Recruitize

Import libraries necessary for Analysis

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import re
import io
import json
import spacy
from spacy.matcher import Matcher
from spacy import displacy
import nltk
from nltk.corpus import stopwords
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
import tabula
from tika import parser
import spacy
from geopy.geocoders import Nominatim
from sklearn.feature_extraction.text import CountVectorizer
from collections import Counter
import en_core_web_sm
from sklearn.feature_extraction.text import TfidfVectorizer
import sklearn
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from matplotlib import pyplot as plt
import numpy as np
from PIL import Image
from datetime import date
import datetime
import operator
import itertools
from urllib.request import urlopen
import requests
from datetime import datetime
from bs4 import BeautifulSoup
import matplotlib as mp
import pickle
import plotly.express as px
import time
import MySQLdb
import math
import glob, os
from sklearn.cluster import KMeans
import pandas as pd
import csv
```

```
import pandas as pd
import numpy as np
from bs4 import BeautifulSoup
from urllib.request import urlopen
import requests
import csv
import time
import PyPDF2
import pytesseract
import cv2
import numpy as np
from PIL import Image
from pyzbar.pyzbar import decode
import spellchecker
from spellchecker import SpellChecker
#from pdf2docx.main import parse
#import mammoth
#nltk.download()
#spacy.load("en_core_web_sm")
```

Load file for CV Analysis

In [2]:
```
with open(r'.\RecruitizeAnalysisData.json') as f:
      match_data = json.load(f)
print("Imported all Analysis Data from json and saved to 'match_data' dictionary Succes
```

```
Imported all Analysis Data from json and saved to 'match_data' dictionary Successfully!
```

Intitialize Database

- MySQL Database is used in the project because it can easily be configured for Machine Learning Applications as per the requirements.

In [3]:
```
db = MySQLdb.connect(host="localhost",    # your host, usually localhost
                     user="root",         # your username
                     passwd="root",  # your password
                     db="recruitize")
cur=db.cursor()
print("Connection to Database Successful!")
```

```
Connection to Database Successful!
```

Database Table creation

Create all the required tables in the database "Recruitize" if not created yet.

In [4]:
```
def create_skill_table():
    skill_branch=list(match_data["Skills"].keys())
    for i in skill_branch:
        for j in match_data["Skills"][i].keys():
            cur.execute("create table IF NOT EXISTS `{}`(Name1 varchar(200) PRIMARY KEY
            cur.execute("create table IF NOT EXISTS `{}_tfidf`(Name1 varchar(200) PRIMA
def create_comp_table():
    for j in match_data["Competencies"].keys():
        cur.execute("create table IF NOT EXISTS `{}`(Name1 varchar(200) PRIMARY KEY)".f
        cur.execute("create table IF NOT EXISTS `{}_tfidf`(Name1 varchar(200) PRIMARY K
```

```
cur.execute("create table IF NOT EXISTS term_freq(Name1 varchar(255) PRIMARY KEY)")
cur.execute("create table IF NOT EXISTS tfidf(Name1 varchar(255) PRIMARY KEY)")
cur.execute("create table IF NOT EXISTS patent(Name1 varchar(255) DEFAULT NULL, id1 var
cur.execute("create table IF NOT EXISTS `Psycographic Analysis`(Name1 varchar(255) PRIM
create_skill_table()
create_comp_table()
no_of_new_tables_created=cur.execute("show tables")
print("No. of tables created in Database=", no_of_new_tables_created)
```

```
No. of tables created in Database= 116
```

Data Extraction and Detail Verification

Authentication of Candidate with his Aadhaar Card

While recruitment for any role it is necesaary to verify the identity of the candidate.
preprocess_img(img_path)- enhances the image for Object Code Recognition and QR scan. The function makes use of Gaussian Blur and Thresholding.
extractAadhaarDetails(image)- from the enhanced image it recognizes the QR code on Aadhaar and performs an OCR on aadhaar for verification.

In [5]:
```
# kindly install tesseract from -https://github.com/UB-Mannheim/tesseract/wiki first an
def preprocess_img(img_path):
    pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract'
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = img.astype('uint8')
    blur = cv2.GaussianBlur(img,(1,1),0)
    ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
    image=th3
    cv2.imshow('image',image)
    data="Extracted Text  -\n",pytesseract.image_to_string(image)
    return image,data;

def extractAadhaarDetails(image):
    details=dict()
    data = decode(image)
    d=str(data[0])
    try:
        try:
            s=d.index("uid")+5
            details["uid"]=d[s:s+12]
        except:print("uid error")
        try:
            s=d.index('name')+6
            details["name"]=d[s:d.index('"',s)]
        except:print("name error")
        try:
            s=d.index('gender')+8
```

```
            details["gender"]=d[s:d.index('"',s)]
        except:print("gender error")
        try:
            s=d.index('state')+7
            details["state"]=d[s:d.index('"',s)]
        except:print("state error")
        try:
            s=d.index(' pc=')+5
            details["pincode"]=d[s:d.index('"',s)]
        except:print("pc error")
        try:
            s=d.index(' yob=')+6
            details["yob"]=d[s:d.index('"',s)]
        except:print("yob error")
    except: print("No QR found\n REFER OCR RESULTS")
    return details
preprocessed_image,ocr_data=preprocess_img(r'.\aadhaar.PNG')
qr_data=extractAadhaarDetails(preprocessed_image)
print("Extracted Data from Candidate's Aadhaar Card:")
if len(qr_data):
    print(qr_data)
else:
    print(ocr_data)
```

```
Extracted Data from Candidate's Aadhaar Card:
{'uid': '980475200990', 'name': 'Shivam Gautam', 'gender': 'MALE', 'state': 'Uttar Prade
sh', 'pincode': '282005', 'yob': '1998'}
```

Face Recognition of the candidate and match it with his Aadhaar ID photo

This helps in 2 factor authentication of the candidate.

For authentication of current user logged in. This will be helpful for any online proctoring practice if required.

The module makes use of Azure Cognitive Services face recognition feature to verify and authenticate the candidate with his face.

In [6]:

```
def match_face(img):
    import asyncio
    import io
    import glob
    import os
    import sys
    import time
    import uuid
    import requests
    from urllib.parse import urlparse
    from io import BytesIO
    from PIL import Image, ImageDraw
    from azure.cognitiveservices.vision.face import FaceClient
    from msrest.authentication import CognitiveServicesCredentials
    from azure.cognitiveservices.vision.face.models import TrainingStatusType, Person

    KEY = "57db4ff2189e4ce8913adba162d7ee61"
    ENDPOINT = "https://deifacerecognition.cognitiveservices.azure.com/"
    face_client = FaceClient(ENDPOINT, CognitiveServicesCredentials(KEY))
```

```
    import cv2
    cam = cv2.VideoCapture(0)
    cv2.namedWindow("test")
    img_counter = 0
    img_name=''
    while True:
        ret, frame = cam.read()
        if not ret:
            print("failed to grab frame")
            break
        cv2.imshow("test", frame)

        k = cv2.waitKey(1)
        if k%256 == 27:
            # ESC pressed
            print("Escape hit, closing...")
            break
        elif k%256 == 32:
            # SPACE pressed
            img_name = "img{}.png".format(img_counter)
            cv2.imwrite(img_name, frame)
            print("{} written!".format(img_name))
            img_counter += 1
        if img_counter==1:
            break;

    cam.release()

    cv2.destroyAllWindows()


    urls=[]
    urls.append('./aadhaar.png')
    urls.append(img_name)
    detected_faces_ids = []
    for image in urls:
        detected_faces = face_client.face.detect_with_stream(open((os.path.join(image))

        detected_faces_ids.append(detected_faces[0].face_id)
        print('{} face(s) detected from image {}.'.format(len(detected_faces), image))


    verify_result_same = face_client.face.verify_face_to_face(detected_faces_ids[0], de
    print('Faces from {} & {} are of the same person, with confidence: {}'
    .format(urls[0], urls[1], verify_result_same.confidence)
        if verify_result_same.is_identical
        else 'Faces from {} & {} are of a different person, with confidence: {}'
            .format(urls[0], urls[1], verify_result_same.confidence))
    return verify_result_same.confidence
```

Text Extraction from Resume

- 2 functions are created for extraction of text. The function "extract_resume_text(pdf_path)" gives best results but may fail for some resume formats thus the second function "extract_resume_text1(pdf_path)" comes into picture for extracttion when the func1 fails.
- **extract_resume_text(pdf_path)**- Runs on Tika server which uses "JAVA" as backend for extraction.

- **extract_resume_text1(pdf_path)**- Uses PyPDF2 module of Python.

In [7]:
```
def extract_resume_text(pdf_path):
    t1=""
    t2=[]
    try:
        raw= parser.from_file(pdf_path)
        txt=''
        bullets=[]
        text_list=[]
        for line in io.StringIO(raw['content']):
                if len(line.strip())!=0:
                    txt=txt+"\n"+line.strip().lower()
                    text_list.append(line.strip())
        t1=txt
        t2=text_list
    except:
        object = PyPDF2.PdfFileReader(pdf_path)
        NumPages = object.getNumPages()
        # extract text and do the search
        flag=False
        print('Verifying....')
        Text=''
        for i in range(0, NumPages):
            PageObj = object.getPage(i)
            Text = Text+PageObj.extractText()
        t1=Text
        t2=Text
    return t1,t2

def extract_resume_text1(pdf_path):
    object = PyPDF2.PdfFileReader(pdf_path)
    NumPages = object.getNumPages()
    # extract text and do the search
    flag=False
    print('Verifying....')
    Text=''
    for i in range(0, NumPages):
        PageObj = object.getPage(i)
        Text = Text+PageObj.extractText()
    return Text, text
```

Preprocessing of Text

- Special characters such as "/", "-", "," are removed.
- Terms related to Highschool and Intermediate are normalized for qualification extraction in future processing.
- **NLTK toolkit** is used for removing stopwords such as "is, and , the..." which are just noise in the analysis from the text.
- Alphanumeric words and digits are removed expect the digits that may represent years
- All the words are **Lemmatized** to get the root word, for eg. "*running, runs, ran*" is changed to "*run*" the root word

In [8]:
```
def preprocess_text(txt):
    txt=txt.replace('/', " ").replace("-"," ").replace(',', ' ')
    for i in match_data['X']:
```

```
        txt=txt.replace(i,'X')
    for i in match_data['XII']:
        txt=txt.replace(i,'XII')
    words=nltk.word_tokenize(txt)
    wordsNS=[word for word in words if word not in stopwords.words('english') and word.
    lemma=WordNetLemmatizer()
    txt_new=" ".join([lemma.lemmatize(word) for word in wordsNS])
    return txt_new
```

Extraction of Personal Details

- **extract_name(resume_text)** - Name of the candidate is extracted using "POS" tag-"PROPN" (proper noun) in spacy module in the beginning of the document (first 200 characters) to get efficient results.
- **extract_phoneNo(t)** - Extraction of Phone No. of candidate using regex
- **extract_email(t)** - Extraction of E-mail of candidate using regex
- **extract_DOB(t)** - Extraction of DOB of candidate using regex
- **calculateAge(birthDate)** - Function to calculate present age of the candidate using "*datetime*" module of python
- **extract_personalDetails(txt)** - Function to extract personal details of the candidate such as - "*Gender, Address, DOB, Father's Name, Mother's Name, contact no., email and Age*". This functions calls the above mentioned functions if the need arrises.

In [9]:
```
def extract_name(resume_text):
    nlp = spacy.load('en_core_web_sm')
    matcher = Matcher(nlp.vocab)
    nlp_text = nlp(resume_text[:200])
    pattern = [{'POS': 'PROPN'}, {'POS': 'PROPN'}]
    matcher.add('NAME', None, pattern)
    matches = matcher(nlp_text)
    for match_id, start, end in matches:
        span = nlp_text[start:end]
        return span.text

def extract_phoneNo(t):
    no= list(re.findall("[6-9][0-9]{9}", t))
    return int(no[0])

def extract_email(t):
    email= re.findall('\S+@\S+[.][a-z]+', t)
    return email[0]

def extract_DOB(t):
    dob=None
    dob= re.findall('[0-9]+/[0-9]+/[0-9]+|[0-9]+-[0-9]+-[0-9]+|[0-9]+[.][0-9]+[.][0-9]+
    return  dob[0]

def calculateAge(birthDate):
    birthDate=datetime.datetime.strptime(birthDate.replace("/","").replace("-","").repl
    today = date.today()
    age = today.year - birthDate.year -  ((today.month, today.day) < (birthDate.month,
    return age

def extract_personalDetails(txt):
    dic={}
```

```
    pd=[]
    for line in txt.splitlines():
        split= re.split(":",line)
        word_tokens=word_tokenize(txt)
        if len(split)==2:
            string=split[0].strip()
            detail=split[1].strip()
            if re.match('name', string, re.IGNORECASE)!=None:
                dic["Name"]=detail
                pd.append(string);pd.extend(word_tokenize(detail))
            if re.search("gender", string, re.IGNORECASE)!=None:
                if re.search("f", detail, re.IGNORECASE)!=None:
                    dic["Gender"]="Female";
                elif re.search("m", detail, re.IGNORECASE)!=None:
                    dic["Gender"]="Male";
                pd.append(string);
            if re.search("contact" or "mobile" or "phone", string, re.IGNORECASE)!=None
                try:
                    dic["Contact No."]=detail[0]
                except: print("Contact Error")
                pd.append(string);pd.extend(detail)
            if re.search("mail", string, re.IGNORECASE)!=None:
                dic["E-Mail"]=detail
                pd.append(string);
                pd.extend(word_tokenize(detail))
            if re.search("address", string, re.IGNORECASE)!=None:
                dic["Address"]=detail
                pd.append(string);pd.extend(word_tokenize(detail))
            if re.search("father", string, re.IGNORECASE)!=None:
                dic["Father's Name"]=detail
                pd.append(string);pd.extend(word_tokenize(detail))
            if re.search("mother", string, re.IGNORECASE)!=None:
                dic["Mother's Name"]=detail
                pd.append(string);pd.extend(word_tokenize(detail))
            dob=None
            if re.search("dob" or "d.o.b", string, re.IGNORECASE)!=None:
                dic["DOB"]=detail;
    if "Name" not in dic.keys():
        dic["Name"]=extract_name(txt)
    if "Contact No." not in dic.keys():
        try:
            dic["Contact No."]=extract_phoneNo(txt)
        except:print("Contact Error")
    if "E-Mail" not in dic.keys():
        try:
            dic["E-Mail"]=extract_email(txt)
        except:print("E-mail Error")
    if "DOB" not in dic.keys():
        try:
            dic["DOB"]=extract_DOB(txt)
        except: print("DOB Error")
    try:
        if "Age"  not in dic.keys() and "DOB" in dic.keys() and len(dic["DOB"])!=0:
            dic["Age"]=calculateAge(dic["DOB"])
    except: print("Error Age")
    pd.extend(["Male","MALE", "Female","FEMALE"])
    return dic, pd
```

Extraction of Qualifications

- **extract_branch(t)** - Extraction of Branch of the candidate using regex
- **extract_year1(txt)** - Function used to check if some year is mentioned in the vicinity or not for qualification year extraction.

The extraction of Qualifications of the candidate is done by following 2 strategies based on the format of the resume.One being if the candidate has used some sort of tabular format to mention his qualifications and the other being normal text without tables

- **extract_qualifications_Tabular(pdf_path)** - First attempt to extract qualifications is made using this module. It attempts to get the details if a table is used to enlist the qualtifications by using regex and normalising the cgpa and percentage.
- **extract_qualifications_nonTabular(txt)** - This function is called when the first attempt fails (i.e. non tablular format is used. Now another attempt is made to extract the details using regex with a slightly different approach. The cgpa and percentage are normalized in this method also.

In [10]:
```
def extract_branch(t):
    branch=[]
    for b in match_data['Branches']:
        if re.search(b,t, re.IGNORECASE)!=None:
            branch.append(b)
    return branch

def extract_year1(txt):
    year=re.findall("[2][0]\d{2}|[1][9]\d{2}", txt, re.IGNORECASE)
    return  year

def extract_qualifications_Tabular(pdf_path):
    dfs = tabula.read_pdf(pdf_path, stream=True, pages='all')
    data=dfs[0]
    data = data.dropna().reset_index(drop=True)
    col=data.columns
    for c in col:
        if re.search("year", c, re.IGNORECASE)!=None:
            data=data.rename(columns={c:"Year"})
        elif re.search("institut", c, re.IGNORECASE)!=None:
            data=data.rename(columns={c:"Institution"})
        elif re.search("quali" or "name" or 'degr', c, re.IGNORECASE)!=None:
            data=data.rename(columns={c:"Qualification"})
        elif re.search("gpa" or "perc", c, re.IGNORECASE)!=None:
            data=data.rename(columns={c:"Percentage"})

    for p in range(len(data["Percentage"])):
        try:
            s=data.at[p,"Percentage"]
            l=re.findall(r"\d+\.\d+",str(s))
            per=float(l[0])
            if per<10:
                per=per*9.5
            data.at[p,"Percentage"] = per
        except: print("Error ignored")
        data_dict = data.to_dict(orient='records')
    return data_dict

def extract_qualifications_nonTabular(txt):
    btech=re.findall("B.*Tech", str(txt), re.IGNORECASE)[0]
    btech_index=txt.index(btech)
    XII=re.findall("XII", txt[btech_index:], re.IGNORECASE)[0]
```

```
    XII_index=txt.index(XII)
    X=re.findall("X", txt[XII_index:], re.IGNORECASE)[1]
    X_index=txt.index(X,XII_index+1)
    x=re.findall("(X)", txt, re.IGNORECASE)
    xii=re.findall("XII", txt, re.IGNORECASE)[0]
    btech_data=txt[btech_index:XII_index]
    XII_data=txt[XII_index: X_index]
    X_data=txt[X_index:]
    btech_year=extract_year1(btech_data)
    btech_year_completion=max(btech_year)
    btech_gpa=float(re.findall("[0-9]{1,2}[.][0-9]{1,2}",btech_data)[0])
    XII_year=extract_year1(XII_data)
    XII_year_completion=max(XII_year)
    XII_gpa=float(re.findall("[0-9]{1,2}[.][0-9]{1,2}",XII_data)[0])
    X_year=extract_year1(X_data)
    X_year_completion=X_year[0]
    X_gpa=float(re.findall("[0-9]{1,2}[.][0-9]{1,2}",X_data)[0])
    li=[]
    dic={}
    dic["Qualification"]=btech
    if(btech_gpa<10):
        btech_gpa=btech_gpa*9.5
    dic["Percentage"]=btech_gpa
    dic["Year"]=btech_year_completion
    li.append(dic)
    dic={}
    dic["Qualification"]=XII
    if(XII_gpa<10):
        btech_gpa=btech_gpa*9.5
    dic["Percentage"]=XII_gpa
    dic["Year"]=XII_year_completion
    li.append(dic)
    dic={}
    dic["Qualification"]=X
    if(X_gpa<10):
        btech_gpa=btech_gpa*9.5
    dic["Percentage"]=X_gpa
    dic["Year"]=X_year_completion
    li.append(dic)
    dic={}
    return li
```

Extraction of Patent Details

Patent ID in India has a specific format. It is a 12 digit no. with format-**YYYYJTNNNNNN**
Where,

- "YYYY" - Year of filing,
- J-"Jurisdiction in numerals (1-Delhi, 2- Mumbai, 3- Kolkata, 4- Chennai)
- T-Type of Application
- "NNNNNN"-6 digit serial no.

Taking advantage of the above format Patent ID is extracted.

- **find_patent(txt)** - Function to extract all the patent IDs from the candidate's resume.

Patent information is not directly available on ipindia website as it as got a field to stop robots from breaking into the system. An alternate way was found out. We researched on this topic and came to

know that everyday a document is published on ipindia website which consists of briefing of all the patents published on that date hence we took advantage of this point and scraped data from that document.

- **get_patent_data(patent_id,name)** - Given a particular patent this function i used to extract information for that ID from the web. The particular patent ID is searched on the web and link to the pdf file is selected dynamically taking advantage of the semantics of the link. A pdf file of the document is downloaded and saved in the system from which information from the page containing that Patent ID is extracted using regex and thereafter saved and returned by the function. It also verifies that name-"*name of the candidate*" is there or not for that patent and hence verify it.

- **get_patent_details(txt, name, pdf_path)**- This is an iterative function which calls the above 2 functions to extract all the patent IDs and their information and store the in the MySQL database.

In [11]:
```
def find_patent(txt):
    p=re.findall("([1][9][4-9][0-9][1-4][1-9]\d{6})|([2][0][0-2][0-9][1-4][1-9]\d{6})",
    patent=[]
    for i in p:
        patent.append(i[1])
    return  patent
def get_patent_data(patent_id,name):
    patent=dict()
    headers_std = {
    'User-Agent': 'Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko; compatible; Googl
    }
    print("Searching patent=", patent_id)
    url="https://www.google.com/search?q="+patent_id+"&rlz=1C1CHBF_enIN799IN799&oq=2020
    html = requests.get(url,headers=headers_std).text
    soup = BeautifulSoup(html,'html.parser')
    file_url=None
    for a in soup.find_all('a', href=True):
        link=a['href']
        if "ipindia" in link:
            print ("Found URL:",link)
            file_url=link
            break;
    if file_url==None:
        print('patent not found')
    if file_url:
        r = requests.get(file_url, stream = True)
        with open("patent.pdf","wb") as pdf:
                for chunk in r.iter_content(chunk_size=1024):
                        if chunk:
                            pdf.write(chunk)
        object = PyPDF2.PdfFileReader("patent.pdf")
        NumPages = object.getNumPages()
        # extract text and do the search
        flag=False
        print('Verifying....')
        for i in range(0, NumPages):
            PageObj = object.getPage(i)
            Text = PageObj.extractText()
            ResSearch = re.search(patent_id, Text)
            if ResSearch:
                flag=True
                break
        print('Extracting Details...')
```

```
        if flag:
            patent['id']=patent_id
            ind1=Text.index('Title of the invention :')+25
            ind2=Text.index('(',ind1)
            if name.lower() in Text.lower():
                patent['status']=True
            else:patent['status']=False
            patent['title']=Text[ind1:ind2].replace('\n',"").strip()
            ind1=Text.index('Date of filing of Application :')+31
            ind2=Text.index('\n',ind1)
            patent['filing-date']=Text[ind1:ind2].strip()
            ind1=Text.index('Publication Date :')+19
            ind2=Text.index('\n',ind1)
            patent['publication-date']=Text[ind1:ind2].strip()
            ind1=Text.index('Abstract :')+11
            ind2=Text.index('No. of Pages',ind1)
            patent['abstract']=Text[ind1:ind2].replace("\n","").replace("  "," ").strip
        return patent
def get_patent_details(txt, name, pdf_path):
    patent_data=[]
    for i in  find_patent(txt):
        try:
            p=get_patent_data(i,name)
            #print(p)
            insert_query="REPLACE INTO patent VALUES ( \'{} \',  \'{} \',  {} ,  \'{} \
            #print(insert_query)
            r=cur.execute(insert_query)
            db.commit()
        except:
            print("failed",i)
    return 1
cur.execute("select *from patent")
print("Data from Patent Table:")
for row in cur.fetchall():
    col_db.append(row[0])
```

```
Data from Patent Table:
```

Psychographic Analysis of the candidate

The Behavior and general mindset of the candidate is looked upon for the analysis. The 5 well known traits viz.,

1. 'Extrovert'
2. 'Neuroticism'
3. 'Agreeable'
4. 'Conscientious'
5. 'Openness'

After the analysis a bar graph is generated to dipict the report.
create_psychograph(text, path) This function makes use of "*Pickle*" *module of python to do the analysis. text-text to be analysed. path-path where the graph generated needs to be saved.*

In [12]:
```
def create_psychograph(text, path):
    cEXT = pickle.load( open(r"Personality_affinities/cEXT.p", "rb"))
    cNEU = pickle.load( open(r"Personality_affinities/cNEU.p", "rb"))
```

```
    cAGR = pickle.load( open(r"Personality_affinities/cAGR.p", "rb"))
    cCON = pickle.load( open(r"Personality_affinities/cCON.p", "rb"))
    cOPN = pickle.load( open(r"Personality_affinities/cOPN.p", "rb"))
    vectorizer_31 = pickle.load( open(r"Personality_affinities//vectorizer_31.p", "rb")
    vectorizer_30 = pickle.load( open(r"Personality_affinities//vectorizer_30.p", "rb")
    scentences = re.split("(?<=[.!?]) +", text)
    text_vector_31 = vectorizer_31.transform(scentences)
    text_vector_30 = vectorizer_30.transform(scentences)
    EXT = cEXT.predict(text_vector_31)
    NEU = cNEU.predict(text_vector_30)
    AGR = cAGR.predict(text_vector_31)
    CON = cCON.predict(text_vector_31)
    OPN = cOPN.predict(text_vector_31)

    predictions = [EXT, NEU, AGR, CON, OPN]
    sum=0
    for i in range(0,len(predictions[0])):
            sum = sum + predictions[0][i]
    sum=(sum*100)/len(predictions[0])
    count=0
    for j in range(0, len(predictions[1])):
          count = count + predictions[1][j]
    count=(count*100)/len(predictions[1])
    total=0
    for j in range(0, len(predictions[2])):
          total = total + predictions[2][j]
    total=(total*100)/len(predictions[2])
    cons=0
    for j in range(0, len(predictions[3])):
          cons = cons + predictions[3][j]
    cons=(cons*100)/len(predictions[3])
    opnn=0
    for j in range(0, len(predictions[4])):
          opnn = opnn + predictions[4][j]
    opnn=(opnn*100)/len(predictions[4])
    final=[int(sum),int(count),int(total),int(cons),int(opnn)]
    height = final
    likeability_scores = np.array(height)
    data_normalizer = mp.colors.Normalize()

    color_map = mp.colors.LinearSegmentedColormap("my_map", {"red": [(0, 0.50, 0.5),(1.
    bars = ('Extrovert', 'Neuroticism', 'Agreeable', 'Conscientious', 'Openness')
    y_pos = np.arange(len(bars))
    plt.barh(y_pos, height,  color=color_map(data_normalizer(likeability_scores)),edgec
    plt.xlabel("Score(in %)", labelpad=20, weight='bold', size=12)

    plt.ylabel("Personality Affinities", labelpad=20, weight='bold', size=12)
    for i, v in enumerate(height):
        plt.text(v + 3, i + .25, str(v)+"%",va='center', color='black')

    plt.yticks(y_pos, bars)
    plt.title('Psychographic Analysis')
    plt.savefig(path)
    cur.execute("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME =
    query="REPLACE INTO `Psycographic Analysis` VALUES ( \"{}\",{}  ,{}  ,{}  ,{}  ,{})
    print(query)
    cur.execute(query)
    return path, height
```

Competencies

Competence is the set of demonstrable characteristics and skills that enable, and improve the efficiency or performance of a job. Key competencies are specific qualities that a company's recruiters consider desirable for employees to possess.
They are often used as benchmarks to rate and evaluate candidates during the recruitment process.
We have taken into account 11 key competencies of the candidate, viz.

1. *Adhering to Principles and Values*
2. *Creative and Innovative*
3. *Decision Making*
4. *Enterpreneurial Thinking*
5. *Leadership Quality*
6. *Management*
7. *Working with People*
8. *Achieving Personal Goals*
9. *Influencing and Pursuading*
10. *Networking and Relating*
11. *Planning and Organising*

extract_competencies(t) *- This function fetches the list of competencies to be analysed for from the database and matches its similarity with the candidate's resume and calls the function* ***extract_competency_score(t, tag)*** *to get the score for each competency.*

extract_competencies(t) *- t- text to be analysed.*
extract_competency_score(t, tag) *- t- text to be analysed, tag - Name of the competency to be looked for.*

In [13]:
```
def extract_competency_score(t, tag):
    comp=dict()
    total_count=0
    for i in match_data["Competencies"][tag]:
        i=i.lower().strip()
        if re.search(i,t, re.IGNORECASE)!=None:
            cc=t.count(i)
            if cc:
                comp[i]=cc
                total_count+=cc
    return comp, total_count
def extract_competencies(t,pdf_path):
    competency_list=match_data["Competencies"].keys()
    competencies=dict()
    total_count=0
    for i in competency_list:
        competencies[i],count1=extract_competency_score(t,i)
        total_count+=count1

    for i in competencies.keys():
        col_db=[]
        col_comp=competencies[i].keys()
        cur.execute("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAM
        for row in cur.fetchall():
```

```
            col_db.append(row[0].lower().strip())
        #print("Trying-.....................", i)
        col_to_add=set(col_comp)-set(col_db)
        if len(col_to_add)!=0:
            query="ALTER TABLE `{}` ADD (`"
            for k in list(col_to_add):
                query+=k.strip()+"` float DEFAULT 0.0,`"
            query=query[:-2]+')'
            query1=query.format(i)
            query2=query.format(i+"_tfidf")
            cur.execute(query1)
            cur.execute(query2)
        if len(col_comp)!=0:
            insert_query="REPLACE INTO `{}` (Name1 ,".format(i)
            for k in col_comp:
                insert_query+="`"+str(k.strip())+'`, '
            insert_query=insert_query[:-2]+") VALUES ( \'{}\'  ,".format(pdf_path)
            for j in col_comp:
                k=competencies[i][j]
                insert_query+=str(k/total_count)+', '
            insert_query=insert_query[:-2]+')'
            #print(insert_query)
            r=cur.execute(insert_query)
            db.commit()
            #print("added to DB")
    return competencies
def competency_tfidf():
    for j in match_data["Competencies"].keys():
        tf_idf(j.lower(),j.lower()+"_tfidf")
    return 1
```

Wordcloud Creation

A word cloud, also known as a tag cloud or word art, is a simple visualization of data in which words are shown in varying sizes depending on how often they appear in your text. We use this representation to get a brief idea about the interests of the candidate so as the hiring organzation can have a look and get an insight about the candidate.

create_wordcloud(t,path) *- t-text for which the wordcloud needs to be created, path-path where the .png file of wordpath needs to be saved.*

In [14]:
```
def create_wordcloud(t,path):
    resume_wordcloud =  WordCloud(width=600,height=600,background_color ='white',min_fo
    plt.figure(figsize=(8,8),facecolor=None)
    plt.imshow(resume_wordcloud)
    plt.axis("off")
    plt.tight_layout(pad=0)
    plt.savefig(path)
    return path
```

Extraction of Certifications

Skills ensure that a candidate can do a job efficiently and a skill certificate is a way of ensuring the recruiters about the candidate having such skills. Therefore, a skill certificate plays an important role in getting you a job.

We at "Recruitize" make sure that every certification of the candidate pays for his/her effort. Currently, we have tested our algorithm with courseera certificates to demonstrate such a thing. The workflow goes as follows-
extract_certifications(links) *-this function identifies the link to coursera certificate of the candidate from a bunch of different links extracted beforehand.*

In [15]:
```
def extract_certifications(links):
    certification=[]
    site={"coursera"}
    for i in site:
        for j in links:
            if re.search(j,i, re.IGNORECASE):
                certification.append(j)
    return certification
```

Verification of Coursera certificates of the Candidate

scrape_coursera(url)- *Webscrapes the coursera website to verify the certification of the candidate.*

In [16]:
```
def scrape_coursera(url):
    headers_std = {
    'User-Agent': 'Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko; compatible; Googl
    }
    print("Scraping---",url)
    html = requests.get(url,headers=headers_std).text
    soup = BeautifulSoup(html,'html.parser')
    details=dict()
    details["cert_link"]=url
    details["candidate_name"]  = soup.find("h3",{ "class" : "_lriijlm"}).strong.text
    details["course_name"]=soup.find("h2",{ "class" : "course-name"}).text
    details["course_date"]=soup.find("div", { "class" : "course-details"}).p.text
    return details
#r=scrape_coursera("https://www.coursera.org/verify/JAQPWWXWYHPH")
```

Term Frequency- Inverse Document Frequency

Terminology :

t — term (word)
d — document (set of words)
N — count of corpus
corpus — the total document set

Term Frequency-

The weight of a term that occurs in a document is simply proportional to the term frequency.
tf(t,d) *= count of t in d / number of words in d*

Document Frequency:

This measures the importance of document in whole set of corpus, this is very similar to TF. The only difference is that TF is frequency counter for a term t in document d, where as DF is the count of occurrences of term t in the document set N.
df(t) *= occurrence of t in documents*

Inverse Document Frequency:

An inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely. IDF is the inverse of the document frequency which measures the informativeness of term t. When we calculate IDF, it will be very low for the most occurring words such as stop words (because stop words such as "is" is present in almost all of the documents, and N/df will give a very low value to that word). This finally gives what we want, a relative weightage.
idf(t) *= N/df*
During the query time, when a word which is not in vocab occurs, the df will be 0. As we cannot divide by 0, we smoothen the value by adding 1 to the denominator.That's the final formula:
Formula :
idf(t) *= log(N/(df + 1))*

TF-IDF

Hence, tf-idf is a the right measure to evaluate how important a word is to a document in a collection or corpus.
Formula :
tf-idf(t, d) *= tf(t, d) * log(N/(df + 1))*

In [17]:
```
def term_freq(txt,name):
    cur = db.cursor()
    col_db=[]
    cur.execute("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME =
    for row in cur.fetchall():
        col_db.append(row[0])
    tf=Counter(txt.split())
    col_cv=tf.keys()
    val_cv=tf.values()
    count_words=sum(val_cv)
    col_to_add=set(col_cv)-set(col_db)
    if len(col_to_add)!=0:
        query ="ALTER TABLE {} ADD (`"
        for i in col_to_add:
            query+=i+"` float DEFAULT 0.0,`"
        query=query[:-2]+');'
        query1=query.format("term_freq")
        query2=query.format("tfidf")
        cur.execute(query1)
        cur.execute(query2)

    insert_query="REPLACE INTO term_freq (" +"Name1 ,"
    for i in col_cv:
        insert_query+="`"+str(i)+'`, '
    insert_query=insert_query[:-2]+") VALUES ( \'{}\'  ,".format(name)
    for i in val_cv:
```

```
        insert_query+=str(i/count_words)+', '
    insert_query=insert_query[:-2]+')'
    #print(insert_query)
    print("Term Freq Added to DB")
    r=cur.execute(insert_query)
    db.commit()
    return r
def inverse_doc_freq(t_freq):
    db = MySQLdb.connect(host="localhost",    # your host, usually localhost
                         user="root",         # your username
                         passwd="root",  # your password
                         db="recruitize")
    cur=db.cursor()
    cur.execute("select count(*) from `{}`".format(t_freq))
    no_of_doc=cur.fetchone()[0]
    #print(no_of_doc)
    cur.execute("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME =
    col_db=[]
    idf=dict()
    for row in cur.fetchall():
        q="select count(*) from `{}` where `{}`>0".format(t_freq,row[0])
        #print(q)
        cur.execute(q)
        no_of_doc_contain_word=cur.fetchone()[0]
        #print(no_of_doc_contain_word)
        try:
            #print(no_of_doc,no_of_doc_contain_word)
            val=no_of_doc/no_of_doc_contain_word
            idf[row[0]]=(val)
        except:
            #print("Zero Error")
            idf[row[0]]=no_of_doc
    return idf
def tf_idf(t_freq, t_idf):
    col_tf=[]
    col_tfidf=[]
    cur.execute("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME =
    for row in cur.fetchall():
        col_tf.append(row[0])
    cur.execute("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME =
    for row in cur.fetchall():
        col_tfidf.append(row[0])
    col_to_add=set(col_tf)-set(col_tfidf)
    if len(col_to_add)!=0:
        query="ALTER TABLE {} ADD (`".format(t_idf)
        for i in col_to_add:
            query+=i+"` float DEFAULT 0.0,`"
        query=query[:-2]+');'
        #print(query)
        cur.execute(query)
    cur.execute("select *from `{}`".format(t_freq))
    user_data=[]
    for row in cur.fetchall():
        user_data.append(list(row))
    #print(user_data)
    idf=inverse_doc_freq(t_freq)
    idf_v=list(idf.values())
    for i in user_data:
        insert_query="REPLACE INTO `{}` VALUES ( \'{}\'  ,".format(t_idf,i[0])
        #print(i[0])#i[0] is name of the applicant
        for j in range(1, len(i)):
```

```
            insert_query+=str(i[j]*idf_v[j])+', '
        insert_query=insert_query[:-2]+')'
       # print(insert_query)
        r=cur.execute(insert_query)
    db.commit()
```

Extraction of Links from the Resume

All links are extracted from the resume which may be further used for profile analysis etc. if the need arises.

In [18]:
```
def extract_links(soup):
    links=soup.findAll("a",href=True)
    link_list=[]
    for i in links:
        link_list.append(i['href'])
    return list(set(link_list))
```

In [19]:
```
def extract_profiles(links):
    profile={}
    site={'linkedin',"github","hackerrank"}
    for i in links:
        for j in site:
            if re.search(j,i, re.IGNORECASE):
                profile[j]=i
                break
```

GitHub Account Analysis

Analysis of GitHub Profile of the Candidate

In [20]:
```
URL = 'https://github.com/shivamgautam98?tab=repositories'
def repositories(URL):
    repo_lists={}
    page = requests.get(URL)

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

    repos = soup.findAll('li', itemprop="owns")
    for repo in repos:
        r={}
        title_repo = repo.find('a', itemprop="name codeRepository")
        lang = repo.find('span', itemprop="programmingLanguage")
        status = repo.find('relative-time', class_="no-wrap")
        repo_url = ("".join(["https://github.com/",title_repo['href']]))
        repo_page = requests.get(repo_url)
        new_soup = BeautifulSoup(repo_page.content, 'html.parser')
        commits_head = new_soup.find('span', class_="d-none d-sm-inline")
        readme_head = new_soup.find('div', class_="Box-body px-5 pb-5")
        if(title_repo.text):
            r["title"] = title_repo.text
        if(lang):
            r['languages'] = lang.text
        if(status):
           r['status'] = status.text
        if(commits_head):
```

```
            commits = commits_head.find('strong')
            r['commits'] = commits.text

        repo_lists.add(r)

    return repo_lists
```

Experience

The total experience acquired by the candidate till now has a significant impact on recruitment. The Functions are used to extract the candidate's experince via various NLP techniques.

In [21]:

```
def convert_doc_to_soup(doc_path):
    with open(doc_path, "rb") as docx_file:
        result = mammoth.convert_to_html(docx_file)
        html = result.value # The generated HTML
        messages = result.messages
        soup=BeautifulSoup(html,'html.parser')
    return soup,html
def extract_year(txt):
    year=re.findall("[2][0]\d{2}|[1][9]\d{2}",txt, re.IGNORECASE)
    y={}
    y['start']=min(year[0],year[1])
    y['end']=max(year[0],year[1])
    return  y
def extract_bold_from_soup(soup):
    table=soup.findAll(["strong","h1", "h2", "h3","h4","h5","h6"])
    bold=[]
    for i in table:
        bold.append(i.text)
    return bold
def extract_companies(bold_list):
    nlp = en_core_web_sm.load()
    company=[]
    for i in bold_list:
        doc = nlp(i)
        for X in doc.ents:
            if X.label_=='ORG' :
                if X.text not in company and len(X.text)>4:
                    company.append(X.text)
    #company = list(dict.fromkeys(company))
    return company
def search_exp(bold_txt,companies):
    exp=[];s=[];e=[];
    for i in range(len(companies)-1):
        ind=bold_txt.index(companies[i])
        ind1=bold_txt.index(companies[i+1])
        #print(companies[i],"--")
        #print(ind, ind1)
        #print('----------')
        ex=[]
        try:
            y=extract_year(bold_txt[ind:ind1])
            if len(exp)==0:
                exp.append({"company":companies[i], "start":y['start'],"end":y['end']})
            elif y['end']<=exp[-1]['start']:
                    exp.append({"company":companies[i], "start":y['start'],"end":y['end
        except: -1
    try:
```

```
        ind=bold_txt.index(companies[i])
        y=extract_year(bold_txt[ind:ind+100])
        if y['end']<=exp[-1]['start']:
            exp.append({"company":companies[-1], "start":y['start'],"end":y['end']})
    except: -1
    return exp
```

Skills

The skills section of a resume shows employers, that the candidate has the abilities required to succeed in the role. Often, employers pay special attention to the skills section to determine who should move on to the next step of the hiring process.

In [22]:

```
def extract_skill_score(t, branch, skill):
    s=dict()
    for i in match_data["Skills"][branch][skill]:
        i=i.lower()
        try:
            c=t.count(i)
            if c:
                s[i]=c
        except:
            print(i)
    total_count=sum(s.values())
    if total_count>0:
        return {"count":total_count,"Skills": s}
    else:return -1
def extract_skill(t, pdf_path):
    skill_branch=list(match_data["Skills"].keys())
    skill={}
    skillset={}
    skill_dic=dict()
    total_count=0
    for i in skill_branch:
        skill={}
        for j in match_data["Skills"][i].keys():
            sk=extract_skill_score(t,i, j)
            if sk!=-1:
                skill_dic[j]=extract_skill_score(t,i, j)
                total_count+=skill_dic[j]["count"]
        if bool(skill):
            skillset[i]=skill

    for j,value in skill_dic.items():
        col_db=[]
        sk=value
        if sk!=-1:
            #print("Trying-.....................................", j)
            skill[j]=sk["Skills"].keys()
            col_skill=sk["Skills"].keys()
            cur.execute("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_
            for row in cur.fetchall():
                col_db.append(row[0].lower())
            col_to_add=set(col_skill)-set(col_db)
            if len(col_to_add)!=0:
                query="ALTER TABLE `{}` ADD (`"
                for k in list(col_to_add):
                    query+=k+"` float DEFAULT 0.0,`"
```

```
                query=query[:-2]+')'
                query1=query.format(j)
                query2=query.format(j+"_tfidf")
                cur.execute(query1)
                cur.execute(query2)
            if len(col_skill)!=0:
                insert_query="REPLACE INTO `{}` (Name1 ,".format(j)
                for k in col_skill:
                    insert_query+="`"+str(k)+'`, '
                insert_query=insert_query[:-2]+") VALUES ( \'{}\'  ,".format(pdf_path)
                for k in sk["Skills"].values():
                    insert_query+=str(k/total_count)+', '
                insert_query=insert_query[:-2]+')'
                #print(insert_query)
                r=cur.execute(insert_query)
                db.commit()
                #print("added to DB")
    return skillset

def skill_tfidf():
    skill_branch=list(match_data["Skills"].keys())
    for i in skill_branch:
        for j in match_data["Skills"][i].keys():
            tf_idf(j,j+"_tfidf")
    return 1
```

Check for Mis-spelled words in Candidate's Resume

This function recommends the candidate to check the spellings of the words which may be mis-spelled.

In [23]:
```
def check_spelling(txt):
    spell=SpellChecker()
    words=spell.split_words(txt)
    misspelled=spell.unknown(words)
    sentence=[spell.correction(word) for word in words]
    return misspelled
```

Candidate-Job Description Matching

The functions match the candidate's profile and the job description of the recruiting firm and score accordingly.

In [24]:
```
def find_jaccardDistance(s1,s2):
    dis = nltk.jaccard_distance(s1, s2)
    return dis
c1=dict()
def compare_skills_jd_Resume(list_jd, list_cv):#pass list===skills/software list
    intrsection=list_jd.intersection(list_cv)
    dif=list_jd.difference(list_cv)
    matchScore=intrsection/list_jd*100
    return intrsection,matchScore, dif
#dif==additional skills/softwares required
#func returns all skills that matched, Match Percentage, Additional skills required
```

Supplementary Functions for processing

In [25]:
```
def count_words(txt):
    return int(len(txt.split()))
def count_freq(txt):
    L_AB=[txt,""]
    cv=CountVectorizer(ngram_range=(1,1), analyzer='word')
    cv_train=cv.fit(L_AB)
    ngram=cv.transform(L_AB)
    f=ngram.toarray()
    freq= dict(zip(cv_train.get_feature_names(), [int(i) for i in f[0]]))
    max_freq=dict(sorted(freq.items(), key=operator.itemgetter(1),reverse=True))
    max_freq=dict(list(max_freq.items())[0: 10])
    return max_freq
def tokenize_resume(txt):
    word_tokens=word_tokenize(txt)
    stop_words = set(stopwords.words('english'))
    t= [w for w in word_tokens if not w in stop_words]
    return list(set(t))
```

Main function to extract data from resume

This function calls all the seperate modules responsible for extracting all the necessary details from the Resume and Social Media Accounts to be stored in the database and to be used for further analysis.

In [26]:
```
def create_user_json(pdf_path):
    data={}
    t,t_list=extract_resume_text(pdf_path)
    data["No. of Words"]=count_words(t)
    data["Personal Details"],remove_list=extract_personalDetails(t)
    data["Patents"]=get_patent_details(t, data["Personal Details"]['Name'], pdf_path)
    try:
        data["Qualifications"]=extract_qualifications_Tabular(pdf_path)
    except:
        try:
            data["Qualifications"]=extract_qualifications_nonTabular(t)
        except:print("Qualification Error")
    ptxt=preprocess_text(t)#preprocessed txt
    data["Competencies"]=extract_competencies(ptxt)
    wpd_txt=ptxt
    for i in remove_list:
        wpd_txt=wpd_txt.lower().replace(i.lower(),"")#wpd_txt=txt without personal deta
    data["Max used Words"]=count_freq(wpd_txt)
    term_freq(wpd_txt, pdf_path)
    data["Word Cloud Path"]=create_wordcloud(wpd_txt,"C:\\Users\\shivam\\Desktop\Word-C
    try:
        data["Psychograph"], score=create_psychograph(t, "C:\\Users\\shivam\\Desktop\ps
    except:print("Psychograph Error")
    try:
        docx_file = pdf_path.replace('.pdf','.docx')
        parse(pdf_path, docx_file, start=0)
        soup=convert_doc_to_soup(docx_file)
        bold_list=extract_bold_from_soup(soup)
        companies=extract_companies(bold_list)
        print(companies)
        bold_txt=" ".join(bold_list)
        data["Experience"]=search_exp(bold_txt,companies)
```

```
    except: print("Experience Error")
    try:
        data["Useful Links"]=extract_links(soup)
        data["Profile Links"]=extract_profiles(data["Useful Links"])
        data["Certifications"]=extract_certifications(data["Useful Links"])
    except:print("Links Error")
    data["Skills"]=extract_skill(t)
    remove_list1=[]
    for i in data['Skills'].keys():
        for j in data['Skills'][i].keys():
            remove_list1+=data['Skills'][i][j]["Skills"]
    mispelled=check_spelling(wpd_txt)
    data["Mispelled Words"]=list(set([_.lower() for _ in mispelled]).difference(set([_.
    #data["Job Recommendations"]=list(best_job(t))
    try:
        with open("data.json", "w") as outfile:
            json.dump(data, outfile)
    except: print("json Error")
    return data
```

Machine Learning in Action!!

After extraction and verification of all the details of the candidate, the next part that comes into picture is the data modelling phase.

Step 1: Gathering Data

Creation of Database of Resume Corpus

- *Mine Resume data of different candidates from resume corpus directory and save the extracted data to MySQL Database 'recruitize'.*

In [ ]:

```
start = time.time()
db = MySQLdb.connect(host="localhost",    # your host, usually localhost
                     user="root",         # your usernametcs
                     passwd="root",  # your password
                     db="recruitize")
os.chdir("C:\\Users\hp\\desktop\\Recruitize\\resume")
f=[]
for file in glob.glob("*.pdf"):
    f.append(file)
scan_s=0
scan_us=0
for i in f:
    try:
        pdf_path=r'C:\\Users\\hp\\Desktop\\Recruitize\\resume\\'+i
        print(pdf_path)
        try:
            txt=preprocess_text(extract_resume_text(pdf_path)[0])
```

```
            print("PDF scan Successful")
            scan_s+=1
        except:
            scan_us+=1
        term_freq(txt, pdf_path)
        extract_skill(txt, pdf_path)
        extract_competencies(txt, pdf_path)
    except:print("Error",pdf_path)
end=time.time()
print("Time Taken=",(end-start))
db.close()
print("Successful=",scan_s)
print("Unsuccessful=",scan_us)
```

In [173…
```
tf_idf("term_freq", "tfidf")
competency_tfidf()
skill_tfidf()
```

In [30]:
```
def getTables():
    tables=[]
    tab=cur.execute("show tables")
    tab1=cur.fetchall()
    for i in tab1:
        if "tfidf" in i[0]:
            tables.append(i[0])
    return tables
tables=getTables()
```

In [ ]:
```
for i in table_sum:
    q="drop table `{}`".format(i)
    #print(q)
    cur.execute("drop table {}".format(i))
```

In [31]:
```
def getSumOfTables():
    table_sum=[]
    for i in tables:
        col_db=[]

        cur.execute("SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAM
        cols=cur.fetchall()[1:]
        if len(cols)>0:
            for row in cols:#[1:] for slicing to skip Name1
                col_db.append(row[0])
            s='`+`'
            try:
                inner_query="create table if not exists `{}` as (select Name1, (`{}`) a
                table_sum.append("`sum_"+i[:i.index('_')]+"`")
            except:
                inner_query="create table if not exists `{}` as (select Name1, (`{}`) a
                table_sum.append("`sum_"+i+"`")
            try:
                cur.execute(inner_query)
            except:print("e")
    return table_sum
table_sum=getSumOfTables()
```

To get sum of individual score of all tables and combine them using LEFT OUTER JOIN for Analysis.

In [32]:
```
query="SELECT *FROM `sum_data science`"
```

```
for i in table_sum[1:]:
    if "data science" not in i:
        query+= " LEFT JOIN {} ON {}.Name1={}.Name1".format(i, "`sum_data science`",i)
create_query="create table Data as ("+query+")"
#print(query)
```

In [156…

```
from sqlalchemy import create_engine
import pandas as pd

db_connection_str = 'mysql+pymysql://root:root@localhost/recruitize'
db_connection = create_engine(db_connection_str)
df = pd.read_sql(query, con=db_connection)
Name_column=df.iloc[:, 0]
df.drop('Name1', axis = 1, inplace = True)
df=df.fillna(0)
df
```

Out[156…

	data science	adhering to principles and values	administration	androiddevelopment	application servers	artifical intelligence	creative and innovative	str a
0	0.025271	0.000000	0.0	0.0	0.213187	0.000000	0.271723	0.
1	0.056497	0.269767	0.0	0.0	0.166667	0.000000	0.717162	0.
2	0.046200	0.000000	0.0	0.0	0.195750	0.000000	0.165423	0.
3	0.035804	0.658156	0.0	0.0	0.393891	0.000000	0.102070	0.
4	0.052595	0.658156	0.0	0.0	0.201434	0.000000	0.102070	0.
…	…	…	…	…	…	…	…	
207	0.011628	0.031133	0.0	0.0	0.225478	0.000000	0.345744	0.
208	0.018018	0.000000	0.0	0.0	0.214901	0.000000	0.863687	0.
209	0.037505	1.327945	0.0	0.0	0.256196	0.000000	0.564095	0.
210	0.048276	0.079890	0.0	0.0	0.204147	0.007088	0.417692	0.
211	0.031652	0.000000	0.0	0.0	0.226802	0.000000	6.284998	0.

212 rows × 36 columns

Step 2: Preparing that Data

- *After gathering the data and converting it to a requisite form the data is preprocessed to that it can be fed to a suitable Machine Learning Model.*
- *As we created the data by our own the data is processed to a great extent but less us check if any further processing is required.*

In [157…

```
df.describe()
```

Out[157…

	data science	adhering to principles and values	administration	androiddevelopment	application servers	artifical intelligence	creativ and innovativ
count	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.00000
mean	0.066926	0.084290	0.000022	0.000336	0.227042	0.005765	0.89938
std	0.112737	0.241445	0.000314	0.001723	0.049061	0.027136	1.76543
min	0.006536	0.000000	0.000000	0.000000	0.154185	0.000000	0.00000
25%	0.026138	0.000000	0.000000	0.000000	0.199307	0.000000	0.22467
50%	0.037399	0.000000	0.000000	0.000000	0.216481	0.000000	0.51950
75%	0.055725	0.030160	0.000000	0.000000	0.236880	0.000000	0.97723
max	1.177011	1.757574	0.004566	0.015734	0.456919	0.264286	19.81336

8 rows × 36 columns

- The data seams to have a varied range hence needs normalization.
- Let us check for empty columns

In [159…
```
df.mean()
```

Out[159…
```
data science                          0.066926
adhering to principles and values     0.084290
administration                        0.000022
androiddevelopment                    0.000336
application servers                   0.227042
artifical intelligence                0.005765
creative and innovative               0.899384
data structures and algo              0.002279
database                              0.001733
decision making                       0.202735
devops                                0.021032
drafting software                     0.005516
enterpreneurial thinking              0.529716
environmental engineering software    0.004851
ide                                   0.010430
influencing and pursuading            0.097833
leadership quality                    0.863471
machine learning                      0.649567
management                            2.777710
networking and relating               0.200246
operating systems                     0.019211
planning and organising               0.215896
programming languages                 0.202765
software                              0.027644
softwaredevelopment                   0.001820
structural analysis software          0.009604
sum                                   0.000000
surveying and gis software            0.013628
technologies  and frameworks          0.024715
testing                               0.000041
tfidf_sum                             0.000000
virtualization                        0.005039
```

```
water resources/ irrigation/ hydrology software    0.009259
web development (back end)                         0.004382
web development (front end)                        0.007856
working with people                                0.190203
dtype: float64
```

Drop Empty/Null columns

- *The data contains 2 empty columns which need to be removed from the dataframe.*

In [160…
```
df.drop(['sum','tfidf_sum'], axis = 1, inplace = True)
```

In [161…
```
df.describe()
```

Out[161…

	data science	adhering to principles and values	administration	androiddevelopment	application servers	artifical intelligence	creativ and innovativ
count	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.00000
mean	0.066926	0.084290	0.000022	0.000336	0.227042	0.005765	0.89938
std	0.112737	0.241445	0.000314	0.001723	0.049061	0.027136	1.76543
min	0.006536	0.000000	0.000000	0.000000	0.154185	0.000000	0.00000
25%	0.026138	0.000000	0.000000	0.000000	0.199307	0.000000	0.22467
50%	0.037399	0.000000	0.000000	0.000000	0.216481	0.000000	0.51950
75%	0.055725	0.030160	0.000000	0.000000	0.236880	0.000000	0.97723
max	1.177011	1.757574	0.004566	0.015734	0.456919	0.264286	19.81336

8 rows × 34 columns

Data Normalization

- *The data needs normalization hence we use MinMaxScaler to normalize the data and make the raneg of each column between 0-1*

In [162…
```
col=df.columns
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(df)
df_normalized = pd.DataFrame(x_scaled)
df=df_normalized
df.columns=col
```

In [163…
```
df.describe()
```

Out[163…

	data science	adhering to principles and values	administration	androiddevelopment	application servers	artifical intelligence	creativ and innovativ

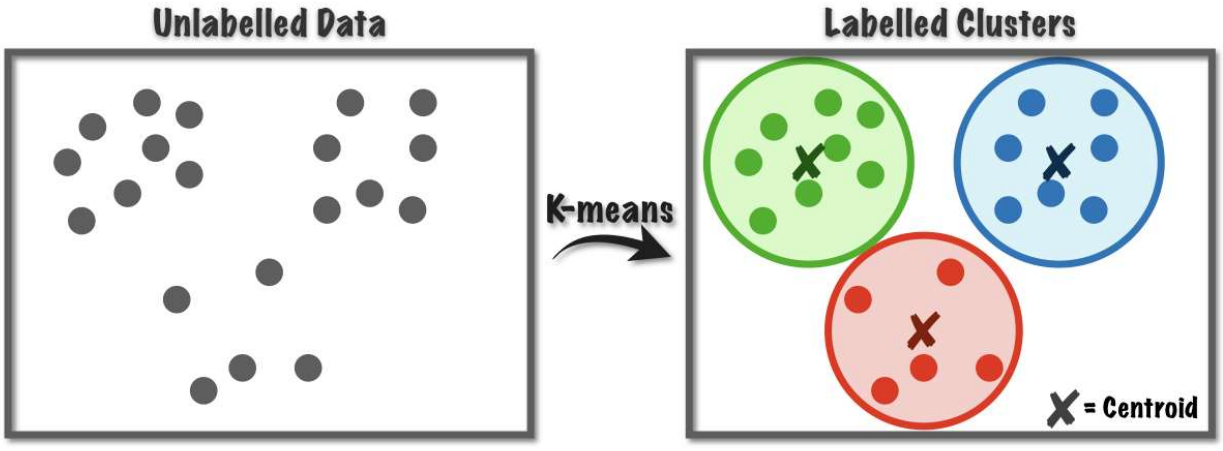
	data science	adhering to principles and values	administration	androiddevelopment	application servers	artifical intelligence	creativ and innovativ
count	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000
mean	0.051595	0.047958	0.004717	0.021370	0.240662	0.021814	0.045393
std	0.096318	0.137374	0.068680	0.109521	0.162059	0.102675	0.089103
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.016747	0.000000	0.000000	0.000000	0.149049	0.000000	0.011339
50%	0.026368	0.000000	0.000000	0.000000	0.205779	0.000000	0.026220
75%	0.042025	0.017160	0.000000	0.000000	0.273159	0.000000	0.049322
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 34 columns

Step 3: Choosing a Model

- *The selection of the model type is our next course of action once we are done with the data-centric steps.*
- *As we lack labelled data for Supervised Learning we will go with the Unsupervised Learning Approach.*
- *We use K-Means Clustering to group the candidates according to their common traits.*

K-Means Clustering



Finding the Optimum Cluster using Elbow method

In [143…

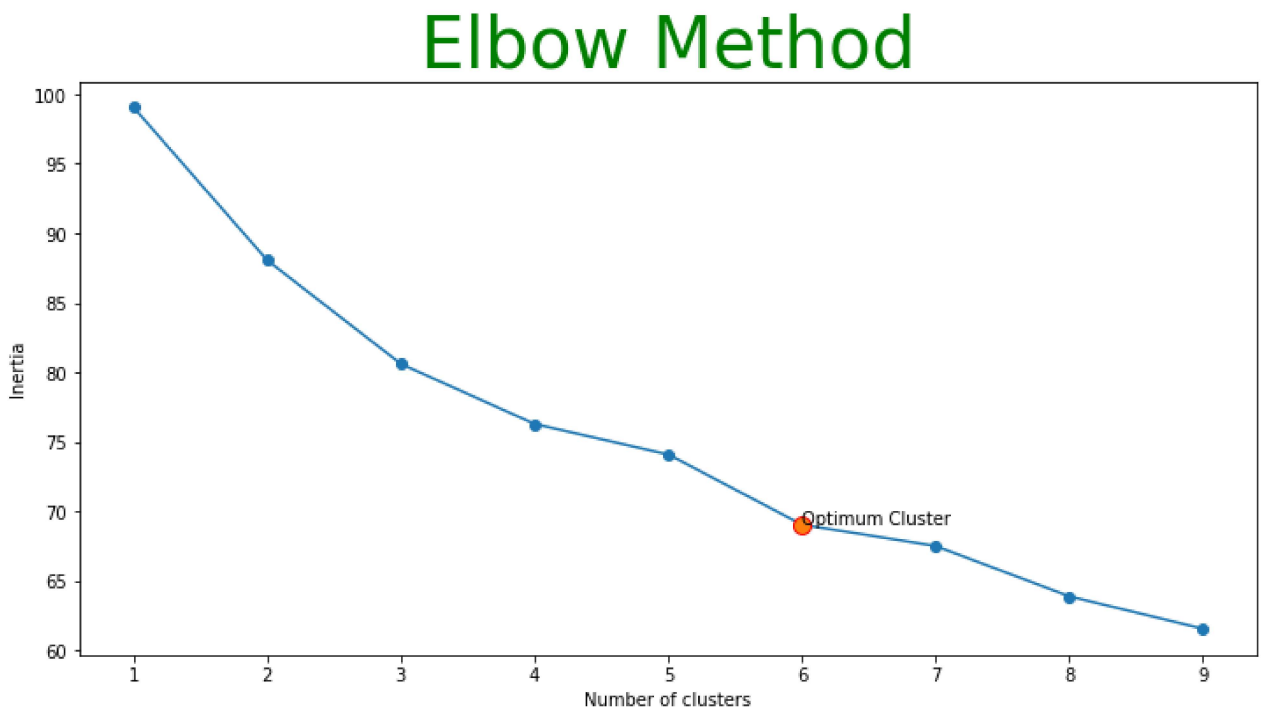
```
from kneed import DataGenerator, KneeLocator
SSE = []
```

```
max_cluster=10
for cluster in range(1,max_cluster):
    kmeans = KMeans(n_jobs = -1, n_clusters = cluster, init='k-means++')
    kmeans.fit(df)
    SSE.append(kmeans.inertia_)
# converting the results into a dataframe and plotting them
frame = pd.DataFrame({'Cluster':range(1,max_cluster), 'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')


k1 = KneeLocator(range(1, 10), SSE, curve="convex", direction="decreasing")
elb=k1.elbow
plt.plot(elb, frame['SSE'][elb-1], marker='o', ms=10,mec='r')
plt.annotate("Optimum Cluster", (elb, frame['SSE'][elb-1]))

plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title("Elbow Method",fontsize=40,
          color="green")
```

Out[143… `Text(0.5, 1.0, 'Elbow Method')`



Optimum no. of Clusters=6 </h1>

Applying K-Means++ Clustering using scikit-learn

In [144…

```
kmeans = KMeans(n_clusters = elb, init='k-means++')
kmeans.fit(df)
pred = kmeans.predict(df)
df["sum"] = df.sum(axis=1)
inertia=kmeans.inertia_
X_dist = kmeans.transform(df)**2
cen_dis=list(np.min(X_dist, axis=1))
```

```
cluster_group=pd.DataFrame({'Name':Name_column,'cluster':pred, "squared centroidal dist
cluster_group=cluster_group.sort_values("squared centroidal distance")
print(cluster_group['cluster'].value_counts())
```

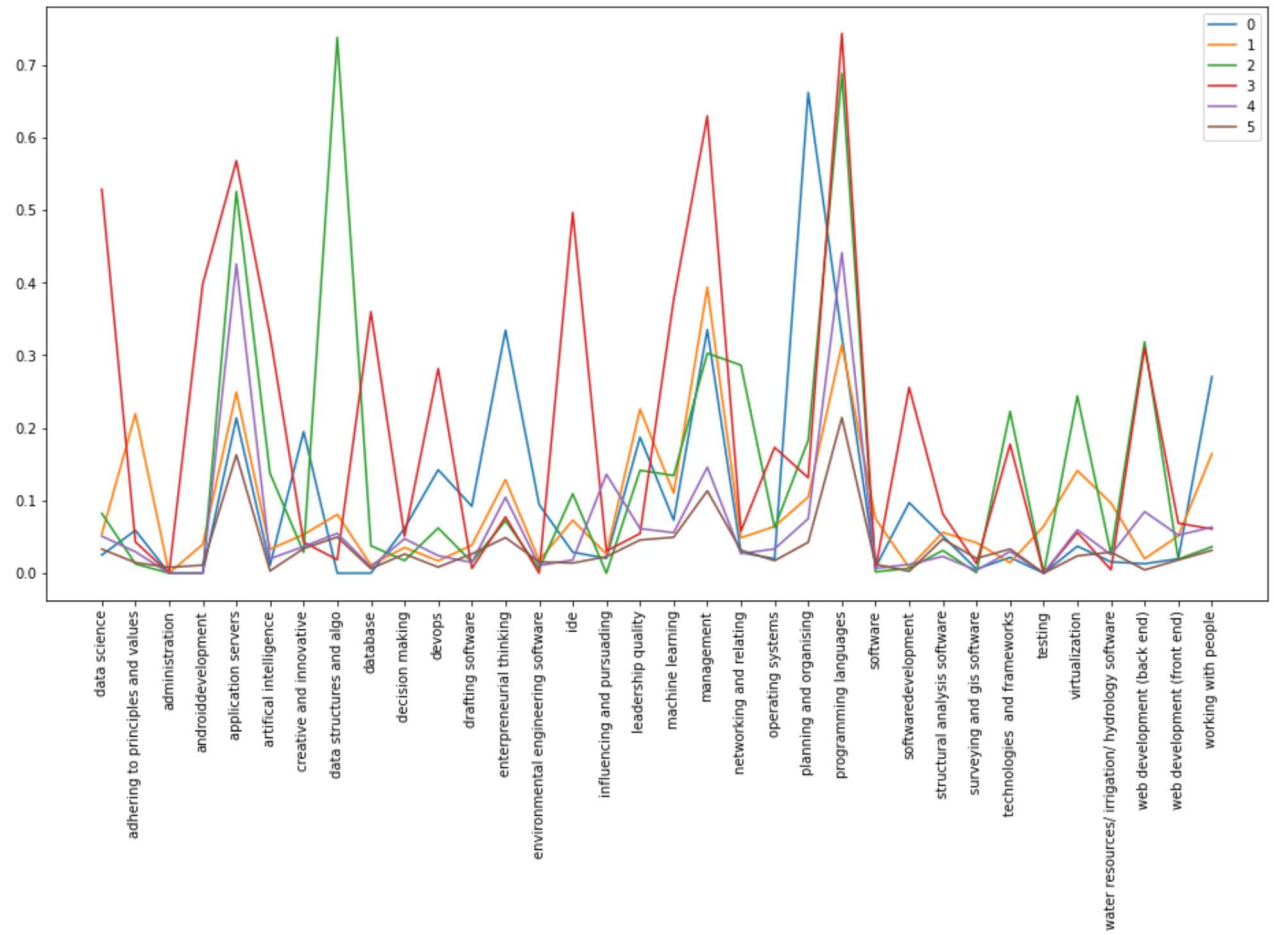
```
5    126
4     35
1     29
0     11
2      6
3      5
Name: cluster, dtype: int64
```

Step 4: Visualizing the Clusters and Trait Prediction

- *After Clustering the candidates the next important task that comes is to make sense out of those clusters.*
- *If we cannot get insights about every cluster then the whole modeling process is useless.*
- *Therefore it is a very important step to be performed.*

In [166…

```
centroids=kmeans.cluster_centers_
f = plt.figure()
f.set_figwidth(16)
f.set_figheight(8)
plt.xticks(rotation='vertical')
col=list(df.columns)
for i in range(elb):
    plt.plot(col, centroids[i])
plt.legend(range(elb))
plt.show()
```



- *The above line graph shows the relevant traits/characteristics for each cluster and hence represent the quality of each candidate present in that cluster.*

Step 5: Grouping each candidates according to their clusters

- *Let us take a glimpse of the clusters classified by the model and analyse them to make sense out of each cluster.*
- *Smaller the "squared centroidal distance", More is the alignment of the candidate's traits to the cluster characteristics*

In [219…

```
cluster_group=pd.DataFrame({'Name':Name_column,'cluster':pred, "squared centroidal dist
cluster_group=cluster_group.sort_values("squared centroidal distance")
gb = cluster_group.groupby(['cluster'])
g=[gb.get_group(x) for x in gb.groups]
```

- *Let us have a look on some of the clusters and their alignment with reality.*

Cluster 2

According to the model 'Cluster 2' contains 6 candidates who are skilled in data structures, application servers, programming languages, frameworks, and good in Management.

In reality also these people are from Computer Science background and have worked as an Intern in Keysight Technologies and hence are well versed with application servers, frameworks etc.

In [220…
```
g[2]
```

Out[220…

	Name	cluster	squared centroidal distance
101	C:UsershpDesktopRecruitize\resume1700741_EE_CS...	2	0.249290
198	C:UsershpDesktopRecruitize\resumeShivamPachaur...	2	0.383776
194	C:UsershpDesktopRecruitize\resumeShashankUpadh...	2	0.474382
13	C:UsershpDesktopRecruitize\resume166243_EE_COR...	2	0.618816
161	C:UsershpDesktopRecruitize\resumeGarima_Kumar_...	2	1.026982
206	C:UsershpDesktopRecruitize\resumeTushar_Varshn...	2	1.111601

Cluster 3

- *According to the model 'Cluster 3' contains 6 candidates who are highly enthusiastic Data Scientists having a good knowledge of databases, application servers, android, programming, devops and many other domains and have excellent management skills.*
- *In reality also these people are from Computer Science Background and have worked on various projects related to Machine Learning, Data Science etc.*

In [221…
```
g[3]
```

Out[221…

	Name	cluster	squared centroidal distance
71	C:UsershpDesktopRecruitize\resume1700702_Adars...	3	0.473697
11	C:UsershpDesktopRecruitize\resume166238_RidhiD...	3	1.087643
69	C:UsershpDesktopRecruitize\resume1700701_Aadha...	3	1.258238
54	C:UsershpDesktopRecruitize\resume1700668_CE_CS...	3	1.434623
72	C:UsershpDesktopRecruitize\resume1700702_EE_CS...	3	1.540092

Cluster 0

- *According to the model 'Cluster 0' contains 11 candidates who are supposed to be young entrepreneurs with a creative mind and good planning and organizing skills who like to collaborate with people and have excellent management skills.*
- *In reality also, these people belong to non-CS background and most of them are placed for Management and Sales jobs*

In [222…
```
g[0]
```

Out[222…

	Name	cluster	squared centroidal distance

	Name	cluster	squared centroidal distance
131	C:UsershpDesktopRecruitize\resume1700811 CE CO...	0	0.176891
189	C:UsershpDesktopRecruitize\resumeSahaj Kumar S...	0	0.200181
186	C:UsershpDesktopRecruitize\resumeRevised CT2 S...	0	0.214624
156	C:UsershpDesktopRecruitize\resumeAshishparasha...	0	0.288756
133	C:UsershpDesktopRecruitize\resume1700821_Vikas...	0	0.340578
147	C:UsershpDesktopRecruitize\resume1700882_RaniP...	0	0.432648
168	C:UsershpDesktopRecruitize\resumeMayank Agarwa...	0	0.916784
146	C:UsershpDesktopRecruitize\resume1700879 P V G...	0	0.925822
151	C:UsershpDesktopRecruitize\resume1700905_T CHI...	0	1.073354
150	C:UsershpDesktopRecruitize\resume1700903_Srash...	0	1.291751
182	C:UsershpDesktopRecruitize\resumeResume_Abhish...	0	1.306408

Future Scope

1. *The modelling is Done for only Computer Science Background Candidates, The model can be extended by adding the details of candidates belonging to varying fields.*
2. *A larger Resume Corpus can be used to increase the model efficiency.*
3. *Tagged Resume Dataset can be used to create Bayesian Models and build supervised models.*
4. *Further K-Nearest Neighbour can be used to classify every new resume added.*