- **Phase-1.** Data Scraping

- I have collected the data from Wuzuf website, and used (BeautifulSoup, requests, pandas) libraries in this phase.
- I have chosen some jobs titles (Data analyst, software engineer, web developer, flutter-developer, Accountant, Customer Service, Civil Engineer). We can add more titles if we want.
- While collecting the data I was careful to collect jobs for entry level people who have 0-1 years' experience.
- The data contains ['job_title','company_name','job_location','job_requirement','job_description'].
- Finally, I saved the data in csv file, and database by using SQLite library in python

- **Phase-2.** Data Analysis

1. Numbers of jobs opportunity for each title:

CUSTOMER SERVICE	120
ACCOUNTANT	81
WEB DEVELOPER	57
CIVIL ENGINEER	56
SOFTWARE ENGINEER	25
FLUTTER DEVELOPER	2

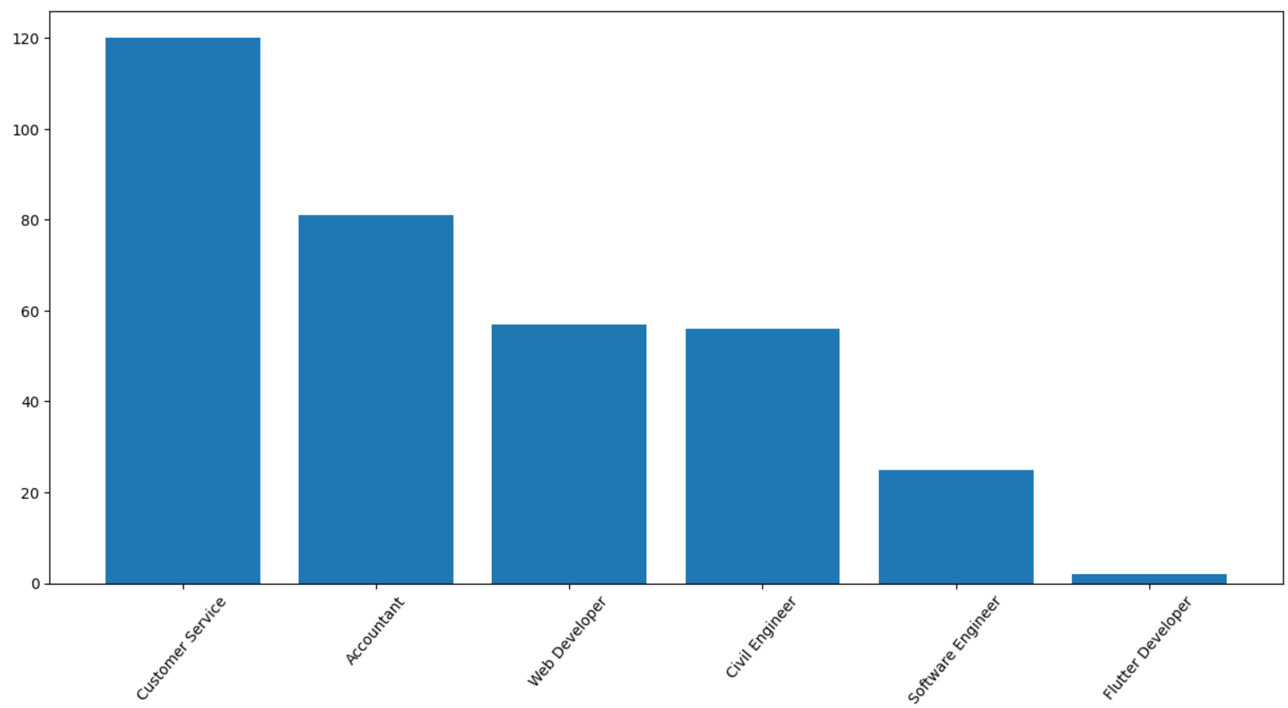
2. The preferred skills mentioned in job descriptions:

   (['xml', 'teamwork', 'statisticals', 'SQL', 'software', 'react', 'python', 'OOP', 'NoSQL', 'Node', '.Net', 'MVC Model', 'microsoft office', 'ML', 'listening', 'javascript', 'java', 'HTML', 'Git', 'financial', 'English', 'Databas', 'css', 'communication', 'C++', 'C#', 'Bachelor of CS', 'analysis', 'accountant'])

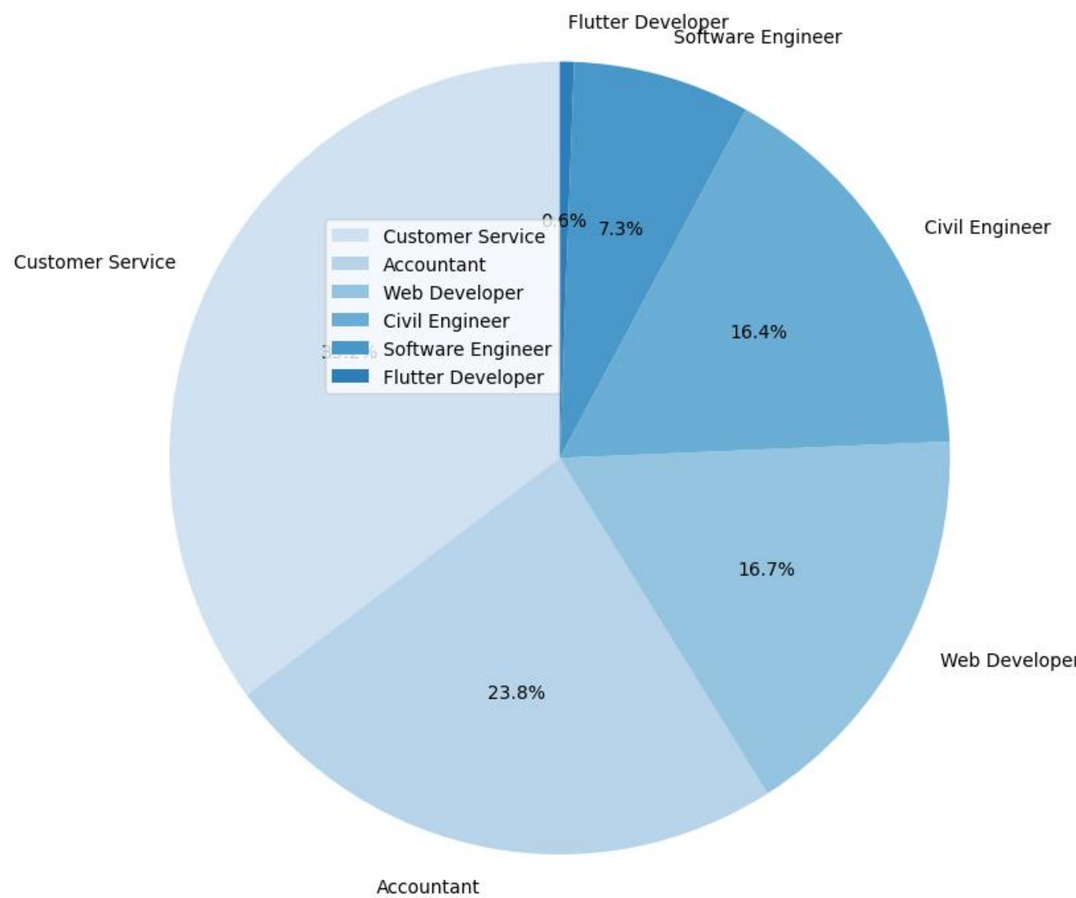
3. Geographic distribution of job opportunities.
   Descending sort for job opportunity in each city:

   ["cairo","giza","alexandria","sharqia","dakahlia","damietta","gharbia","minya","suef","assiut","beheira","said","sinai"]

- **Phase-3.** Data visualization.
  1. Most in demand job titles

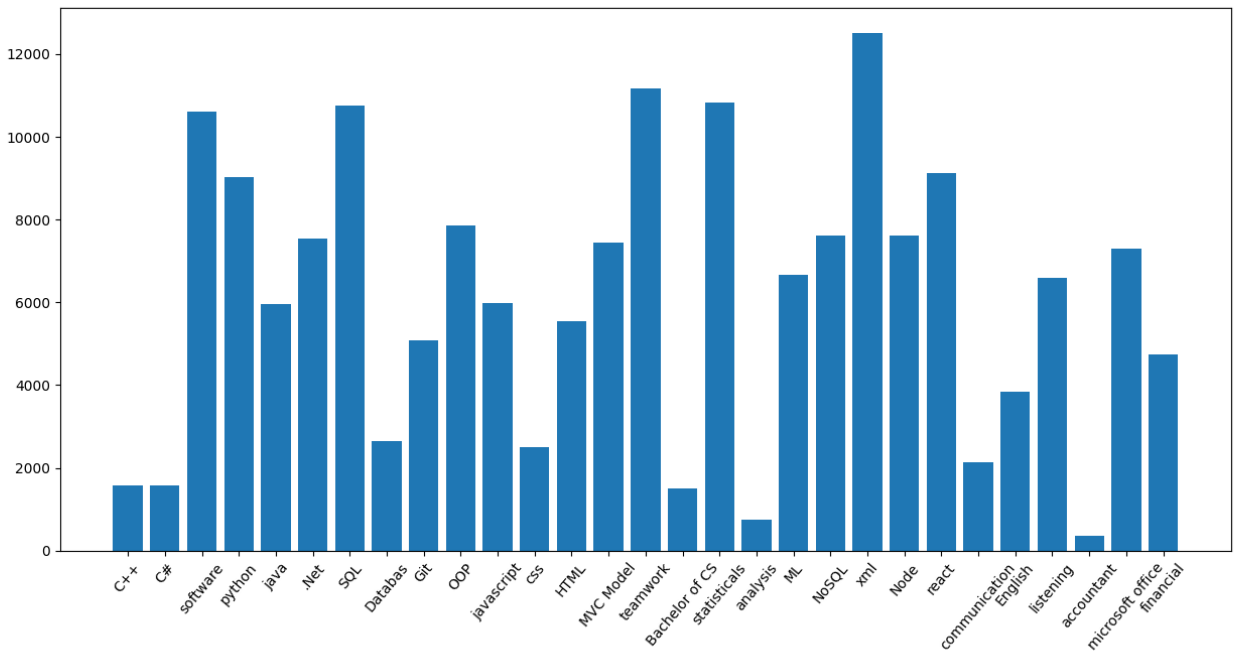


120
100
80
60
40
20
0
Customer Service
Accountant
Web Developer
Civil Engineer
Software Engineer
Flutter Developer

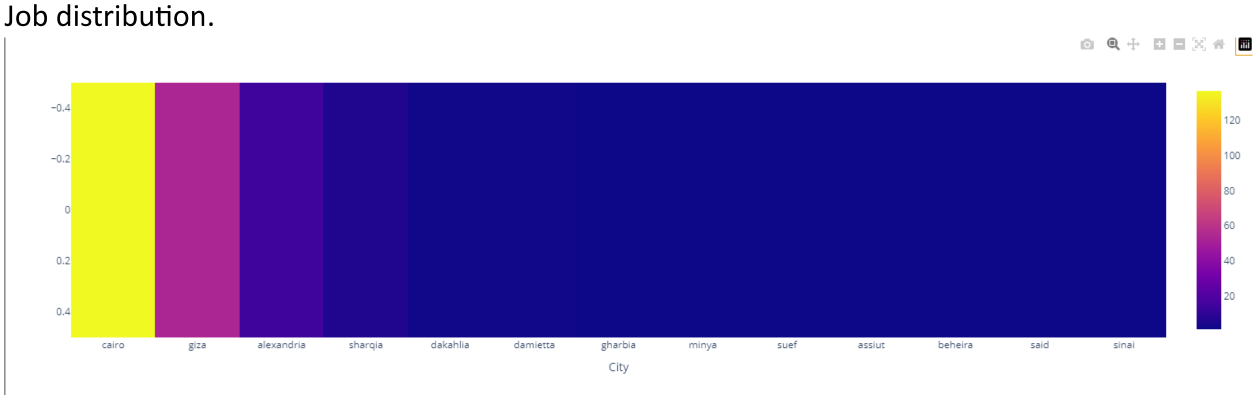


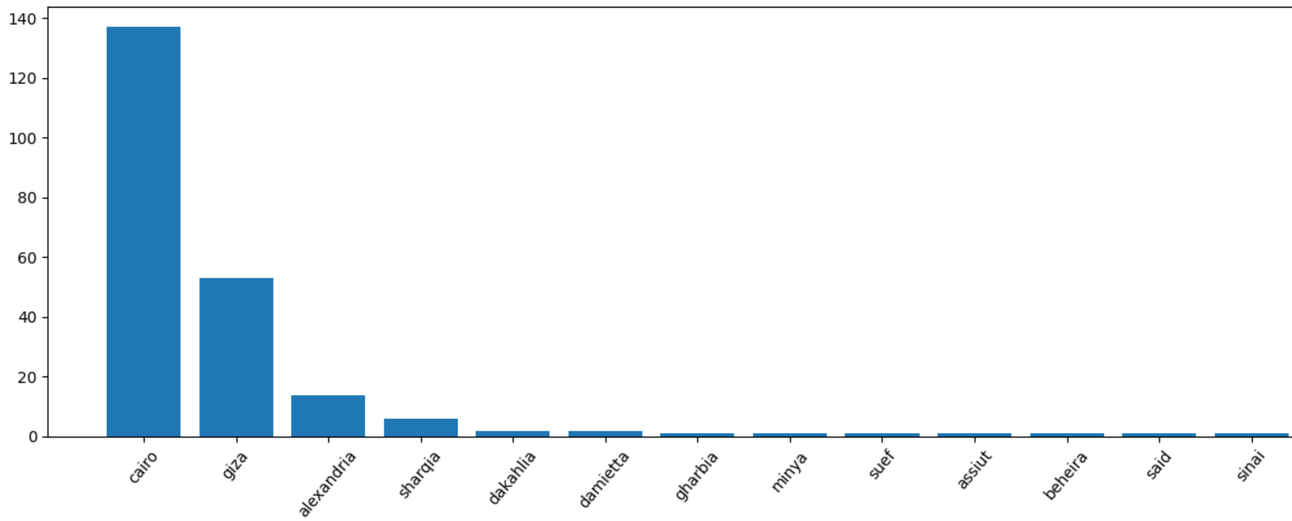
Flutter Developer
Software Engineer
Civil Engineer
Customer Service
Web Developer
Accountant
0.6%
7.3%
16.4%
16.7%
23.8%
Customer Service
Accountant
Web Developer
Civil Engineer
Software Engineer
Flutter Developer

2. Frequently mentioned skills in job description and requirements



3. Job distribution.





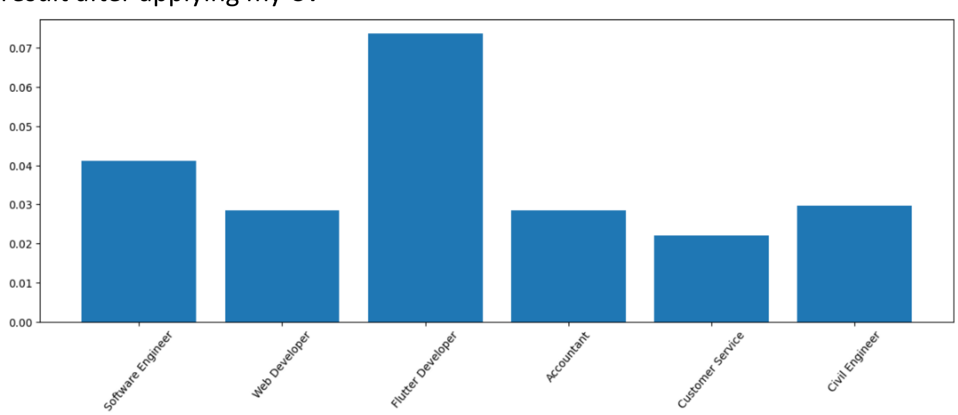
- **Phase-4.** CV matching algorithm.

  1. **First approach:** keywords matching.

- I have merged job description column with job requirement column.
- Then prepare the data by cleaning it, removing stop word, numbers, and punctuation, tokenize it, and remove duplicate words.
- Before preprocessing stage, I have separated the job listing based on job title.
- Now we have terms for each job title.
- Next step is taking the user CV and preprocess it.
- Simply I've implemented a function that takes the words of each class (job title), and the terms in the user's CV, compare the terms in cv and each class, then it takes the average.

```
CV_score(CV_terms,class_terms):
    For term1 in CV_terms:
        For term2 in class_terms:
            If term1 == term2:
                Count ++
    Avg = Count/len(class_terms)
    return Avg
```

result after applying my CV



2. **Second approach:** TF-IDF matching algorithm
   (term frequency-inverse document frequency):
   Statistical method to find importance of a term within a document.
   Calculates a numerical score for each term using combination of term frequency (TF) and inverse document frequency (IDF).
   - Term Frequency (TF): measures how often a specific word appears in a document. For example, in the below document word "cat" appeared three times.
     TF = number of times the term appears in the document / total number of unique words in the document
   - IDF: is to find or understand importance of a term within a collection of documents. This technique helps to identify how rare a term or word is across all the documents.
     IDF = log (total number of documents/number of documents contains the terms)
   - Once we have TF and IDF scores separately, we can calculate TF-IDF for a specific term or word by multiplying the TF value with its IDF value. TF-IDF = TF * IDF.

   Result:

```python
similarity_scores = cosine_similarity(new_tfidf_vector, tfidf_matrix)
copy_Score2 = similarity_scores[0].copy()
sorted = np.sort(-copy_Score2)
maxTen = sorted[0] * -1
index = np.where(similarity_scores[0]==maxTen)
job = data.iloc[index[0][0]]["job_title"]
print("the most appropriate job is:",job)
```

```
the most appropriate job is: Software Engineer
```