column	Description
x	the first dimension of result from TSNE
y	the second dimension of result from TSNE
Feature 1	the word corresponding to the vector
Feature 2	the kmeans cluster label

Below is a sample of what the dataframe could look like:

x	y	Feature 1	Feature 2
7.154159	9.251100	lips	8

x	y	Feature 1	Feature 2
-53.254147	-13.514915	them	9
34.049191	-13.402843	topic	0
-32.515320	28.699677	sofa	24
13.006302	-4.270626	half-past	21

```
#your code here
google_df = pd.DataFrame(data_embed[:,:2],columns=["x","y"])
google_df['Feature 1'] = new_unique_words
google_df['Feature 2'] = kmeans.labels_
google_df.head()
```

	x	y	Feature 1	Feature 2
0	7.498052	7.210917	</s>	11
1	4.880733	3.773450	in	22
2	6.664927	2.946017	for	11
3	12.032755	5.872264	that	11
4	26.025517	-6.327523	is	11

Q7e: Visualization

In this question, you are required to visualize and explore the reduced dataset **from the pretrained model** you created in Q7d using the d3-scatterplot library.

Note: The first code-chunk at the top in this notebook clones the libary from github. Make sure that it has been executed before you proceed.

Save your dataset as a tsv file 'd3-scatterplot/google_mytext.tsv'

Example:

```
google_df.to_csv('d3-scatterplot/google_mytext.tsv', sep='\t', index=False)
```

Note 1: The TSV file needs to be stored in the `d3-scatterplot` folder so that the d3-scatterplot library can access it.

Note 2: Make sure to save the TSV file WITHOUT the row index i.e. use `index=False`.

```
#your code here
google_df.to_csv('d3-scatterplot/google_mytext.tsv', sep='\t', index=False)
```

Visualize the reduced vectors by running the following code

```
def show_port(port, data_file, width=600, height=800):
  display(Javascript("""
  (async ()=>{
    fm = document.createElement('iframe')
    fm.src = await google.colab.kernel.proxyPort(%d) + '/index.html?dataset=%s'
    fm.width = '90%%'
    fm.height = '%d'
    fm.frameBorder = 0
    document.body.append(fm)
  })();
  """ % (port, data_file, height)))

port = 8001
data_file = 'google_mytext.tsv'
height = 1400

get_ipython().system_raw('cd d3-scatterplot && python3 -m http.server %d &' % port)
show_port(port, data_file, height)
```

```
        Armstrong  23
        Reed       23
        Gray       23
        Cole       23
        Ellis      23
        Kent       23
        Ferguson   23
        Stevens    23
        Harper     23
        Foster     23
        Barnes     23
        Carroll    23
        Brooks     23
        Bond       23
        Simpson    23
```

Color the points by their kmeans cluster. You can do this by choosing "Feature 2" as the variable in the "Color" drop-down in the visualization.

```
        Duncan     23
```

Please include a snapshot of your visualization in the notebook. Refer to the tutorial video, and/or follow the steps below:

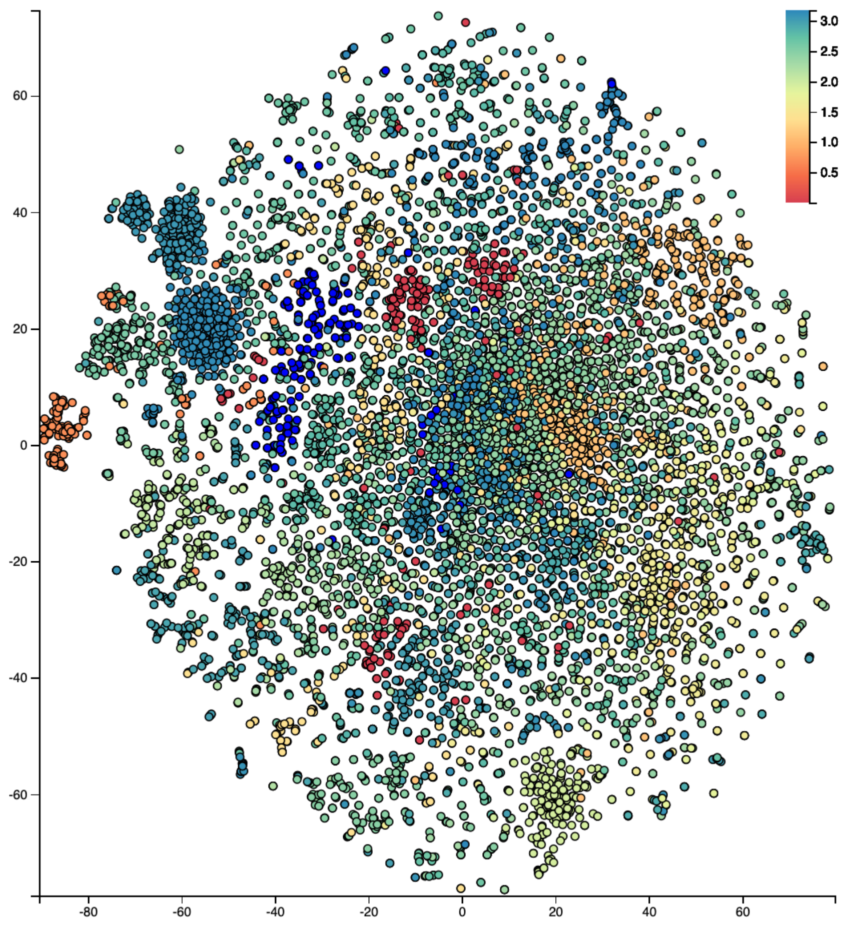
1) Take a snapshot of the visualization and save it on your computer with the filename `google_trained_scatter.png`

2) Upload the `google_trained_scatter.png` by clicking on the 'Files' icon on the left sidebar and clicking on the upload icon (the one with an upward arrow on top left)

3) Run the code below to display the image

```
        Manning    23
```

```
#This is an autograded cell, do not edit/delete
Image('google_trained_scatter.png')
```



Q8: Exploration (0.5 points)

This is an open-ended question.

On the visualizations in Q6 & Q7, lasso-select a group of points with the left mouse button and look at summaries of the group on the right-side of the plot. (Refer to the tutorial video for a demo on the lasso selection). Also look at the words / features of the selected points.

Comment on any patterns / similarities you see in the selected words in the visualization for the pre-trained vectors and the vectors trained on your corpus. Are you able to find any group of points that are close to each other in the 2D space that also have semantic similarity?

--your answer here-- I think the words in Q7 around (20, -60) are mostly belonged to cluster 7. Most of words serve for adverbial part of sentences. Also, words are labeled as deep blue around (-40, 20) are mostly relative to playing games.

I also found out that words around (-60, 20) and (-60, 40) are representing last names and first names of characters. So that's why they are close in the 2D space.