import pandas as pd

import numpy as np

import nltk

df\_jobs = pd.read\_csv("Combined\_Jobs\_Final.csv")

df\_jobs.head(2)

df\_jobs.info()

df\_jobs.isnull().sum()

cols = ['Job.ID']+['Title']+['Position']+ ['Company']+['City']+['Employment.Type']+['Job.Description']

df\_jobs =df\_jobs[cols]

df\_jobs.columns = ['Job.ID', 'Title', 'Position', 'Company','City', 'Empl\_type','Job\_Description']

df\_jobs.head()

df\_jobs.isnull().sum()

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

x = df\_jobs.columns

y = df\_jobs.isnull().sum()

plt.figure(figsize=(10,6))

sns.set()

sns.barplot(x,y)

ax = plt.gca()

for p in ax.patches:

    height = p.get\_height()

    ax.text(p.get\_x(),

            height + 2,

            str(100\*round(int(height)/df\_jobs.shape[0], 3)) +"%",

            fontsize=10, ha='center', va='bottom')

ax.set\_xlabel("Columns")

ax.set\_ylabel("NA's")

plt.xticks(rotation=80)

plt.show()

df\_nan\_city = df\_jobs[pd.isnull(df\_jobs['City'])]

print(df\_nan\_city.shape)

df\_nan\_city.head()

df\_nan\_city.groupby(['Company'])['City'].count()

df\_jobs['Company'] = df\_jobs['Company'].replace(['Genesis Health Systems'], 'Genesis Health System')

df\_jobs.loc[df\_jobs.Company == 'CHI Payment Systems', 'City'] = 'Illinois'

df\_jobs.loc[df\_jobs.Company == 'Academic Year In America', 'City'] = 'Stamford'

df\_jobs.loc[df\_jobs.Company == 'CBS Healthcare Services and Staffing ', 'City'] = 'Urbandale'

df\_jobs.loc[df\_jobs.Company == 'Driveline Retail', 'City'] = 'Coppell'

df\_jobs.loc[df\_jobs.Company == 'Educational Testing Services', 'City'] = 'New Jersey'

df\_jobs.loc[df\_jobs.Company == 'Genesis Health System', 'City'] = 'Davennport'

df\_jobs.loc[df\_jobs.Company == 'Home Instead Senior Care', 'City'] = 'Nebraska'

df\_jobs.loc[df\_jobs.Company == 'St. Francis Hospital', 'City'] = 'New York'

df\_jobs.loc[df\_jobs.Company == 'Volvo Group', 'City'] = 'Washington'

df\_jobs.loc[df\_jobs.Company == 'CBS Healthcare Services and Staffing', 'City'] = 'Urbandale'

df\_jobs.isnull().sum()

df\_nan\_emp = df\_jobs[pd.isnull(df\_jobs['Empl\_type'])]

df\_nan\_emp.head()

df\_jobs['Empl\_type'].unique()

df\_jobs['Empl\_type']=df\_jobs['Empl\_type'].fillna('Full-Time/Part-Time')

df\_jobs.groupby(['Empl\_type'])['Company'].count()

df\_jobs.head()

df\_jobs["text"] = df\_jobs["Position"].map(str) + " " + df\_jobs["Company"] +" "+ df\_jobs["City"]+ " "+df\_jobs['Empl\_type']+" "+df\_jobs['Job\_Description'] +" "+df\_jobs['Title']

df\_jobs.head(2)

df\_all = df\_jobs[['Job.ID', 'text', 'Title']]

df\_all = df\_all.fillna(" ")

df\_all.head()

df\_all.shape

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

nltk.download('averaged\_perceptron\_tagger')

from nltk.corpus import stopwords

import re

import string

from nltk.stem import WordNetLemmatizer

from nltk import word\_tokenize

from nltk.corpus import stopwords

stop = stopwords.words('english')

stop\_words\_ = set(stopwords.words('english'))

wn = WordNetLemmatizer()

def black\_txt(token):

    return  token not in stop\_words\_ and token not in list(string.punctuation)  and len(token)>2

def clean\_txt(text):

  clean\_text = []

  clean\_text2 = []

  text = re.sub("'", "",text)

  text=re.sub("(\\d|\\W)+"," ",text)

  text = text.replace("nbsp", "")

  clean\_text = [ wn.lemmatize(word, pos="v") for word in word\_tokenize(text.lower()) if black\_txt(word)]

  clean\_text2 = [word for word in clean\_text if black\_txt(word)]

  return " ".join(clean\_text2)

df\_all['text'] = df\_all['text'].apply(clean\_txt)

df\_all.head()

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_jobid = tfidf\_vectorizer.fit\_transform((df\_all['text'])) #fitting and transforming the vector

tfidf\_jobid

df\_job\_view = pd.read\_csv("Job\_Views.csv")

df\_job\_view.head(2)

df\_job\_view = df\_job\_view[['Applicant.ID', 'Job.ID', 'Position', 'Company','City']]

df\_job\_view["select\_pos\_com\_city"] = df\_job\_view["Position"].map(str) + "  " + df\_job\_view["Company"] +"  "+ df\_job\_view["City"]

df\_job\_view['select\_pos\_com\_city'] = df\_job\_view['select\_pos\_com\_city'].map(str).apply(clean\_txt)

df\_job\_view['select\_pos\_com\_city'] = df\_job\_view['select\_pos\_com\_city'].str.lower()

df\_job\_view = df\_job\_view[['Applicant.ID','select\_pos\_com\_city']]

df\_job\_view.head()

df\_experience = pd.read\_csv("Experience.csv")

df\_experience.head(2)

from scipy.stats import pearsonr

from scipy.stats import spearmanr

df\_experience.corr(method='pearson')

df\_experience= df\_experience[['Applicant.ID','Position.Name']]

#cleaning the text

df\_experience['Position.Name'] = df\_experience['Position.Name'].map(str).apply(clean\_txt)

df\_experience.head()

df\_experience =  df\_experience.sort\_values(by='Applicant.ID')

df\_experience = df\_experience.fillna(" ")

df\_experience.head()

df\_experience = df\_experience.groupby('Applicant.ID', sort=False)['Position.Name'].apply(' '.join).reset\_index()

df\_experience.head(5)

df\_poi =  pd.read\_csv("Positions\_Of\_Interest.csv", sep=',')

df\_poi = df\_poi.sort\_values(by='Applicant.ID')

df\_poi.head()

df\_poi = df\_poi.drop('Updated.At', 1)

df\_poi = df\_poi.drop('Created.At', 1)

#cleaning the text

df\_poi['Position.Of.Interest']=df\_poi['Position.Of.Interest'].map(str).apply(clean\_txt)

df\_poi = df\_poi.fillna(" ")

df\_poi.head(10)

df\_poi = df\_poi.groupby('Applicant.ID', sort=True)['Position.Of.Interest'].apply(' '.join).reset\_index()

df\_poi.head()

df\_jobs\_exp = df\_job\_view.merge(df\_experience, left\_on='Applicant.ID', right\_on='Applicant.ID', how='outer')

df\_jobs\_exp = df\_jobs\_exp.fillna(' ')

df\_jobs\_exp = df\_jobs\_exp.sort\_values(by='Applicant.ID')

df\_jobs\_exp.head()

df\_jobs\_exp\_poi = df\_jobs\_exp.merge(df\_poi, left\_on='Applicant.ID', right\_on='Applicant.ID', how='outer')

df\_jobs\_exp\_poi = df\_jobs\_exp\_poi.fillna(' ')

df\_jobs\_exp\_poi = df\_jobs\_exp\_poi.sort\_values(by='Applicant.ID')

df\_jobs\_exp\_poi.head()

df\_jobs\_exp\_poi["text"] = df\_jobs\_exp\_poi["select\_pos\_com\_city"].map(str) + df\_jobs\_exp\_poi["Position.Name"] +" "+ df\_jobs\_exp\_poi["Position.Of.Interest"]

df\_jobs\_exp\_poi.head()

df\_final\_person= df\_jobs\_exp\_poi[['Applicant.ID','text']]

df\_final\_person.head()

df\_final\_person.columns = ['Applicant\_id','text']

df\_final\_person.head()

df\_final\_person['text'] = df\_final\_person['text'].apply(clean\_txt)

df\_final\_person.head()

u = 145

index = np.where(df\_final\_person['Applicant\_id'] == u)[0][0]

user\_q = df\_final\_person.iloc[[index]]

user\_q

user\_q.text

from sklearn.metrics.pairwise import cosine\_similarity

user\_tfidf = tfidf\_vectorizer.transform(user\_q['text'])

cos\_similarity\_tfidf = map(lambda x: cosine\_similarity(user\_tfidf, x),tfidf\_jobid)

output2 = list(cos\_similarity\_tfidf)

def get\_recommendation(top, df\_all, scores):

  recommendation = pd.DataFrame(columns = ['ApplicantID', 'JobID',  'title', 'score'])

  count = 0

  for i in top:

      recommendation.at[count, 'ApplicantID'] = u

      recommendation.at[count, 'JobID'] = df\_all['Job.ID'][i]

      recommendation.at[count, 'title'] = df\_all['Title'][i]

      recommendation.at[count, 'score'] =  scores[count]

      count += 1

  return recommendation

top = sorted(range(len(output2)), key=lambda i: output2[i], reverse=True)[:6]

list\_scores = [output2[i][0][0] for i in top]

tfcosine=get\_recommendation(top,df\_all, list\_scores)

from sklearn.feature\_extraction.text import CountVectorizer

count\_vectorizer = CountVectorizer()

count\_jobid = count\_vectorizer.fit\_transform((df\_all['text'])) #fitting and transforming the vector

count\_jobid

from sklearn.metrics.pairwise import cosine\_similarity

user\_count = count\_vectorizer.transform(user\_q['text'])

cos\_similarity\_countv = map(lambda x: cosine\_similarity(user\_count, x),count\_jobid)

output2 = list(cos\_similarity\_countv)

top = sorted(range(len(output2)), key=lambda i: output2[i], reverse=True)[:6]

list\_scores = [output2[i][0][0] for i in top]

get\_recommendation(top, df\_all, list\_scores)

import spacy

!python -m spacy download en\_core\_web\_lg

def calculateSimWithSpaCy(nlp, df, user\_text, n=6):

    # Calculate similarity using spaCy

    list\_sim =[]

    doc1 = nlp("u'" + user\_text + "'")

    for i in df.index:

      try:

            doc2 = list\_docs[i][0]

            score = doc1.similarity(doc2)

            list\_sim.append((doc1, doc2, list\_docs[i][1],score))

      except:

        continue

    return  list\_sim

from sklearn.neighbors import NearestNeighbors

n\_neighbors = 11

KNN = NearestNeighbors(n\_neighbors, p=2)

KNN.fit(tfidf\_jobid)

NNs = KNN.kneighbors(user\_tfidf, return\_distance=True)

top = NNs[1][0][1:]

index\_score = NNs[0][0][1:]

get\_recommendation(top, df\_all, index\_score)

from PIL import Image

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

bunch\_text = " ".join(text for text in df\_all.tail(10000).text)

stopwords = set(STOPWORDS)

wordcloud = WordCloud(stopwords=stopwords, background\_color="white", colormap= "magma").generate(bunch\_text)

plt.figure(figsize=[11,11])

plt.imshow(wordcloud, interpolation="sinc")

plt.axis("off")

plt.show()