```
import config
import glob
import numpy as np
import matplotlib.pyplot as plt
from FunctionsForJobRecommendation import FunctionsForJobRecommendation
import os
import json

def main():
    # The data scraped from web is obtained from reference dataset which is stored in JSON file
    exists = os.path.isfile(config.JOBS_INFO_JSON_FILE)
    if exists:
        with open(config.JOBS_INFO_JSON_FILE, 'r') as fp:
            JobsInfo = json.load(fp)

    # Initialize skill_keyword_match with JobsInfo
    skill_match = FunctionsForJobRecommendation(JobsInfo)

    # Extract skill keywords from job descriptions
    skill_match.ExtractJobDescKeywords()

    # Extract resume skills from given resume and store them in a list
    for resumePDF in glob.glob(config.SAMPLE_RESUME_PDF_DIR+"SampleResume*.pdf"):
        print("=====================================================")
        print("Processing the resume : ",resumePDF)
        print("=====================================================")
        ResumeSkills = skill_match.ExtractResumeKeywords(resumePDF)
        ResumeSkills.reset_index(inplace=True)
        ResumeSkills.rename(columns={'index': 'skillsinresume'}, inplace=True)
        ResumeSkillList = ResumeSkills['skillsinresume'].tolist()
```

```
resume_skill_list_dummy =
['azure','sql','mysql','c++','excel','power','keras','agile','r','tableau','google']


print("Skills extracted from resume are : \n",ResumeSkillList)


# Calculate similarity of skills from a resume and job post and get top10 job descriptions

MainTop10JDs = skill_match.CalculateSimilarity(ResumeSkillList)

# copy of the dataframe as "MainTop10JDs2" to keep them different for static and dynamic
approach

MainTop10JDs2 = MainTop10JDs.copy()


# Extract 20 similar Job description for each of the top10 job descriptions

# Explicit and Implicit skills extracted for static weight approach

ImplicitStatic,finalSkillWeightList = skill_match.Extract20SimilarJDs(0,MainTop10JDs,
ResumeSkillList)


# Calculating Final cosine score based on term frequency and weighted cosine similarity

FinalJDPrev = skill_match.WeightedCosineSimilarity(ResumeSkillList, ImplicitStatic)

print("Below is the reference approach job listing ranking\n",FinalJDPrev[['Jobid','final_cosine']])


# Extract 20 similar Job description for each of the top10 job descriptions

# Explicit and Implicit skills extracted for dynamic weight approach

ImplicitDynamic,finalSkillWeightList = skill_match.Extract20SimilarJDs(1,MainTop10JDs2,
ResumeSkillList)

# Calculating Final cosine score based on term frequency and weighted cosine similarity


FinalJD = skill_match.WeightedCosineSimilarity(ResumeSkillList, ImplicitDynamic)

print("Below is the proposed approach job listing ranking\n",FinalJD[['Jobid','final_cosine']])

topIndex = FinalJD['Jobid'][0]

allTopSkills = ImplicitDynamic.loc[topIndex]['keywords']


topExSkills = []
```

```
    topImpSkills = []
    for skill, weight in allTopSkills:
        if weight ==1:
            topExSkills.append(skill)
        else:
            topImpSkills.append(skill)
    print("Explicit skills to upskill : ",np.setdiff1d(topExSkills,ResumeSkillList))
    diffImpSkills = np.setdiff1d(topImpSkills,ResumeSkillList)
    if len(diffImpSkills)>5:
        print("Implicit skills to upskill : ",np.setdiff1d(topImpSkills,ResumeSkillList)[0:5])
    else:
        print("Implicit skills to upskill : ",np.setdiff1d(topImpSkills,ResumeSkillList))


    # Graph plot with explicit and implicit skills that match the resume for static approach
    ImplicitStaticGraph =
FinalJDPrev[["Jobid","final_cosine","exSkillCountResumeMatch","impSkillCountResumeMatch"]]
    # skill_match.GraphPlotsForEvaluation(ImplicitStaticGraph,finalSkillWeightList,0)


    # Graph plot with explicit and implicit skills that match the resume for dynamic approach
    # Graph plot to show how the ranking of the top10 job postings differ due to the Implicit
weightage of skills
    ImplicitDynamicGraph =
FinalJD[["Jobid","final_cosine","exSkillCountResumeMatch","impSkillCountResumeMatch"]]
    # skill_match.GraphPlotsForEvaluation(ImplicitDynamicGraph,finalSkillWeightList,1)
    if(resumePDF.count(r'SampleResume1') == 1):
        plt.figure()

skill_match.AllGraphPlotsForEvaluation(ImplicitStaticGraph,ImplicitDynamicGraph,finalSkillWeightLis
t,1)


if __name__ == "__main__":
  main()
```