Extract text from resume

```
#!pip install sentence_transformers
```

```
import string
import re
import nltk
import statistics
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sentence_transformers import SentenceTransformer
from tabulate import tabulate
import matplotlib.pyplot as plt
import numpy as np
import math
```

```
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
score_array=[]
top_institutes = ["IIT","NIT","IIIT","BITS"]
equvalent_courses = ["B.E","MSC","MCA","B.Tech","Bachelors","Information science","
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Data cleaning

```
def start(actual,expected):
  removeStopwords(actual,expected)
```

```
def checkTopInstitutes(education,requiredEducation):
  for institute in top_institutes:
    if institute in education:
      education = requiredEducation
      break
    else:
      education.join(',')
```

```
def checkEquvalentCourses(education,requiredEducation):
```

```
    for course in equvalent_courses:
      if course in education:
        education = requiredEducation
      else:
        education.join(',')


  def removeStopwords(actual,expected):
    stop_words = set(stopwords.words('english'))
    candidateExperienceTokens = word_tokenize(actual)
    requiredExperienceTokens = word_tokenize(expected)
    cleanCandidateExperience = [word for word in candidateExperienceTokens if not wor
    cleanrequiredExperience = [word for word in requiredExperienceTokens if not word
    lemmatisation(cleanCandidateExperience,cleanrequiredExperience)


  def lemmatisation(actual,expected):
    CandidateExperience = ' '.join(map(str, actual))
    RequiredExperience = ' '.join(map(str, expected))
    lemmatizer = WordNetLemmatizer()
    lemCandidateExperience =lemmatizer.lemmatize(CandidateExperience)
    lemRequiredExperience = lemmatizer.lemmatize(RequiredExperience)
    removePunctuation(lemCandidateExperience,lemRequiredExperience)


  def removePunctuation(actual,expected):
    noPunCandidateExperience = ""
    noPunRequiredExperience = ""
    for character in actual:
        if character.isalnum():
            noPunCandidateExperience += character
        else:
          noPunCandidateExperience += " "

    for character in expected:
        if character.isalnum():
            noPunRequiredExperience += character
        else:
          noPunRequiredExperience += " "
    model(noPunCandidateExperience,noPunRequiredExperience)


  def drawPieChart():
    y=[]
    for i in range(len(score_array)):
      if(i==2):
        y.append(score_array[i]*1.5)
      else:
        y.append(score_array[i])
    mylabels = ["Skills", "Education", "Experience"]
    explode = (0, 0, 0.1 )
    plt.pie(y, labels = mylabels, explode=explode, shadow=True)
    plt.title("Distribution of attributes.")
    plt.show()


  def analytics():
```

```
def analytics():
  print("\n")
  print("REPORT")
  final_score=0
  for score in score_array:
    final_score+=score
  avg = (final_score/300)*100
  score_array.append(avg)
  s1=score_array[0]/2
  s2=score_array[1]/2
  s3=score_array[0]
  resume_score = ((s1+s1+s3)/200)*100
  score_array.append(math.floor(resume_score))
  if(avg>min_criteria):
    score_array.append("PASSED")
  elif(avg>=70 and avg<min_criteria):
    score_array.append("Manual review needed")
  else:
    score_array.append("FAILED")
  print(tabulate([score_array], headers=["Skills", "Education", "Experience","AVG",
  #clear data
  score_array.clear()
```

Modelling

```
def model(actual,expected):
  vectorizer = TfidfVectorizer()
  vectorizer.fit([actual])
  vectorizer.fit([expected])
  vectorA = vectorizer.transform([actual])
  vectorB = vectorizer.transform([expected])
  similarity_index = cosine_similarity(vectorA, vectorB)
  final_score = similarity_index[0][0]*100
  score_array.append(int(final_score))
```

```
def sentenceTranformerModel(actual,expected):
  modelName = "bert-base-nli-mean-tokens"
  model = SentenceTransformer(modelName)
  vectorA = model.encode([actual])
  vectorB = model.encode([expected])
  similarity_index = cosine_similarity(vectorA, vectorB)
  final_score = similarity_index[0][0]*100
  score_array.append(int(final_score))
  #print(tabulate([score_array], headers=["Skills", "Education", "Experience","AVG"
```

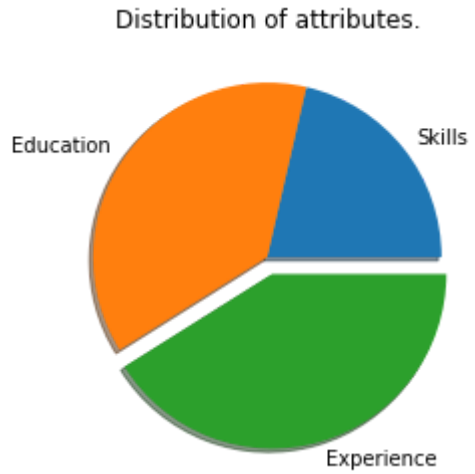
Analysis

Make changes below to describe what is needed.

```
#Candidate
skills = "JAVA, SPRING BOOT, Hybernate, SQL, DBMS, Angular, GIT, AI/ML, Jira, Jenki
education = "BE in computer science from BMS"


#Organisation
requireTopTierEducation = False
min_criteria = 80
requiredSkills = "Java, REACT JS, Spring, Hibernate, CI/ CD, Docker, Kubernetes, Pu
if(requireTopTierEducation):
  requiredEducation = "BE in computer science from" + ' '.join([str(elem) for elem
  checkTopInstitutes(education,requiredEducation)
else:
  requiredEducation = "BE in computer science "
  checkEquvalentCourses(education,requiredEducation)


requiredExperience = "Deep expertise and hands on experience in Core java. · Hands-

candidateExperience = "Have Over 6+ years of experience in core java, Spring and Hi
```

Trigger Run

```
start(skills,requiredSkills)
start(education,requiredEducation)
sentenceTranformerModel(candidateExperience.lower(),requiredExperience.lower())
drawPieChart()
analytics()
```




```
REPORT
  Skills    Education    Experience      AVG    Resume score  Automatic screer
--------  -----------  ------------  -------  --------------  ----------------
      57          100            73  76.6667              56  Manual review ne
```

```
# Templates
# Example 1
# Microsoft
# requiredSkills = "JAVA/C, SPRING BOOT, Hybernate, SQL, DBMS, Angular, node.js"
```

```
# requiredSkills = "Java/J, Spring Boot, Hybernate, SQL, DBMS, Angular, Node.js

# candidateExperience = "4 years of software design and development experience in d

# requiredExperience = " 3+ years of software design and development experience of
# ------------------------------------------------------------------------------
# Example 2
# Hotstar
# requiredSkills = "JAVA, SPRING BOOT, Hybernate, SQL, DBMS, Angular, GIT, AI/ML, J
# requiredExperience = "4-8 years of experience in software development with strong

# candidateExperience = "Have 3 years of development experiance in working with rea

# ------------------------------------------------------------------------------
# Example 3
# Amazon
# requiredSkills = "Java, REACT JS, distributed, multi-tiered systems, algorithms,

# requiredExperience = "Currently enrolled in a Bachelor’s or Master’s Degree in Co

# candidateExperience = "Pursuing Bachelors Degree in Information science. profecie
# ------------------------------------------------------------------------------
# Example 4
# Societe generale

# requiredSkills = "Java, REACT JS, Spring, Hibernate, CI/ CD, Docker, Kubernetes,

# requiredExperience = "Deep expertise and hands on experience in Core java. · Hand

# candidateExperience = "Have Over 6+ years of experience in core java, Spring and
```