**Scrapping Code**

**#Importing the python packages**

import requests

import csv

from datetime import datetime

from bs4 import BeautifulSoup

for i in range(1, 10):

page = requests.get("https://www.monsterindia.com/jobs-in-chennai.html-%s" %i)

soup = BeautifulSoup(page.content, 'html.parser')

data = soup.find\_all("div",{"class":"jobwrap"})

item = data[0]

**#Creating the csv file in write mode**

filename="jobs.csv"

f=open(filename,"w")

headers="Job\_title,Company\n"

f.write(headers)

**#Extracting raw data**

for item in data:

job\_title=item.find\_all("span",{"class":"title\_in"})

job\_company=item.find\_all("div",{"class" : "jtxt orange"})

job\_loc=item.find\_all("div",{"class":"jtxt jico ico1"})

job\_exp=item.find\_all("div",{"class":"jtxt jico ico2"})

**#Cleaning the data**

title=job\_title[0].text.strip()

industry="HR"

company=job\_company[0].text

loc=job\_loc[0].text

exp=job\_exp[0].text

print(exp)

f.write(title.replace(",","|") + ","+industry.replace(",","|")+"," + company.replace(",","|") + ","+loc.replace(",","|")+ ","+exp+"\n")

f.close()

**Content Based Recommendation**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

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

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("D/data/job\_recommend.csv")

self.training\_range=int(len(self.df2.loc[:,'uniq\_id']))

def check\_threshold(self,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):

# #Predefined categories

#Compare similarities of word embeddings

nlp=spacy.load('en\_core\_web\_sm')

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','Application Development','Big Data','Data Analyst','Software Developer','DevOps','Software Testing','Front End','Back End','Full Stack','Web Development','Information Security','Mobile developer','System Administrator','Business Analyst','Manager','Cloud']

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=str(job\_t\_d[1])

job\_d=str(job\_t\_d[2])

#if type(job\_i) is str:

job\_title=nlp(job\_i.lower())

#job\_title=nlp(job\_title)

#if type(job\_d) is str:

job\_description=nlp(job\_d.lower())

#job\_description=nlp(job\_description)

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 represents max

#if(match\_cat\_title[0][1]>0.5 or match\_cat\_description[0][1]>0.5):

a = match\_cat\_title[0]

#print(a)

cat\_description= lambda x: self.check\_threshold(match\_cat\_title,x)

match\_cat\_description=list(filter(cat\_description,match\_cat\_description))

l=len(match\_cat\_description)

if(l!=0):

#print(match\_cat\_description)

#print(id\_job)

#b=match\_cat\_description[0]

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):

#print(i)

#Method 1: Manual pre-processing

job\_id=self.df2.iloc[i,-1]

#Method 2:Using NLTK

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

#print(job\_skills[i])

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\_csv("D/data/languages.csv")

df\_frameworks=pd.read\_csv("D/data/frameworks.csv")

df\_database=pd.read\_csv("D/data/database.csv")

df\_os=pd.read\_csv("D/data/operating\_systems.csv")

df\_plat=pd.read\_csv("D/data/platforms.csv")

frameworks=df\_frameworks.iloc[:,1].tolist()

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

#frameworks=[str(x).split(",")[0] for x in df\_frameworks.iloc[:,1]]

languages=list(df\_languages.iloc[:,0])

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

#frameworks=[x.lower().strip().split('\t')[0] for x in frameworks]

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()

#print(platforms)

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

#print(frameworks)

new\_extracted=dict()

for ele in extracted\_skills.keys():

final\_lang=''

final\_frame=''

final\_others=''

final\_database=''

final\_plat=''

final\_os=''

#print(extracted\_skills[ele])

for skill in extracted\_skills[ele]:

skill\_base=skill.lower().strip()

#print(skill\_base)

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

#new\_extracted[ele]=[final\_lang,final\_frame,final\_database,final\_os,final\_plat,final\_others]

extracted\_skills\_df=pd.DataFrame.from\_dict(new\_extracted,orient='index',columns=['Language','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())):

#Languages

l=lang.split(",")

if(lang!=np.nan or lang!=''):

for ele in l:

if(ele==''):

continue

if(ele not in languages\_df.columns):

#languages.append(ele)

languages\_df[ele]=np.nan

languages\_df.loc[job,ele]=1

#Frameworks

l=frame.split(",")

if(frame!=np.nan or frame!=''):

for ele in l:

if(ele==''):

continue

if(ele not in frameworks\_df.columns):

#languages.append(ele)

frameworks\_df[ele]=np.nan

frameworks\_df.loc[job,ele]=1

#Platforms

l=plat.split(",")

if(plat!=np.nan or plat!=''):

for ele in l:

if(ele==''):

continue

if(ele not in platforms\_df.columns):

#languages.append(ele)

platforms\_df[ele]=np.nan

platforms\_df.loc[job,ele]=1

#Databases

l=datab.split(",")

if(datab!=np.nan or datab!=''):

for ele in l:

if(ele==''):

continue

if(ele not in databases\_df.columns):

#languages.append(ele)

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("D/data/languages\_job\_profile.csv")

frameworks\_df.to\_csv("D/data/frameworks\_job\_profile.csv")

platforms\_df.to\_csv("D/data/platforms\_job\_profile.csv")

databases\_df.to\_csv("D/data/databases\_job\_profile.csv")

domain\_df.to\_csv("D/data/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

#Input is two dataframes

def create\_common\_profile(self,flag=0):

if(flag==0):

#Domain

userprofile=pd.read\_csv("D/data/DevType.csv",index\_col='Respondent')

jobprofile=pd.read\_csv("D/data/domain\_job\_profile.csv",index\_col=False)

#print("Read from file")

print(jobprofile.index)

#jobprofile=jobprofile.reset\_index()

#userprofile=userprofile.reset\_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)

#print(jobprofile.loc[:,'uniq\_id'])

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)

#Present in userprofile but not in jobprofile

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.set\_index('Respondent')

#jobprofile=jobprofile.set\_index('uniq\_id')

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

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

#Exclude

#print(userprofile.columns==jobprofile.columns)

#print(userprofile.columns)

#print(jobprofile.columns)

userprofile=userprofile[userprofile.columns.tolist()]

jobprofile=jobprofile[jobprofile.columns.tolist()]

userprofile.to\_csv("D/data/domain\_user\_profile.csv")

jobprofile.to\_csv("D/data/domain\_job\_profile.csv")

#Languages

df\_user=pd.read\_csv("D/data/LanguageWorkedWith.csv",index\_col='Respondent')

df\_job=pd.read\_csv("D/data/languages\_job\_profile.csv",index\_col=False)

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.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())]

#df\_user=userprofile.reindex\_axis(sorted(df\_user.columns), axis=1)

#df\_job=jobprofile.reindex\_axis(sorted(df\_job.columns), axis=1)

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("D/data/languages\_profile\_user.csv")

df\_job.to\_csv("D/data/languages\_profile\_job.csv")

#Frameworks

df\_user=pd.read\_csv("Documents/data/FrameworkWorkedWith.csv",index\_col='Respondent')

df\_job=pd.read\_csv("Documents/data/frameworks\_job\_profile.csv",index\_col=False)

df\_job.index.name='uniq\_id'

#print(df\_user.columns)

#print(df\_job.columns)

df\_user.drop('Unnamed: 0', axis=1, inplace=True)

#df\_job.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))

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

#userprofile=userprofile.reindex\_axis(sorted(userprofile.columns), axis=1)

#jobprofile=jobprofile.reindex\_axis(sorted(jobprofile.columns), axis=1)

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,'Framework')

df\_user.to\_csv("Documents/data/frameworks\_profile\_user.csv")

df\_job.to\_csv("Documents/data/frameworks\_profile\_job.csv")

#Platforms

df\_user=pd.read\_csv("Documents/data/PlatformWorkedWith.csv",index\_col='Respondent')

df\_job=pd.read\_csv("Documents/data/platforms\_job\_profile.csv",index\_col=False)

print(df\_user.columns)

df\_job.index.name='uniq\_id'

print(df\_job.columns)

df\_user.drop('Unnamed: 0', axis=1, inplace=True)

#df\_job.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))

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,'Platform')

df\_user.to\_csv("D/data/platforms\_profile\_user.csv")

df\_job.to\_csv("D/data/platforms\_profile\_job.csv")

#Databases

df\_user=pd.read\_csv("D/data/DatabaseWorkedWith.csv",index\_col='Respondent')

df\_job=pd.read\_csv("D/data/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\_job.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))

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("D/data/databases\_profile\_user.csv")

df\_job.to\_csv("D/data/databases\_profile\_job.csv")

#flag indicates that a new user profile

def match\_profile(self,input\_path,user\_id,flag=0):

#Match a given user\_id with all jobs in the database

#Check if user id exists

df=pd.read\_csv("D/data/domain\_user\_profile.csv",index\_col='Respondent')

#print(df.columns)

matches=dict()

if(flag==0):

if(user\_id in df.index):

userdomain=df.loc[user\_id,:]

#print(userdomain)

#If it does, retrieve the user profile from input\_path

df=pd.read\_csv("D/data/languages\_profile\_user.csv",index\_col='Respondent')

userlanguages=df.loc[user\_id,:]

df=pd.read\_csv("D/data/frameworks\_profile\_user.csv",index\_col='Respondent')

userframeworks=df.loc[user\_id,:]

df=pd.read\_csv("D/data/platforms\_profile\_user.csv",index\_col='Respondent')

userplatforms=df.loc[user\_id,:]

df=pd.read\_csv("D/data/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))

#print(userdomain)

else:

print("error! user id not in Dataset")

#If it doesn't,take user profile as input

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")

#domains=list(map(lambda x:x.lower(),domains))

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("D/data/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("D/data/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("D/data/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("D/data/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)

#print(userdomain)

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("D/data/domain\_job\_profile.csv",index\_col='uniq\_id')

joblanguages=pd.read\_csv('D/data/languages\_profile\_job.csv',index\_col='uniq\_id')

jobframeworks=pd.read\_csv('D/data/frameworks\_profile\_job.csv',index\_col='uniq\_id')

jobplatforms=pd.read\_csv('D/data/platforms\_profile\_job.csv',index\_col='uniq\_id')

jobdatabases=pd.read\_csv('D/data/databases\_profile\_job.csv',index\_col='uniq\_id')

#print(len(jobdomain.index),len(joblanguages.index))

for i in jobdomain.index:

#print(i)

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)

#print(len(domain),len(userdomain))

score=(0.7\*cosine\_similarity(domain,userdomain))+(0.3\*(cosine\_similarity(language,userlanguages)+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,userlanguages)+cosine\_similarity(framework,userframeworks)+cosine\_similarity(platform,userplatforms)+cosine\_similarity(database,userdatabases)))

#Initializing job profiles for later access

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]

#print("recommendations are")

#print(recommendations)

rows=pd.DataFrame(columns=self.df2.columns)

count=0

for i in recommendations:

row=self.df2[self.df2['uniq\_id']==i[0]]

#rows[count]=np.asarray(row.values.T.tolist()[0])

rows=rows.append(row.iloc[0])

count=count+1

#print(row)

return rows

obj=job\_postings("D/data/job\_sample.csv")

#final\_cat=obj.categorize\_jobs()

#final\_cat.to\_csv("D/data/preprocessed\_df.csv")

#extracted\_skills=obj.clean\_skills()

#extracted\_skills\_df=obj.extract\_skills(extracted\_skills)

#print(extracted\_skills\_df)

#domain\_df=pd.read\_csv("D/data/preprocessed\_df.csv")

#obj.create\_job\_profile(extracted\_skills\_df,domain\_df)

#obj.create\_common\_profile("D/data/","Documents/data/")

#Path represents the location where final job and user profiles

#df\_user=pd.read\_csv("D/data/survey\_results\_public.csv")

#df\_job=pd.read\_csv("D/data/job\_recommend.csv")

#Pass a third parameter(flag) as 1 in order to get your recommendations!

rows=obj.match\_profile("Documents/data/",3)

rows

#rows

#recommendations\_1000=pd.DataFrame(columns=df\_job.columns)

#for ele in df\_user.loc[:,'Respondent'].tolist()[:5000]:

#rows=obj.match\_profile("Documents/data/",ele)

#recommendations\_1000=recommendations\_1000.append(rows.iloc[0,:],ignore\_index=True)

#recommendations\_1000.to\_csv("D/data/recommend.csv")