intelligent-job-alignment-system

August 13, 2024

1 CAPSTONE PROJECT:

Author:

Wachuka R. Kinyanjui

1.1 1.0 Introduction

1.1.1 1.1 Business Understanding

We aim to address the challenge of job searching by developing a system that simplifies the process of finding relevant job postings. We aim to enhance the job search experience by making it more efficient and tailored to individual qualifications and preferences. Having personally experienced the tedious and time-consuming nature of job searching, we recognise the need for a solution that can streamline this process. By improving the job search experience, we believe we can benefit both job seekers and recruiters. Job seekers will be able to quickly find positions that match their skills and interests, while recruiters will have an easier time identifying suitable candidates for their openings.

This project is relevant across all industries, as job searching is a universal activity that affects professionals in every field. The primary users of the job recommendation system include job seekers looking for new opportunities, professionals seeking career advancement, and recruiters and hiring managers aiming to find the right candidates for their job openings. If implemented, this job recommendation system would significantly ease the job search process for individuals, leading to quicker and more accurate matches between job seekers and available positions. Consequently, recruiters would benefit from a more efficient hiring process, reducing the time and effort required to find qualified candidates.

Our project builds on existing research in the field of job recommendations. For instance, we have explored papers such as "Enhancing Job Recommendations Using NLP and Machine Learning Techniques" by Narula, Rachna, Kumar, Vijay, Arora, Renuka, and Bhatia, Rishu (2023). This research highlights the potential of natural language processing (NLP) and machine learning techniques to improve job recommendation systems. Our primary motivation for this project is to create a solution that addresses a real-world problem and has a positive societal impact. By developing a job recommendation system, we hope to contribute to the betterment of the job search experience for individuals and the recruitment process for organizations.

1.1.2 1.2 Problem Statement

The traditional job search process is often cumbersome and inefficient, requiring job seekers to sift through numerous job postings that may not align with their skills and interests. This mismatch leads to frustration among job seekers and inefficiencies for recruiters who struggle to find suitable candidates. The problem is compounded by the volume of data available, which can overwhelm job seekers and recruiters alike. There is a need for a streamlined solution that can intelligently match job seekers with relevant job postings based on their unique qualifications and preferences. ### 1.3 Specific Objectives #### Develop an Intelligent Matching Algorithm: Create an algorithm that matches job seekers with relevant job postings based on their skills, experience, and preferences using recommendation systems and machine learning techniques. #### Enhance Recruiter Experience: Help recruiters quickly identify qualified candidates by providing them with a curated list of potential matches, thereby improving the overall efficiency of the hiring process. Reduce the time and effort required for job seekers to find suitable job openings by providing personalized job recommendations. #### Develop a User-Friendly Web Application: Create a web application that allows users to input their profiles and receive personalized job recommendations, ensuring an intuitive and seamless user experience. #### Deployment and Maintenance: Deploy the job recommendation system as a web application, ensuring regular updates and maintenance to improve functionality and accuracy over time.

1.1.3 1.4 Data Understanding

For this project, we will collect data on skills, experience, and career interests from job seekers. From recruiters, we will gather information on job postings, including role descriptions and job requirements. Our raw data is sourced from Kaggle, a platform known for its extensive datasets. The data is readily available on Kaggle, so we will download it directly from the platform.

The datasets include:
Combined_Jobs_Final.csv: Details about job openings available
Experience.csv: Job seeker's experience in previous roles
Job_views.csv: Time duration (in seconds) a job seeker spent looking at an opening. Also contains a few info about an opening.
Positions_Of_Interest.csv: Information about various positions a job seeker is interested in.
job_data.csv: Further description about a job opening

The Combined_Jobs_Final dataset contains: `Job.ID` `Provider` `Status` `Slug` `Title` `Position` `Company` `City` `State.Name` `State.Code` `Address` `Latitude` `Longitude` `Industry` `Job.Description` `Requirements` `Salary` `Listing.Start` `Listing.End` `Employment.Type` `Education.Required` `Created.At` `Updated.At`

The Experience dataset contains: `Applicant.ID` `Position.Name` `Employer.Name` `City` `State.Name` `State.Code` `Start.Date` `End.Date` `Job.Description` `Salary` `Can.Contact.Employer` `Created.At` `Updated.At`

The Job_views dataset contains: `Applicant.ID` `Job.ID` `Title` `Position` `Company` `City` `State.Name` `State.Code` `Industry` `View.Start` `View.End` `View.Duration`:(Time in seconds) `Created.At` `Updated.At`

The Positions_Of_Interest dataset contains: `Applicant.ID` `Position.Of.Interest` `Created.At` `Updated.At`

The job_data dataset contains: `Job.ID` `text`

[1]:
```
# Import libs
import os
```

```python
import pandas as pd
import numpy as np
import re

from collections import Counter, defaultdict
from operator import itemgetter
from time import time

# Libs for visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Text preprocessing
import nltk
import subprocess
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import word_tokenize, pos_tag, pos_tag_sents
nltk.download("stopwords")
from sklearn.feature_extraction.text import TfidfVectorizer
from string import punctuation

# Metrics
from scipy.stats import normaltest # D'Agostino's K-squared test
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
from sklearn.metrics import silhouette_score, pairwise_distances
from gensim.models.coherencemodel import CoherenceModel

# Dimensionality reduction
from sklearn.decomposition import TruncatedSVD
from sklearn.manifold import TSNE

# Clustering algorithms, Topic modeling
from sklearn.cluster import KMeans, MiniBatchKMeans, DBSCAN,␣
↪AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.neighbors import NearestNeighbors
from sklearn.decomposition import NMF
from gensim.models.nmf import Nmf
from gensim import corpora
from gensim.corpora.dictionary import Dictionary

import warnings
warnings.filterwarnings("ignore")
```

```
[nltk_data] Downloading package stopwords to
```

```
[nltk_data]     /Users/shilton/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[2]:
```
#Fixing RANDOM_SEED and package versions to make results reproducible
RANDOM_SEED = 42

!pip freeze > requirements.txt
```

1.2 Step 1: Data Wrangling and Cleaning

We load the datasets into pandas data frames so that we can begin data wrangling and cleaning.

[5]:
```
df_jd = pd.read_csv("Combined_Jobs_Final.csv")
df_cv = pd.read_csv("Experience.csv")
df_jbdt = pd.read_csv("job_data.csv")
df_views = pd.read_csv("Job_Views.csv")
df_poi = pd.read_csv("Positions_Of_Interest.csv")
```

Inspect the datasets

[7]:
```
df_jd.head()
```

[7]:
```
   Job.ID  Provider Status                                           Slug  \
0     111         1   open                   palo-alto-ca-tacolicious-server
1     113         1   open     san-francisco-ca-claude-lane-kitchen-staff-chef
2     117         1   open  san-francisco-ca-machka-restaurants-corp-barte…
3     121         1   open                  brisbane-ca-teriyaki-house-server
4     127         1   open  los-angeles-ca-rosa-mexicano-sunset-kitchen-st…

                                        Title            Position  \
0                        Server @ Tacolicious              Server
1              Kitchen Staff/Chef @ Claude Lane  Kitchen Staff/Chef
2          Bartender @ Machka Restaurants Corp.           Bartender
3                      Server @ Teriyaki House              Server
4  Kitchen Staff/Chef @ Rosa Mexicano - Sunset  Kitchen Staff/Chef

                      Company           City State.Name State.Code  …  \
0                 Tacolicious      Palo Alto California         CA  …
1                 Claude Lane  San Francisco California         CA  …
2  Machka Restaurants Corp.  San Francisco California         CA  …
3              Teriyaki House       Brisbane California         CA  …
4    Rosa Mexicano - Sunset    Los Angeles California         CA  …

             Industry                                    Job.Description  \
0  Food and Beverages  Tacolicious' first Palo Alto store just opened…
1  Food and Beverages   \r\n\r\nNew French Brasserie in S.F. Financia…
2  Food and Beverages  We are a popular Mediterranean wine bar and re…
3  Food and Beverages    Serve food/drinks to customers in a profess…
```

```
4  Food and Beverages  Located at the heart of Hollywood, we are one …

   Requirements Salary Listing.Start  Listing.End  Employment.Type  \
0           NaN   8.00           NaN          NaN        Part-Time
1           NaN   0.00           NaN          NaN        Part-Time
2           NaN  11.00           NaN          NaN        Part-Time
3           NaN  10.55           NaN          NaN        Part-Time
4           NaN  10.55           NaN          NaN        Part-Time

  Education.Required               Created.At               Updated.At
0                NaN  2013-03-12 02:08:28 UTC  2014-08-16 15:35:36 UTC
1                NaN  2013-04-12 08:36:36 UTC  2014-08-16 15:35:36 UTC
2                NaN  2013-07-16 09:34:10 UTC  2014-08-16 15:35:37 UTC
3                NaN  2013-09-04 15:40:30 UTC  2014-08-16 15:35:38 UTC
4                NaN  2013-07-17 15:26:18 UTC  2014-08-16 15:35:40 UTC

[5 rows x 23 columns]
```

We display the head of all 5 tables for inspection so as to pick the data we need and discard what we don't need

[9]:
```python
df_cv.head()
```

[9]:
```
   Applicant.ID                                     Position.Name  \
0         10001  Account Manager / Sales Administration / Quali…
1         10001    Electronics Technician / Item Master Controller
2         10001                                  Machine Operator
3         10003                           maintenance technician
4         10003                                 Electrical Helper

             Employer.Name         City           State.Name State.Code  \
0        Barcode Resourcing   Bellingham           Washington         WA
1              Ryzex Group   Bellingham           Washington         WA
2              comptec inc       Custer           Washington         WA
3           Winn residental   washington  District of Columbia         DC
4  michael and son services   alexandria             Virginia         VA

   Start.Date    End.Date                                  Job.Description  \
0  2012-10-15         NaN                                              NaN
1  2001-12-01  2012-04-01                                              NaN
2  1997-01-01  1999-01-01                                              NaN
3         NaN         NaN  Necessary maintenance for "Make Ready" Plumbin…
4         NaN         NaN    repair and services of electrical construction

   Salary Can.Contact.Employer               Created.At  \
0     NaN                  NaN  2014-12-12 20:10:02 UTC
1     NaN                  NaN  2014-12-12 20:10:02 UTC
```

```
2     NaN                  NaN  2014-12-12 20:10:02 UTC
3    10.0                False  2014-12-12 21:27:05 UTC
4     NaN                False  2014-12-12 21:27:05 UTC

                Updated.At
0  2014-12-12 20:10:02 UTC
1  2014-12-12 20:10:02 UTC
2  2014-12-12 20:10:02 UTC
3  2014-12-12 21:27:05 UTC
4  2014-12-12 21:27:05 UTC
```

[11]:
```
df_jbdt.head()
```

[11]:
```
   Unnamed: 0  Job.ID                                               text
0           0     111  server tacolici palo alto part time tacolici f…
1           1     113  kitchen staff chef claud lane san francisco pa…
2           2     117  bartend machka restaur corp. san francisco par…
3           3     121  server teriyaki hous brisban part time serv fo…
4           4     127  kitchen staff chef rosa mexicano sunset lo ang…
```

[13]:
```
df_views.head(10)
```

[13]:
```
   Applicant.ID  Job.ID                                              Title  \
0         10000   73666              Cashiers & Valets Needed! @ WallyPark
1         10000   96655  Macy's Seasonal Retail Fragrance Cashier  - Ga…
2         10001   84141  Part Time Showroom Sales / Cashier @ Grizzly I…
3         10002   77989  Event Specialist Part Time @ Advantage Sales &…
4         10002   69568          Bonefish - Kitchen Staff @ Bonefish Grill
5         10003   48200  Entry Level Security Officer @ Securitas Secur…
6         10004  139880  PT Teller - Chester/East 36th Cleveland @ KeyBank
7         10006  132042  Housekeeper / Caregiver - Senior Living - San …
8         10006  118687  Front Desk Coordinator for Immediate Opportuni…
9         10006   82500  Macy's Seasonal Retail Selling Floor Recovery,…

                                         Position  \
0                       Cashiers & Valets Needed!
1  Macy's Seasonal Retail Fragrance Cashier  - Ga…
2               Part Time Showroom Sales / Cashier
3                       Event Specialist Part Time
4                         Bonefish - Kitchen Staff
5                     Entry Level Security Officer
6          PT Teller - Chester/East 36th Cleveland
7  Housekeeper / Caregiver - Senior Living - San …
8  Front Desk Coordinator for Immediate Opportunity!
9  Macy's Seasonal Retail Selling Floor Recovery,…

                              Company         City     State.Name  \
```

```
0                         WallyPark        Newark      New Jersey
1                            Macy's   Garden City        New York
2              Grizzly Industrial Inc.    Bellingham      Washington
3             Advantage Sales & Marketing  Simpsonville  South Carolina
4                      Bonefish Grill    Greenville  South Carolina
5  Securitas Security Services USA, Inc.     Annandale        Virginia
6                             KeyBank     Cleveland            Ohio
7       Belmont Village at Sabre Springs     San Diego      California
8                          OfficeTeam      El Cajon      California
9                              Macy's      El Cajon      California

  State.Code Industry               View.Start                 View.End  \
0         NJ      NaN  2014-12-12 20:12:35 UTC  2014-12-12 20:31:24 UTC
1         NY      NaN  2014-12-12 20:08:50 UTC  2014-12-12 20:10:15 UTC
2         WA      NaN  2014-12-12 20:12:32 UTC  2014-12-12 20:17:18 UTC
3         SC      NaN  2014-12-12 20:39:23 UTC  2014-12-12 20:42:13 UTC
4         SC      NaN  2014-12-12 20:43:25 UTC  2014-12-12 20:43:58 UTC
5         VA    Other  2014-12-12 21:18:01 UTC  2014-12-12 21:18:19 UTC
6         OH      NaN  2014-12-12 22:04:19 UTC                      NaN
7         CA      NaN  2014-12-12 22:54:14 UTC  2014-12-13 01:04:58 UTC
8         CA      NaN  2014-12-13 01:20:24 UTC  2014-12-13 01:22:54 UTC
9         CA      NaN  2014-12-13 01:24:12 UTC  2014-12-13 18:03:29 UTC

   View.Duration               Created.At               Updated.At
0         1129.0  2014-12-12 20:12:35 UTC  2014-12-12 20:12:35 UTC
1           84.0  2014-12-12 20:08:50 UTC  2014-12-12 20:08:50 UTC
2          286.0  2014-12-12 20:12:32 UTC  2014-12-12 20:12:32 UTC
3          170.0  2014-12-12 20:39:23 UTC  2014-12-12 20:39:23 UTC
4           33.0  2014-12-12 20:43:25 UTC  2014-12-12 20:43:25 UTC
5           17.0  2014-12-12 21:18:01 UTC  2014-12-12 21:18:01 UTC
6            NaN  2014-12-12 22:04:19 UTC  2014-12-12 22:04:19 UTC
7         7843.0  2014-12-12 22:54:14 UTC  2014-12-12 22:54:14 UTC
8          150.0  2014-12-13 01:20:24 UTC  2014-12-13 01:20:24 UTC
9        59956.0  2014-12-13 01:24:12 UTC  2014-12-13 01:24:12 UTC
```

[15]:
```
df_poi.head()
```

[15]:
```
   Applicant.ID Position.Of.Interest               Created.At  \
0         10003     security officer  2014-12-12 21:20:54 UTC
1         10007               Server  2014-08-14 15:56:42 UTC
2         10007            Bartender  2014-08-14 15:56:44 UTC
3         10008                 Host  2014-08-14 15:56:42 UTC
4         10008              Barista  2014-08-14 15:56:43 UTC

               Updated.At
0  2014-12-12 21:20:54 UTC
1  2015-02-26 20:35:12 UTC
```

```
2   2015-02-19 23:21:28 UTC
3   2015-02-26 20:35:12 UTC
4   2015-02-18 02:35:06 UTC
```

After visually inspecting the data in the 5 tables, we decided to proceed with Combined_jobs_final and Experience

Cleaning Combined_jobs_final and Experience

COMBINED JOBS FINAL CLEANING

[17]:
```
#Inspecting Combined_jobs_final
df_jd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84090 entries, 0 to 84089
Data columns (total 23 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Job.ID              84090 non-null  int64
 1   Provider            84090 non-null  int64
 2   Status              84090 non-null  object
 3   Slug                84090 non-null  object
 4   Title               84090 non-null  object
 5   Position            84090 non-null  object
 6   Company             81819 non-null  object
 7   City                83955 non-null  object
 8   State.Name          83919 non-null  object
 9   State.Code          83919 non-null  object
 10  Address             36 non-null     object
 11  Latitude            84090 non-null  float64
 12  Longitude           84090 non-null  float64
 13  Industry            267 non-null    object
 14  Job.Description     84034 non-null  object
 15  Requirements        0 non-null      float64
 16  Salary              229 non-null    float64
 17  Listing.Start       83407 non-null  object
 18  Listing.End         83923 non-null  object
 19  Employment.Type     84080 non-null  object
 20  Education.Required  83823 non-null  object
 21  Created.At          84090 non-null  object
 22  Updated.At          84090 non-null  object
dtypes: float64(4), int64(2), object(17)
memory usage: 14.8+ MB
```

[19]:
```
df_jd.info
```

[19]:
```
<bound method DataFrame.info of        Job.ID  Provider Status  \
0         111         1   open
1         113         1   open
2         117         1   open
3         121         1   open
4         127         1   open
…          …         …    …
84085      82         1   open
84086      83         1   open
84087      84         1   open
84088      88         1   open
84089      92         1   open

                                                    Slug  \
0                     palo-alto-ca-tacolicious-server
1        san-francisco-ca-claude-lane-kitchen-staff-chef
2      san-francisco-ca-machka-restaurants-corp-barte…
3                    brisbane-ca-teriyaki-house-server
4      los-angeles-ca-rosa-mexicano-sunset-kitchen-st…
…                                                   …
84085  san-francisco-ca-national-japanese-american-hi…
84086       larkspur-ca-emporio-rulli-kitchen-staff-chef
84087                san-francisco-ca-onigilly-driver-84
84088  san-francisco-ca-machka-restaurants-corp-line-…
84089              san-jose-ca-kazoo-restaurant-cashier

                                                   Title           Position  \
0                                 Server @ Tacolicious             Server
1                         Kitchen Staff/Chef @ Claude Lane  Kitchen Staff/Chef
2                     Bartender @ Machka Restaurants Corp.          Bartender
3                                Server @ Teriyaki House             Server
4             Kitchen Staff/Chef @ Rosa Mexicano - Sunset  Kitchen Staff/Chef
…                                                      …                 …
84085  Book Keeper @ National Japanese American Histo…        Book Keeper
84086                 Kitchen Staff/Chef @ Emporio Rulli  Kitchen Staff/Chef
84087                                  Driver @ Onigilly             Driver
84088                 Line Cook @ Machka Restaurants Corp.          Line Cook
84089                          Cashier @ Kazoo Restaurant            Cashier

                                             Company           City  \
0                                        Tacolicious      Palo Alto
1                                        Claude Lane  San Francisco
2                            Machka Restaurants Corp.  San Francisco
3                                      Teriyaki House       Brisbane
4                               Rosa Mexicano - Sunset    Los Angeles
…                                                  …            …
84085  National Japanese American Historical Society  San Francisco
```

```
84086                                 Emporio Rulli       Larkspur
84087                                      Onigilly  San Francisco
84088                      Machka Restaurants Corp.  San Francisco
84089                             Kazoo Restaurant       San Jose

       State.Name State.Code  …               Industry  \
0      California         CA  …     Food and Beverages
1      California         CA  …     Food and Beverages
2      California         CA  …     Food and Beverages
3      California         CA  …     Food and Beverages
4      California         CA  …     Food and Beverages
…             …          …  …                      …
84085  California         CA  …  Office Administration
84086  California         CA  …     Food and Beverages
84087  California         CA  …     Food and Beverages
84088  California         CA  …     Food and Beverages
84089  California         CA  …     Food and Beverages

                                         Job.Description  Requirements Salary  \
0      Tacolicious' first Palo Alto store just opened…           NaN   8.00
1       \r\n\r\nNew French Brasserie in S.F. Financia…           NaN   0.00
2      We are a popular Mediterranean wine bar and re…           NaN  11.00
3         Serve food/drinks to customers in a profess…           NaN  10.55
4      Located at the heart of Hollywood, we are one …           NaN  10.55
…                                                    …           …    …
84085  NJAHS stands for National Japanese American Hi…           NaN  20.00
84086  Weekend Brunch Line Cook \r\n  Other shifts ma…           NaN  10.55
84087  ONIGILLY (Japanese rice ball wraps) seeks outg…           NaN  11.00
84088  We are a popular Mediterranean restaurant in F…           NaN  13.00
84089   We are looking for a cashier! \r\n\r\n   Take…           NaN  10.00

      Listing.Start  Listing.End  Employment.Type Education.Required  \
0               NaN          NaN        Part-Time                NaN
1               NaN          NaN        Part-Time                NaN
2               NaN          NaN        Part-Time                NaN
3               NaN          NaN        Part-Time                NaN
4               NaN          NaN        Part-Time                NaN
…                 …            …              …                 …
84085           NaN          NaN        Part-Time                NaN
84086           NaN          NaN        Part-Time                NaN
84087           NaN          NaN        Part-Time                NaN
84088           NaN          NaN        Part-Time                NaN
84089           NaN          NaN        Part-Time                NaN

                    Created.At               Updated.At
0      2013-03-12 02:08:28 UTC  2014-08-16 15:35:36 UTC
1      2013-04-12 08:36:36 UTC  2014-08-16 15:35:36 UTC
```

```
2      2013-07-16 09:34:10 UTC  2014-08-16 15:35:37 UTC
3      2013-09-04 15:40:30 UTC  2014-08-16 15:35:38 UTC
4      2013-07-17 15:26:18 UTC  2014-08-16 15:35:40 UTC
…                         …                        …
84085  2013-03-20 06:35:01 UTC  2014-08-16 15:35:27 UTC
84086  2013-03-20 08:06:43 UTC  2014-08-16 15:35:27 UTC
84087  2013-03-12 01:47:13 UTC  2014-08-16 15:35:27 UTC
84088  2013-07-16 08:55:22 UTC  2014-08-16 15:35:28 UTC
84089  2013-03-27 09:35:04 UTC  2014-08-16 15:35:30 UTC

[84090 rows x 23 columns]>
```

[21]:
```
#dropping unnecessary columns
df_jd = df_jd.drop(columns = ['Listing.Start', 'Listing.End', 'Created.At',␣
 ↪'Updated.At', 'Provider', 'Status', 'Slug', 'Title'])
```

[23]:
```
df_jd.isnull().sum()
```

[23]:
```
Job.ID                  0
Position                0
Company              2271
City                  135
State.Name            171
State.Code            171
Address             84054
Latitude                0
Longitude               0
Industry            83823
Job.Description        56
Requirements        84090
Salary              83861
Employment.Type        10
Education.Required    267
dtype: int64
```

[25]:
```
#checking for duplicates
jd_duplicates = df_jd.duplicated().sum()
jd_duplicates
```

[25]: 0

Filling in as much data as accurately possible

[27]:
```
#inspecting null values in Employment.type column
empl_typeNaN = df_jd[df_jd["Employment.Type"].isnull()]
display(empl_typeNaN)
```

```
      Job.ID         Position Company          City      State.Name  \
```

```
10768  153197  Driving Partner    Uber  San Francisco     California
10769  153198  Driving Partner    Uber    Los Angeles     California
10770  153199  Driving Partner    Uber        Chicago       Illinois
10771  153200  Driving Partner    Uber         Boston  Massachusetts
10772  153201  Driving Partner    Uber      Ann Arbor       Michigan
10773  153202  Driving Partner    Uber       Oklahoma       Oklahoma
10774  153203  Driving Partner    Uber          Omaha       Nebraska
10775  153204  Driving Partner    Uber        Lincoln       Nebraska
10776  153205  Driving Partner    Uber    Minneapolis      Minnesota
10777  153206  Driving Partner    Uber       St. Paul      Minnesota

      State.Code Address   Latitude   Longitude        Industry  \
10768         CA     NaN  37.774929 -122.419415  Transportation
10769         CA     NaN  34.052234 -118.243685  Transportation
10770         IL     NaN  41.878114  -87.629798  Transportation
10771         MA     NaN  42.358431  -71.059773  Transportation
10772         MI     NaN  42.280826  -83.743038  Transportation
10773         OK     NaN  35.467560  -97.516428  Transportation
10774         NE     NaN  41.252363  -95.997988  Transportation
10775         NE     NaN  40.809722  -96.675278  Transportation
10776         MN     NaN  44.983334  -93.266670  Transportation
10777         MN     NaN  44.953703  -93.089958  Transportation

                                         Job.Description  Requirements  \
10768  Uber is changing the way the world moves. From…           NaN
10769  Uber is changing the way the world moves. From…           NaN
10770  Uber is changing the way the world moves. From…           NaN
10771  Uber is changing the way the world moves. From…           NaN
10772  Uber is changing the way the world moves. From…           NaN
10773  Uber is changing the way the world moves. From…           NaN
10774  Uber is changing the way the world moves. From…           NaN
10775  Uber is changing the way the world moves. From…           NaN
10776  Uber is changing the way the world moves. From…           NaN
10777  Uber is changing the way the world moves. From…           NaN

       Salary Employment.Type Education.Required
10768     NaN             NaN                NaN
10769     NaN             NaN                NaN
10770     NaN             NaN                NaN
10771     NaN             NaN                NaN
10772     NaN             NaN                NaN
10773     NaN             NaN                NaN
10774     NaN             NaN                NaN
10775     NaN             NaN                NaN
10776     NaN             NaN                NaN
10777     NaN             NaN                NaN
```

From this information we can conclude that the missing employment types are Full-Time/Part-

Time

[29]:
```python
#filling null values in employment_type
df_jd.loc[:, "Employment.Type"] = df_jd["Employment.Type"].fillna("Full-Time/
↪Part-Time")
```

[31]:
```python
#inspecting null values in State.Code column
st_codeNaN = df_jd[df_jd["State.Code"].isnull()]
st_codeNaN
```

[31]:
```
       Job.ID                                           Position  \
204    134544               Pool Attendant (Regular Part-time)
205    134545           Towel Hut Attendant (Regular Part-time)
3433   142054  Sales Representative - Business Development Op…
3434   142055                            New Business Executive
3435   142056  Outside Sales Representative (Business Develop…
…         …                                                  …
79814  315008                                 Account Executive
80617  315811  Sales Representative / Sales Associate ( Entry…
80694  315888  Sales Representative / Sales Associate ( Entry…
82516  317709     Site Manager - Apartment Community (Part Time)
83966  319158                                   Account Manager

                         Company          City State.Name State.Code Address  \
204          Wyndham Hotel Group    Rio Grande        NaN        NaN     NaN
205          Wyndham Hotel Group    Rio Grande        NaN        NaN     NaN
3433         CHI Payment Systems           NaN        NaN        NaN     NaN
3434         CHI Payment Systems           NaN        NaN        NaN     NaN
3435         CHI Payment Systems           NaN        NaN        NaN     NaN
…                              …             …        …          …       …
79814        CHI Payment Systems           NaN        NaN        NaN     NaN
80617           Vector Marketing      San Juan        NaN        NaN     NaN
80694           Vector Marketing         Ponce        NaN        NaN     NaN
82516  Webrecruit North America  Saint Thomas        NaN        NaN     NaN
83966        CHI Payment Systems           NaN        NaN        NaN     NaN

       Latitude  Longitude Industry  \
204     18.3018   -66.0800      NaN
205     18.3018   -66.0800      NaN
3433     0.0000     0.0000      NaN
3434     0.0000     0.0000      NaN
3435     0.0000     0.0000      NaN
…           …          …        …
79814    0.0000     0.0000      NaN
80617   18.3905   -66.0903      NaN
80694   18.1144   -66.8568      NaN
82516   18.3436   -64.9322      NaN
83966    0.0000     0.0000      NaN
```

```
                                          Job.Description  Requirements  \
204    The Pool Attendant is responsible for ensuring…          NaN
205    The Towel Hut Attendant is responsible for ens…          NaN
3433   If you&rsquo;re energetic, motivated, hardwork…          NaN
3434   If you&rsquo;re energetic, motivated, hardwork…          NaN
3435   If you&rsquo;re energetic, motivated, hardwork…          NaN
…                                                  …            …
79814  CHI Payment Systems, a leading merchant servic…          NaN
80617  \r\n \r\nIf you are eager to learn, we ha…          NaN
80694  \r\n \r\nIf you are eager to learn, we ha…          NaN
82516  Site Manager &ndash; Apartment Community …          NaN
83966  If you&rsquo;re energetic, motivated, hardwork…          NaN

       Salary      Employment.Type   Education.Required
204       NaN            Part-Time  High School Diploma
205       NaN            Part-Time  High School Diploma
3433      NaN  Full-Time/Part-Time  High School Diploma
3434      NaN  Full-Time/Part-Time  High School Diploma
3435      NaN  Full-Time/Part-Time  High School Diploma
…         …                  …                    …
79814     NaN  Full-Time/Part-Time  High School Diploma
80617     NaN  Full-Time/Part-Time        Not Specified
80694     NaN  Full-Time/Part-Time        Not Specified
82516     NaN            Part-Time        Not Specified
83966     NaN  Full-Time/Part-Time  High School Diploma

[171 rows x 15 columns]
```

From this information, the missing state is Puerto Rico

[33]:
```python
#filing null values in State.Code with "PR" for Puerto Rico
df_jd.loc[:, "State.Code"] = df_jd["State.Code"].fillna("PR")
```

[35]:
```python
df_jd.isnull().sum()
```

[35]:
```
Job.ID                   0
Position                 0
Company               2271
City                   135
State.Name             171
State.Code               0
Address              84054
Latitude                 0
Longitude                0
Industry             83823
Job.Description         56
```

```
Requirements          84090
Salary                83861
Employment.Type           0
Education.Required      267
dtype: int64
```

[37]:
```
# Handling the missing values
df_jd['Employment.Type'] = df_jd['Employment.Type'].fillna('Full-Time/
  ↪Part-Time')
df_jd['State.Code'] = df_jd['State.Code'].fillna('UNKNOWN')
df_jd['Industry'] = df_jd['Industry'].fillna('Not Specified')
df_jd['Salary'] = df_jd['Salary'].fillna(0.0)
df_jd['Job.Description'] = df_jd['Job.Description'].fillna('No Description␣
  ↪Provided')
df_jd['Education.Required'] = df_jd['Education.Required'].fillna('Not␣
  ↪Specified')

# Handle any remaining missing values if necessary (e.g., for other columns)
df_jd = df_jd.fillna('Unknown')

print("Missing values handled and cleaned dataset saved.")
```

```
Missing values handled and cleaned dataset saved.
```

[39]:
```
df_jd.isnull().sum()
```

[39]:
```
Job.ID                0
Position              0
Company               0
City                  0
State.Name            0
State.Code            0
Address               0
Latitude              0
Longitude             0
Industry              0
Job.Description       0
Requirements          0
Salary                0
Employment.Type       0
Education.Required    0
dtype: int64
```

[41]:
```
df_jd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84090 entries, 0 to 84089
```

```
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Job.ID             84090 non-null  int64
 1   Position           84090 non-null  object
 2   Company            84090 non-null  object
 3   City               84090 non-null  object
 4   State.Name         84090 non-null  object
 5   State.Code         84090 non-null  object
 6   Address            84090 non-null  object
 7   Latitude           84090 non-null  float64
 8   Longitude          84090 non-null  float64
 9   Industry           84090 non-null  object
 10  Job.Description    84090 non-null  object
 11  Requirements       84090 non-null  object
 12  Salary             84090 non-null  float64
 13  Employment.Type    84090 non-null  object
 14  Education.Required 84090 non-null  object
dtypes: float64(3), int64(1), object(11)
memory usage: 9.6+ MB
```

EXPERIENCE

[43]:
```
#inspecting experience
df_cv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8653 entries, 0 to 8652
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Applicant.ID         8653 non-null   int64
 1   Position.Name        7703 non-null   object
 2   Employer.Name        8620 non-null   object
 3   City                 4922 non-null   object
 4   State.Name           4595 non-null   object
 5   State.Code           4595 non-null   object
 6   Start.Date           6618 non-null   object
 7   End.Date             4906 non-null   object
 8   Job.Description      5692 non-null   object
 9   Salary               2798 non-null   float64
 10  Can.Contact.Employer 3581 non-null   object
 11  Created.At           8653 non-null   object
 12  Updated.At           8653 non-null   object
dtypes: float64(1), int64(1), object(11)
memory usage: 878.9+ KB
```

[45]:
```
df_cv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8653 entries, 0 to 8652
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Applicant.ID          8653 non-null   int64
 1   Position.Name         7703 non-null   object
 2   Employer.Name         8620 non-null   object
 3   City                  4922 non-null   object
 4   State.Name            4595 non-null   object
 5   State.Code            4595 non-null   object
 6   Start.Date            6618 non-null   object
 7   End.Date              4906 non-null   object
 8   Job.Description       5692 non-null   object
 9   Salary                2798 non-null   float64
 10  Can.Contact.Employer  3581 non-null   object
 11  Created.At            8653 non-null   object
 12  Updated.At            8653 non-null   object
dtypes: float64(1), int64(1), object(11)
memory usage: 878.9+ KB
```

[47]:
```
#checking for null values
df_cv.isnull().mean()*100
```

[47]:
```
Applicant.ID             0.000000
Position.Name           10.978851
Employer.Name            0.381371
City                    43.117994
State.Name              46.897030
State.Code              46.897030
Start.Date              23.517855
End.Date                43.302901
Job.Description         34.219346
Salary                  67.664394
Can.Contact.Employer    58.615509
Created.At               0.000000
Updated.At               0.000000
dtype: float64
```

[49]:
```
#removing unnecessary columns
df_cv = df_cv.drop(columns = ['Can.Contact.Employer', 'Created.At', 'Updated.
 ↪At', 'Employer.Name', 'Salary'])
```

[51]:
```
#Removing null values including placeholders
df_cv["Job.Description"] = df_cv["Job.Description"].apply(lambda x: x if str(x).
 ↪lower().replace(' ', '') != "none" and x is not None else np.nan)
```

```
df_cv["Position.Name"] = df_cv["Position.Name"].apply(lambda x: x if str(x).
↪lower().replace(' ', '') != "none" and x is not None else np.nan)

df_cv = df_cv.dropna(subset=["Job.Description", "Position.Name"])
```

[53]:
```
#Checking for and dropping duplicates
cv_duplicates = df_cv.duplicated().sum()
cv_duplicates
```

[53]:
```
9
```

[55]:
```
df_cv = df_cv.drop_duplicates()
```

[57]:
```
df_cv.isnull().sum()
```

[57]:
```
Applicant.ID          0
Position.Name         0
City               1667
State.Name         1814
State.Code         1814
Start.Date          554
End.Date           1660
Job.Description       0
dtype: int64
```

[59]:
```
# Handling missing values

df_cv['City'] = df_cv['City'].fillna('Unknown')
df_cv['State.Name'] = df_cv['State.Name'].fillna('Unknown')
df_cv['State.Code'] = df_cv['State.Code'].fillna('Unknown')
df_cv['Start.Date'] = df_cv['Start.Date'].fillna('1900-01-01')
df_cv['End.Date'] = df_cv['End.Date'].fillna('1900-01-01')
```

[61]:
```
df_cv.isnull().sum()
```

[61]:
```
Applicant.ID       0
Position.Name      0
City               0
State.Name         0
State.Code         0
Start.Date         0
End.Date           0
Job.Description    0
dtype: int64
```

[63]:
```
df_cv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5563 entries, 3 to 8652
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Applicant.ID     5563 non-null   int64
 1   Position.Name    5563 non-null   object
 2   City             5563 non-null   object
 3   State.Name       5563 non-null   object
 4   State.Code       5563 non-null   object
 5   Start.Date       5563 non-null   object
 6   End.Date         5563 non-null   object
 7   Job.Description  5563 non-null   object
dtypes: int64(1), object(7)
memory usage: 391.1+ KB
```

[65]:

```
# Convert column names to lowercase and remove white spaces for df_jd
df_jd.columns = df_jd.columns.str.lower().str.replace(' ', '_')

# Convert column names to lowercase and remove white spaces for df_cv
df_cv.columns = df_cv.columns.str.lower().str.replace(' ', '_')
```

[67]:

```
df_jd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84090 entries, 0 to 84089
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   job.id              84090 non-null  int64
 1   position            84090 non-null  object
 2   company             84090 non-null  object
 3   city                84090 non-null  object
 4   state.name          84090 non-null  object
 5   state.code          84090 non-null  object
 6   address             84090 non-null  object
 7   latitude            84090 non-null  float64
 8   longitude           84090 non-null  float64
 9   industry            84090 non-null  object
 10  job.description     84090 non-null  object
 11  requirements        84090 non-null  object
 12  salary              84090 non-null  float64
 13  employment.type     84090 non-null  object
 14  education.required  84090 non-null  object
dtypes: float64(3), int64(1), object(11)
memory usage: 9.6+ MB
```

[69]:
```
df_cv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5563 entries, 3 to 8652
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   applicant.id     5563 non-null   int64
 1   position.name    5563 non-null   object
 2   city             5563 non-null   object
 3   state.name       5563 non-null   object
 4   state.code       5563 non-null   object
 5   start.date       5563 non-null   object
 6   end.date         5563 non-null   object
 7   job.description  5563 non-null   object
dtypes: int64(1), object(7)
memory usage: 391.1+ KB
```

1.3 VIEWS

[71]:
```
df_views = df_views[['Applicant.ID', 'Job.ID', 'Position']]
```

1.4 POSITION OF INTEREST

[73]:
```
df_poi = df_poi[['Applicant.ID', 'Position.Of.Interest']]
```

1.5 STEP 2: EDA

1.5.1 1. UNIVARIATE ANALYSIS

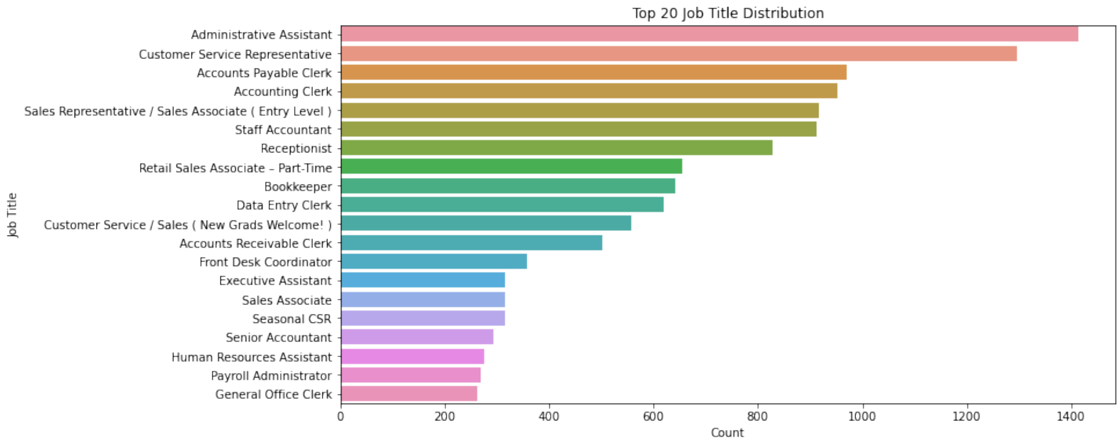
Plot for Numerical Columns Distribution

Distribution of Jobs

Top 20 position Distribution

[75]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Get the top 20 job titles by the number of occurrences
top_job_titles = df_jd['position'].value_counts().index[:20]

# Filter the DataFrame to include only these top job titles
df_top_job_titles = df_jd[df_jd['position'].isin(top_job_titles)]

# Plot the countplot for the top 20 job titles
plt.figure(figsize=(12, 6))
sns.countplot(y='position', data=df_top_job_titles, order=top_job_titles)
```

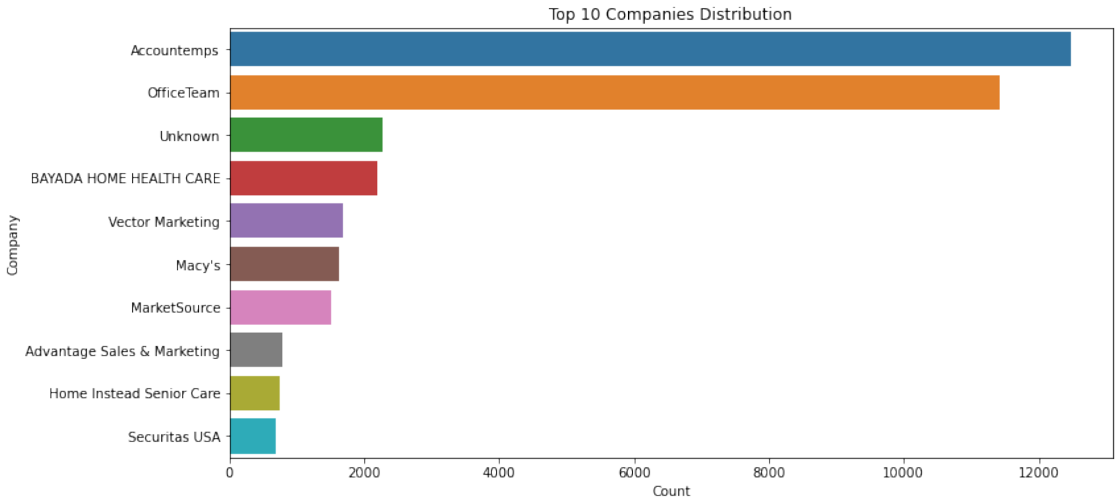
```
plt.title('Top 20 Job Title Distribution')
plt.xlabel('Count')
plt.ylabel('Job Title')
plt.show()
```



Insights The top 20 job titles can reveal the most sought-after positions. High counts for specific titles like administrative assistans indicate high demand in certain roles.

Top 10 Companies Distribution

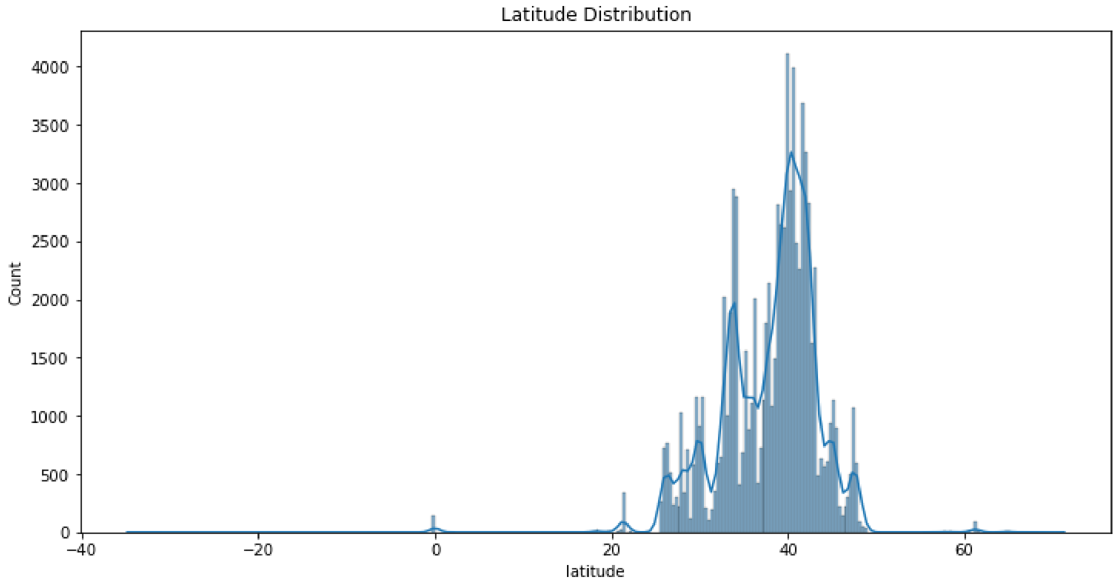
[77]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Get the top 10 companies by the number of occurrences
top_companies = df_jd['company'].value_counts().index[:10]

# Filter the DataFrame to include only these top companies
df_top_companies = df_jd[df_jd['company'].isin(top_companies)]

# Plot the countplot for the top 10 companies
plt.figure(figsize=(12, 6))
sns.countplot(y='company', data=df_top_companies, order=top_companies)
plt.title('Top 10 Companies Distribution')
plt.xlabel('Count')
plt.ylabel('Company')
plt.show()
```

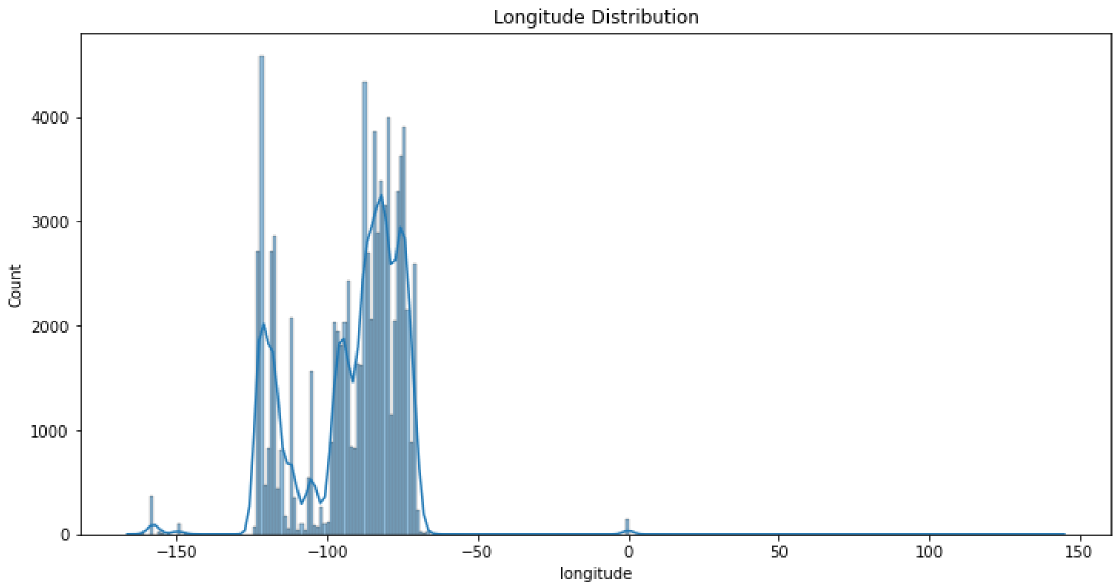


Insights Identifying the top 10 companies by job postings helps to understand which companies are the most active in hiring, potentially indicating market leaders.

Longitude and Latitide Distribution

[79]:
```
import matplotlib.pyplot as plt
import seaborn as sns
# Distribution of Latitude
plt.figure(figsize=(12, 6))
sns.histplot(df_jd['latitude'], kde=True)
plt.title('Latitude Distribution')
plt.show()

# Distribution of Longitude
plt.figure(figsize=(12, 6))
sns.histplot(df_jd['longitude'], kde=True)
plt.title('Longitude Distribution')
plt.show()
```



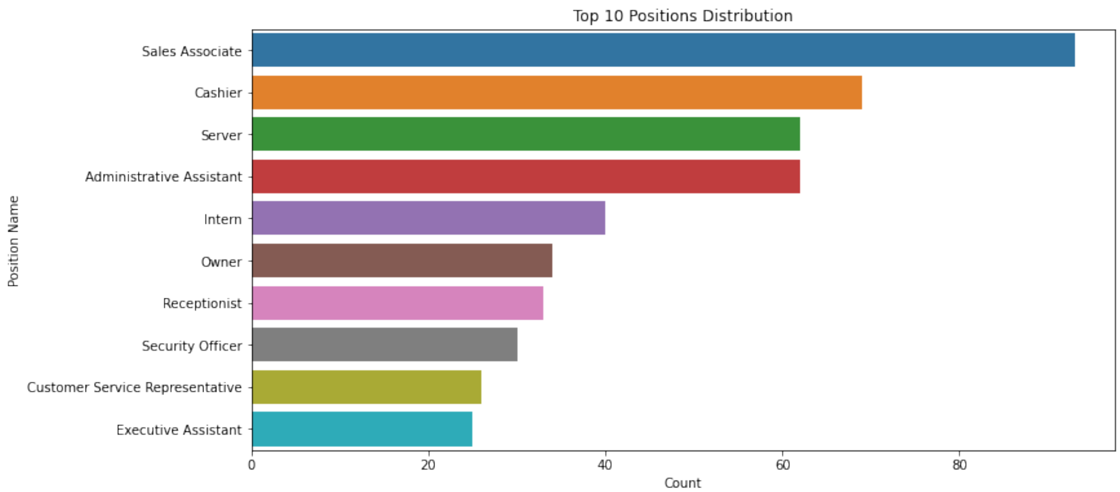


Insights The histograms for latitude and longitude give insights into the geographical distribution of job listings. Peaks in these distributions can point to densely populated job areas.

Top 10 Positions Distribution

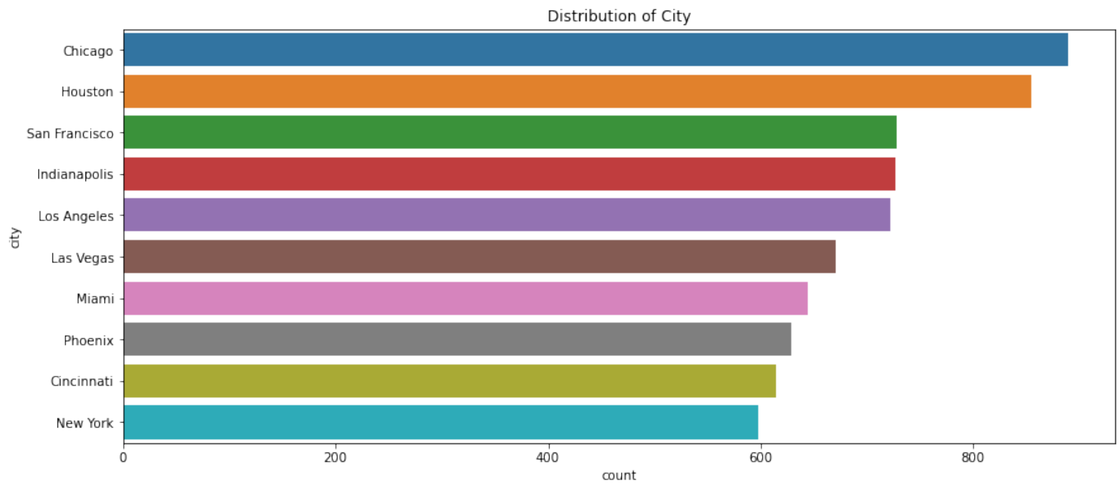
[81]:
```
top_positions = df_cv['position.name'].value_counts().nlargest(10).index

plt.figure(figsize=(12, 6))
```

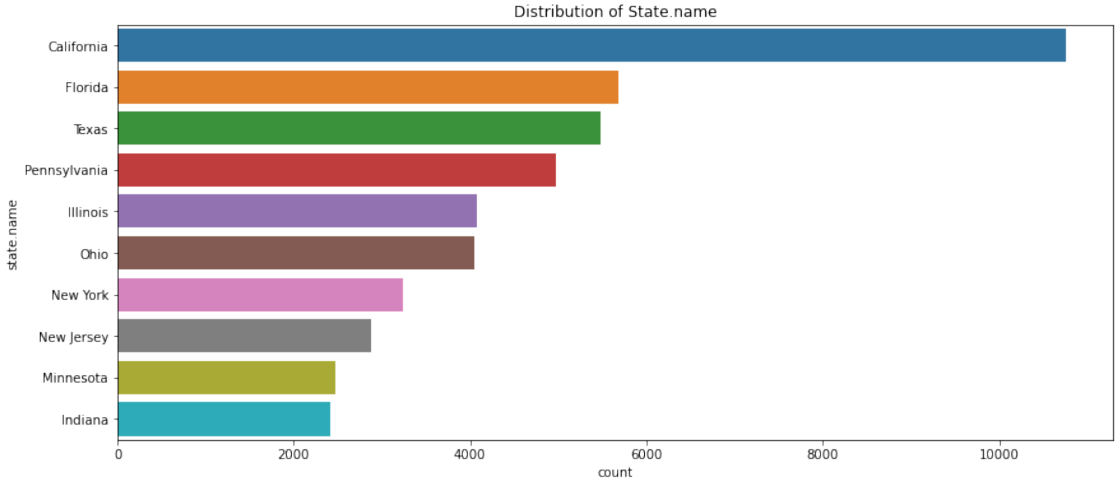
```
sns.countplot(y='position.name', data=df_cv, order=top_positions)
plt.title('Top 10 Positions Distribution')
plt.xlabel('Count')
plt.ylabel('Position Name')
plt.show()
```



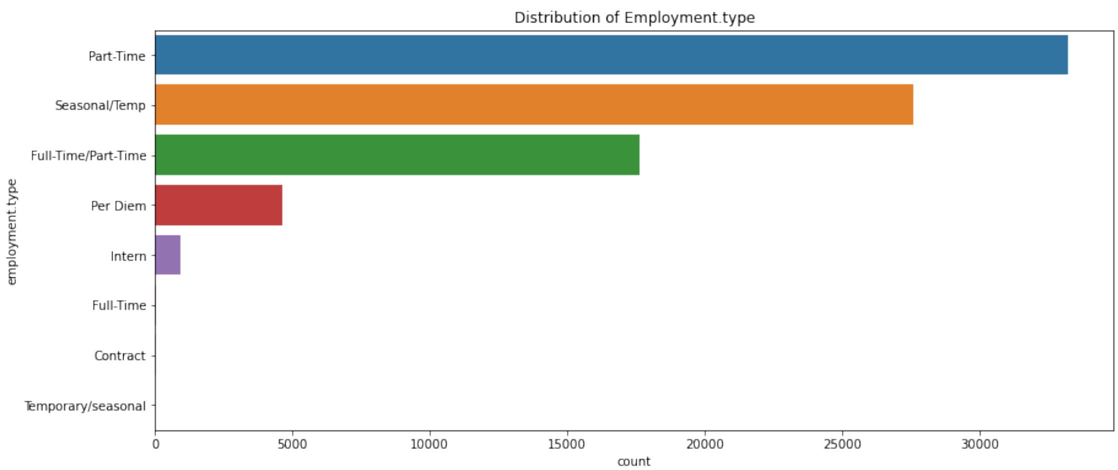
Distribution Of Categorical Columns

[83]:
```
for col in ['city', 'state.name', 'employment.type', 'education.required']:
    plt.figure(figsize=(14, 6))
    sns.countplot(y=col, data=df_jd, order=df_jd[col].value_counts().index[:10])
    plt.title(f'Distribution of {col.capitalize()}')
    plt.show()
```

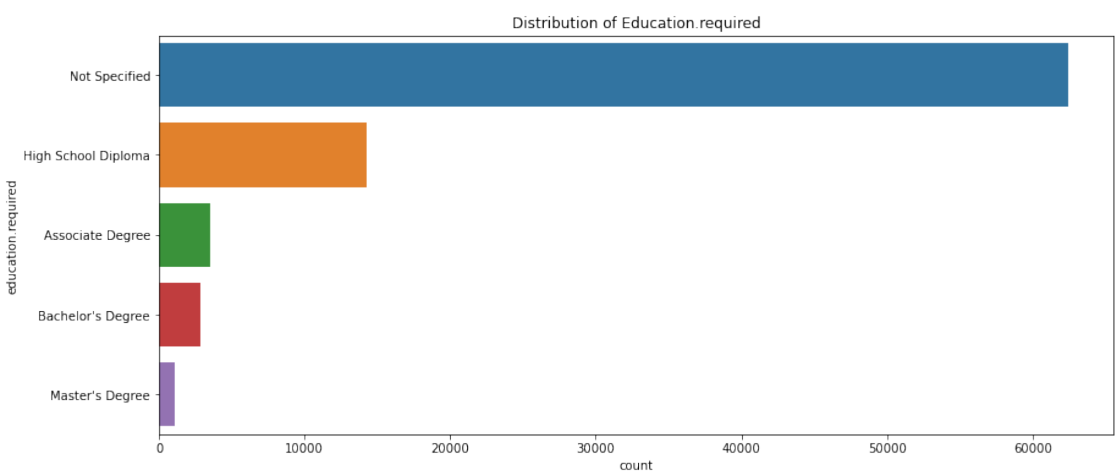




Distribution of State.name
California
Florida
Texas
Pennsylvania
Illinois
Ohio
New York
New Jersey
Minnesota
Indiana
state.name
0
2000
4000
6000
8000
10000
count



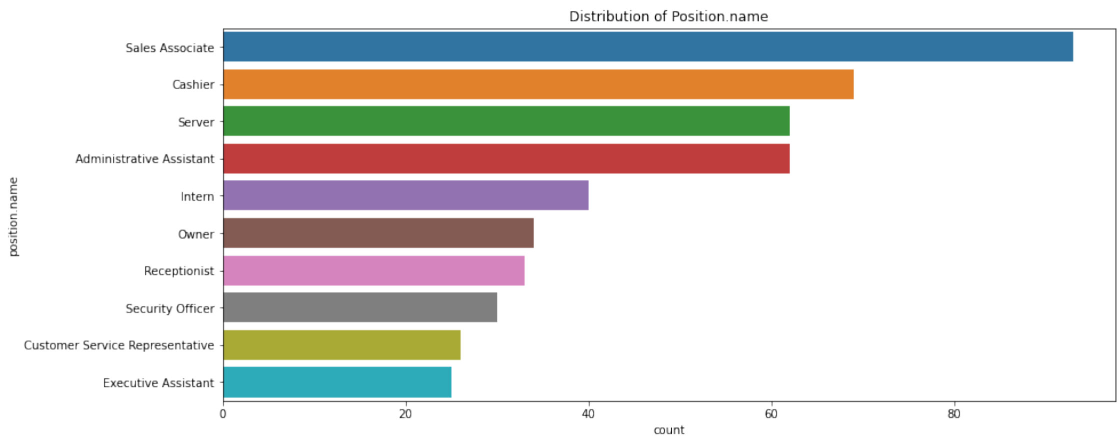
Distribution of Employment.type
Part-Time
Seasonal/Temp
Full-Time/Part-Time
Per Diem
Intern
Full-Time
Contract
Temporary/seasonal
employment.type
0
5000
10000
15000
20000
25000
30000
count

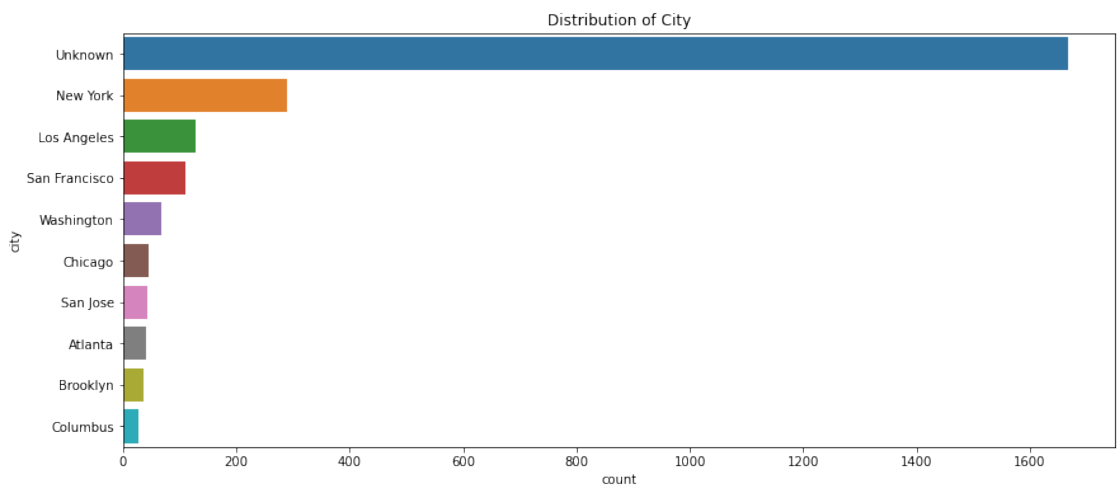


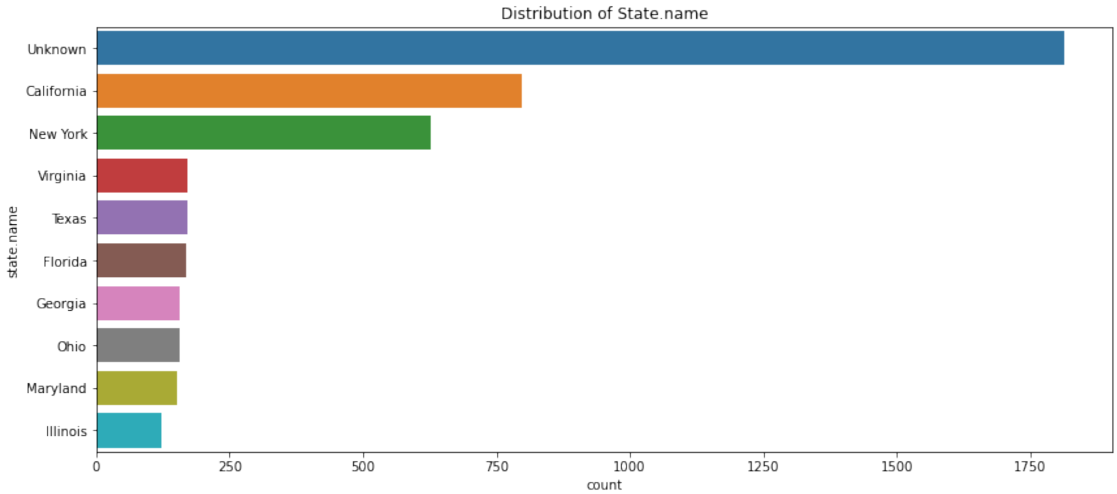
Distribution of Education.required
Not Specified
High School Diploma
Associate Degree
Bachelor's Degree
Master's Degree
education.required
0
10000
20000
30000
40000
50000
60000
count

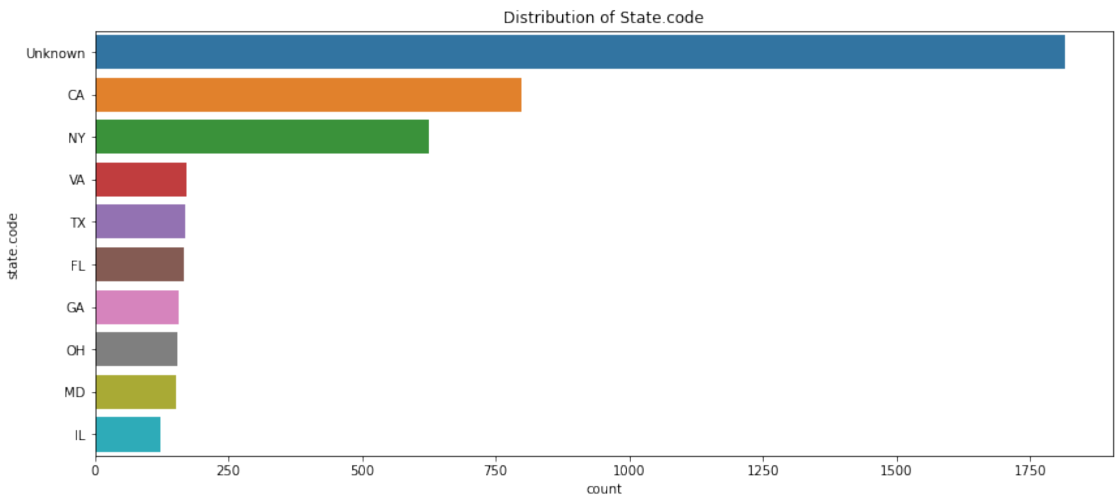
[85]:

```python
## Categorical Data - df_cv
for col in ['position.name', 'city', 'state.name', 'state.code']:
    plt.figure(figsize=(14, 6))
    sns.countplot(y=col, data=df_cv, order=df_cv[col].value_counts().index[:10])
    plt.title(f'Distribution of {col.capitalize()}')
    plt.show()
```







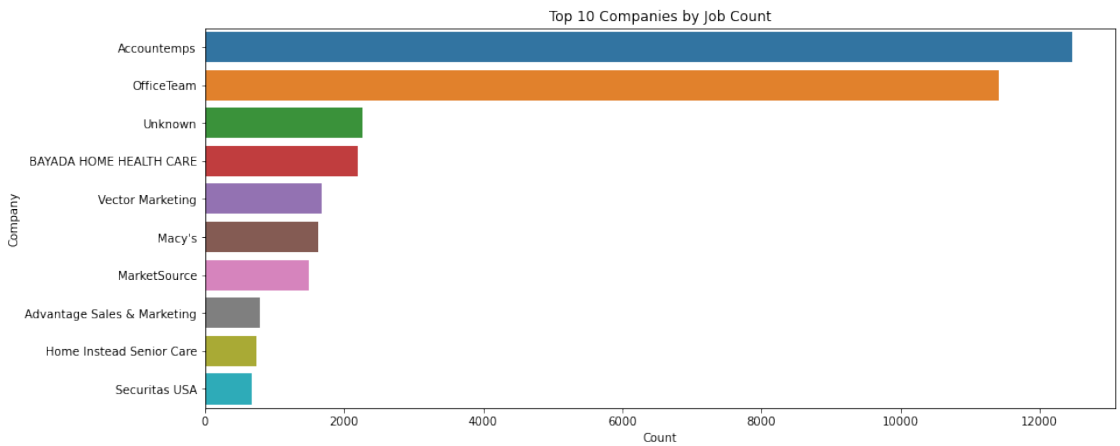


Insights Distributions of categorical columns provide insights into the most common cities, states, employment types, and education levels required, which helps tailor job recommendations to user preference

Top 10 Companies by Job count

[87]:
```
plt.figure(figsize=(14, 6))
top_companies = df_jd['company'].value_counts().nlargest(10).index  # Top 10␣
  ↪companies
sns.countplot(y='company', data=df_jd[df_jd['company'].isin(top_companies)],␣
  ↪order=top_companies)
plt.title('Top 10 Companies by Job Count')
plt.xlabel('Count')
```

```
plt.ylabel('Company')
plt.show()
```



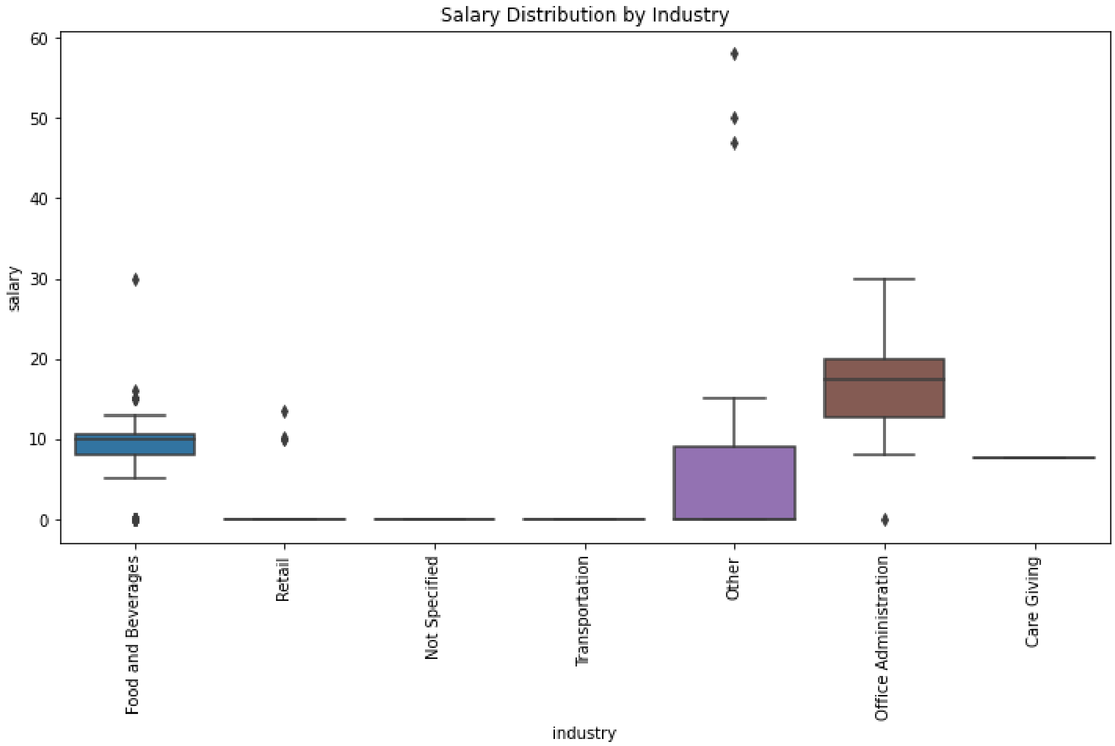
Insights The bar chart reveals the top 10 companies that are actively hiring, showing which companies dominate the job market in the dataset.

The distribution provides an understanding of which companies are most likely to have job openings, which is crucial for job seekers targeting specific employers.

1.5.2 2.BIVARIATE ANALYSIS

Salary Distribution by Industry

[89]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='industry', y='salary', data=df_jd)
plt.title('Salary Distribution by Industry')
plt.xticks(rotation=90)
plt.show()
```

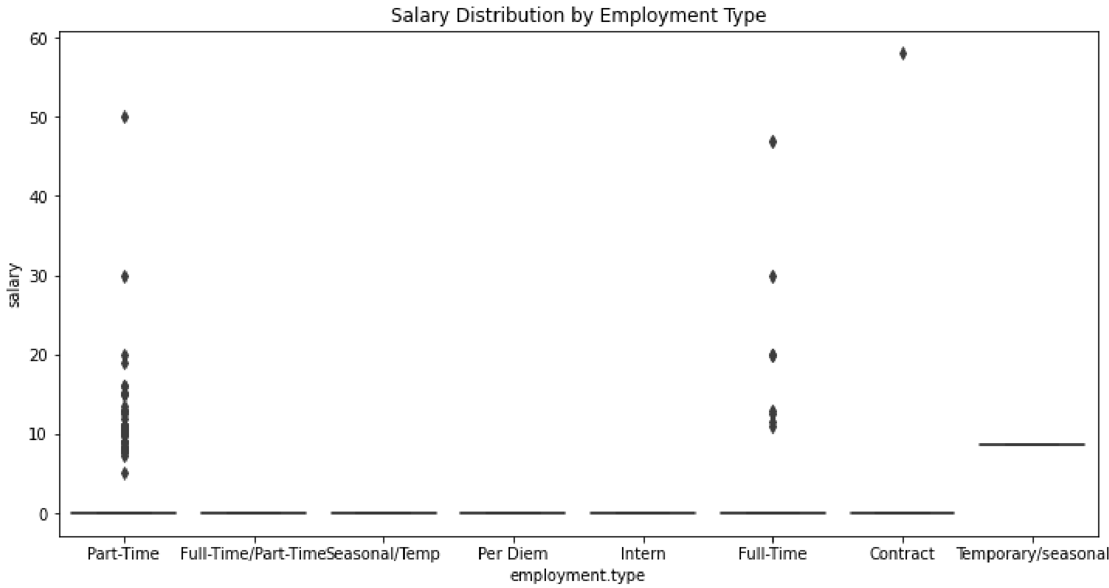


Insights The boxplot demonstrates significant variations in salary across different industries.

Certain industries have a broader salary range, while others are more consistent, which can guide job seekers on which industries might offer higher compensation.

Salary vs Employment type

[91]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='employment.type', y='salary', data=df_jd)
plt.title('Salary Distribution by Employment Type')
plt.show()
```

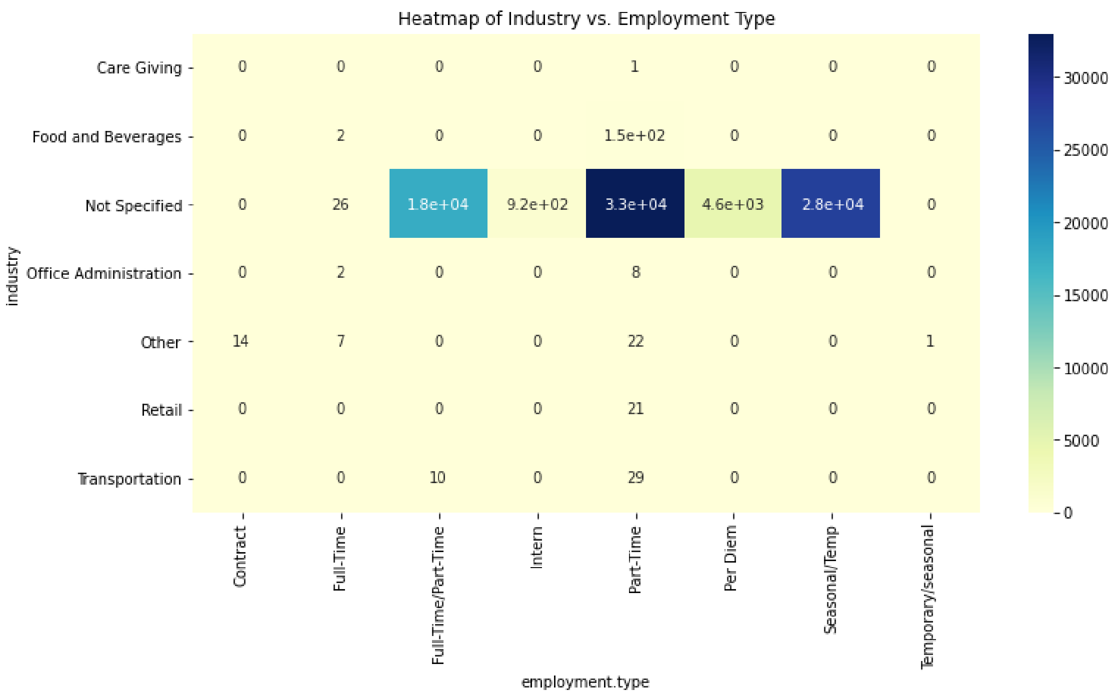


Insights This plot highlights that full-time employment types generally offer higher salaries compared to part-time or contract positions.

The variation in salaries within employment types might be due to differences in job roles and industries.

Correlation Heatmap for numerical columns

[93]:
```
pivot_table = pd.crosstab(df_jd['industry'], df_jd['employment.type'])
plt.figure(figsize=(12, 6))
sns.heatmap(pivot_table, annot=True, cmap='YlGnBu')
plt.title('Heatmap of Industry vs. Employment Type')
plt.show()
```

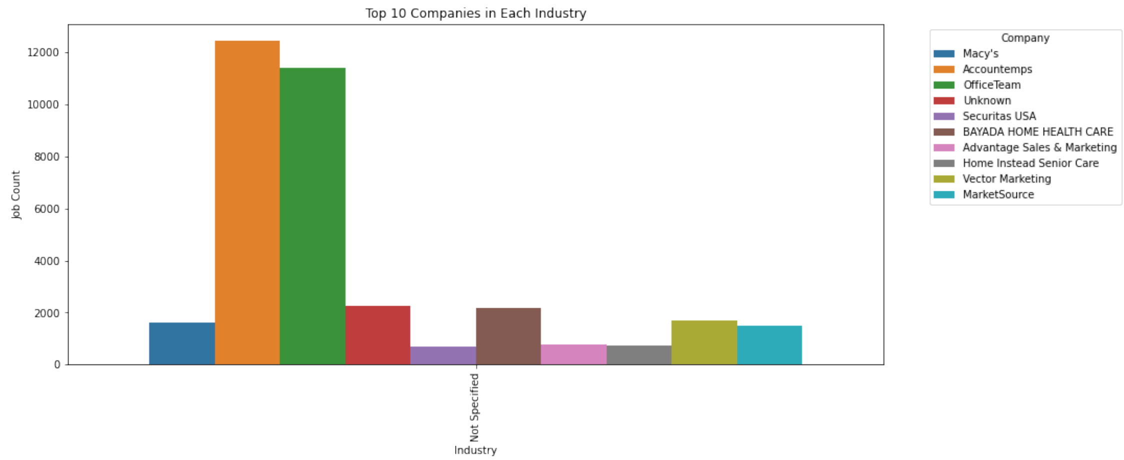


Insights: The heatmap shows how certain industries favor specific employment types. For example, some industries may have more part-time roles, while others might offer more contract positions.

This can help job seekers understand industry trends in employment types, aiding in more targeted job applications.

Industry by Company

[95]:
```python
#Relationship between Industry and Company
plt.figure(figsize=(14, 6))
sns.countplot(x='industry', hue='company', data=df_jd[df_jd['company'].isin(top_companies)])
plt.title('Top 10 Companies in Each Industry')
plt.xticks(rotation=90)
plt.xlabel('Industry')
plt.ylabel('Job Count')
plt.legend(title='Company', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

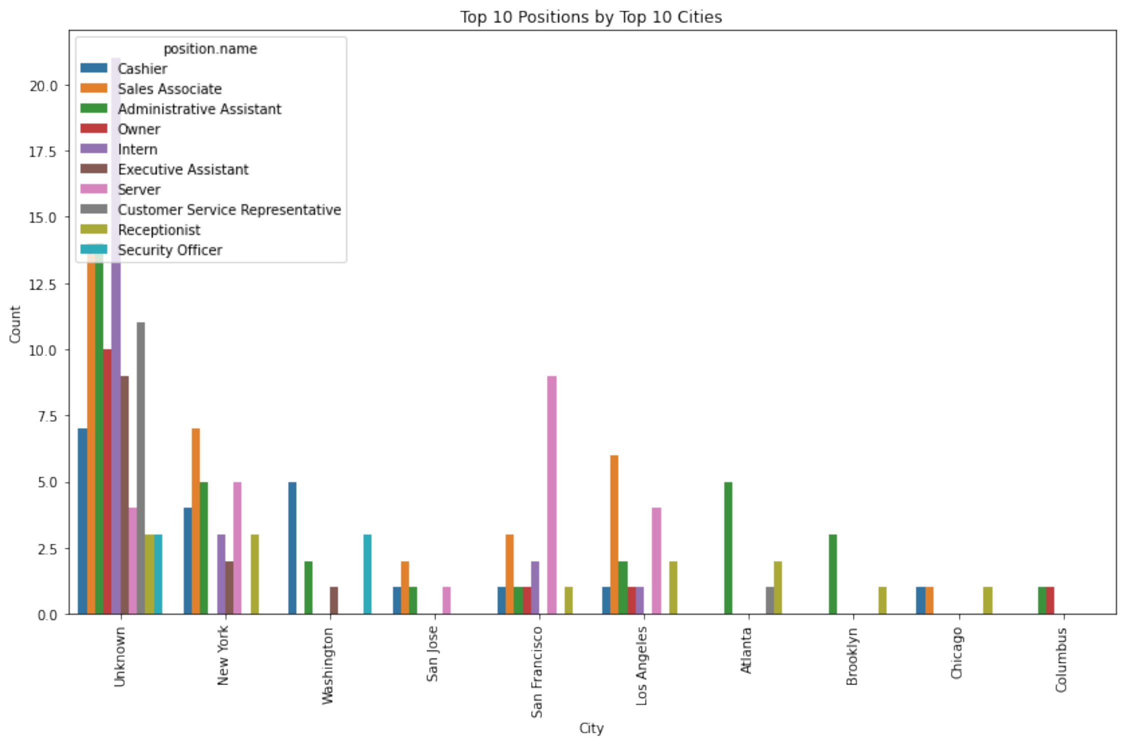


Insights: This plot illustrates which companies are leading in specific industries, providing insights into the competitive landscape within sectors.

It can help job seekers identify the major players in their industry of interest and target these companies.

Positions by City

[97]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Get the top 10 cities
top_cities = df_cv['city'].value_counts().nlargest(10).index

# Get the top 10 position names
top_positions = df_cv['position.name'].value_counts().nlargest(10).index

# Filter the DataFrame for the top cities and positions
df_cv_filtered = df_cv[df_cv['city'].isin(top_cities) & df_cv['position.name'].
↪isin(top_positions)]

# Plot the countplot
plt.figure(figsize=(14, 8))
sns.countplot(x='city', hue='position.name', data=df_cv_filtered)
plt.title('Top 10 Positions by Top 10 Cities')
plt.xticks(rotation=90)
plt.xlabel('City')
plt.ylabel('Count')
plt.show()
```

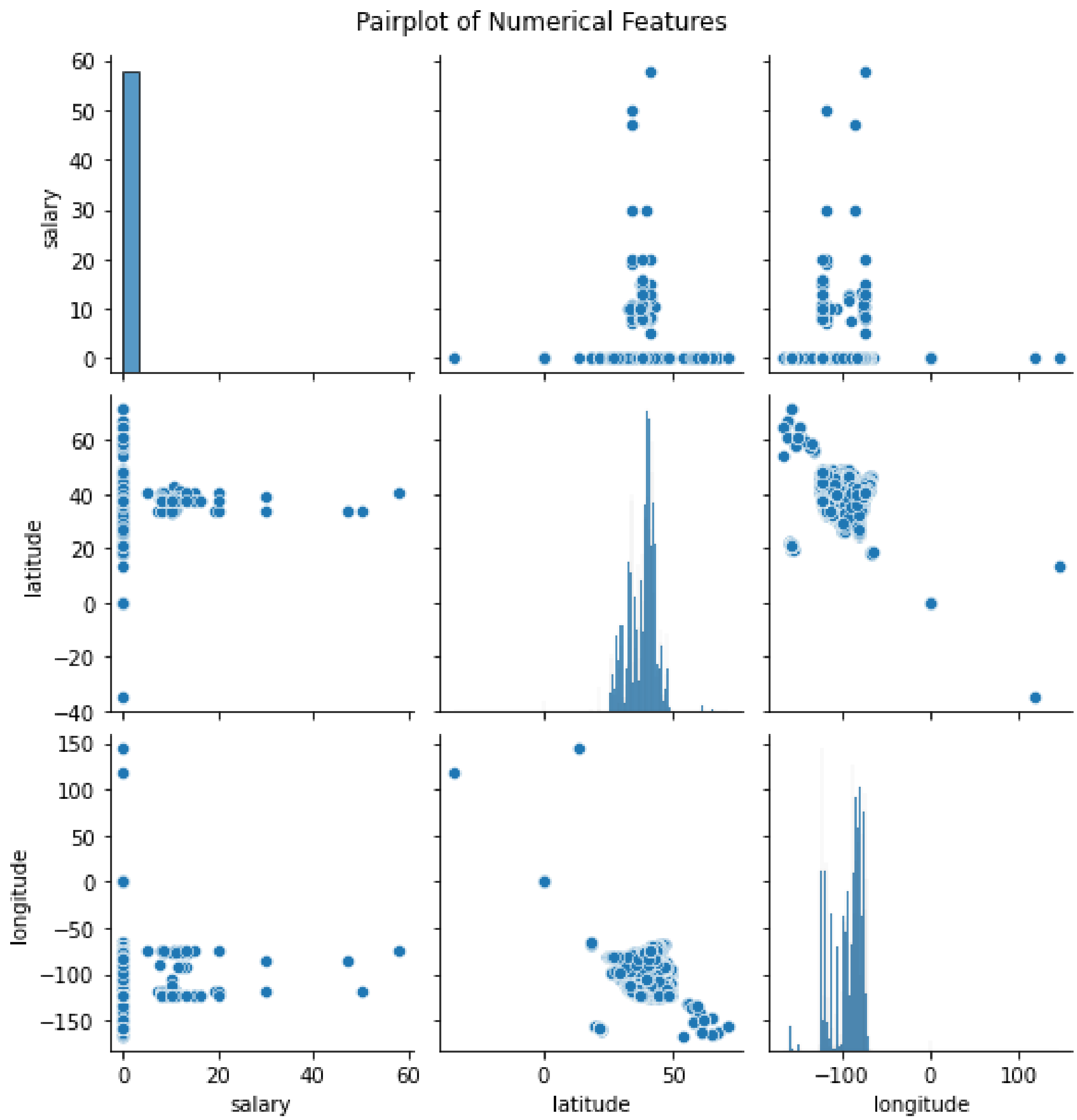


Insights: This visualization shows which positions are most common in the top 10 cities, revealing geographical preferences for certain job roles. It can assist job seekers in identifying where their desired positions are most likely to be available.

1.5.3 3.MULTIVARIATE

Pairplot of Numerical Columns

[99]:
```
sns.pairplot(df_jd[['salary', 'latitude', 'longitude']])
plt.suptitle('Pairplot of Numerical Features', y=1.02)
plt.show()
```



Insights: The pairplot reveals relationships between numerical variables salary, latitude, and longitude, indicating potential geographic salary trends. Clustering patterns might suggest areas with higher or lower salary offerings, helping in location-based job searches.

Salary vs. Industry and Employment type

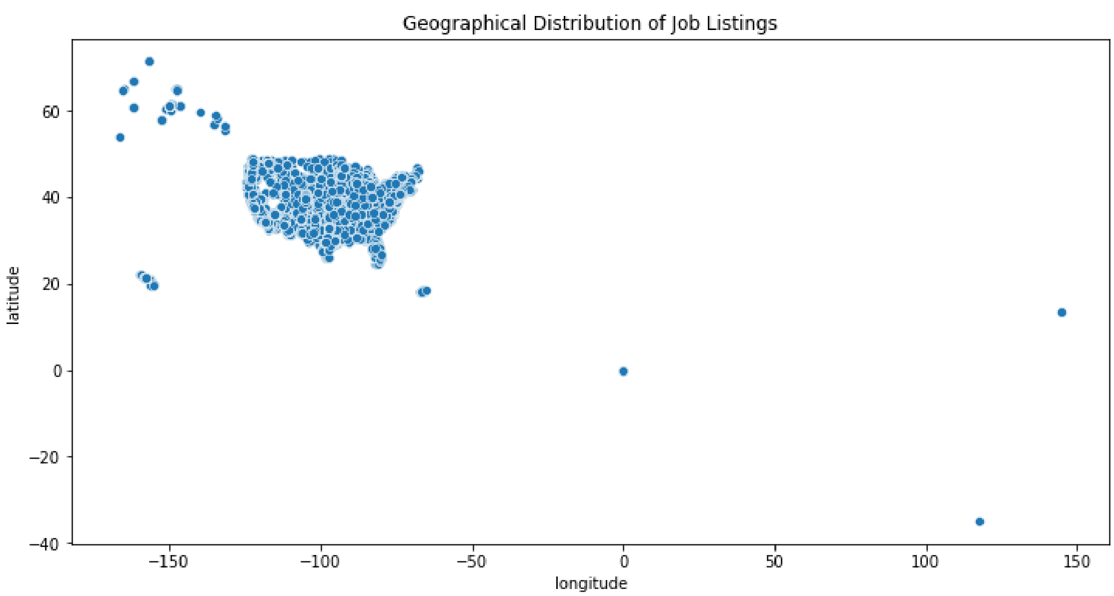
[101]:
```
plt.figure(figsize=(12, 6))
sns.scatterplot(x='salary', y='industry', hue='employment.type', data=df_jd)
plt.title('Salary vs. Industry by Employment Type')
plt.show()
```

Insights: The scatterplot shows the distribution of salaries across industries and employment types, allowing for a detailed comparison.

It highlights how certain employment types within industries may offer higher salaries, useful for negotiation and decision-making.

Geographical Distribution of Job Listings

[103]:
```
plt.figure(figsize=(12, 6))
sns.scatterplot(x='longitude', y='latitude', data=df_jd)
plt.title('Geographical Distribution of Job Listings')
plt.show()
```

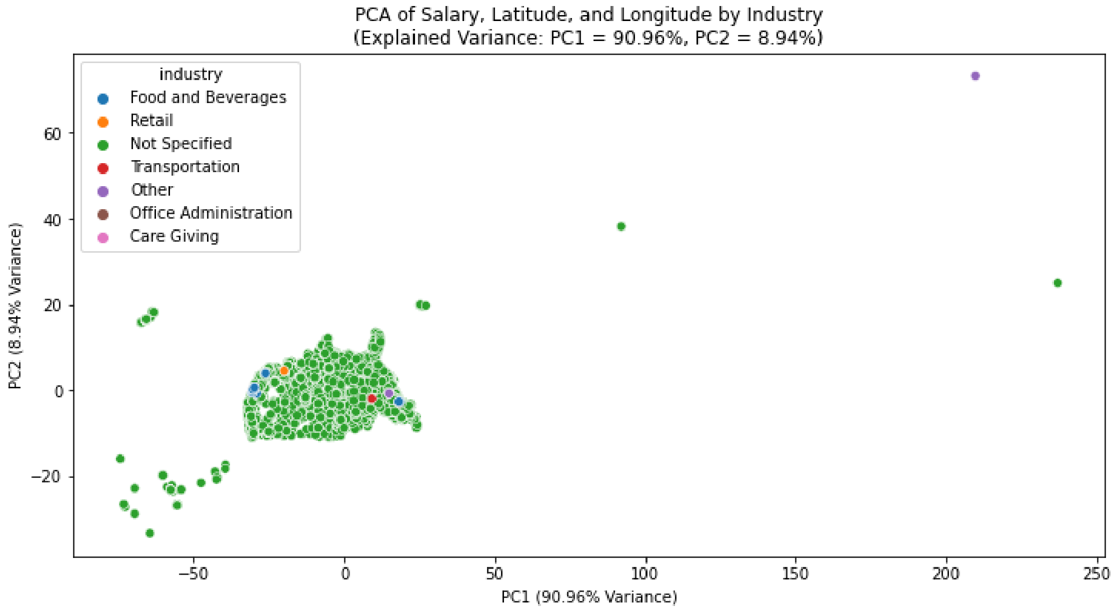


Insights This scatterplot provides a geographical overview of job listings, identifying key regions with dense job availability.

It is beneficial for job seekers considering relocation, as it highlights areas with more job opportunities.

PCA for Numerical Columns

[105]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Perform PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(df_jd[['salary', 'latitude',␣
↪'longitude']])

# Calculate the percentage of variance explained by each principal component
explained_variance = pca.explained_variance_ratio_ * 100

# Create a DataFrame for the PCA results
df_jd_pca = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])

# Plot the PCA results
plt.figure(figsize=(12, 6))
sns.scatterplot(x='PC1', y='PC2', data=df_jd_pca, hue=df_jd['industry'])
plt.title(f'PCA of Salary, Latitude, and Longitude by Industry\n'
          f'(Explained Variance: PC1 = {explained_variance[0]:.2f}%, PC2 =␣
↪{explained_variance[1]:.2f}%)')
plt.xlabel(f'PC1 ({explained_variance[0]:.2f}% Variance)')
plt.ylabel(f'PC2 ({explained_variance[1]:.2f}% Variance)')
plt.show()
```



Insights Principal Component Analysis (PCA) Results:

1.PC1 (Principal Component 1): Explains 90.96% of the variance. This indicates that PC1 captures the most significant variance in the data, which is likely driven by one or a combination of the features (salary, latitude, longitude).

2.PC2 (Principal Component 2): Explains 8.945% of the variance. This component explains a much smaller proportion of the variance compared to PC1, suggesting that its contribution is less significant.

Interpretation: 1.PC1's Dominance: The fact that PC1 accounts for 90.96% of the variance implies that the majority of the variance in the dataset can be attributed to the primary component. This component likely represents the most dominant feature or a combination of features affecting job listings.

2.PC2's Minor Role: PC2's contribution is relatively minor, capturing only about 8.945% of the variance. It may represent less significant features or secondary patterns in the data.

Industry Visualization: Industry Clusters: By plotting PC1 and PC2, we can observe how different industries are distributed across these principal components. Industries that are clustered together along PC1 suggest that their job listings share similar characteristics related to salary, latitude, and longitude.

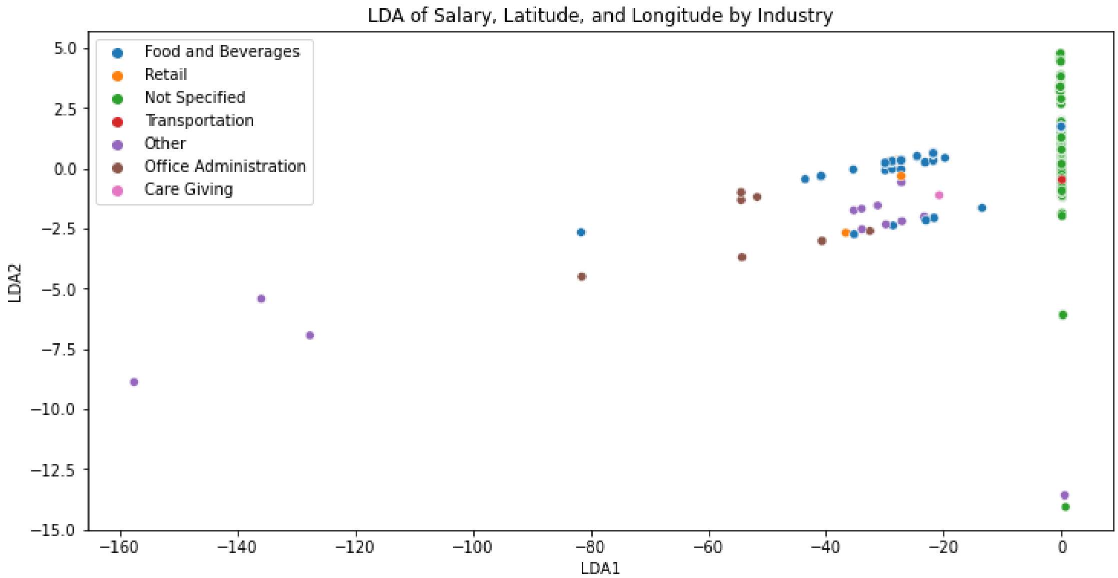
Application: Dimensionality Reduction: PCA is useful for reducing the dimensionality of the dataset while retaining most of the variance. This makes it easier to visualize and interpret complex datasets.

Feature Analysis: Understanding the features that contribute most to PC1 and PC2 can help in identifying key drivers in job listings and tailoring job recommendations or analyses accordingly.

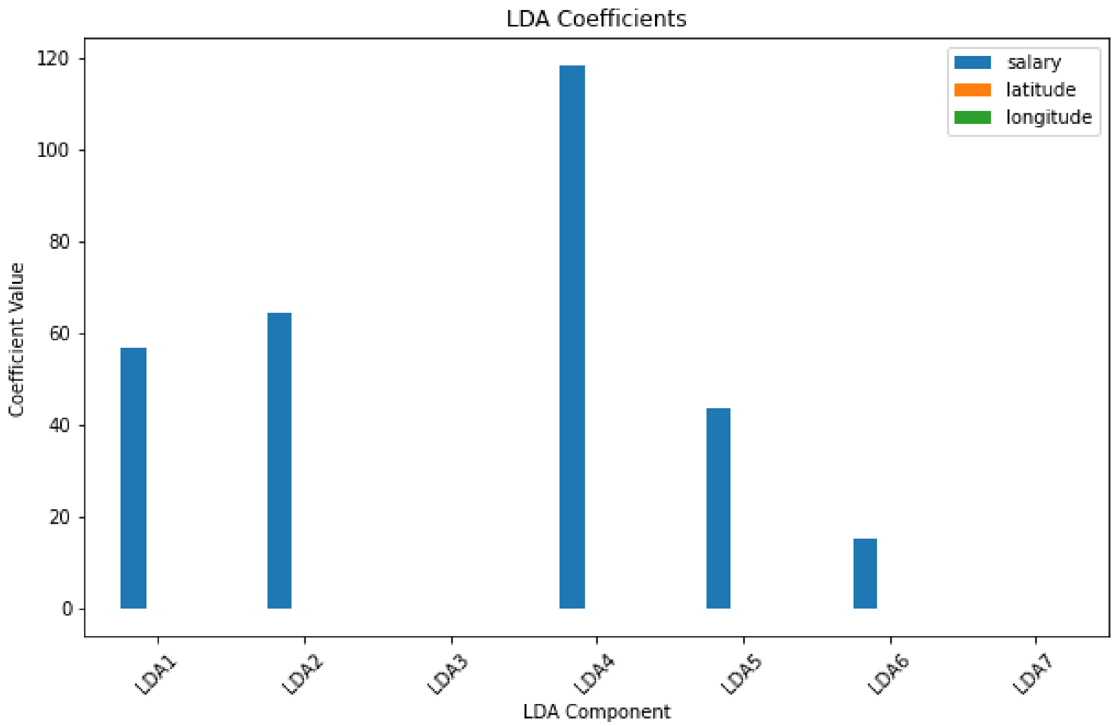
LDA FOR NUMERICAL COLUMNS

[109]:
```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Checking for missing values
if df_jd[['salary', 'latitude', 'longitude', 'industry']].isnull().any().any():
    df_jd = df_jd.dropna(subset=['salary', 'latitude', 'longitude', 'industry'])

# Preparing the Data
X = df_jd[['salary', 'latitude', 'longitude']]
y = df_jd['industry']

# Fitting LDA
lda = LDA(n_components=2)
X_lda = lda.fit_transform(X, y)

# Creating DataFrame for LDA results
df_jd_lda = pd.DataFrame(X_lda, columns=['LDA1', 'LDA2'])
df_jd_lda['industry'] = y.values

# Plotting LDA results
plt.figure(figsize=(12, 6))
sns.scatterplot(x='LDA1', y='LDA2', data=df_jd_lda, hue='industry',␣
 ↪palette='tab10')
plt.title('LDA of Salary, Latitude, and Longitude by Industry')
plt.xlabel('LDA1')
plt.ylabel('LDA2')
plt.legend(loc='best')
plt.show()

# Examining LDA coefficients
coefficients = lda.coef_
features = ['salary', 'latitude', 'longitude']
coef_df = pd.DataFrame(coefficients, columns=features, index=[f'LDA{i+1}' for i␣
 ↪in range(coefficients.shape[0])])
print("LDA Coefficients:")
print(coef_df)

# Plotting LDA coefficients
coef_df.plot(kind='bar', figsize=(10, 6))
plt.title('LDA Coefficients')
plt.xlabel('LDA Component')
```

```
plt.ylabel('Coefficient Value')
plt.xticks(rotation=45)
plt.show()
```




```
LDA Coefficients:
           salary  latitude  longitude
LDA1    56.527155  0.010183   0.003534
LDA2    64.348121 -0.025505  -0.090859
LDA3    -0.158066  0.000113   0.000203
LDA4   118.403816 -0.017885  -0.040009
LDA5    43.480817 -0.103462  -0.006474
LDA6    15.282083 -0.031761  -0.024223
LDA7    -0.154598 -0.005052  -0.051335
```



Insights

1. Salary is a Major Driver:

Salary has high coefficients in LDA1 and LDA2, indicating it plays a significant role in distinguishing between industries. This suggests salary differences are key in separating the industry categories in your dataset.

2. Geographic Factors (Latitude and Longitude) Have Minor Impact:

Latitude and longitude have relatively small coefficients in LDA1 and LDA2, showing they contribute less to distinguishing industries compared to salary.

3. LDA Components and Variance:

LDA1 and LDA2, with their high salary coefficients, capture the most meaningful variance related to industry classification. LDA3 to LDA7 have lower coefficients, indicating they contribute less to the separation between industries and may capture less relevant variance or noise.

4. Focus on Top Components:

The high coefficients in LDA1 and LDA2 suggest these components are the most informative for understanding how salary impacts industry classification. Simplifying your analysis to these components can provide clearer insights.

5. Visual Representation:

Visualizing the coefficients helps confirm the relative importance of each feature and component, making it easier to interpret the contributions and focus on the most significant factors.

PLOTLY

[111]:
```
import plotly.express as px

fig = px.scatter_mapbox(df_jd, lat="latitude", lon="longitude",␣
 ↪color="industry", size="salary",
                        color_continuous_scale=px.colors.cyclical.IceFire,␣
 ↪size_max=15, zoom=10)
fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

Insights

1. Industry Distribution:

The map shows how various industries are geographically distributed, highlighting regions with high concentrations of specific industries.

2. Salary Trends:

Marker size represents salary levels, indicating areas with high or low salaries. This helps identify regions with higher-paying job opportunities.

3. Industry Clusters:

Clusters of markers suggest where particular industries are more prevalent, indicating job availability and regional industry concentration.

4. Regional Salary Insights:

Variations in marker size across regions reveal salary trends, highlighting areas with significant salary differences.

1.6 STEP 3:Data Preprocessing

1.6.1 Selecting columns

[113]:

```
#Selecting columns for preprocessing and modelling
jd_fin = df_jd.loc[:, ['job.id', 'position', 'job.description']]

jd_fin = jd_fin.rename(columns={"job.id": "Job_ID",
                                "position": "Job_position",
                                "job.description": "Job_description"})

jd_fin = jd_fin.dropna(subset=["Job_description"])
jd_fin = jd_fin.drop_duplicates(subset=["Job_description"])

jd_fin.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59270 entries, 0 to 84089
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Job_ID           59270 non-null  int64
 1   Job_position     59270 non-null  object
 2   Job_description  59270 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.8+ MB
```

[115]:

```
#Selecting columns for preprocessing and modelling
cv_fin = df_cv.loc[:, ['applicant.id', 'position.name', 'job.description']]
cv_fin = cv_fin.rename(columns={"applicant.id": "Applicant_ID",
                                "position.name": "Current_position",
                                "job.description": "Current_jd"})

cv_fin = cv_fin.drop_duplicates(subset=["Current_jd"])

cv_fin.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5490 entries, 3 to 8652
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Applicant_ID      5490 non-null   int64
 1   Current_position  5490 non-null   object
 2   Current_jd        5490 non-null   object
dtypes: int64(1), object(2)
memory usage: 171.6+ KB
```

[117]:
```
# Combining columns to create new column
jd_fin["Job_pos_and_desc"] = jd_fin["Job_position"].map(str) + "  " +␣
↪jd_fin["Job_description"]
jd_fin.head()
```

[117]:
```
   Job_ID        Job_position  \
0     111              Server
1     113  Kitchen Staff/Chef
2     117           Bartender
3     121              Server
4     127  Kitchen Staff/Chef

                                     Job_description  \
0  Tacolicious' first Palo Alto store just opened…
1   \r\n\r\nNew French Brasserie in S.F. Financia…
2  We are a popular Mediterranean wine bar and re…
3    Serve food/drinks to customers in a profess…
4  Located at the heart of Hollywood, we are one …

                                    Job_pos_and_desc
0  Server  Tacolicious' first Palo Alto store jus…
1  Kitchen Staff/Chef   \r\n\r\nNew French Brasse…
2  Bartender  We are a popular Mediterranean wine…
3  Server     Serve food/drinks to customers in a…
4  Kitchen Staff/Chef  Located at the heart of Ho…
```

[119]:
```
# Combining all JDs for each Applicant
cv_fin = cv_fin.groupby("Applicant_ID").agg({"Current_position": " ".
↪join,"Current_jd": " ".join}).reset_index()
cv_fin.head()
```

[119]:
```
   Applicant_ID                                   Current_position  \
0             2                  Writer for the Uloop Blog Volunteer
1            38                    Sales Person & Phone Receptionist
2            78                                   Impact team member
3            89  Healthcare Specialist / Combat Medic Clerk's h…
4            96                  Cashier Receptionist Cashiet/Waiter

                                          Current_jd
0  * Wrote articles for the "Uloop Blog," which i…
1  Asking customer if they need any assistance an…
2  Help maintain merchandise flow, Work on fillin…
3  Clinical and field medicine, Healthcare educat…
4  Greeting people and introducing/recommend food…
```

[121]:
```
#converting to lowercase and splitting into tokens at caps
def text_preprocessing(data):
```

```
    data["Job_pos_and_desc"] = data["Job_pos_and_desc"].apply(lambda x: re.sub(␣
↪r"([A-Z][^a-z]*)", r" \1", x))
    data["Job_pos_and_desc"] = data["Job_pos_and_desc"].str.lower()

    return data
```

[ ]:
```
#tagging
def process_data(data):
    pos_noninformative = {".", "CC", "CD", "DT", "IN", "LS", "MD", "POS", "PRP",
                          "PRP$", "TO", "UH", "WDT", "WP", "WP$", "WRB"}

# Tokenizing, tagging POS and filtering non-informative tokens
    data["POS"] = pos_tag_sents(map(word_tokenize, data["Job_pos_and_desc"].
↪tolist()))
    data["POS_clean"] = data["POS"].apply(lambda x: [pair for pair in x if␣
↪pair[0] != "nbsp" and pair[1] not in pos_noninformative])
    data["clean_token"] = data["POS_clean"].apply(lambda x: [word[0] for word␣
↪in x])
    data["token_number"] = data["clean_token"].apply(len)

    return data
# Processing the data
jd_fin = process_data(jd_fin)

jd_fin.head()
```

[ ]:
```
# Filtering tokens based on their length
jd_fin["clean_token"] = jd_fin["clean_token"].apply(lambda x: [subelt for␣
↪subelt in x if len(subelt) > 1])

# Updating the token_number column
def update_token_count(data):
    data["token_number"] = data["clean_token"].apply(len)
    return data

jd_fin = update_token_count(jd_fin)

jd_fin.head()
```

[ ]:
```
# Returning tokens into one single string for lemmatization
jd_fin["clean_jd"] = [" ".join(x) for x in jd_fin["clean_token"]]
jd_fin.head()
```

[ ]:
```
wnl = WordNetLemmatizer()
patterns = "[^a-zA-Z \n\.]"
```

```
# Using stopwords list from nltk and adding extra stopwords
stopwords_eng = stopwords.words("english")
additional_stopwords = ["also", "new", "etc", "part", "time", "hours", "hour",
                        "week", "per", "please", "offer", "part time",␣
↪"example",
                        "monday", "tuesday", "wednesday", "thursday", "friday",
                        "saturday", "sunday", "pm", "am"]
stopwords_eng.extend(additional_stopwords)

def lemmatize_sentence(text):
    text = re.sub(patterns, " ", text)
    tokens = [wnl.lemmatize(token.strip()) for token in text.split() if token␣
↪and token not in stopwords_eng]
    return " ".join(tokens)

print("Before lemmatization:\n", jd_fin["clean_jd"].iloc[1])
print("\nAfter lemmatization:\n", lemmatize_sentence(jd_fin["clean_jd"].
↪iloc[1]))
```

[ ]:
```
# Getting new column with lemmatize text
jd_fin["jd_lem"] = jd_fin["clean_jd"].apply(lemmatize_sentence)
jd_fin.head()
```

[ ]:
```
# Updating token_num_aft_lem column
jd_fin["token_num_aft_lem"] = [len(word.split()) for word in jd_fin["jd_lem"]]
jd_fin.head()
```

[ ]:
```
# Data visualization: histogram and boxplot
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))

sns.histplot(
    data=jd_fin,
    x="token_num_aft_lem",
    bins=25,
    kde=True,
    ax=axes[0]
).set_title("Distribution of word counts in job description", fontsize=16)

sns.boxplot(
    data=jd_fin,
    x="token_num_aft_lem",
    orient="h",
    width=0.9,
    ax=axes[1]
)
plt.tight_layout()
```

```
[ ]: # Number of Job Descriptions with less than 10 words
display(jd_fin[jd_fin["token_num_aft_lem"] <= 10]["Job_position"].count())
```

```
[ ]: # Number of Job Descriptions with more than 290 words
display(len(jd_fin[jd_fin["token_num_aft_lem"] > 290]["Job_ID"]))
```

```
[ ]: # Removing outliers (JDs with <= 10 or >290 words)
jd_fin = jd_fin[(jd_fin["token_num_aft_lem"] <=␣
 ↪290)&(jd_fin["token_num_aft_lem"] > 10)]
jd_fin.shape[0]
```

```
[ ]: # Data visualization: histogram and boxplot
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))

sns.histplot(
    data=jd_fin,
    x="token_num_aft_lem",
    bins=25,
    kde=True,
    ax=axes[0]
).set_title("Distribution of word counts in job description", fontsize=16)

sns.boxplot(
    data=jd_fin,
    x="token_num_aft_lem",
    orient="h",
    width=0.9,
    ax=axes[1]
)
plt.tight_layout()
```

```
[ ]: jd_mod = jd_fin.reset_index(drop=True)
display(jd_mod.head())
display(jd_mod.iloc[-2:])
jd_mod.shape
```

```
[ ]: jd_mod = jd_mod[["Job_ID", "Job_position", "Job_description",␣
 ↪"Job_pos_and_desc", "jd_lem"]].iloc[:]
display(jd_mod.head())
jd_mod.shape
```

1.7 Step 4: Modelling

1.7.1 Content Based model

[12]:
```
# Generating Tfidf matrix
tfidf_vect = TfidfVectorizer(min_df=5,
                             max_df=0.95
                            )

tfidf = tfidf_vect.fit(jd_mod["jd_lem"])
tfidf
```

[12]:
```
TfidfVectorizer(max_df=0.95, min_df=5)
```

[13]:
```
# Vectorize applicants' job experience
cv_tfidf = tfidf_vect.transform(cv_fin["Current_jd"])
cv_tfidf
```

[13]:
```
<2313x16171 sparse matrix of type '<class 'numpy.float64'>'
        with 119232 stored elements in Compressed Sparse Row format>
```

[14]:
```
def Applicant_Job (job_id):
    # Check if the job ID exists in the job descriptions DataFrame
    if job_id in jd_mod["Job_ID"].tolist():
        index = np.where(jd_mod["Job_ID"] == job_id)[0]
        jd_q = jd_mod.iloc[index[0]:(index[-1] + 1)]

        jd_tfidf = tfidf_vect.transform(jd_q["Job_description"])

        similarity_score = [linear_kernel(jd_tfidf, cv_tfidf[i])[0][0] for i in␣
↪range(cv_tfidf.shape[0])]

        # Getting the top 10 recommendations
        top_indices = sorted(range(len(similarity_score)), key=lambda i:␣
↪similarity_score[i], reverse=True)[:10]
        recommendation = pd.DataFrame({
            "Job_ID": [job_id] * 10,
            "Recommended_Applicant_ID": [cv_fin["Applicant_ID"].iloc[i] for i␣
↪in top_indices],
            "Score": [similarity_score[i] for i in top_indices]
        })

        nearest_candidates = recommendation["Recommended_Applicant_ID"]
        applicant_recommended = pd.DataFrame(columns=["Job_ID", "Job_position",␣
↪"Recommended_Applicant_ID", "Work_experience", "Previous_job", "Score"])

        for count, applicant_id in enumerate(nearest_candidates):
```

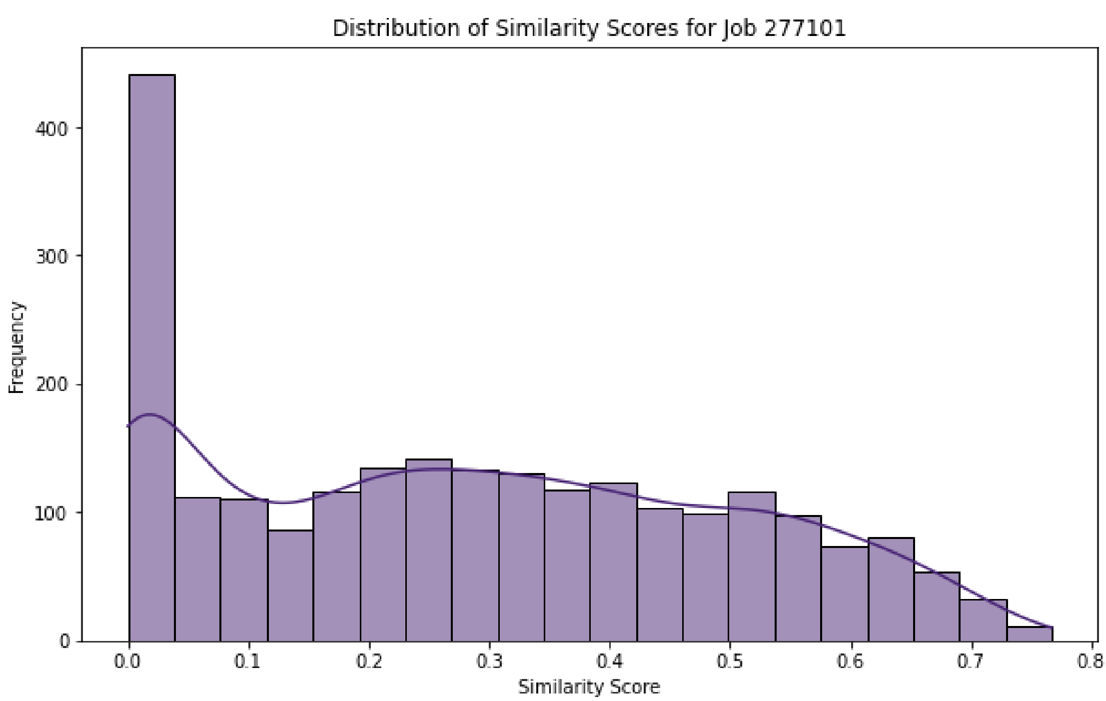
```
            index_resume = cv_fin.index[cv_fin["Applicant_ID"] ==␣
↪applicant_id][0]
            job_position = jd_mod[jd_mod["Job_ID"] == job_id]["Job_position"].
↪iloc[0]
            work_experience = cv_fin["Current_jd"].iloc[index_resume]
            previous_job = cv_fin["Current_position"].iloc[index_resume]
            score = recommendation["Score"].iloc[count]

            applicant_recommended.loc[count] = [job_id, job_position,␣
↪applicant_id, work_experience, previous_job, score]

        return applicant_recommended
    else:
        return "This Job_ID is not in the Jobs' list"
```

[15]:
```
Applicant_Job (277101)
```

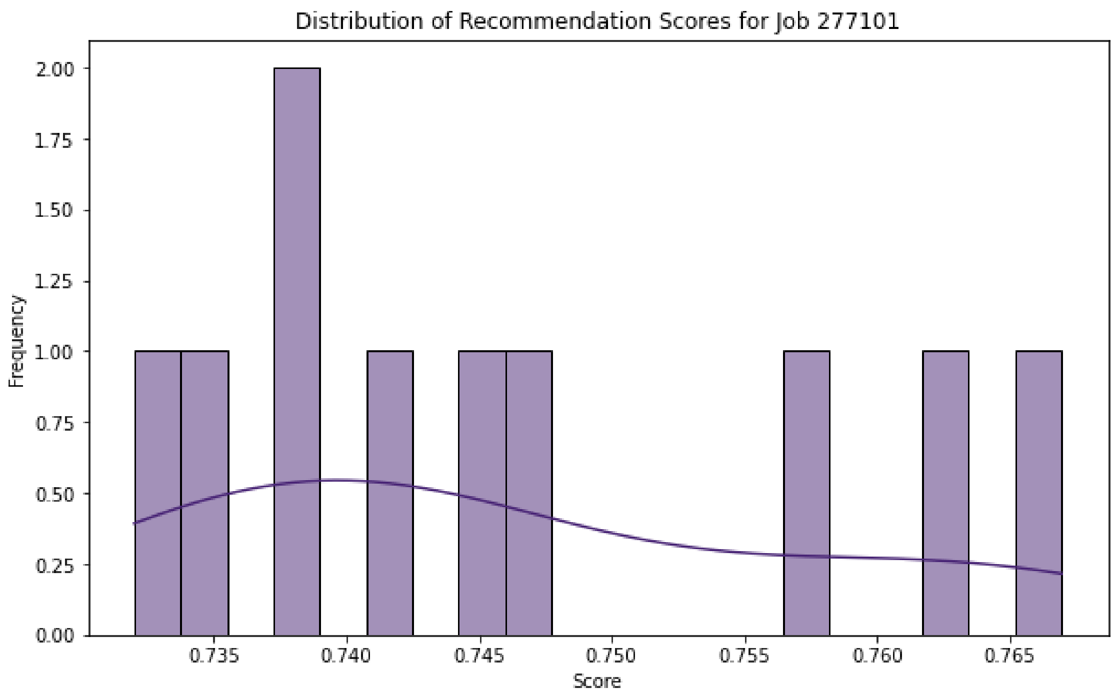
[15]:
```
   Job_ID    Job_position Recommended_Applicant_ID  \
0  277101  Office Manager                    13876
1  277101  Office Manager                     4693
2  277101  Office Manager                     3544
3  277101  Office Manager                    14542
4  277101  Office Manager                     2352
5  277101  Office Manager                    11710
6  277101  Office Manager                    11509
7  277101  Office Manager                    13852
8  277101  Office Manager                    10774
9  277101  Office Manager                     3775

                                     Work_experience  \
0  *Manage and organize electronic data according…
1  Troubleshoot, analyze, detect, identify and co…
2  Construction Coordinator overseeing / performi…
3  Answered all inbound calls relating to medical…
4  Reporting to the COO, providing strategic lead…
5  Certified Business Mentor, providing small bus…
6  Iintership Event Coordinator, contributing to …
7  Painter, Entrepreneur, Blogger, Organizer, Res…
8  CEG owns and operates English as a second lang…
9  > Improve operational systems to support bette…

                                        Previous_job     Score
0  HRMS Data Analyst (Temp) HRIS Ops Analyst HRIS…  0.766952
1  HRIS Specialist HRIS Analyst HRB Client Suppor…  0.762605
2  Administrative Construction Coordinator Execut…  0.757563
3  Member Services Specialist (temp) Member Servi…  0.746041
4  Director of Operations Quality Manager Manager…  0.744535
```

```
5  SCORE Fox Valley Branch Manager for Lisle Bran…  0.741553
6  ATHGO forum 2011 Finance Controller Independen…  0.738170
7  Painter Entrepreneur, Blogger, Painter, Organi…  0.737715
8  US IT Support IT Service Delivery Lead Tech Co…  0.735139
9  Director of Operations Assistant to the Managi…  0.732003
```

[16]:
```python
#getting all similarity scores for a given job
def get_similarity_scores_for_job(job_id):
    index = np.where(jd_mod["Job_ID"] == job_id)[0]
    jd_q = jd_mod.iloc[index[0]:(index[-1] + 1)]
    jd_tfidf = tfidf_vect.transform(jd_q["Job_description"])
    similarity_scores = [linear_kernel(jd_tfidf, cv_tfidf[i])[0][0] for i in␣
 ↪range(cv_tfidf.shape[0])]
    return similarity_scores

job_id = 277101
similarity_scores = get_similarity_scores_for_job(job_id)

# Plotting the distribution of similarity scores
plt.figure(figsize=(10, 6))
sns.histplot(similarity_scores, bins=20, kde=True, color=plt.cm.viridis(0.1))
plt.xlabel('Similarity Score')
plt.ylabel('Frequency')
plt.title(f'Distribution of Similarity Scores for Job {job_id}')
plt.show()
```



[17]:
```
applicant_recommended = Applicant_Job(job_id)

plt.figure(figsize=(10, 6))
sns.histplot(applicant_recommended['Score'], bins=20, kde=True, color=plt.cm.
↪viridis(0.1))
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.title(f'Distribution of Recommendation Scores for Job {job_id}')
plt.show()
```



[18]:
```
def Job_Applicant(applicant_id):
    if applicant_id not in cv_fin["Applicant_ID"].tolist():
        return "This Applicant_ID is not in Applicants' list"

    index = np.where(cv_fin["Applicant_ID"] == applicant_id)[0]
    cv_q = cv_fin.iloc[index[0]:(index[-1] + 1)]

    cv_tfidf = tfidf_vect.transform(cv_q["Current_jd"])
    jd_tfidf = tfidf_vect.transform(jd_mod["Job_description"])

    similarity_scores = [linear_kernel(cv_tfidf, jd_tfidf[i])[0][0] for i in␣
↪range(jd_tfidf.shape[0])]
```

```python
    if len(cv_q) > 1:
        print("\nThis Applicant has more than 1 resume (different job␣
↪description)\n")
    else:
        print("\nThis Applicant has only 1 resume\n")

    # Get the top 10 recommendations
    top_indices = sorted(range(len(similarity_scores)), key=lambda i:␣
↪similarity_scores[i], reverse=True)[:10]
    recommendation = pd.DataFrame({
        "Applicant_ID": [applicant_id] * 10,
        "Recommended_Job_ID": [jd_mod["Job_ID"].iloc[i] for i in top_indices],
        "Score": [similarity_scores[i] for i in top_indices]
    })

    job_recommended = pd.DataFrame(columns=["Applicant_ID",␣
↪"Applicant_job_title", "Recommended_Job_ID", "Job_description", "Job_title",␣
↪"Score"])

    for count, job_id in enumerate(recommendation["Recommended_Job_ID"]):
        index_vacancy = jd_mod.index[jd_mod["Job_ID"] == job_id][0]
        applicant_job_title = cv_fin["Current_position"].iloc[index[0]:
↪(index[-1] + 1)].tolist()
        job_description = jd_mod["Job_description"].iloc[index_vacancy]
        job_title = jd_mod["Job_position"].iloc[index_vacancy]
        score = recommendation["Score"].iloc[count]

        job_recommended.loc[count] = [applicant_id, applicant_job_title,␣
↪job_id, job_description, job_title, score]

    return job_recommended
```

[19]:

```python
Job_Applicant(1801)
```

```
This Applicant has only 1 resume
```

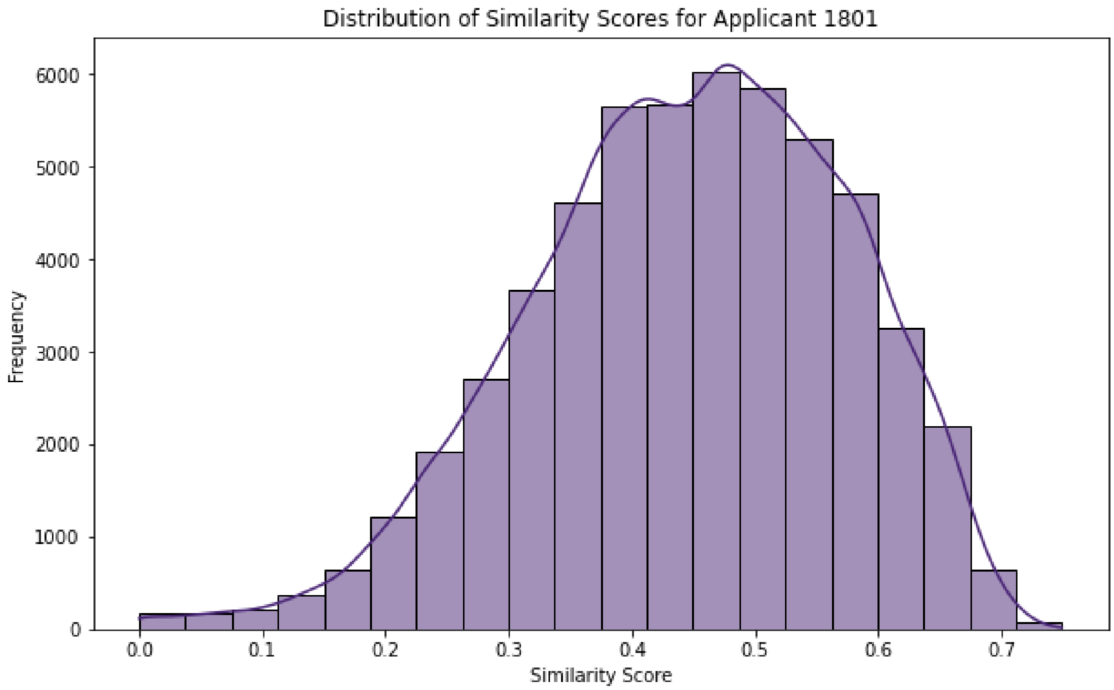
[19]:

```
  Applicant_ID                                 Applicant_job_title  \
0         1801  [Credit Analyst Server Deputy Program Manager …
1         1801  [Credit Analyst Server Deputy Program Manager …
2         1801  [Credit Analyst Server Deputy Program Manager …
3         1801  [Credit Analyst Server Deputy Program Manager …
4         1801  [Credit Analyst Server Deputy Program Manager …
5         1801  [Credit Analyst Server Deputy Program Manager …
6         1801  [Credit Analyst Server Deputy Program Manager …
7         1801  [Credit Analyst Server Deputy Program Manager …
```

```
8         1801  [Credit Analyst Server Deputy Program Manager …
9         1801  [Credit Analyst Server Deputy Program Manager …

  Recommended_Job_ID                                   Job_description  \
0             267571  Ref ID: 00370-9735287Classification: Customer …
1             288547  Job Summary: \r\n\r\n\r\n\r\nThis person must …
2             273426  This is a part-time position for the Kins…
3             253908  \r\n\r\nPOSITION SUMMARY:\r\n\r\nThe Registere…
4             273937  Duties Responsibilities\r\n\r\nConceptualizing…
5             313161  \r\n\r\nPOSITION SUMMARY:\r\n\r\n\r\n\r\nThe R…
6             260627  \r\n\r\nPOSITION SUMMARY:\r\n\r\n\r\n\r\nThe R…
7             260814  Ref ID:04250-107317Classification:Accountant -…
8             278483  \r\n\r\nPOSITION SUMMARY:\r\n\r\n\r\n\r\nThe R…
9             276614  JOB SUMMARY\r\nThe Senior Financial Business A…

                                          Job_title     Score
0                       Customer Service Supervisor  0.749123
1  CNA II (Critical Care, Part Time - 40 hours / …  0.742250
2                      Private Duty, RN/LPN (Kinston)  0.741244
3    RN Endo/GI PRN - Mountain West Endoscopy Center  0.738272
4  Merchandiser at Greenville-Spartanburg Int'l A…  0.737349
5                     RN OR PRN - Sahara Surgery Center  0.733558
6                  RN OR PRN - Wasatch Endoscopy Center  0.732831
7                                  Senior Accountant  0.732804
8                   RN OR PRN - Flamingo Surgery Center  0.732597
9                     Senior Financial Business Analyst  0.732555
```

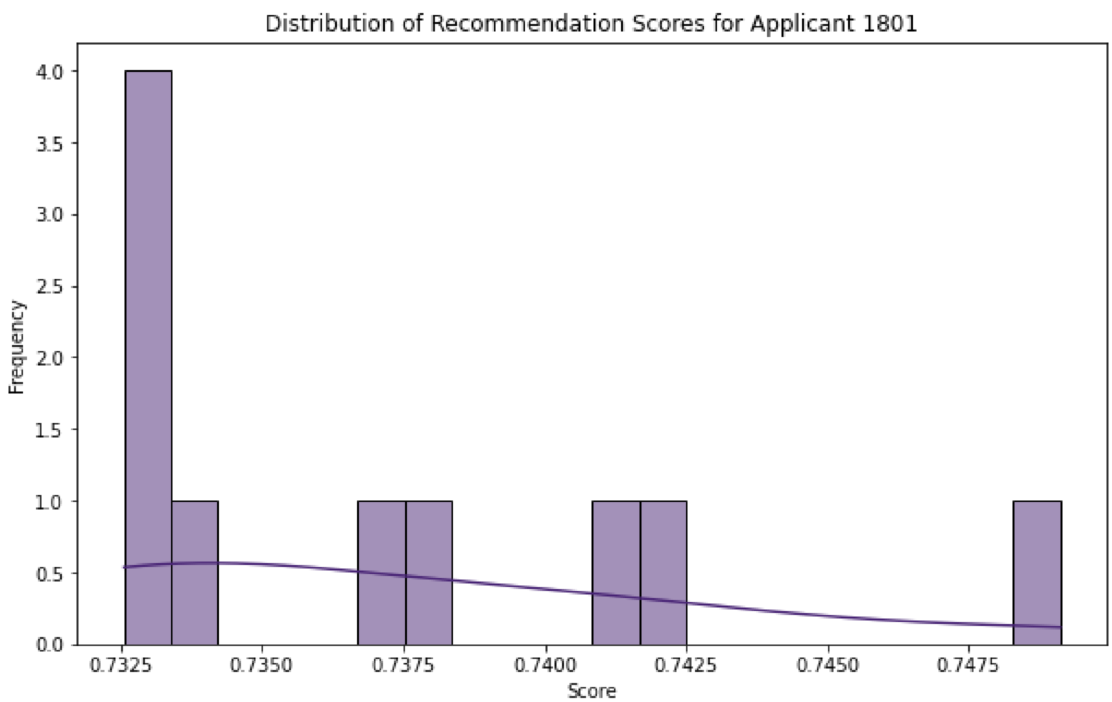
[20]:
```python
# Getting similarity scores for a given applicant
def get_similarity_scores_for_applicant(applicant_id):
    index = np.where(cv_fin["Applicant_ID"] == applicant_id)[0]

    if len(index) == 0:
        print(f"No data found for Applicant_ID {applicant_id}")
        return []

    cv_q = cv_fin.iloc[index[0]:(index[-1] + 1)]
    cv_tfidf = tfidf_vect.transform(cv_q["Current_jd"])

    similarity_scores = [linear_kernel(cv_tfidf, tfidf_vect.
 ↪transform([job_desc]))[0][0] for job_desc in jd_mod["Job_description"]]

    return similarity_scores

applicant_id = 1801
similarity_scores = get_similarity_scores_for_applicant(applicant_id)

if similarity_scores:
```

```
    # Plotting the distribution of similarity scores
    plt.figure(figsize=(10, 6))
    sns.histplot(similarity_scores, bins=20, kde=True, color=plt.cm.viridis(0.
↪1))
    plt.xlabel('Similarity Score')
    plt.ylabel('Frequency')
    plt.title(f'Distribution of Similarity Scores for Applicant {applicant_id}')
    plt.show()
```



[21]:
```
job_recommended = Job_Applicant(applicant_id)

plt.figure(figsize=(10, 6))
sns.histplot(job_recommended['Score'], bins=20, kde=True, color=plt.cm.
↪viridis(0.1))
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.title(f'Distribution of Recommendation Scores for Applicant {applicant_id}')
plt.show()
```

This Applicant has only 1 resume



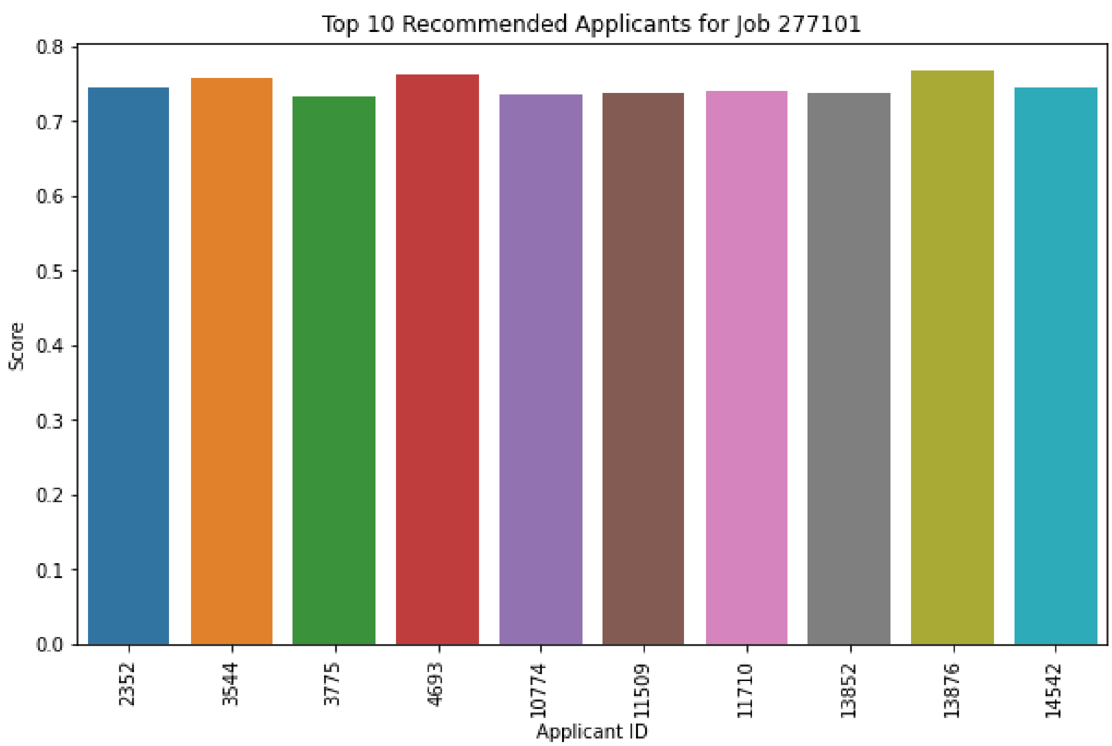
[22]:
```
import numpy as np

def precision_at_k(recommendations, ground_truth, k):
    precision_sum = 0
    count = 0
    for job, gt_list in ground_truth.items():
        rec_list = recommendations.get(job, [])
        relevant_recs = set(rec_list[:k]) & set(gt_list)
        precision_sum += len(relevant_recs) / k
        count += 1
    return precision_sum / count if count > 0 else 0

def recall_at_k(recommendations, ground_truth, k):
    recall_sum = 0
    count = 0
    for job, gt_list in ground_truth.items():
        rec_list = recommendations.get(job, [])
        relevant_recs = set(rec_list[:k]) & set(gt_list)
        recall_sum += len(relevant_recs) / len(gt_list)
        count += 1
    return recall_sum / count if count > 0 else 0

def mrr(recommendations, ground_truth):
    mrr_sum = 0
```

```python
    count = 0
    for job, gt_list in ground_truth.items():
        rec_list = recommendations.get(job, [])
        for i, rec in enumerate(rec_list):
            if rec in gt_list:
                mrr_sum += 1 / (i + 1)
                break
        count += 1
    return mrr_sum / count if count > 0 else 0

def ndcg_at_k(recommendations, ground_truth, k):
    ndcg_sum = 0
    count = 0
    for job, gt_list in ground_truth.items():
        rec_list = recommendations.get(job, [])
        ideal_dcg = sum(1 / np.log2(i + 2) for i, rec in␣
↪enumerate(sorted(gt_list, key=lambda x: rec_list.index(x) if x in rec_list␣
↪else len(rec_list))))
        dcg = sum(1 / np.log2(i + 2) if rec in gt_list else 0 for i, rec in␣
↪enumerate(rec_list[:k]))
        ndcg_sum += dcg / ideal_dcg if ideal_dcg > 0 else 0
        count += 1
    return ndcg_sum / count if count > 0 else 0

ground_truth = {
    'Accounting': [4842, 4407, 4444, 2193, 2107, 13851, 6028, 3419],
    'Child care': [4908, 3382, 4575, 4936, 5309, 2903, 2915, 3275],
    'Education': [3427, 14492, 3303, 3223, 4598, 3068, 7370, 12926],
    'Human resources': [4598, 3360, 3326, 13860, 14192, 13871, 13841, 13870],
    'Office manager': [13944, 3460, 5921, 10023, 14178, 14230, 14278, 2353]
}

recommendations = {
    'Accounting': [4842, 4407, 6028, 3419, 13003, 1754, 3163, 10424],
    'Child care': [3382, 4936, 4575, 14542, 5309, 3275, 2903, 8764],
    'Education': [3427, 3303, 7370, 14492, 10200, 3059, 11509, 9152],
    'Human resources': [3360, 13860, 4623, 13870, 13871, 14542, 11191, 13937],
    'Office manager': [13944, 3460, 14278, 2352, 10774, 5921, 3544, 13876]
}


# Calculating metrics
precision = precision_at_k(recommendations, ground_truth, k=3)
recall = recall_at_k(recommendations, ground_truth, k=3)
mrr_value = mrr(recommendations, ground_truth)
ndcg = ndcg_at_k(recommendations, ground_truth, k=3)
```

```
print('Precision: ', precision)
print('Recall: ', recall)
print('mrr_value: ', mrr_value)
print('ndcg: ', ndcg)
```

```
Precision:  0.9333333333333332
Recall:  0.35
mrr_value:  1.0
ndcg:  0.5137088609996164
```

[23]:
```
import matplotlib.pyplot as plt
import seaborn as sns

top_k = 10
plt.figure(figsize=(10, 6))
sns.barplot(x=applicant_recommended['Recommended_Applicant_ID'],␣
 ↪y=applicant_recommended['Score'])
plt.xlabel('Applicant ID')
plt.ylabel('Score')
plt.title(f'Top {top_k} Recommended Applicants for Job {job_id}')
plt.xticks(rotation=90)
plt.show()
```

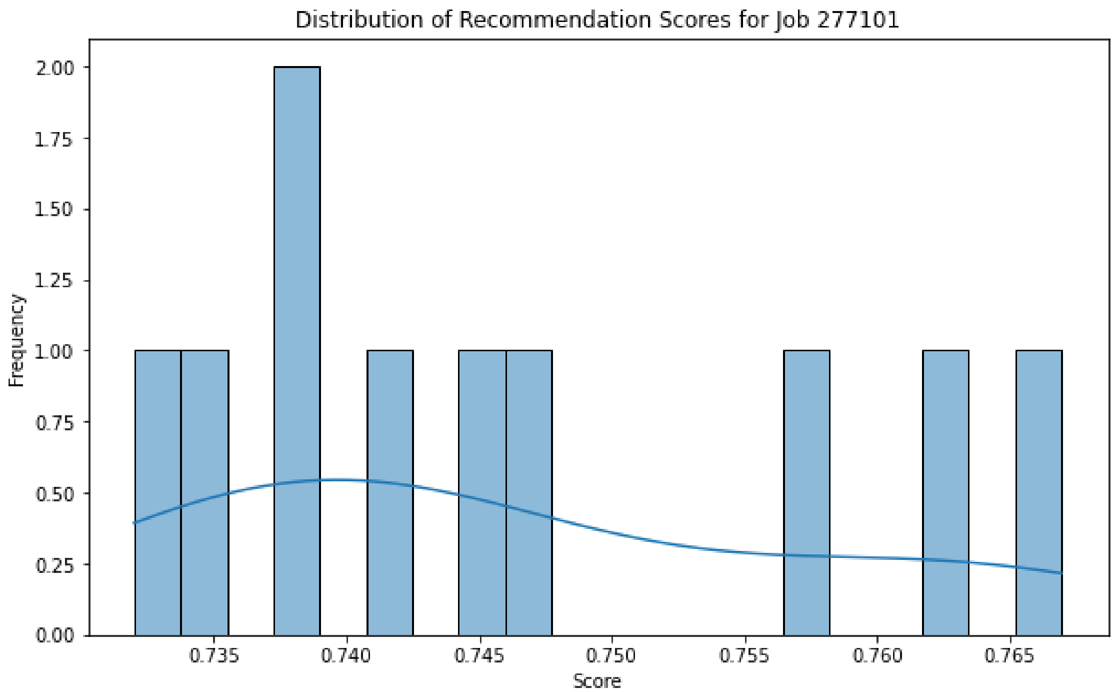


[24]:
```
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


def Applicant_Job(job_id):
    if job_id in jd_mod["Job_ID"].tolist():
        index = np.where(jd_mod["Job_ID"] == job_id)[0]
        jd_q = jd_mod.iloc[index[0]:(index[-1] + 1)]

        jd_tfidf = tfidf_vect.transform(jd_q["Job_description"])
        similarity_score = cosine_similarity(jd_tfidf, cv_tfidf)
        mean_similarity_score = similarity_score.mean(axis=0)

        # Get the top 10 recommendation
        top_indices = sorted(range(len(mean_similarity_score)), key=lambda i:␣
↪mean_similarity_score[i], reverse=True)[:10]
        recommendation = pd.DataFrame({
            "Job_ID": [job_id] * 10,
            "Recommended_Applicant_ID": [cv_fin["Applicant_ID"].iloc[i] for i␣
↪in top_indices],
            "Score": [mean_similarity_score[i] for i in top_indices]
        })

        nearest_candidates = recommendation["Recommended_Applicant_ID"]
        applicant_recommended = pd.DataFrame(columns=["Job_ID", "Job_position",␣
↪"Recommended_Applicant_ID", "Work_experience", "Previous_job", "Score"])

        for count, applicant_id in enumerate(nearest_candidates):
            index_resume = cv_fin.index[cv_fin["Applicant_ID"] ==␣
↪applicant_id][0]
            job_position = jd_mod[jd_mod["Job_ID"] == job_id]["Job_position"].
↪iloc[0]
            work_experience = cv_fin["Current_jd"].iloc[index_resume]
            previous_job = cv_fin["Current_position"].iloc[index_resume]
            score = recommendation["Score"].iloc[count]

            applicant_recommended.loc[count] = [job_id, job_position,␣
↪applicant_id, work_experience, previous_job, score]

        return applicant_recommended
    else:
        return "This Job_ID is not in the Jobs' list"

def Job_Applicant(applicant_id):
```

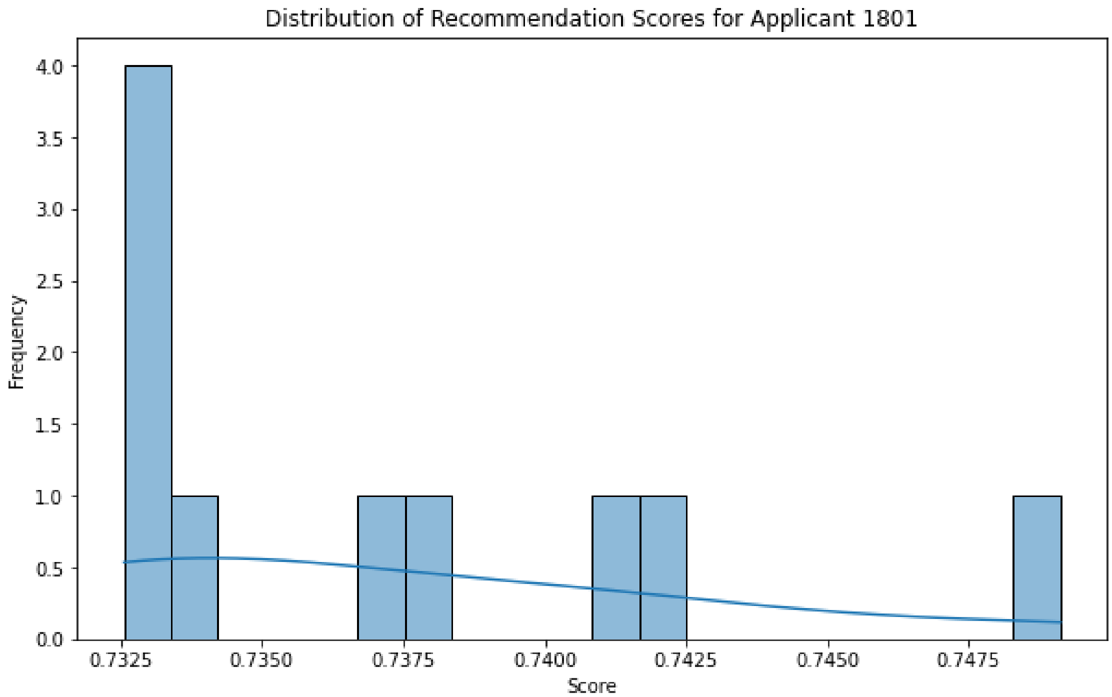
```
    if applicant_id not in cv_fin["Applicant_ID"].tolist():
        return "This Applicant_ID is not in Applicants' list"

    index = np.where(cv_fin["Applicant_ID"] == applicant_id)[0]
    cv_q = cv_fin.iloc[index[0]:(index[-1] + 1)]

    cv_tfidf = tfidf_vect.transform(cv_q["Current_jd"])
    jd_tfidf = tfidf_vect.transform(jd_mod["Job_description"])

    similarity_scores = cosine_similarity(cv_tfidf, jd_tfidf)
    mean_similarity_scores = similarity_scores.mean(axis=0)

    if len(cv_q) > 1:
        print("\nThis Applicant has more than 1 resume (different job␣
↪description)\n")
    else:
        print("\nThis Applicant has only 1 resume\n")

    # Get the top 10 recommendations
    top_indices = sorted(range(len(mean_similarity_scores)), key=lambda i:␣
↪mean_similarity_scores[i], reverse=True)[:10]
    recommendation = pd.DataFrame({
        "Applicant_ID": [applicant_id] * 10,
        "Recommended_Job_ID": [jd_mod["Job_ID"].iloc[i] for i in top_indices],
        "Score": [mean_similarity_scores[i] for i in top_indices]
    })

    job_recommended = pd.DataFrame(columns=["Applicant_ID",␣
↪"Applicant_job_title", "Recommended_Job_ID", "Job_description", "Job_title",␣
↪"Score"])

    for count, job_id in enumerate(recommendation["Recommended_Job_ID"]):
        index_vacancy = jd_mod.index[jd_mod["Job_ID"] == job_id][0]
        applicant_job_title = cv_fin["Current_position"].iloc[index[0]:
↪(index[-1] + 1)].tolist()
        job_description = jd_mod["Job_description"].iloc[index_vacancy]
        job_title = jd_mod["Job_position"].iloc[index_vacancy]
        score = recommendation["Score"].iloc[count]

        job_recommended.loc[count] = [applicant_id, applicant_job_title,␣
↪job_id, job_description, job_title, score]

    return job_recommended
```

[25]:
```
# Job Recommendations
plt.figure(figsize=(10, 6))
sns.histplot(applicant_recommended['Score'], bins=20, kde=True)
```

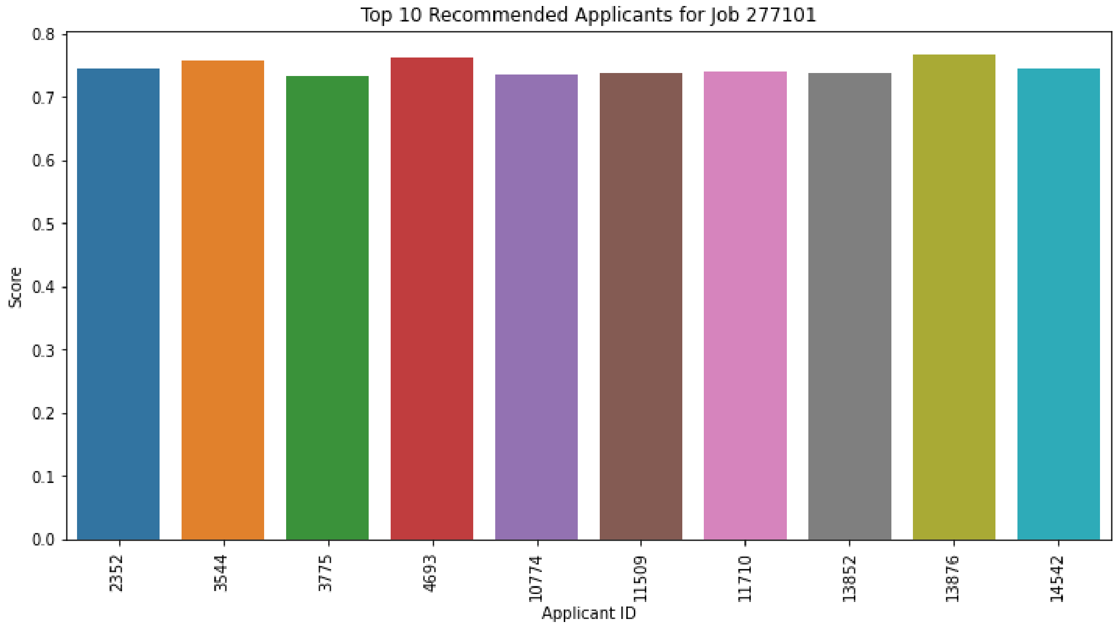
```
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.title(f'Distribution of Recommendation Scores for Job {job_id}')
plt.show()
```



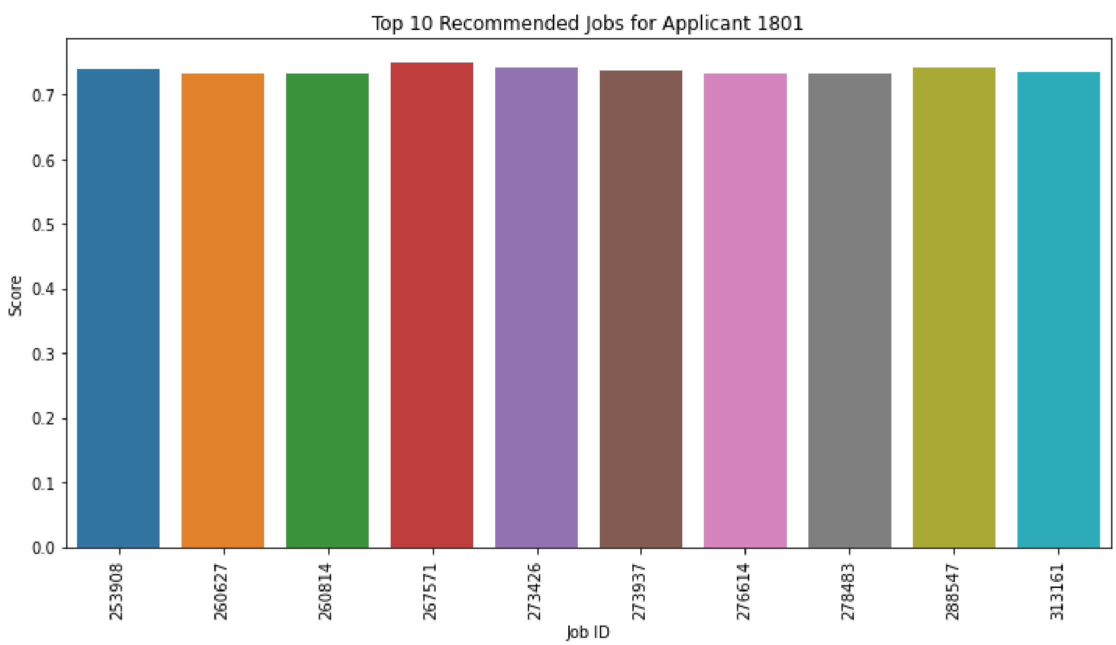
[26]:
```
# Applicant Recommendations
plt.figure(figsize=(10, 6))
sns.histplot(job_recommended['Score'], bins=20, kde=True)
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.title(f'Distribution of Recommendation Scores for Applicant {applicant_id}')
plt.show()
```



[27]:
```
# Top K Recommendations
top_k = 10
plt.figure(figsize=(12, 6))
sns.barplot(x=applicant_recommended['Recommended_Applicant_ID'],␣
↪y=applicant_recommended['Score'])
plt.xlabel('Applicant ID')
plt.ylabel('Score')
plt.title(f'Top {top_k} Recommended Applicants for Job {job_id}')
plt.xticks(rotation=90)
plt.show()
```



[28]:
```
# Top K Recommendations for Jobs
plt.figure(figsize=(12, 6))
sns.barplot(x=job_recommended['Recommended_Job_ID'], y=job_recommended['Score'])
plt.xlabel('Job ID')
plt.ylabel('Score')
plt.title(f'Top {top_k} Recommended Jobs for Applicant {applicant_id}')
plt.xticks(rotation=90)
plt.show()
```



1.7.2 Collaborative Filtering

1.7.3 KNNBasic

[31]:
```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder  # Import LabelEncoder
from surprise import Dataset, Reader, KNNBasic
from surprise.model_selection import GridSearchCV, train_test_split

df_views = pd.DataFrame({
     'Applicant.ID': [10000, 10000, 10000, 10001, 10001, 10002, 10002, 10002],
    'Job.ID': [4842, 4407, 4444, 4908, 3382, 4598, 3360, 3326],
    'Position': ['Accounting', 'Accounting', 'Accounting', 'Child care', 'Child␣
↪care', 'Human resources', 'Human resources', 'Human resources']
})


label_encoder = LabelEncoder()
df_views['Position_Encoded'] = label_encoder.fit_transform(df_views['Position'])


reader = Reader(rating_scale=(0, 1))
data = Dataset.load_from_df(df_views[['Applicant.ID', 'Job.ID',␣
↪'Position_Encoded']], reader)

trainset, testset = train_test_split(data, test_size=0.2)

algo_knn_default = KNNBasic(sim_options={'name': 'cosine', 'user_based': True})

algo_knn_default.fit(trainset)
predictions_default = algo_knn_default.test(testset)


def precision_at_k(recommended_ids, relevant_ids, k):
    recommended_at_k = recommended_ids[:k]
    relevant_set = set(relevant_ids)
    recommended_set = set(recommended_at_k)
    true_positives = len(recommended_set & relevant_set)
    return true_positives / k

def recall_at_k(recommended_ids, relevant_ids, k):
    recommended_at_k = recommended_ids[:k]
    relevant_set = set(relevant_ids)
    recommended_set = set(recommended_at_k)
```

```python
    true_positives = len(recommended_set & relevant_set)
    return true_positives / len(relevant_ids)

def mean_reciprocal_rank(recommended_ids, relevant_ids):
    relevant_set = set(relevant_ids)
    for rank, rec_id in enumerate(recommended_ids, start=1):
        if rec_id in relevant_set:
            return 1 / rank
    return 0

def ndcg_at_k(recommended_ids, relevant_ids, k):
    def dcg(recs, rels, k):
        dcg_score = 0.0
        for i, rec in enumerate(recs[:k]):
            if rec in rels:
                dcg_score += 1 / np.log2(i + 2)
        return dcg_score

    ideal_dcg = dcg(relevant_ids, relevant_ids, k)
    actual_dcg = dcg(recommended_ids, relevant_ids, k)
    return actual_dcg / ideal_dcg if ideal_dcg > 0 else 0

def get_top_n_recommendations(algo, applicant_id, n=5):
    job_ids = df_views['Job.ID'].unique()
    predictions = [algo.predict(str(applicant_id), str(job_id)) for job_id in job_ids]
    predictions.sort(key=lambda x: x.est, reverse=True)
    return [int(pred.iid) for pred in predictions[:n]]

applicant_id = 10000
ground_truth = df_views[df_views['Applicant.ID'] == applicant_id]['Job.ID'].tolist()

recommendations_knn_default = get_top_n_recommendations(algo_knn_default, applicant_id, n=5)
precision_knn_default = precision_at_k(recommendations_knn_default, ground_truth, k=5)
recall_knn_default = recall_at_k(recommendations_knn_default, ground_truth, k=5)
mrr_knn_default = mean_reciprocal_rank(recommendations_knn_default, ground_truth)
ndcg_knn_default = ndcg_at_k(recommendations_knn_default, ground_truth, k=5)

print("KNN (Default) Metrics:")
print(f"Precision@5: {precision_knn_default}")
print(f"Recall@5: {recall_knn_default}")
print(f"MRR: {mrr_knn_default}")
print(f"NDCG@5: {ndcg_knn_default}")
```

```python
try:
    param_grid_knn = {
        'k': [10, 20, 30],
        'sim_options': {
            'name': ['cosine'],
            'user_based': [True]
        }
    }
    gs_knn = GridSearchCV(KNNBasic, param_grid_knn, measures=['rmse'], cv=3)
    gs_knn.fit(data)


    best_algo_knn = gs_knn.best_estimator['rmse']
    best_algo_knn.fit(trainset)


    recommendations_knn_tuned = get_top_n_recommendations(best_algo_knn,␣
 ↪applicant_id, n=5)
    precision_knn_tuned = precision_at_k(recommendations_knn_tuned,␣
 ↪ground_truth, k=5)
    recall_knn_tuned = recall_at_k(recommendations_knn_tuned, ground_truth, k=5)
    mrr_knn_tuned = mean_reciprocal_rank(recommendations_knn_tuned,␣
 ↪ground_truth)
    ndcg_knn_tuned = ndcg_at_k(recommendations_knn_tuned, ground_truth, k=5)


    print("\nKNN (Tuned) Metrics:")
    print(f"Precision@5: {precision_knn_tuned}")
    print(f"Recall@5: {recall_knn_tuned}")
    print(f"MRR: {mrr_knn_tuned}")
    print(f"NDCG@5: {ndcg_knn_tuned}")
except ZeroDivisionError:
    print("Error during KNN hyperparameter tuning. Consider checking data␣
 ↪sparsity or similarity options.")
except AttributeError:
    print("Error accessing the best estimator from GridSearchCV.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Computing the cosine similarity matrix…
Done computing similarity matrix.
KNN (Default) Metrics:
Precision@5: 0.6
Recall@5: 1.0
MRR: 1.0
NDCG@5: 1.0
```

```
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.

KNN (Tuned) Metrics:
Precision@5: 0.6
Recall@5: 1.0
MRR: 1.0
NDCG@5: 1.0
```

1.7.4 SVD

[33]:
```python
import numpy as np
import pandas as pd
from surprise import Dataset, Reader, SVD
from surprise.model_selection import cross_validate, GridSearchCV
from sklearn.preprocessing import LabelEncoder


def precision_at_k(recommended_ids, relevant_ids, k):
    recommended_at_k = recommended_ids[:k]
    relevant_set = set(relevant_ids)
    recommended_set = set(recommended_at_k)
    true_positives = len(recommended_set & relevant_set)
    return true_positives / k

def recall_at_k(recommended_ids, relevant_ids, k):
    recommended_at_k = recommended_ids[:k]
    relevant_set = set(relevant_ids)
    recommended_set = set(recommended_at_k)
```

```python
    true_positives = len(recommended_set & relevant_set)
    return true_positives / len(relevant_ids)

def mean_reciprocal_rank(recommended_ids, relevant_ids):
    relevant_set = set(relevant_ids)
    for rank, rec_id in enumerate(recommended_ids, start=1):
        if rec_id in relevant_set:
            return 1 / rank
    return 0

def ndcg_at_k(recommended_ids, relevant_ids, k):
    def dcg(recs, rels, k):
        dcg_score = 0.0
        for i, rec in enumerate(recs[:k]):
            if rec in rels:
                dcg_score += 1 / np.log2(i + 2)
        return dcg_score

    ideal_dcg = dcg(relevant_ids, relevant_ids, k)
    actual_dcg = dcg(recommended_ids, relevant_ids, k)
    return actual_dcg / ideal_dcg if ideal_dcg > 0 else 0


df_views = pd.DataFrame({
     'Applicant.ID': [10000, 10000, 10000, 10001, 10001, 10002, 10002, 10002],
    'Job.ID': [4842, 4407, 4444, 4908, 3382, 4598, 3360, 3326],
    'Position': ['Accounting', 'Accounting', 'Accounting', 'Child care', 'Child care', 'Human resources', 'Human resources', 'Human resources']
})


label_encoder = LabelEncoder()
df_views['Position_Encoded'] = label_encoder.fit_transform(df_views['Position'])


reader = Reader(rating_scale=(0, 1))  # Fixed the syntax issue here
data = Dataset.load_from_df(df_views[['Applicant.ID', 'Job.ID', 'Position_Encoded']], reader)


algo = SVD()
cross_val_results = cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)


trainset = data.build_full_trainset()
algo.fit(trainset)
```

```
def get_top_n_recommendations(algo, applicant_id, n=5):
    job_ids = df_views['Job.ID'].unique()
    predictions = [algo.predict(str(applicant_id), str(job_id)) for job_id in␣
 ↪job_ids]
    predictions.sort(key=lambda x: x.est, reverse=True)
    return [int(pred.iid) for pred in predictions[:n]]


param_grid = {
    'n_factors': [50, 100, 150],
    'n_epochs': [20, 30, 40],
    'lr_all': [0.002, 0.005, 0.01],
    'reg_all': [0.02, 0.05, 0.1]
}
gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)
gs.fit(data)


best_algo = gs.best_estimator['rmse']
best_algo.fit(trainset)


applicant_id = 10000
ground_truth = df_views[df_views['Applicant.ID'] == applicant_id]['Job.ID'].
 ↪tolist()


recommendations_svd = get_top_n_recommendations(algo, applicant_id, n=5)
precision_svd = precision_at_k(recommendations_svd, ground_truth, k=5)
recall_svd = recall_at_k(recommendations_svd, ground_truth, k=5)
mrr_svd = mean_reciprocal_rank(recommendations_svd, ground_truth)
ndcg_svd = ndcg_at_k(recommendations_svd, ground_truth, k=5)

# SVD (Tuned)
recommendations_svd_tuned = get_top_n_recommendations(best_algo, applicant_id,␣
 ↪n=5)
precision_svd_tuned = precision_at_k(recommendations_svd_tuned, ground_truth,␣
 ↪k=5)
recall_svd_tuned = recall_at_k(recommendations_svd_tuned, ground_truth, k=5)
mrr_svd_tuned = mean_reciprocal_rank(recommendations_svd_tuned, ground_truth)
ndcg_svd_tuned = ndcg_at_k(recommendations_svd_tuned, ground_truth, k=5)


print("SVD (Default) Metrics:")
print(f"Precision@5: {precision_svd}")
```

```
print(f"Recall@5: {recall_svd}")
print(f"MRR: {mrr_svd}")
print(f"NDCG@5: {ndcg_svd}")

print("\nSVD (Tuned) Metrics:")
print(f"Precision@5: {precision_svd_tuned}")
print(f"Recall@5: {recall_svd_tuned}")
print(f"MRR: {mrr_svd_tuned}")
print(f"NDCG@5: {ndcg_svd_tuned}")
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9212  0.9248  0.6983  0.0000  1.0000  0.7088  0.3685
MAE (testset)     0.9174  0.9214  0.4938  0.0000  1.0000  0.6665  0.3777
Fit time          0.01    0.00    0.00    0.00    0.00    0.00    0.00
Test time         0.00    0.00    0.00    0.00    0.00    0.00    0.00
SVD (Default) Metrics:
Precision@5: 0.6
Recall@5: 1.0
MRR: 1.0
NDCG@5: 1.0

SVD (Tuned) Metrics:
Precision@5: 0.6
Recall@5: 1.0
MRR: 1.0
NDCG@5: 1.0
```

1.7.5 RECOMMENDATIONS:

Job Seekers

- Utilize Advanced Job Recommendation Systems:

Leverage platforms that use advanced job recommendation algorithms to receive personalized job listings. Regularly update your profile with accurate and detailed information to improve the relevance of job recommendations. * Provide Feedback:

Actively provide feedback on the recommendations you receive to help improve the accuracy and relevance of the system. Rate and review job matches to ensure the system learns your preferences over time. * Keep Skills and Qualifications Updated:

Regularly update your resume and profile with new skills, certifications, and job experiences to ensure you are matched with the most relevant positions. Highlight key skills and experiences that align with your career goals.

Recruiters

- Adopt Advanced Recruitment Tools:

Use advanced job recommendation systems and recruitment platforms that leverage AI and machine learning to identify suitable candidates quickly. Integrate these tools into your existing recruitment processes to enhance efficiency. * Provide Detailed Job Descriptions:

Ensure that job postings are detailed and accurately describe the role, responsibilities, and required qualifications to improve the quality of candidate matches. Highlight key skills, experiences, and attributes that are critical for the role. * Regularly Update Job Listings:

Keep job postings up-to-date with any changes in job requirements or company policies to ensure accurate matches. Remove outdated job listings to prevent confusion and maintain the relevance of recommendations.

1.7.6 Implications Of The Study

1. For Job Seekers

- Enhanced User Experience: The study aims to simplify and streamline the job search process, reducing the time and effort required to find relevant job postings.
- Personalized Recommendations: By tailoring job recommendations to individual qualifications and preferences, job seekers receive more relevant job listings, increasing their chances of finding suitable positions.
- Increased Job Match Accuracy: Improved algorithms ensure that job seekers are matched with positions that better align with their skills and career aspirations, leading to higher job satisfaction and retention rates.

2. For Recruiters and Hiring Managers

- Efficient Recruitment Process: The system helps recruiters quickly identify suitable candidates, reducing the time and resources spent on screening and shortlisting applicants.
- Access to a Wider Talent Pool: Improved matching algorithms can uncover potential candidates who might not have been considered through traditional recruitment methods.
- Better Quality of Hires: By ensuring a closer match between job requirements and candidate skills, the system can lead to higher quality hires, which can positively impact organizational performance.

3. For Organizations and the Labor Market

- Reduced Turnover Rates: Better job matches lead to increased employee satisfaction and lower turnover rates, saving organizations the costs associated with frequent hiring and training.
- Increased Productivity: Employees who are well-matched to their roles are likely to be more productive, contributing to the overall efficiency and effectiveness of organizations.
- Support for Career Development: The system can aid in career advancement by identifying opportunities that align with an individual's career path and growth potential.

4. Technological and Research Advancements

- Advancement in AI and Machine Learning: The study contributes to the body of research in natural language processing (NLP) and machine learning, particularly in the context of job recommendation systems.
- Benchmarking and Best Practices: The results and methodologies from the study can serve as benchmarks and best practices for future research and development in job recommendation systems.

1.7.7 Limitations

1. User Behavior and Preferences Dynamic Preferences: Job seekers' preferences and qualifications can change over time, making it challenging for the system to provide continuously relevant recommendations. User Feedback Loop: Incorporating real-time user feedback into the system to improve recommendations dynamically can be complex.
2. Technical and Implementation Challenges Integration with Job Platforms: Integrating the recommendation system with various job platforms and databases can be technically challenging and require significant resources. Real-Time Processing: Ensuring that the recommendation system operates in real-time or near-real-time to provide timely suggestions is a technical challenge.
3. Generalizability Industry Specificity: The system may perform well in certain industries but not in others. Generalizing the findings across all industries might be difficult without extensive testing and adaptation. Geographical Limitations: Job markets vary significantly across different regions, and the system may need localization to be effective in diverse geographical areas.
4. Ethical Considerations Transparency: Ensuring the transparency of the recommendation algorithms is important but challenging, particularly in explaining how recommendations are generated to end-users. Ethical Implications: There may be ethical concerns regarding the influence of algorithmic recommendations on career choices and the potential for manipulation or unintended consequences.

1.7.8 Conclusion

The development of a sophisticated job recommendation system has the potential to revolutionize the job search and recruitment processes, offering significant benefits to job seekers, recruiters, and the broader labor market. By leveraging advanced algorithms and techniques such as natural language processing (NLP) and machine learning, the system aims to provide personalized, efficient, and accurate job recommendations that align with individual qualifications and preferences.

Key Findings:

- Enhanced User Experience: The system simplifies the job search process, making it less time-consuming and more efficient for job seekers.
- Improved Job Matching: By tailoring recommendations, the system increases the likelihood of finding suitable job positions, leading to higher job satisfaction and retention rates.
- Efficient Recruitment: Recruiters benefit from a streamlined hiring process, allowing them to quickly identify and engage with qualified candidates.
- Positive Societal Impact: The system contributes to reduced unemployment and underemployment, supporting economic growth and social mobility.