Importing all essential libraries

```
#Importing all libraries
import pandas as pd
import sqlite3
import numpy as np
import pickle as pickle
from sklearn.decomposition import PCA
import subprocess
import io

#Optional Libraries
#from gensim.models import word2vec
#import gensim.downloader as api
#from gensim.models import KeyedVectors


#Glove pretrained word embeddings: https://nlp.stanford.edu/projects/glove/


#Connect your drive as file system (If you have your files on drive)
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

Word Embeddings part (Pre-trained embeddings used: Glove)

Reducing the vector dimensions unsing PCA

```
#Reducing Glove word embeddings from 50 to 10 dimensions
Glove = {}
with io.open('/content/drive/My Drive/NLP_Data/glove.6B.50d.txt', encoding='utf8')
#f = open('/content/drive/mydrive/glove.6B.50d.txt')

    print("Loading Glove vectors.")
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        Glove[word] = coefs
    f.close()

    print("Done.")
    X_train = []
    X_train_names = []
    for x in Glove:
            X_train.append(Glove[x])
            X_train_names.append(x)

    X_train = np.asarray(X_train)
    pca_embeddings = {}
```

```
    pca_embeddings = {}

# PCA to get Top Components
    pca =  PCA(n_components = 50)
    X_train = X_train - np.mean(X_train)
    X_fit = pca.fit_transform(X_train)
    U1 = pca.components_

    z = []

# Removing Projections on Top Components
    for i, x in enumerate(X_train):
      for u in U1[0:7]:
              x = x - np.dot(u.transpose(),x) * u
      z.append(x)

    z = np.asarray(z)

# PCA Dim Reduction
    pca =  PCA(n_components = 10)
    X_train = z - np.mean(z)
    X_new_final = pca.fit_transform(X_train)


# PCA to do Post-Processing Again
    pca =  PCA(n_components = 10)
    X_new = X_new_final - np.mean(X_new_final)
    X_new = pca.fit_transform(X_new)
    Ufit = pca.components_

    X_new_final = X_new_final - np.mean(X_new_final)

    final_pca_embeddings = {}
    embedding_file = open('/content/drive/My Drive/NLP_Data/pca_embed2.txt', 'w')

    for i, x in enumerate(X_train_names):
      final_pca_embeddings[x] = X_new_final[i]
      embedding_file.write("%s\t" % x)
      for u in Ufit[0:7]:
        final_pca_embeddings[x] = final_pca_embeddings[x] - np.dot(u.transpose(),fi

      for t in final_pca_embeddings[x]:
        embedding_file.write("%f\t" % t)

      embedding_file.write("\n")


    print("Reduced the dimensionality of the vector to 10 dimensions! \nPlease chec
```

```
Loading Glove vectors.
Done.
Reduced the dimensionality of the vector to 10 dimensions!
Please check pca_embed2.txt file
```

```
#Function to get 10 dimensional vector from txt file
def get_vector(given_word):
```

```
def get_vector(given_word):
  Glove = {}
  with io.open('/content/drive/My Drive/NLP_Data/pca_embed2.txt', encoding='utf8')
  #f = open('/content/drive/My Drive/NLP_Data/pca_embed2.txt')

      #print("Loading Glove vectors.")
      for line in f:
          values = line.split()
          word = values[0]
          if word == given_word:
            coefs = np.asarray(values[1:], dtype='float32')
            given_word_vector = coefs
            break
  f.close()
  return given_word_vector


#Getting Vectors for available posts
engineer_vector = get_vector('engineer')
manager_vector = get_vector('manager')
developer_vector = get_vector('developer')
ceo_vector = get_vector('ceo')
cto_vector = get_vector('cto')
coo_vector = get_vector('coo')
waiter_vector = get_vector('waiter')

#Getting Vectors for available cities
victoria_vector = get_vector('victoria')
vancouver_vector = get_vector('vancouver')
delhi_vector = get_vector('delhi')
pune_vector = get_vector('pune')
ottawa_vector = get_vector('ottawa')
toronto_vector = get_vector('toronto')
mumbai_vector = get_vector('mumbai')

#Stored these vectors in CSV Files
#File names:
#1. table1_name_post_city.csv
#2. table2_vectors_for_post_city.csv
#3. table3_vectors_for_posts.csv
#4. table4_vectors_for_cities.csv


#Defining Cosine Similarity Function
def cos_sim(a, b):
    """Takes 2 vectors a, b and returns the cosine similarity according
    to the definition of the dot product
    """
    dot_product = np.dot(a, b)
    norm_a = np.linalg.norm(a)
    norm_b = np.linalg.norm(b)
    return dot_product / (norm_a * norm_b)

#Testing the function
similarity = cos_sim(engineer_vector,manager_vector)
print(similarity)
```

```
0.8390401
```

Database creation part

1. Reading data with Pandas

```
#Opening & Reading CSV files into pandas dataframe

#Dataframe with Person name, Applicable Post and City
data = pd.read_csv (r'/content/drive/My Drive/NLP_Data/table1_name_post_city.csv')
df1 = pd.DataFrame(data, columns= ['Name','pi1','pi2','pi3','pi4','pi5','pi6','pi7'
#print(df1)

#Dataframe with Post Available and City
data = pd.read_csv (r'/content/drive/My Drive/NLP_Data/table2_vectors_for_post_city
df2 = pd.DataFrame(data, columns= ['pi1','pi2','pi3','pi4','pi5','pi6','pi7','pi8',
#print(df2)

#Dataframe with Vectors for respective posts and post
data = pd.read_csv (r'/content/drive/My Drive/NLP_Data/table3_vectors_for_posts.csv
df3 = pd.DataFrame(data, columns= ['pi1','pi2','pi3','pi4','pi5','pi6','pi7','pi8',
#print(df3)

#Dataframe with Vectors for respective cities and city
data = pd.read_csv (r'/content/drive/My Drive/NLP_Data/table4_vectors_for_cities.cs
df4 = pd.DataFrame(data, columns= ['ci1','ci2','ci3','ci4','ci5','ci6','ci7','ci8',
#print(df4)
```

Database creation part

2. Creating Sqlite database

```
#Creating database
connection = sqlite3.connect("position_city_database_with_embeddings.db")
crsr = connection.cursor()

#Comment the table creation and insertion of data into the table if the database is

#Creating table1 with name, embeddings of post, and embeddings of city
crsr.execute('CREATE TABLE name_post_city (NAME nvarchar(50),pi1 float,pi2 float,pi
df1.to_sql('name_post_city', connection, if_exists='replace', index = False)
crsr.execute('''SELECT * FROM name_post_city''')
print("Table 1: Name_Post_City Data")
for row in crsr.fetchall():
    print (row)

#Creating table2 with embeddings of post and embeddings of city
crsr.execute('CREATE TABLE post_city (ci1 float,ci2 float,ci3 float,ci4 float,ci5 f
df2.to_sql('post_city', connection, if_exists='replace', index = False)
print("Table 2: Post_City Data")
```

```
crsr.execute('''SELECT * FROM post_city''')
for row in crsr.fetchall():
    print (row)

#Creating table3 with embeddings of post and name of posts
crsr.execute('CREATE TABLE em_post_name (pi1 float,pi2 float,pi3 float,pi4 float,pi
df3.to_sql('em_post_name', connection, if_exists='replace', index = False)
print("Table 3: Em_Post_Name Data")
crsr.execute('''SELECT * FROM em_post_name''')
for row in crsr.fetchall():
    print (row)

#Creating table4 with embeddings of city and name of cities
crsr.execute('CREATE TABLE em_city_name (ci1 float,ci2 float,ci3 float,ci4 float,ci
df4.to_sql('em_city_name', connection, if_exists='replace', index = False)
print("Table 4: Em_City_Name Data")
crsr.execute('''SELECT * FROM em_city_name''')
for row in crsr.fetchall():
    print (row)

connection.commit()

#Incase of you want to drop all tables:
#crsr.execute('DROP TABLE name_post_city')
#crsr.execute('DROP TABLE post_city')
#crsr.execute('DROP TABLE em_post_name')
#crsr.execute('DROP TABLE em_city_name')
#connection.commit()
```

```
Table 1: Name_Post_City Data
('Tom', 0.0, -1e-06, 0.0, 0.0, -1e-06, 1.1e-05, -4e-06, -0.252, 0.00328, 1.39,
('Henry', 0.0, -1e-06, 1e-06, 0.0, -4e-06, 5e-06, 1.4999999999999999e-05, 0.47
('Bush', 0.0, -1e-06, 1e-06, 0.0, -1e-06, 6e-06, 1.2e-05, 0.32, -0.31, 0.966,
('Ram', 0.0, -1e-06, 1e-06, 0.0, -4e-06, 5e-06, 1.4999999999999999e-05, 0.478,
('Bharat', 0.0, -2e-06, 2e-06, 0.0, -3e-06, 5e-06, 2.4e-05, 0.733, -0.16899999
('Laxman', 0.0, 0.0, -2e-06, 0.0, -1e-06, 6e-06, -2.2e-05, -0.736, 0.625, 0.42
('Krishna', 0.0, -2e-06, 2e-06, 0.0, -1e-06, 8e-06, 1.2e-05, 0.28, -0.44, 1.17
('Josh', 0.0, -1e-06, 0.0, 0.0, 2e-06, 6e-06, -9e-06, -0.424, -0.6579999999999
('Jelly', 0.0, -1e-06, 1e-06, 0.0, -4e-06, 5e-06, 1.4999999999999999e-05, 0.47
('Akash', 0.0, -1e-06, 1e-06, 0.0, -4e-06, 5e-06, 1.4999999999999999e-05, 0.47
('Ritul', 0.0, -1e-06, 1e-06, 0.0, -1e-06, 6e-06, 1.2e-05, 0.32, -0.31, 0.966,
('Rosie', 0.0, -1e-06, 1e-06, 0.0, -1e-06, 6e-06, 1.2e-05, 0.32, -0.31, 0.966,
('Alexa', 0.0, -2e-06, 2e-06, 0.0, -3e-06, 5e-06, 2.4e-05, 0.733, -0.16899999
('Sita', 0.0, -1e-06, 0.0, 0.0, -1e-06, 1.1e-05, -4e-06, -0.252, 0.00328, 1.39
('Erik', 0.0, -1e-06, 0.0, 0.0, -1e-06, 1.1e-05, -4e-06, -0.252, 0.00328, 1.39
('Deepika', 0.0, -1e-06, 1e-06, 0.0, -4e-06, 5e-06, 1.4999999999999999e-05, 0.
('Jerry', 0.0, -1e-06, 0.0, 0.0, 2e-06, 6e-06, -9e-06, -0.424, -0.657999999999
('Angelina', 0.0, -1e-06, 1e-06, 0.0, -1e-06, 6e-06, 1.2e-05, 0.32, -0.31, 0.9
('Selena', 0.0, -1e-06, 0.0, 0.0, -1e-06, 1.1e-05, -4e-06, -0.252, 0.00328, 1.
Table 2: Post_City Data
(0.0, -1e-06, 0.0, 0.0, -1e-06, 1.1e-05, -4e-06, -0.252, 0.00328, 1.39, 0.0, 0
(0.0, -1e-06, 1e-06, 0.0, -4e-06, 5e-06, 1.4999999999999999e-05, 0.478, 0.431,
(0.0, -1e-06, 1e-06, 0.0, -4e-06, 5e-06, 1.4999999999999999e-05, 0.478, 0.431,
(0.0, -1e-06, 0.0, 0.0, -1e-06, 1.1e-05, -4e-06, -0.252, 0.00328, 1.39, 0.0, -
(0.0, -1e-06, 1e-06, 0.0, -1e-06, 6e-06, 1.2e-05, 0.32, -0.31, 0.966, 0.0, 1e-
(0.0, -2e-06, 2e-06, 0.0, -3e-06, 5e-06, 2.4e-05, 0.733, -0.16899999999999998,
(0.0, -1e-06, 1e-06, 0.0, -1e-06, 6e-06, 1.2e-05, 0.32, -0.31, 0.966, 0.0, 1e-
(0.0, -1e-06, 0.0, 0.0, 2e-06, 6e-06, -9e-06, -0.424, -0.6579999999999999, 0.4
(0.0, -1e-06, 1e-06, 0.0, -4e-06, 5e-06, 1.4999999999999999e-05, 0.478, 0.431,
(0.0, -1e-06, 1e-06, 0.0, -1e-06, 6e-06, 1.2e-05, 0.32, -0.31, 0.966, 0.0, -1e
(0.0, -1e-06, 1e-06, 0.0, -1e-06, 6e-06, 1.2e-05, 0.32, -0.31, 0.966, 0.0, -1e
```

Function for calculating cosine similarity on:

---

1. Posts (Engineer, Developer, etc)

```
(0.0, -1e-06, 1e-06, 0.0, -1e-06, 6e-06, 1.2e-05, 0.32, -0.31, 0.966, 0.0, -2e
```

```
connection = sqlite3.connect("position_city_database_with_embeddings.db")
crsr = connection.cursor()


def sel_func_post(q):
  crsr.execute("select pi1,pi2,pi3,pi4,pi5,pi6,pi7,pi8,pi9,pi10 from em_post_name w
  q_vect = crsr.fetchall()
  crsr.execute("select pi1,pi2,pi3,pi4,pi5,pi6,pi7,pi8,pi9,pi10,post from em_post_n
  t_vect = crsr.fetchall()
  vect_t = [tuple(list(x)[0:10]) for x in t_vect]
  vect_names = [list(x).pop(-1) for x in t_vect]
  #print(vect_t)
  print(vect_names)
  similarity_array = [cos_sim(q_vect, x)[0] for x in vect_t]
  print(similarity_array)
  #print([x for x,y in vect_names,similarity_array if y > 0.8])

sel_func_post('engineer')
```

```
['engineer', 'developer', 'manager', 'coo', 'cto', 'ceo', 'waiter']
[1.0  0.79271217716575...  0.83916563940563...  0.54799252779612...  0.01408213198...
```

Function for calculating cosine similarity on:

---

2. Cities (Victoria, Pune, etc)

```
def sel_func_city(q):
  crsr.execute("select ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10 from em_city_name w
  q_vect = crsr.fetchall()
  crsr.execute("select ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,city from em_city_n
  t_vect = crsr.fetchall()
  vect_t = [tuple(list(x)[0:10]) for x in t_vect]
  vect_names = [list(x).pop(-1) for x in t_vect]
  #print(vect_t)
  print(vect_names)
  similarity_array = [cos_sim(q_vect, x)[0] for x in vect_t]
  print(similarity_array)
  a = np.array(similarity_array)
  index = np.where(a > 0.85)[0]
  print(index)
  new_vect_t = [vect_t[x] for x in index]
  #print(list(np.array(vect_t)[index][0]))
  join_list = []
  for x in range(0,len(new_vect_t[0])):
    join_list.append([i[x] for i in new_vect_t])
  print(join_list)
  return join_list
#sel_func_city('victoria')
```

Joining two tables according to word cosine similarities

In progress

```
connection = sqlite3.connect("position_city_database_with_embeddings.db")
crsr = connection.cursor()

#join_list = sel_func_city('victoria')
#format_strings = ','.join(['%s'] * len(join_list[0]))
#print(format_strings)
# Testing done by ninad, not yet working
join_list = sel_func_city('victoria')
format_strings = ','.join(['%s'] * len(join_list[0]))
crsr.execute('''SELECT *
FROM post_city a
INNER JOIN name_post_city b
ON a.ci1 = b.ci1 AND
a.ci2 = b.ci2 AND
a.ci3 = b.ci3 AND
a.ci4 = b.ci4 AND
a.ci5 = b.ci5 AND
a.ci6 = b.ci6 AND
a.ci7 = b.ci7 AND
```

```
a.ci7 = b.ci7 AND
a.ci8 = b.ci8 AND
a.ci9 = b.ci9 AND
a.ci10 = b.ci10
WHERE
a.ci1 IN '%s',
a.ci2 IN '%s',
a.ci3 IN '%s',
a.ci4 IN '%s',
a.ci5 IN '%s',
a.ci6 IN '%s',
a.ci7 IN '%s',
a.ci8 IN '%s',
a.ci9 IN '%s',
a.ci10 IN '%s'
'''
% tuple(join_list[0]),
tuple(join_list[1]),
tuple(join_list[2]),
tuple(join_list[3]),
tuple(join_list[4]),
tuple(join_list[5]),
tuple(join_list[6]),
tuple(join_list[7]),
tuple(join_list[8]),
tuple(join_list[9])
)



join_list = sel_func_city('victoria')
format_strings = ','.join(['%s'] * len(join_list[0]))
crsr.execute('''SELECT ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10
FROM post_city a
INNER JOIN name_post_city b
ON a.ci1 = b.ci1 AND
a.ci2 = b.ci2 AND
a.ci3 = b.ci3 AND
a.ci4 = b.ci4 AND
a.ci5 = b.ci5 AND
a.ci6 = b.ci6 AND
a.ci7 = b.ci7 AND
a.ci8 = b.ci8 AND
a.ci9 = b.ci9 AND
a.ci10 = b.ci10
WHERE
a.ci1 IN '%s',
a.ci2 IN '%s',
a.ci3 IN '%s',
a.ci4 IN '%s',
a.ci5 IN '%s',
a.ci6 IN '%s',
a.ci7 IN '%s',
a.ci8 IN '%s',
a.ci9 IN '%s',
a.ci10 IN '%s'
```

```
'''
% tuple(join_list[0]),
tuple(join_list[1]),
tuple(join_list[2]),
tuple(join_list[3]),
tuple(join_list[4]),
tuple(join_list[5]),
tuple(join_list[6]),
tuple(join_list[7]),
tuple(join_list[8]),
tuple(join_list[9])
)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-139-c6e6db5d0b0a> in <module>()
     26 a.ci10 IN '%s'
     27 '''
---> 28 % tuple(join_list[0]),
     29 tuple(join_list[1]),
     30 tuple(join_list[2]),

TypeError: not enough arguments for format string
```