JOB RECOMMENTATION SYSTEM

Code for importing libraries

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

import nltk

from nltk.corpus import stopwords

import re

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans

import json

from os import listdir

import glob

from scipy import spatial

import spacy
```

For fetching the details form the downloaded csv files

```
def cosine_similarity(arr1,arr2):

    ans=1- spatial.distance.cosine(arr1,arr2)

    if(np.isnan(ans)):

        return 0

    else:

        return ans
```

```
class job_postings:

    def __init__(self,link):

        self.df2=pd.read_csv(link)

        self.training_range=int(len(self.df2.loc[:,'uniq_id']))

    def check_threshold(threshold,ele):

        if(ele[0]!=threshold[0][0] and abs(ele[1]-threshold[0][1])<0.03):

            return True

        else:

            return False

    def categorize_jobs(self):

        nlp=spacy.load('en_core_web_lg')

        job_id=self.df2.loc[:,'uniq_id'].tolist()[:self.training_range]

        job_titles=self.df2.loc[:,'jobtitle'].tolist()[:self.training_range]

        job_descriptions=self.df2.loc[:,'jobdescription'].tolist()[:self.training_range]

        final_cat=pd.DataFrame(index=job_id)

        categories=['Network Engineer','Full stack','QA/Test Developer','Enterprise
application','DevOps','Mobile Developer','Back End','Database Administrator(DBA)','Front
End','Game developer','System Administrator','Data Scientist','Business analyst','Sales
professional','Product Manager','Information Security','Software Developer/Java
Developer','Web Developer','Cloud Computing']

        for category in categories:

            final_cat[category]=np.nan

        for job_t_d in list(zip(job_id,job_titles,job_descriptions)):

            id_job=job_t_d[0]

            job_i=job_t_d[1]

            job_d=job_t_d[2]
            job_title=nlp(job_i.lower())
            job_description=nlp(job_d.lower())
            match_cat_title=dict()
            match_cat_description=dict()
            for category in categories:
                word=nlp(category.lower())
                match_cat_title[category]=job_title.similarity(word)
                match_cat_description[category]=job_description.similarity(word)
            match_cat_title=sorted(match_cat_title.items(),key=lambda x:x[1],reverse=True)
            match_cat_description=sorted(match_cat_description.items(),key=lambda
x:x[1],reverse=True)

            a=match_cat_title[0]
            #print(a)
            match_cat_description=list(filter(lambda x:
self.check_threshold(match_cat_title,x),match_cat_description))
            if(len(match_cat_description)!=0):
                print(match_cat_description)
                print(id_job)
                final_cat.loc[id_job,a[0]]=1
                match_cat_description.extend([(match_cat_title[0][0],1)])
                sum_proportion=sum([x[1] for x in match_cat_description])
                for ele in match_cat_description:
                    final_cat.loc[id_job,ele[0]]=ele[1]/sum_proportion
            else:
                print(id_job)
                final_cat.loc[id_job,a[0]]=1
        return final_cat
    def clean_skills(self):
        extracted_skills=dict()
        job_skills=np.asarray(self.df2.loc[:,"skills"])
        for i in range(self.training_range):
            job_id=self.df2.iloc[i,-1]
            tokenizer=nltk.tokenize.RegexpTokenizer(r'\w+')
            if(pd.isnull(job_skills[i])):
                continue
            stopwords_list=stopwords.words("english")
            tokens=re.split("|".join([","," and","/"," AND"," or"," OR",";"]),job_skills[i])
            tokens=list(set(tokens))
            extracted_skills[job_id]=[]
            extracted_skills[job_id].extend(tokens)
        return extracted_skills
    def extract_skills(self,extracted_skills):
        df_languages=pd.read_excel('./data/job_profile/languages.xlsx')
        df_frameworks=pd.read_csv("./data/job_profile/frameworks.csv")
        df_database=pd.read_csv("./data/job_profile/database.csv")
        df_os=pd.read_csv("./data/job_profile/operating_systems.csv")
        df_plat=pd.read_csv("./data/job_profile/platforms.csv")
        frameworks=df_frameworks.iloc[:,1].tolist()
```

```
frameworks=[x.lower().strip() for x in frameworks]

languages=list(df_languages.iloc[:,0])

languages=[x.lower().strip() for x in languages]

databases=df_database.iloc[:,0].tolist()

databases=[x.lower().strip() for x in databases]

op_systems=df_os.iloc[:,0].tolist()

op_systems=[x.lower().strip() for x in op_systems]

platforms=df_plat.iloc[:,1].tolist()

platforms=[x.lower().strip() for x in platforms]

new_extracted=dict()

for ele in extracted_skills.keys():

    final_lang=''

    final_frame=''

    final_others=''

    final_database=''

    final_plat=''

    final_os=''

    for skill in extracted_skills[ele]:

        skill_base=skill.lower().strip()

        if(skill_base in languages):

            if(final_lang==''):

                final_lang=skill_base

            else:

                final_lang=final_lang+","+skill_base
```

```
elif(skill_base in frameworks):
    if(final_frame==''):
        final_frame=skill_base
    else:
        final_frame=final_frame+","+skill_base
elif(skill_base in databases):
    if(final_database==''):
        final_database=skill_base
    else:
        final_database=final_database+","+skill_base
elif(skill_base in op_systems):
    if(final_os==''):
        final_os=skill_base
    else:
        final_os=final_os+","+skill_base
elif(skill_base in platforms):
    if(final_plat==''):
        final_plat=skill_base
    else:
        final_plat=final_plat+","+skill_base
else:
    if(final_others==''):
        final_others=skill_base
    else:
```

```
                    final_others=final_others+","+skill_base

new_extracted[ele]=[final_lang,final_frame,final_database,final_os,final_plat,final_others]
        print((list(new_extracted.items()))[:100])
        for ele,describe in
list(zip(self.df2.loc[:,'uniq_id'],self.df2.loc[:,'jobdescription'].tolist()))[:self.training_range]:
            doc=nlp(describe)
            final_lang=''
            final_frame=''
            final_others=''
            final_database=''
            final_plat=''
            final_os=''
            for ent in doc.ents:
                word=ent.text
                word=word.lower().strip()
                if(word in languages and word not in final_lang and word not in
new_extracted[ele][0].split(",")):
                    if(final_lang==''):
                        final_lang=word
                    else:
                        final_lang=final_lang+","+word
                elif(word in frameworks and word not in final_frame and word not in
new_extracted[ele][1].split(",")):
                    if(final_frame==''):
                        final_frame=word
                    else:
                        final_frame=final_frame+","+word
                elif(word in databases and word not in final_database and word not in
new_extracted[ele][2].split(",")):
                    if(final_database==''):
                        final_database=word
                    else:
                        final_database=final_database+","+word
                elif(word in op_systems and word not in final_os and word not in
new_extracted[ele][3].split(",")):
                    if(final_os==''):
                        final_os=word
                    else:
                        final_os=final_os+","+word
                elif(word in platforms and word not in final_plat and word not in
new_extracted[ele][4].split(",")):
                    if(final_plat==''):
                        final_plat=word
                    else:
                        final_plat=final_plat+","+word
                else:
                    if(final_others==''):
                        final_others=word
                    else:
                        final_others=final_others+","+word
```

```
            if(final_lang!=''):
                new_extracted[ele][0]+=","+final_lang
            if(final_frame!=''):
                new_extracted[ele][1]+=","+final_frame
            if(final_database!=''):
                new_extracted[ele][2]+=","+final_database
            if(final_os!=''):
                new_extracted[ele][3]+=","+final_os
            if(final_plat!=''):
                new_extracted[ele][4]+=","+final_plat
            if(final_others!=''):
                new_extracted[ele][5]+=","+final_others

extracted_skills_df=pd.DataFrame.from_dict(new_extracted,orient='index',columns=['Languag
e','Framework','Database','OS','Platform','Others'])
        return extracted_skills_df
    def create_job_profile(self,extracted_skills_df,domain_df):
        job_id=extracted_skills_df.index.tolist()
        languages_df=pd.DataFrame(index=job_id)
        platforms_df=pd.DataFrame(index=job_id)
        frameworks_df=pd.DataFrame(index=job_id)
        databases_df=pd.DataFrame(index=job_id)


        for job,lang,frame,plat,datab in
list(zip(job_id,extracted_skills_df.loc[:,'Language'].tolist(),extracted_skills_df.loc[:,'Framework'].
tolist(),extracted_skills_df.loc[:,'Platform'].tolist(),extracted_skills_df.loc[:,'Database'].tolist())):
```

```
l=lang.split(",")
if(lang!=np.nan or lang!=''):
    for ele in l:
        if(ele==''):
            continue
        if(ele not in languages_df.columns):
            languages_df[ele]=np.nan
        languages_df.loc[job,ele]=1


l=frame.split(",")
if(frame!=np.nan or frame!=''):
    for ele in l:
        if(ele==''):
            continue
        if(ele not in frameworks_df.columns):
            frameworks_df[ele]=np.nan
        frameworks_df.loc[job,ele]=1


l=plat.split(",")
if(plat!=np.nan or plat!=''):
    for ele in l:
        if(ele==''):
                continue
            if(ele not in platforms_df.columns):
                platforms_df[ele]=np.nan
            platforms_df.loc[job,ele]=1


    l=datab.split(",")
    if(datab!=np.nan or datab!=''):
        for ele in l:
            if(ele==''):
                continue
            if(ele not in databases_df.columns):

                databases_df[ele]=np.nan
            databases_df.loc[job,ele]=1
  languages_df=languages_df.reindex_axis(sorted(languages_df.columns), axis=1)
  frameworks_df=frameworks_df.reindex_axis(sorted(frameworks_df.columns), axis=1)
  platforms_df=platforms_df.reindex_axis(sorted(platforms_df.columns), axis=1)
  databases_df=databases_df.reindex_axis(sorted(databases_df.columns), axis=1)
  domain_df=domain_df.reindex_axis(sorted(domain_df.columns), axis=1)

languages_df.index.name=frameworks_df.index.name=platforms_df.index.name=databases_df
.index.name=domain_df.index.name='uniq_id'
  languages_df.to_csv("./data/job_profile/languages_job_profile.csv")
```

```
    frameworks_df.to_csv("./data/job_profile/frameworks_job_profile.csv")
    platforms_df.to_csv("./data/job_profile/platforms_job_profile.csv")
    databases_df.to_csv("./data/job_profile/databases_job_profile.csv")
    domain_df.to_csv("./data/job_profile/domain_job_profile.csv")
    print(languages_df.columns)

def clean_common_profile(self,df_user,df_job,flag):
    #Shift .net from languages to frameworks
    if(flag=='Language'):
        print(df_job.columns.tolist())
        #bash and bash/shell
        count=0
        for ele in df_user.loc[:,'bash/shell']:
            if(ele==1.0):
                df_user.ix[count,'bash']=1.0
            count=count+1
        df_user=df_user.drop('bash/shell',axis=1)
        count=0
        for ele in df_job.loc[:,'bash/shell']:
            if(ele==1.0):
                df_job.ix[count,'bash']=1.0
            count=count+1
        df_job=df_job.drop('bash/shell',axis=1)
```

```
if(flag=='Framework'):
    print(df_user.columns.tolist())
    count=0
    for ele in df_user.loc[:,'nodejs']:
        if(ele==1.0):
            df_user.ix[count,'node.js']=1.0
        count=count+1
    df_user=df_user.drop('nodejs',axis=1)
    count=0
    for ele in df_job.loc[:,'nodejs']:
        if(ele==1.0):
            df_job.ix[count,'node.js']=1.0
        count=count+1
    df_job=df_job.drop('nodejs',axis=1)

    count=0
    for ele in df_user.loc[:,'angularjs']:
        if(ele==1.0):
            df_user.ix[count,'angular']=1.0
        count=count+1
    df_user=df_user.drop('angularjs',axis=1)
    count=0
    for ele in df_job.loc[:,'angularjs']:
        if(ele==1.0):
                df_job.ix[count,'angular']=1.0
            count=count+1
        df_job=df_job.drop('angularjs',axis=1)

    if(flag=='Platform'):
        print(df_user.columns.tolist())
    if(flag=='Database'):
        print(df_user.columns.tolist())
        count=0
        for ele in df_user.loc[:,'microsoft sql server']:
            if(ele==1.0):
                df_user.ix[count,'sql server']=1.0
            count=count+1
        df_user=df_user.drop('microsoft sql server',axis=1)
        count=0
        for ele in df_job.loc[:,'microsoft sql server']:
            if(ele==1.0):
                df_job.ix[count,'sql server']=1.0
            count=count+1
        df_job=df_job.drop('microsoft sql server',axis=1)
    return df_user,df_job


def create_common_profile(self,job_profile_path,user_profile_path,output_path,flag=0):
    if(flag==0):

        userprofile=pd.read_csv(user_profile_path+"DevType.csv",index_col='Respondent')

jobprofile=pd.read_csv(job_profile_path+"domain_job_profile.csv",index_col='Unnamed: 0')
        print("Read from file")
        print(jobprofile.index)
        userprofile.drop('Unnamed: 0', axis=1, inplace=True)
        jobprofile.drop('uniq_id', axis=1, inplace=True)
        jobprofile.index.name='uniq_id'
        print("index 2in domain")
        print(jobprofile.index)
        userprofile.rename(columns={'Product manager':'Product Manager','Back-end
developer':'Back End','C-suite executive (CEO, CTO, etc.)':'C-suite executive','Data scientist or
machine learning specialist':'Data Scientist','Database administrator':'Database
Administrator(DBA)','Mobile developer':'Mobile Developer','Desktop or enterprise applications
developer':'Enterprise application','DevOps specialist':'DevOps','Front-end developer':'Front
End','Full-stack developer':'Full stack','Marketing or sales professional':'Sales professional','QA
or test developer':'QA/Test Developer','System administrator':'System Administrator','Game or
graphics developer':'Game developer'},inplace=True)
        jobprofile.rename(columns={'Business analyst':'Data or business analyst'},inplace=True)
        print(userprofile.columns)
        print(jobprofile.columns)
        print("index in domain")
        print(jobprofile.index)
        a=list(set(userprofile.columns)-set(jobprofile.columns))
        print(a)
        for i in a:
           if(i!='Respondent'):
               jobprofile[i]=0
        b=list(set(jobprofile.columns)-set(userprofile.columns))
        print(b)
        for i in b:
           if(i!='uniq_id'):
               userprofile[i]=0
        userprofile=userprofile[sorted(userprofile.columns.tolist())]
        jobprofile=jobprofile[sorted(jobprofile.columns.tolist())]

        print(userprofile.columns==jobprofile.columns)


        print(userprofile.columns)
        print(jobprofile.columns)
        userprofile=userprofile[userprofile.columns.tolist()]
        jobprofile=jobprofile[jobprofile.columns.tolist()]
        userprofile.to_csv(output_path+"domain_user_profile.csv")
        jobprofile.to_csv(output_path+"domain_job_profile.csv")




df_user=pd.read_csv(user_profile_path+"LanguageWorkedWith.csv",index_col='Respondent')
        df_job=pd.read_csv(job_profile_path+"languages_job_profile.csv",index_col=0)
```

```
df_job.index.name='uniq_id'

print("index is")

print(df_job.index)

print(df_user.columns)

print(df_job.columns)

df_user.drop('Unnamed: 0', axis=1, inplace=True)


df_job.rename(columns={'visual basic .net':'vb.net'},inplace=True)

df_user.columns=list(map(lambda x:x.lower(),df_user.columns))

df_job.columns=list(map(lambda x:x.lower(),df_job.columns))

columns_to_add=[]

a=list(set(df_user.columns)-(set(df_job.columns)))

print(a)

for i in a:

    if(i!='Respondent'):

        df_job[i]=0

b=list(set(df_job.columns)-set(df_user.columns))

print(b)

for i in b:

    if(i!='uniq_id'):

        df_user[i]=0

print(df_job.index)

df_user=df_user[sorted(df_user.columns.tolist())]

df_job=df_job[sorted(df_job.columns.tolist())]
```

```
        print("index 2")

        print(df_job.index)

        print(len(set(df_user.columns).intersection(df_job.columns)),len(df_user.columns))

        df_user,df_job=self.clean_common_profile(df_user,df_job,'Language')

        print("language is")

        print(df_job.index[0])

        print(df_job.loc[df_job.index[0],:])

        df_user.to_csv(output_path+"languages_profile_user.csv")

        df_job.to_csv(output_path+"languages_profile_job.csv")



df_user=pd.read_csv(user_profile_path+"FrameworkWorkedWith.csv",index_col='Respondent'
)

        df_job=pd.read_csv(job_profile_path+"frameworks_job_profile.csv",index_col=0)

        df_job.index.name='uniq_id'

        print(df_user.columns)

        print(df_job.columns)

        df_user.drop('Unnamed: 0', axis=1, inplace=True)

        df_user.columns=list(map(lambda x:x.lower(),df_user.columns))

        df_job.columns=list(map(lambda x:x.lower(),df_job.columns))


        a=list(set(df_user.columns)-(set(df_job.columns)))

        print(a)

        for i in a:

          if(i!='Respondent'):
```

```
        df_job[i]=0
    b=list(set(df_job.columns)-set(df_user.columns))
    print(b)
    for i in b:
        if(i!='uniq_id'):
            df_user[i]=0
    df_user=df_user[sorted(df_user.columns.tolist())]
    df_job=df_job[sorted(df_job.columns.tolist())]

    print(len(set(df_user.c))
    for i in b:
        if(i!='uniq_id'):
            df_user[i]=0
    df_user=df_user[sorted(df_user.columns.tolist())]
    df_job=df_job[sorted(df_job.columns.tolist())]

    print(len(set(df_user.columns).intersection(df_job.columns)),len(df_user.columns))
    df_user,df_job=self.clean_common_profile(df_user,df_job,'Platform')
    df_user.to_csv(output_path+"platforms_profile_user.csv")
    df_job.to_csv(output_path+"platforms_profile_job.csv")


df_user=pd.read_csv(user_profile_path+"DatabaseWorkedWith.csv",index_col='Respondent')
    df_job=pd.read_csv(job_profile_path+"databases_job_profile.csv",index_col=0)
```

```
df_job.index.name='uniq_id'

print(df_user.columns)

print(df_job.columns)

df_user.drop('Unnamed: 0', axis=1, inplace=True)

df_user.columns=list(map(lambda x:x.lower(),df_user.columns))

df_job.columns=list(map(lambda x:x.lower(),df_job.columns))


a=list(set(df_user.columns)-(set(df_job.columns)))

print(a)

for i in a:

   if(i!='Respondent'):

      df_job[i]=0

b=list(set(df_job.columns)-set(df_user.columns))

print(b)

for i in b:

   if(i!='uniq_id'):

      df_user[i]=0

df_user=df_user[sorted(df_user.columns.tolist())]

df_job=df_job[sorted(df_job.columns.tolist())]


print(len(set(df_user.columns).intersection(df_job.columns)),len(df_user.columns))

df_user,df_job=self.clean_common_profile(df_user,df_job,'Database')

df_user.to_csv(output_path+"databases_profile_user.csv")

df_job.to_csv(output_path+"databases_profile_job.csv")
```

```
def match_profile(self,input_path,user_id,flag=0):
    df=pd.read_csv(input_path+"domain_user_profile.csv",index_col='Respondent')
    matches=dict()
    if(flag==0):
        if(user_id in df.index):
            userdomain=df.loc[user_id,:]
            df=pd.read_csv(input_path+"languages_profile_user.csv",index_col='Respondent')
            userlanguages=df.loc[user_id,:]

            df=pd.read_csv(input_path+"frameworks_profile_user.csv",index_col='Respondent')
            userframeworks=df.loc[user_id,:]

            df=pd.read_csv(input_path+"platforms_profile_user.csv",index_col='Respondent')
            userplatforms=df.loc[user_id,:]

            df=pd.read_csv(input_path+"databases_profile_user.csv",index_col='Respondent')
            userdatabases=df.loc[user_id,:]

            userdomain=np.asarray(userdomain.fillna(0))
            userlanguages=np.asarray(userlanguages.fillna(0))
            userframeworks=np.asarray(userframeworks.fillna(0))
            userplatforms=np.asarray(userplatforms.fillna(0))
            userdatabases=np.asarray(userdatabases.fillna(0))
        else:
            print("error! user id not in Dataset")
    else:


        print("New user!Enter details..")
        name=input("Enter full name")
        skills=input("Enter skills(comma separated). These are programming languages,
frameworks,platforms or databases you have experience with").split(",")
        domains=''
        flag=1
        while(1):
            print("Enter domain(s) of interest separated by commas(Names are case sensitive).
Should be one of the following:")
            for i in df.columns:
                print(i,end=",")
            domains=input().split(",")
            for domain in domains:
                if(domain not in df.columns):
                    flag=0
                    break
            if(flag==1):
                break
            else:
                print("Please enter valid domain")
        skills=list(map(lambda x:x.lower(),skills))
```

```
userdomain=pd.DataFrame(columns=df.columns)

dictionary=dict()

for domain in domains:

    dictionary[domain]=1.0

userdomain=userdomain.append(dictionary,ignore_index=True)



df=pd.read_csv(input_path+"languages_profile_user.csv",index_col='Respondent')

userlanguages=pd.DataFrame(columns=df.columns)

dictionary=dict()

for skill in skills:

    if(skill in df.columns):

        dictionary[skill]=1.0

userlanguages=userlanguages.append(dictionary,ignore_index=True)


df=pd.read_csv(input_path+"frameworks_profile_user.csv",index_col='Respondent')

userframeworks=pd.DataFrame(columns=df.columns)

dictionary=dict()

for skill in skills:

    if(skill in df.columns):

        dictionary[skill]=1.0

userframeworks=userframeworks.append(dictionary,ignore_index=True)
```

```
    df=pd.read_csv(input_path+"platforms_profile_user.csv",index_col='Respondent')
    userplatforms=pd.DataFrame(columns=df.columns)
    dictionary=dict()
    for skill in skills:
        if(skill in df.columns):
            dictionary[skill]=1.0
    userplatforms=userplatforms.append(dictionary,ignore_index=True)

    df=pd.read_csv(input_path+"databases_profile_user.csv",index_col='Respondent')
    userdatabases=pd.DataFrame(columns=df.columns)
    dictionary=dict()
    for skill in skills:
        if(skill in df.columns):
            dictionary[skill]=1.0
    userdatabases=userdatabases.append(dictionary,ignore_index=True)
    userdomain=np.asarray(userdomain.iloc[0,:].fillna(0))
    userlanguages=np.asarray(userlanguages.iloc[0,:].fillna(0))
    userframeworks=np.asarray(userframeworks.iloc[0,:].fillna(0))
    userplatforms=np.asarray(userplatforms.iloc[0,:].fillna(0))
    userdatabases=np.asarray(userdatabases.iloc[0,:].fillna(0))

jobdomain=pd.read_csv(input_path+"domain_job_profile.csv",index_col='uniq_id')
joblanguages=pd.read_csv(input_path+'languages_profile_job.csv',index_col='uniq_id')
jobframeworks=pd.read_csv(input_path+'frameworks_profile_job.csv',index_col='uniq_id')
```

```
        jobplatforms=pd.read_csv(input_path+'platforms_profile_job.csv',index_col='uniq_id')

        jobdatabases=pd.read_csv(input_path+'databases_profile_job.csv',index_col='uniq_id')

        for i in jobdomain.index:

            domain=jobdomain.loc[i,:].fillna(0)

            language=joblanguages.loc[i,:].fillna(0)

            framework=jobframeworks.loc[i,:].fillna(0)

            platform=jobplatforms.loc[i,:].fillna(0)

            database=jobdatabases.loc[i,:].fillna(0)

            job_id=str(i)

            domain=np.asarray(domain)

            language=np.asarray(language)

            framework=np.asarray(framework)

            platform=np.asarray(platform)

            database=np.asarray(database)

score=(0.7*cosine_similarity(domain,userdomain))+(0.3*(cosine_similarity(language,userlangu
ages)+cosine_similarity(framework,userframeworks)+cosine_similarity(platform,userplatforms)
+cosine_similarity(database,userdatabases)))

            matches[job_id]=score

score=(0.7*cosine_similarity(domain,userdomain))+(0.3*(cosine_similarity(language,userlangu
ages)+cosine_similarity(framework,userframeworks)+cosine_similarity(platform,userplatforms)
+cosine_similarity(database,userdatabases)))

            self.job_domain=domain

            self.job_language=language

            self.job_framework=framework
            self.job_platform=platform
            self.job_database=database

            self.user_domain=userdomain
            self.user_language=userlanguages
            self.user_framework=userframeworks
            self.user_platform=userplatforms
            self.user_database=userdatabases
        matches=sorted(matches.items(),key=lambda x:x[1],reverse=True)

        recommendations=matches[:10]
        rows=pd.DataFrame(columns=self.df2.columns)
        count=0
        for i in recommendations:
            row=self.df2[self.df2['uniq_id']==i[0]]
            rows=rows.append(row.iloc[0])
            count=count+1
        return rows



obj=job_postings("./data/dice_com-job_us_sample.csv");
```

IDENTIFYING VARIOUS CATEGORIES IN JOB POSTING CODE

```
df2=pd.read_csv("../input/us-technology-jobs-on-dicecom/dice_com-job_us_sample.csv")
print(df2.head())
```

```
jobs=[]

for job_title  in df2.jobtitle:

    if(job_title.lower() not in jobs):

        jobs.append(job_title)

job_skills=np.asarray(df2.loc[:,"skills"])

print(len(job_description[0:5]))


def remove_whitespace_entities(doc):

    doc.ents=[x for x in doc.ents if not (x.text.isspace())]

    return doc

extracted_skills=dict()

training_range=int(0.7*len(job_skills))

for i in range(training_range):

    job_id=df2.iloc[i,-1]

    tokenizer=nltk.tokenize.RegexpTokenizer(r'\w+')

    if(pd.isnull(job_skills[i])):

        continue

    stopwords_list=stopwords.words("english")

    tokens=re.split("|".join([","," and","/"," AND"," or"," OR",";"]),job_skills[i])

    tokens=list(set(tokens))

    extracted_skills[job_id]=[]

    extracted_skills[job_id].extend(tokens)

print(extracted_skills)
```

Applying TF-IDF ON DATASETS

```
count=0
docs=[]
for i in range(len(job_description[:100])):
    if(job_description[i]==np.nan):
        continue
    doc=[x for x in job_description[i].split(" ") if x not in stopwords_list]
    docs.append(" ".join(doc))
print(len(docs))
vectorizer=TfidfVectorizer(ngram_range=(1,2),max_df=0.6,max_features=50)
response=vectorizer.fit_transform(docs)
name_to_index=vectorizer.get_feature_names()
response=response.toarray()
scores=pd.DataFrame(data=response[:,:],index=range(len(response)),columns=name_to_index)
print(scores)
max_col_scores={}
for col in range(len(scores.iloc[0,:])):
    col_score=sum(scores.iloc[:,col])
    max_col_scores[name_to_index[col]]=col_score
max_col_scores=sorted(max_col_scores.items(),reverse=True,key=lambda x:x[1])[:50]
print(max_col_scores)
```

TO SEPARATE THE DOMAIN FIELDS IN DATASET

```
def cluster_job_titles():
    job_titles=df2.loc[:,'jobtitle'].tolist()
    #Tokenization
    docs=[]
    for i in range(len(job_titles[:training_range])):
        if(job_titles[i]==np.nan):
            continue
        doc=[x for x in job_description[i].split(" ") if x not in stopwords_list]
        docs.append(" ".join(doc))
    print(len(docs))
    vectorizer=TfidfVectorizer(ngram_range=(1,2),max_df=1.0,max_features=50)
    response=vectorizer.fit_transform(docs)
    model=KMeans(n_clusters=10,init='k-means++')
    model.fit(response)
    labels=model.labels_
    return labels
```

FOR EXTRACTING YEARS OF EXPERIENCE

```
nlp=spacy.load('en_core_web_lg')
job_description=df2.loc[:,'jobdescription'].tolist()
id_job=df2.loc[:,'uniq_id'].tolist()
experience_regex=['\d+ years \w+ $\.',r'\d+ experience']
matches=dict()
entities=dict()
for job_id,description in list(zip(id_job,job_description))[:10]:
    l=re.findall(r"[^.]*experience[^.]*\.",description)
    matches[job_id]=l
    for string in matches[job_id]:

        print(string)

        doc=nlp(string)


        for token in doc:

            print(token.text,token.dep_,token.head.text)

print(matches)
```

LINKEDIN COLLAB

```
files=glob.glob("../scraped profiles/*.json")

for file in files[:1]:

    f=open("../scraped profiles/"+file)

    data=json.load(f)

    print(data[0:2])
```

FOR SPLITING THE DATA FOR TEST

```
import pandas as pd

import numpy as np

df=pd.read_csv("user_profile.csv")

df.shape[1]

df.count(axis=1)

df1=df.dropna(thresh=6)

df1.shape[0]

train=df1.sample(frac=0.80)

test=pd.concat([df1,train]).drop_duplicates(keep=False)
```

```
print(train.shape[0])

test.shape[0]

df1.shape[0]

test.to_csv("test_user.csv")

train.to_csv("train_user.csv")
```

FOR ANALYSING USER PROFILE

```
import pandas as pd

import numpy as np

from sklearn import datasets, linear_model

from sklearn.model_selection import train_test_split

from matplotlib import pyplot as plt

df1=pd.read_csv("DatabaseWorkedwith.csv")

df2=pd.read_csv("DevType.csv")

df3=pd.read_csv("FrameworkWorkedwith.csv")

df4=pd.read_csv("LanguageWorkedwith.csv")

df5=pd.read_csv("Operating_Systems.csv")

df6=pd.read_csv("PlatformWorkedwith.csv")

d=[df3,df4,df5,df6];

df=pd.DataFrame();

def merge_datasets():

    df=pd.merge(df2,df1);

    for i in range(len(d)):

        df=pd.merge(df,d[i]);

    return df;
```

```
ds=merge_datasets();

print(ds.head())

print(ds.columns)

ds.to_csv("user_profile.csv")
```

FOR JOB POSTING PREPROCESSING

```
df2=pd.read_csv("../dice_com-job_us_sample.csv")

print(df2.head())

jobs=[]

for job_title  in df2.jobtitle:

   if(job_title.lower() not in jobs):

      jobs.append(job_title)

job_description=np.asarray(df2.loc[:,"jobdescription"])

print(len(job_description[0:5]))


def remove_whitespace_entities(doc):

   doc.ents=[x for x in doc.ents if not (x.text.isspace())]

   return doc

df_languages=pd.read_excel('./data/job_profile/languages.xlsx')

df_frameworks=pd.read_excel("./data/job_profile/frameworks.xlsx",header=None,error_bad_li
nes=False,delim_whitespace=True)

experience_regex=['\d+ years','\d+ experience','']

frame=[str(x).split(",")[0] for x in df_frameworks.iloc[:,0]]

print(len(df_frameworks.columns))
```

```
dictionary=list(df_languages.iloc[:,0])

dictionary.extend(frame)

print(dictionary)

dictionary=[x.lower() for x in dictionary]

extracted_jobs=dict()

for i in range(len(job_description[:10000])):

    job_id=df2.iloc[i,-1]

    job=df2.iloc[i,3]

    flag=0

    for word in job.split(" "):

        word=word.lower()

        if word in dictionary:

            flag=1

            if job_id not in extracted_jobs.keys():

                extracted_jobs[job_id]=[]

            if word not in extracted_jobs[job_id]:

                extracted_jobs[job_id].append(word)

    if(flag==0):

        print(job_id)

print(extracted_jobs)

print(len(extracted_jobs))
```

FOR ERROR VALIDATION

```
import pickle as pkl

with open('1000centroids.pkl','rb') as f:
    lis=pkl.load( f)

print(lis)

df2=pd.read_csv("databases_profile_user.csv")

df1=pd.read_csv("domain_user_profile.csv")

df3=pd.read_csv("frameworks_profile_user.csv")

df4=pd.read_csv("languages_profile_user.csv")

df5=pd.read_csv("platforms_profile_user.csv")





df1 = df1.replace(np.nan, 0, regex=True)

df2 = df2.replace(np.nan, 0, regex=True)

df3 = df3.replace(np.nan, 0, regex=True)

df4 = df4.replace(np.nan, 0, regex=True)

df5 = df5.replace(np.nan, 0, regex=True)

print("fbv");

d=[df3,df4,df5];

df=pd.DataFrame();

def merge_datasets():

    df=pd.merge(df1,df2,on='Respondent');

    for i in range(len(d)):

        df=pd.merge(df,d[i],on='Respondent');

    return df;
```

```
du=merge_datasets();

print(du.head())

df2=pd.read_csv("databases_profile_job.csv")

df1=pd.read_csv("domain_job_profile.csv")

df3=pd.read_csv("frameworks_profile_job.csv")

df4=pd.read_csv("languages_profile_job.csv")

df5=pd.read_csv("platforms_profile_job.csv")


df1 = df1.replace(np.nan, 0, regex=True)

df2 = df2.replace(np.nan, 0, regex=True)

df3 = df3.replace(np.nan, 0, regex=True)

df4 = df4.replace(np.nan, 0, regex=True)

df5 = df5.replace(np.nan, 0, regex=True)



d=[df3,df4,df5];

df=pd.DataFrame();

def merge_datasets1():

   df=pd.merge(df2,df1,on='uniq_id');

   for i in range(len(d)):

       df=pd.merge(df,d[i],on='uniq_id');

   return df;

dj=merge_datasets1();

print(dj.head())
```

```
du=du.drop(['Respondent'],axis=1)

print(du.head())



dj=dj.drop(['uniq_id'],axis=1)

print(dj.head())


kmeans1 = KMeans(n_clusters=50);

time_s=time.time()

kmeans1.fit(du);

time_e=time.time()

print(time_e-time_s);


import pickle as pkl

print(kmeans1.cluster_centers_)

arr= kmeans1.cluster_centers_

with open('userclustercentres.pkl','wb') as f:

    pkl.dump(arr, f)


time_s=time.time()

kmeans2=KMeans(n_clusters=50);

kmeans2.fit(dj);

time_e=time.time()

print(time_e-time_s);
```

```
import pickle as pkl

print(kmeans2.cluster_centers_)

arr1= kmeans2.cluster_centers_

with open('jobclustercentres.pkl','wb') as f:

    pkl.dump(arr1, f)




np.seterr(divide='ignore', invalid='ignore')

def cosine_similarity(arr1,arr2):

    ans=1-spatial.distance.cosine(arr1,arr2)

    if(np.isnan(ans)):

        return 0

    else:

        return ans

v=0

m=0;

lis=[];

nrows1=du.shape[0];

for i in range(len(kmeans1.cluster_centers_)):

    c=[]

    for j in range(len(kmeans2.cluster_centers_));

            cos=cosine_similarity(i,j);

            if(cos>m):

                m=cos;
            v=kmeans2.cluster_centers_[j];


   lis.append(v);


print("done")

print(lis)

with open('centriodsimilarity.pkl','wb') as f:

   pkl.dump(lis, f);
```