Job Recommendation System

Summary

Business and Data Understanding

The objective is to provide job recommendations based on a given job title, thus addressing the need for users to discover related opportunities.

The dataset used in this project is the 'Combined Jobs Final' dataset obtained from Kaggle-[https://www.kaggle.com/datasets/kandij/job-recommendation-datasets](https://www.kaggle.com/datasets/kandij/job-recommendation-datasets%5D) , which contains various job listings with information such as job title, company, location, and job description.

The data is well-suited for the problem as it includes descriptive job details that can be leveraged to calculate similarities.

Data Preparation

The data was preprocessed by removing missing values from critical columns such as 'Title' and 'Job Description'.

The TF-IDF Vectorizer from Scikit-Learn was used to convert the textual data into numerical form for modeling purposes.

Modeling

The modeling approach uses a content-based recommendation system built using the TF-IDF Vectorizer and Knearest Neighbours with a `cosine_similariry metric` .

The `cosine_similarity` function was used to find related job listings.

This technique is appropriate as it measures text similarity effectively for recommendation.

Evaluation

The recommendation system was qualitatively evaluated by testing different input job titles.

Given the unsupervised nature of this content-based recommender, metrics such as accuracy are less applicable in this case.

Instead, similarity-based relevance and user feedback were used for validation.

In [138]:
```
#Importing  libraries

import pandas as pd
import numpy as np
import  matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import re
import string
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import precision_score, recall_score, pairwise_distan
from sklearn.model_selection import GridSearchCV
import pickle
from flask import Flask, request, jsonify
```

The Data

The Combined Jobs Final dataset contains 84,090 rows and 23 columns.

The columns are:

1. Job.ID: Unique identifier for each job listing.

2. Provider: Platform or source of the job listing.

3. Status: Current state of the job (e.g., open, closed).

4. Slug: URL-friendly string for the job.

5. Title: Job title or role and the @ refers to the company or location to where the job is at

6. Position: Job position type.

7. Company: Name of the hiring company.

8. City: City where the job is located.

9. State.Name: Name of the state where the job is located.

10. State.Code: Abbreviation or code for the state.

11. Address: Detailed address of the job location.

12. Latitude: Latitude coordinate of the job location.

13. Longitude: Longitude coordinate of the job location.

14. Industry: Industry related to the job.

15. Job.Description: Detailed description of the job role.

16. **Requirements:** Qualifications and skills required for the job.

17. **Salary:** Salary offered for the position.

18. **Listing.Start:** Date the job listing became active.

19. **Listing.End:** Date the job listing ends or expires.

20. **Employment.Type:** Type of employment (e.g., full-time, part-time).

21. **Education.Required:** Educational qualifications required.

22. **Created.At:** Timestamp when the listing was created.

23. **Updated.At:** Timestamp of the last update made to the listing.

In [99]:
```
#Loading the data
combined_jobs = pd.read_csv('Combined_Jobs_Final.csv')
combined_jobs.head() #preview the data
```

Out[99]:

	Job.ID	Provider	Status	Slug	Title	Position	Company	City	State
0	111	1	open	palo-alto-ca-tacolicious-server	Server @ Tacolicious	Server	Tacolicious	Palo Alto	Ca
1	113	1	open	san-francisco-ca-claude-lane-kitchen-staff-chef	Kitchen Staff/Chef @ Claude Lane	Kitchen Staff/Chef	Claude Lane	San Francisco	Ca
2	117	1	open	san-francisco-ca-machka-restaurants-corp-barte...	Bartender @ Machka Restaurants Corp.	Bartender	Machka Restaurants Corp.	San Francisco	Ca
3	121	1	open	brisbane-ca-teriyaki-house-server	Server @ Teriyaki House	Server	Teriyaki House	Brisbane	Ca
4	127	1	open	los-angeles-ca-rosa-mexicano-sunset-kitchen-st...	Kitchen Staff/Chef @ Rosa Mexicano - Sunset	Kitchen Staff/Chef	Rosa Mexicano - Sunset	Los Angeles	Ca

5 rows × 23 columns

In [100]:
```
#preview column for better understanding
combined_jobs['Slug'].head()
```

Out[100]:
```
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...
Name: Slug, dtype: object
```

Slug is like a link to the job location

In [101]:
```
#preview column for better understanding
combined_jobs['Title'].head()
```

Out[101]:
```
0                           Server @ Tacolicious
1                Kitchen Staff/Chef @ Claude Lane
2            Bartender @ Machka Restaurants Corp.
3                        Server @ Teriyaki House
4     Kitchen Staff/Chef @ Rosa Mexicano - Sunset
Name: Title, dtype: object
```

The @ after the title shows the job location

In [102]:
```
pd.reset_option('display.max_colwidth')#resetting to max column width

combined_jobs['Job.Description'].head(1) # display the first job descripti
```

Out[102]:
```
0    Tacolicious' first Palo Alto store just opened...
Name: Job.Description, dtype: object
```

Job decription is what the recruit will entirely like the first example in our data is:

Tacolicious' first Palo Alto store just opened recently, and we are hiring! If you love tacos, you will love working at our restaurant!

- *Serve food/drinks to customers in a professional manner*
- *Act as a cashier when needed*
- *Clean up the dining space*
- *Train the new staff*

Name: Job.Description, dtype: object

In [103]:
```
combined_jobs.shape # checking data shape
```

Out[103]:
```
(84090, 23)
```

In [104]:
```
combined_jobs.columns # inspecting the columns
```

Out[104]:
```
Index(['Job.ID', 'Provider', 'Status', 'Slug', 'Title', 'Position', 'Comp
any',
       'City', 'State.Name', 'State.Code', 'Address', 'Latitude', 'Longit
ude',
       'Industry', 'Job.Description', 'Requirements', 'Salary',
       'Listing.Start', 'Listing.End', 'Employment.Type', 'Education.Requ
ired',
       'Created.At', 'Updated.At'],
      dtype='object')
```

In [105]:
```
combined_jobs.info() # checking data types
```

```
<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
```

In [106]:

```
combined_jobs.describe() # checking data statistics
```

Out[106]:

	Job.ID	Provider	Latitude	Longitude	Requirements	Salary
count	84090.000000	84090.000000	84090.000000	84090.000000	0.0	229.000000
mean	258490.774979	1.997063	37.967134	-92.151257	NaN	7.832227
std	52653.870401	0.056272	5.458651	17.412900	NaN	7.566016
min	3.000000	1.000000	-34.887672	-166.539760	NaN	0.000000
25%	250415.250000	2.000000	34.072600	-104.249780	NaN	0.000000
50%	271452.500000	2.000000	39.218300	-86.941440	NaN	8.000000
75%	293672.750000	2.000000	41.598965	-79.997460	NaN	10.550000
max	319174.000000	3.000000	71.294700	144.885800	NaN	58.000000

Data Cleaning

In [107]:

```
#checking the percentage of null values in every column
combined_jobs.isnull().sum().sort_values(ascending=False) / len(combined_j
```

Out[107]:

```
Requirements          100.000000
Address                99.957189
Salary                 99.727673
Industry               99.682483
Company                 2.700678
Listing.Start           0.812225
Education.Required      0.317517
State.Name              0.203354
State.Code              0.203354
Listing.End             0.198597
City                    0.160542
Job.Description         0.066595
Employment.Type         0.011892
Job.ID                  0.000000
Created.At              0.000000
Latitude                0.000000
Longitude               0.000000
Provider                0.000000
Position                0.000000
Title                   0.000000
Slug                    0.000000
Status                  0.000000
Updated.At              0.000000
dtype: float64
```

In [108]:

```
#Dropping columns with more than 50% missing values
columns_to_drop = ['Requirements', 'Address', 'Salary', 'Industry', 'Creat

combined_jobs = combined_jobs.drop(columns=columns_to_drop)


#Using Mode to replace missing values
for column in ['Listing.Start', 'Education.Required', 'State.Code', 'State
               'Listing.End', 'City', 'Employment.Type']:
    combined_jobs[column].fillna(combined_jobs[column].mode()[0], inplace=


#Dropping rows with missing values
combined_jobs = combined_jobs.dropna(subset=['Company', 'Job.Description']
```

```
C:\Users\FLEX 5\AppData\Local\Temp\ipykernel_18312\3296909428.py:10: Futu
reWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never wo
rk because the intermediate object on which we are setting values always
behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(valu
e) instead, to perform the operation inplace on the original object.


  combined_jobs[column].fillna(combined_jobs[column].mode()[0], inplace=T
rue)
```

In [109]:

```
combined_jobs.isnull().sum() # Reconfirming that there are no null values
```

Out[109]:

```
Job.ID                0
Provider              0
Status                0
Slug                  0
Title                 0
Position              0
Company               0
City                  0
State.Name            0
State.Code            0
Latitude              0
Longitude             0
Job.Description       0
Listing.Start         0
Listing.End           0
Employment.Type       0
Education.Required    0
dtype: int64
```

In [110]:
```
#checking duplicated values esp for JobID
combined_jobs.duplicated().sum()
```
Out[110]: 0

In [111]:
```
combined_jobs.head() # previewing the data
```
Out[111]:

	Job.ID	Provider	Status	Slug	Title	Position	Company	City	State
0	111	1	open	palo-alto-ca-tacolicious-server	Server @ Tacolicious	Server	Tacolicious	Palo Alto	Ca
1	113	1	open	san-francisco-ca-claude-lane-kitchen-staff-chef	Kitchen Staff/Chef @ Claude Lane	Kitchen Staff/Chef	Claude Lane	San Francisco	Ca
2	117	1	open	san-francisco-ca-machka-restaurants-corp-barte...	Bartender @ Machka Restaurants Corp.	Bartender	Machka Restaurants Corp.	San Francisco	Ca
3	121	1	open	brisbane-ca-teriyaki-house-server	Server @ Teriyaki House	Server	Teriyaki House	Brisbane	Ca
4	127	1	open	los-angeles-ca-rosa-mexicano-sunset-kitchen-st...	Kitchen Staff/Chef @ Rosa Mexicano - Sunset	Kitchen Staff/Chef	Rosa Mexicano - Sunset	Los Angeles	Ca

In [112]:
```
#Dropping columns with unimportant information
combined_jobs = combined_jobs.drop(columns=['Listing.End', 'Listing.Start'
                                            'State.Code', 'Status', 'Title
```

In [113]:
```
combined_jobs.shape # checking data shape of the cleaned dataset
```
Out[113]: (81766, 9)

In [114]:
```
combined_jobs['Employment.Type'].value_counts() # checking employment type
```

Out[114]:
```
Employment.Type
Part-Time              32160
Seasonal/Temp          27389
Full-Time/Part-Time    16759
Per Diem                4502
Intern                   904
Full-Time                 37
Contract                  14
Temporary/seasonal         1
Name: count, dtype: int64
```

In [115]:
```
combined_jobs.loc[:, "Employment.Type"] = combined_jobs["Employment.Type"]
```

In [116]:
```
combined_jobs['Employment.Type'].value_counts() # reconfirming changes
```

Out[116]:
```
Employment.Type
Part-Time              32160
Seasonal/Temp          27390
Full-Time/Part-Time    16759
Per Diem                4502
Intern                   904
Full-Time                 37
Contract                  14
Name: count, dtype: int64
```

In [117]:
```
combined_jobs['Education.Required'].value_counts() # checking education re
```

Out[117]:
```
Education.Required
Not Specified          60919
High School Diploma    13704
Associate Degree        3381
Bachelor's Degree       2753
Master's Degree         1009
Name: count, dtype: int64
```

In [118]:
```
combined_jobs['Provider'].value_counts() # checking provider counts
```

Out[118]:
```
Provider
2    81499
1      257
3       10
Name: count, dtype: int64
```

In [119]:

```
combined_jobs['Company'].value_counts() # checking company counts
```

Out[119]:

```
Company
Accountemps                                                   12471
OfficeTeam                                                    11423
BAYADA HOME HEALTH CARE                                        2194
Vector Marketing                                               1681
Macy's                                                         1625
                                                              ...
Carriage Court of Hilliard, A Good Neighbor Care Community        1
Heartis of Cleburne, a Good Neighbor Care managed community       1
Shawnee Gardens Healthcare and Rehabilitation Center, LLC         1
Lehigh University                                                 1
National Japanese American Historical Society                     1
Name: count, Length: 8334, dtype: int64
```

In [120]:

```
combined_jobs['Position'].value_counts() # checking position counts
```

Out[120]:

```
Position
Administrative Assistant                                    1392
Customer Service Representative                             1270
Accounts Payable Clerk                                       968
Accounting Clerk                                             950
Sales Representative / Sales Associate ( Entry Level )       917
                                                            ...
Immediate Need Full Charge Bookkeeper                          1
Accounts Receivable Clerk needed for a fortune 500 company!    1
SOX Auditor Clerk                                              1
Proactive Accounts Receivable Clerk Needed Immediatley!        1
Book Keeper                                                    1
Name: count, Length: 35325, dtype: int64
```

In [121]:
```
combined_jobs['State.Name'].value_counts() # checking state name counts
```

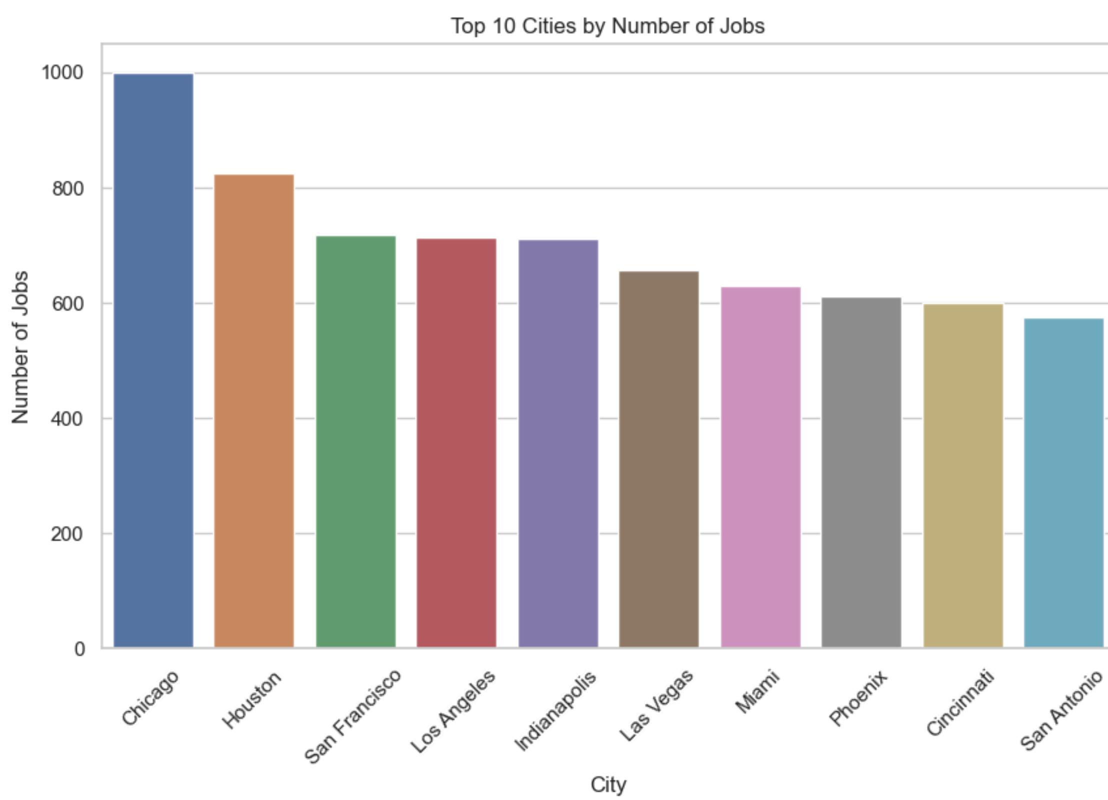
Out[121]:
```
State.Name
California              10761
Florida                  5465
Texas                    5313
Pennsylvania             4857
Illinois                 3960
Ohio                     3938
New York                 3122
New Jersey               2757
Minnesota                2414
Washington               2389
Indiana                  2351
Massachusetts            2195
Michigan                 2144
North Carolina           1937
Wisconsin                1859
Virginia                 1760
Arizona                  1700
Tennessee                1672
Maryland                 1651
Georgia                  1641
Colorado                 1633
Connecticut              1365
Missouri                 1344
Iowa                     1146
Oregon                   1088
Kentucky                 1081
Nevada                    987
South Carolina            893
Kansas                    798
Louisiana                 746
Utah                      674
Alabama                   583
New Hampshire             570
Nebraska                  515
Oklahoma                  493
Hawaii                    395
Delaware                  391
New Mexico                341
Arkansas                  313
Idaho                     290
District of Columbia      286
Rhode Island              285
Mississippi               258
Vermont                   243
Maine                     222
South Dakota              213
Montana                   210
West Virginia             167
Alaska                    152
North Dakota              126
Wyoming                    72
Name: count, dtype: int64
```

Exploratory Data Analysis

Univariate Analysis

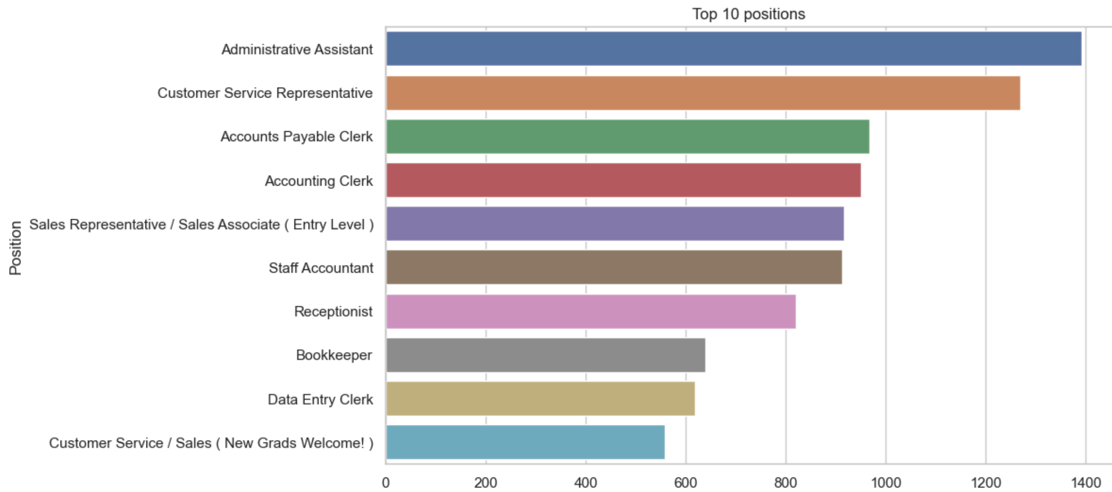
In [122]:

```
# Plotting top 10 cities with the highest number of job listings
plt.figure(figsize=(10, 6))

# Get the top 10 cities
top_10_cities = combined_jobs['City'].value_counts().nlargest(10).index

# Filter the dataset for top 10 cities
sns.countplot(data=combined_jobs[combined_jobs['City'].isin(top_10_cities)
              x='City',
              order=top_10_cities)

plt.title('Top 10 Cities by Number of Jobs')
plt.xlabel('City')
plt.ylabel('Number of Jobs')
plt.xticks(rotation=45)
plt.show()
```



Chicago and Houston have the most job listings

In [123]:
```
#top 10 positions  available
top_positions = combined_jobs['Position'].value_counts().head(10)

plt.figure(figsize=(10, 6))
sns.barplot(y=top_positions.index, x=top_positions.values)
plt.title('Top 10 positions ')
plt.show()
```



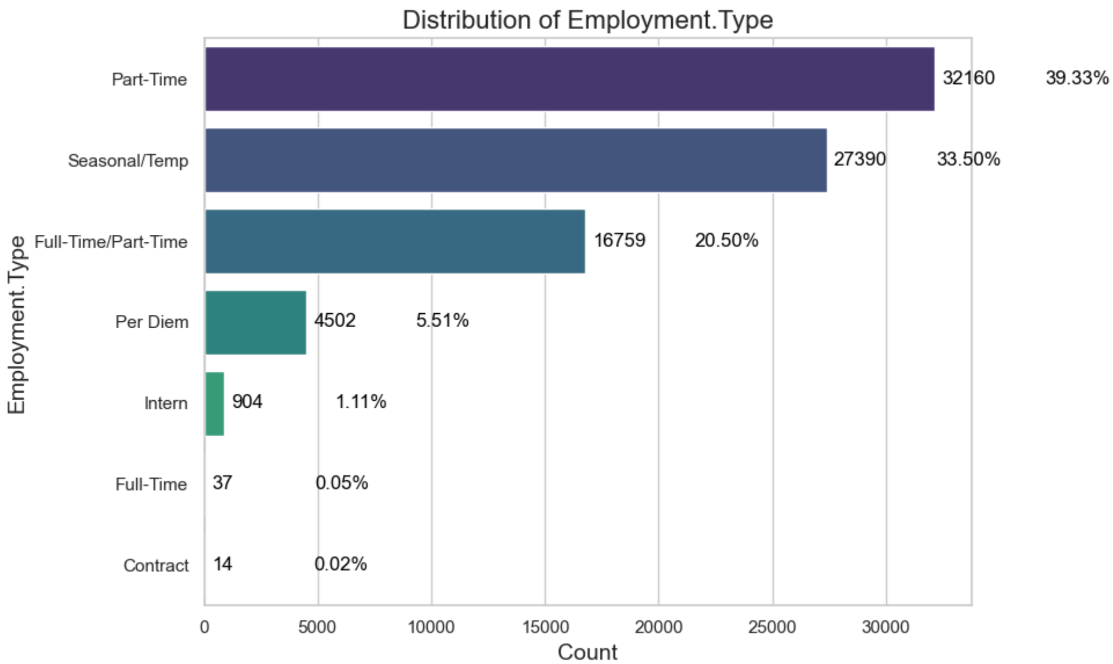
Most Jobs are Admin jobs

In [124]:

```
#Creating a function to further explore and visualize the categorical feat
def plot_categorical_distribution(df, column_name, top_n=10, show_counts=T

    # To Check if the column exists in the DataFrame
    if column_name not in df.columns:
        print(f"Column '{column_name}' does not exist in the DataFrame.")
        return

    # To Check if the column is categorical
    if not pd.api.types.is_object_dtype(df[column_name]) and not pd.api.ty
        print(f"Column '{column_name}' is not a categorical data type.")
        return

    # Dropping missing values
    data = df[column_name].dropna()

    # Calculating value counts
    counts = data.value_counts()

    # To Handle  high-cardinality by selecting top_n categories
    if len(counts) > top_n:
        top_categories = counts.nlargest(top_n)
        other_count = counts.sum() - top_categories.sum()
        counts = pd.concat([top_categories, pd.Series({'Other': other_coun

    # Calculating percentages
    percentages = (counts / counts.sum()) * 100


    plot_df = pd.DataFrame({
        'Category': counts.index,
        'Count': counts.values,
        'Percentage': percentages.values
    })  #Create a DataFrame for plotting


    sns.set(style="whitegrid")

    plt.figure(figsize=(10, 6))

    barplot = sns.barplot(x='Count', y='Category', data=plot_df, palette='

    # Adding count labels and percantages
    if show_counts:
        for index, row in plot_df.iterrows():
            barplot.text(row['Count'] + plot_df['Count'].max()*0.01, index
                        f"{row['Count']}", color='black', va="center")


    if show_percentages:
        for index, row in plot_df.iterrows():
            barplot.text(row['Count'] + plot_df['Count'].max()*0.15, index
                        f"{row['Percentage']:.2f}%", color='black', va="ce


    plt.title(f'Distribution of {column_name}', fontsize=16)
    plt.xlabel('Count', fontsize=14)
```

```
    plt.ylabel(column_name, fontsize=14)

    plt.tight_layout()
    plt.show();
```

In [125]:
```
plot_categorical_distribution(combined_jobs, 'Employment.Type', top_n=10)
```



Most Employment_type is Part time

In [126]:

```
#Renaming columns

cols = ["Job.ID", "Position", "Company", "City", "State.Name",
         "Education.Required", "Employment.Type", "Job.Description"]
combined_jobs = combined_jobs[cols].rename(columns={
    "Job.ID": "Job_ID",
    "State.Name": "State_Name",
    "Employment.Type": "Employment_Type",
    "Job.Description": "Job_Description",
    "Education.Required": "Education_Required"
})

combined_jobs.head()
```

Out[126]:

	Job_ID	Position	Company	City	State_Name	Education_Required	Employment_
0	111	Server	Tacolicious	Palo Alto	California	Not Specified	Part-
1	113	Kitchen Staff/Chef	Claude Lane	San Francisco	California	Not Specified	Part-
2	117	Bartender	Machka Restaurants Corp.	San Francisco	California	Not Specified	Part-
3	121	Server	Teriyaki House	Brisbane	California	Not Specified	Part-
4	127	Kitchen Staff/Chef	Rosa Mexicano - Sunset	Los Angeles	California	Not Specified	Part-

Text Preprocessing

In [127]:
```
#Creating a function
def preprocess_text(text):
    """
    Preprocesses the text data by removing punctuation, digits, and specia

    """
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Remove digits
    text = text.translate(str.maketrans('', '', string.digits))

    # Remove special characters
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    text = re.sub(r'\s+', ' ', text).strip()
    text = text.lower()

    return text
```

In [128]:
```
#Applying function to text data
combined_jobs['Job_Description'] = combined_jobs['Job_Description'].apply(
combined_jobs['Job_Description'].head()
```

Out[128]:
```
0    tacolicious first palo alto store just opened ...
1    new french brasserie in sf financial district ...
2    we are a popular mediterranean wine bar and re...
3    serve fooddrinks to customers in a professiona...
4    located at the heart of hollywood we are one o...
Name: Job_Description, dtype: object
```

In [129]:
```
combined_jobs['Position'] = combined_jobs['Position'].apply(preprocess_tex
```

In [130]:
```
combined_jobs['Position'].head()
```

Out[130]:
```
0              server
1    kitchen staffchef
2           bartender
3              server
4    kitchen staffchef
Name: Position, dtype: object
```

Text Vectorization

In [131]:

```
#Combining textual columns for TF-IDF
combined_jobs['combined_text'] = combined_jobs[['Position', 'Company', 'Ci

# Initializing TF-IDF Vectorizer
tfidf = TfidfVectorizer(max_features=100)  # Adjust max_features as needed

#Fitting and transforming the 'combined_text' column
tfidf_matrix = tfidf.fit_transform(combined_jobs['combined_text'])

#Converting to DataFrame for better viewing
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf.get_feature_

# Viewing the TF-IDF matrix
print(tfidf_df)
```

```
       ability  accounts       all        an       and       are        a
s  \
0          0.0  0.000000  0.000000  0.000000  0.102198  0.151824  0.14616
8
1          0.0  0.000000  0.000000  0.263989  0.378154  0.374518  0.12018
9
2          0.0  0.000000  0.000000  0.164878  0.314909  0.311881  0.15013
2
3          0.0  0.000000  0.000000  0.000000  0.000000  0.000000  0.18453
9
4          0.0  0.000000  0.000000  0.000000  0.000000  0.424470  0.00000
0
...        ...       ...       ...       ...       ...       ...
...
81761      0.0  0.555725  0.169615  0.000000  0.000000  0.271330  0.13061
2
81762      0.0  0.000000  0.000000  0.000000  0.000000  0.000000  0.00000
0
81763      0.0  0.000000  0.000000  0.000000  0.581643  0.078552  0.07562
6
81764      0.0  0.000000  0.211415  0.178790  0.113827  0.338197  0.16280
0
81765      0.0  0.000000  0.000000  0.000000  0.253469  0.376548  0.00000
0

       assigned  assistant        at  ...        up        we       who
\
0           0.0        0.0  0.180403  ...  0.269479  0.190371  0.000000
1           0.0        0.0  0.000000  ...  0.000000  0.156535  0.100976
2           0.0        0.0  0.000000  ...  0.000000  0.586599  0.000000
3           0.0        0.0  0.000000  ...  0.340220  0.000000  0.000000
4           0.0        0.0  0.168124  ...  0.251138  0.354827  0.228888
...         ...        ...       ...  ...       ...       ...       ...
81761       0.0        0.0  0.161202  ...  0.000000  0.340219  0.219464
81762       0.0        0.0  0.000000  ...  0.000000  0.000000  0.000000
81763       0.0        0.0  0.000000  ...  0.000000  0.196993  0.000000
81764       0.0        0.0  0.000000  ...  0.000000  0.424063  0.000000
81765       0.0        0.0  0.000000  ...  0.000000  0.472151  0.000000

           will      with  within      work   working       you      your
0      0.154078  0.000000     0.0  0.000000  0.246901  0.345938  0.000000
1      0.063347  0.201280     0.0  0.070400  0.101509  0.000000  0.000000
2      0.000000  0.000000     0.0  0.175879  0.000000  0.000000  0.000000
3      0.000000  0.000000     0.0  0.000000  0.000000  0.000000  0.000000
4      0.000000  0.000000     0.0  0.000000  0.000000  0.161196  0.000000
...         ...       ...     ...       ...       ...       ...       ...
81761  0.000000  0.000000     0.0  0.153011  0.000000  0.000000  0.000000
81762  0.000000  0.236686     0.0  0.331137  0.000000  0.000000  0.000000
81763  0.000000  0.189976     0.0  0.000000  0.000000  0.268478  0.100698
81764  0.000000  0.000000     0.0  0.381438  0.000000  0.000000  0.000000
81765  0.000000  0.000000     0.0  0.000000  0.000000  0.000000  0.000000

[81766 rows x 100 columns]
```

Modelling

In [132]:

```
#Using Nearest Neighbors for Finding Similar Jobs
n_neighbors = 11  #similar 10 jobs plus job itself
nn = NearestNeighbors(n_neighbors=n_neighbors, metric='cosine')
nn.fit(tfidf_df)

# Trying out similar jobs for a given job index
job_index = 0
distances, indices = nn.kneighbors([tfidf_df.iloc[job_index]])
recommended_job_indices = indices[0][1:]

for i, rec_index in enumerate(recommended_job_indices, 1):
    job = combined_jobs.iloc[rec_index]
    print(f"Recommendation {i}:")
    print(f"Company: {job.get('Company', 'N/A')}")
    print(f"Location: {job.get('City', 'N/A')}, {job.get('State_Name', 'N/
    print(f"Employment Type: {job.get('Employment_Type', 'N/A')}")
    print("-" * 40)

#Creating a function to get job recommendations
def get_recommendations(job_index, tfidf_df, df, top_n=10):
    """
    Given a job_index, returns the top_n most similar jobs using cosine si

    Parameters:
    - job_index: Index of the job in the DataFrame to base recommendations
    - tfidf_df: DataFrame containing TF-IDF features for jobs
    - df: Original DataFrame with job details
    - top_n: Number of recommendations to return (default is 10)

    Returns:
    - top_n similar job listings
    """
    # Fitting  Nearest Neighbors on TF-IDF matrix again for recommendation
    nn = NearestNeighbors(n_neighbors=top_n + 1, metric='cosine')
    nn.fit(tfidf_df)

    # Getting similarity scores for the given job
    distances, indices = nn.kneighbors([tfidf_df.iloc[job_index]])
    recommended_indices = indices[0][1:]

    return df.iloc[recommended_indices][['Position', 'Company', 'City', 'S
```

```
Recommendation 1:
Company: Sushi Kai
Location: Milpitas, California
Employment Type: Part-Time
----------------------------------------
Recommendation 2:
Company: Ajisen Ramen
Location: San Mateo, California
Employment Type: Part-Time
----------------------------------------
Recommendation 3:
Company: Ajito Izakaya
Location: Cupertino, California
Employment Type: Part-Time
----------------------------------------
Recommendation 4:
Company: Luce
Location: San Francisco, California
Employment Type: Part-Time
----------------------------------------
Recommendation 5:
Company: La Mar CebicherÃa Peruana
Location: San Francisco, California
Employment Type: Part-Time
----------------------------------------
Recommendation 6:
Company: Sakae Sushi
Location: Burlingame, California
Employment Type: Part-Time
----------------------------------------
Recommendation 7:
Company: Luna Park - LA
Location: Los Angeles, California
Employment Type: Part-Time
----------------------------------------
Recommendation 8:
Company: Sesame Korean Cuisine
Location: Burlingame, California
Employment Type: Part-Time
----------------------------------------
Recommendation 9:
Company: Piperade
Location: San Francisco, California
Employment Type: Part-Time
----------------------------------------
Recommendation 10:
Company: Akane Japanese Restaurant
Location: Los Altos, California
Employment Type: Part-Time
----------------------------------------

c:\Users\FLEX 5\anaconda03\envs\learn-env\Lib\site-packages\sklearn\base.
py:493: UserWarning: X does not have valid feature names, but NearestNeig
hbors was fitted with feature names
  warnings.warn(
```

In [133]:
```
# Trying out the function for confirmation
recommended_jobs = get_recommendations(job_index, tfidf_df, combined_jobs,

print("Top recommended jobs:")
print(recommended_jobs)
```

```
c:\Users\FLEX 5\anaconda03\envs\learn-env\Lib\site-packages\sklearn\base.
py:493: UserWarning: X does not have valid feature names, but NearestNeig
hbors was fitted with feature names
  warnings.warn(

Top recommended jobs:
      Position                   Company           City  State_Name  \
84041   server                 Sushi Kai       Milpitas  California
84073   server              Ajisen Ramen      San Mateo  California
84016   server             Ajito Izakaya      Cupertino  California
74233   server                      Luce  San Francisco  California
69233   server  La Mar CebicherÃa Peruana  San Francisco  California
54458   server               Sakae Sushi     Burlingame  California
62553   server            Luna Park - LA    Los Angeles  California
84018   server     Sesame Korean Cuisine     Burlingame  California
75234   server                  Piperade  San Francisco  California
84003   server   Akane Japanese Restaurant      Los Altos  California

      Employment_Type                                    Job_Description
84041       Part-Time  we are located in milpitas if you are energeti...
84073       Part-Time  ajisen ramen located right inside downtown san...
84016       Part-Time  we are opening very soon hiring motivated wait...
74233       Part-Time  we are one of the most popular american restau...
69233       Part-Time  we are located on the pier we are one of the m...
54458       Part-Time  located at the heart of burlingame sakae sushi...
62553       Part-Time  luna park is proud to be one of los angeles tr...
84018       Part-Time  located at the heart of burlingame sesame kore...
75234       Part-Time  piperade is a spanish restaurant located in no...
84003       Part-Time  having just celebrated our th anniversary akan...
```

In [136]:

```
# Evaluating  the model using a sample job index
def evaluate_model(job_index, model, tfidf_df, df, top_n=10):


    recommended_jobs = get_recommendations(job_index, tfidf_df, combined_j
    actual_position = df.iloc[job_index]['Position']
    recommended_positions = recommended_jobs['Position'].values

    # Calculating precision and recall
    true_positives = sum([1 for pos in recommended_positions if pos == act
    precision = true_positives / top_n
    recall = true_positives / sum(df['Position'] == actual_position)

    return precision, recall

# Example evaluation
precision, recall = evaluate_model(job_index, nn, tfidf_df, combined_jobs,
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

```
Precision: 1.0
Recall: 0.0847457627118644

c:\Users\FLEX 5\anaconda03\envs\learn-env\Lib\site-packages\sklearn\base.
py:493: UserWarning: X does not have valid feature names, but NearestNeig
hbors was fitted with feature names
  warnings.warn(
```

Interpretation

Precision: 1.0

Precision represents the ratio of correctly recommended items (true positives) to the total number of items recommended.

In this case, a precision of 1.0 means that all of the jobs recommended by the model were relevant, meaning they had the same job position as the original job.

This is ideal and shows that the recommendations are highly accurate regarding relevance.

Recall: 0.0847

Recall represents the ratio of correctly recommended items (true positives) to the total number of relevant items available in the dataset.

In this case, a recall of approximately 0.0847 (or 8.47%) means that the model only found a small fraction of the total relevant jobs.

In other words, while the model's recommendations are accurate, it is missing many other relevant jobs.

Summary

Precision of 1.0 indicates that the recommendations are perfectly relevant.

Recall of 0.0847 suggests that there are many other jobs with the same position that the model did not recommend.

This implies that while the quality of the recommendations is good, the model's coverage (i.e., its ability to find all relevant jobs) is quite low.

Way Forward

Model Hypertuning

To improve recall, you could adjust the number of neighbors (n_neighbors) to include more results, which may capture more relevant jobs.

Alternatively, improving the diversity and comprehensiveness of the features used for similarity might also help improve recall

Model Hypertuning

In [147]:

```
# Performing a  Manual Hyperparameter Search
param_grid = {'n_neighbors': [5, 7, 9, 11, 13, 15], 'metric': ['cosine', '

best_params = None
best_score = float('inf')

for n_neighbors in param_grid['n_neighbors']:
    for metric in param_grid['metric']:
        nn = NearestNeighbors(n_neighbors=n_neighbors, metric=metric)
        nn.fit(tfidf_df)
        distances, _ = nn.kneighbors(tfidf_df)
        mean_distance = np.mean(distances[:, 1:])  # Exclude the distance

        #  best parameters based on minimum mean distance
        if mean_distance < best_score:
            best_score = mean_distance
            best_params = {'n_neighbors': n_neighbors, 'metric': metric}

# Updating the model with the best parameters
nn = NearestNeighbors(n_neighbors=best_params['n_neighbors'], metric=best_
nn.fit(tfidf_df)

print(f"Best Parameters from Manual Search: {best_params}")
```

```
Best Parameters from Manual Search: {'n_neighbors': 5, 'metric': 'cosin
e'}
```

In [149]:

```
#Using Nearest Neighbors for Finding Similar Jobs
n_neighbors =5
knn = NearestNeighbors(n_neighbors=n_neighbors, metric='cosine')
knn.fit(tfidf_df)


job_index = 0
distances, indices = knn.kneighbors([tfidf_df.iloc[job_index]])
recommended_job_indices = indices[0][1:]

for i, rec_index in enumerate(recommended_job_indices, 1):
    job = combined_jobs.iloc[rec_index]
    print(f"Recommendation {i}:")
    print(f"Company: {job.get('Company', 'N/A')}")
    print(f"Location: {job.get('City', 'N/A')}, {job.get('State_Name', 'N/
    print(f"Employment Type: {job.get('Employment_Type', 'N/A')}")
    print("-" * 40)

#Creating a function to get job recommendations
def get_recommendations(job_index, tfidf_df, df, top_n=4):

    # Fitting  Nearest Neighbors on TF-IDF matrix again for recommendation
    knn = NearestNeighbors(n_neighbors=top_n + 1, metric='cosine')
    knn.fit(tfidf_df)

    # Getting similarity scores for the given job
    distances, indices = knn.kneighbors([tfidf_df.iloc[job_index]])
    recommended_indices = indices[0][1:]

    return df.iloc[recommended_indices][['Position', 'Company', 'City', 'S

recommended_jobs = get_recommendations(job_index, tfidf_df, combined_jobs,

print("Top recommended jobs:")
print(recommended_jobs)
```

```
Recommendation 1:
Company: Sushi Kai
Location: Milpitas, California
Employment Type: Part-Time
----------------------------------------
Recommendation 2:
Company: Ajisen Ramen
Location: San Mateo, California
Employment Type: Part-Time
----------------------------------------
Recommendation 3:
Company: Ajito Izakaya
Location: Cupertino, California
Employment Type: Part-Time
----------------------------------------
Recommendation 4:
Company: Luce
Location: San Francisco, California
Employment Type: Part-Time
----------------------------------------
Top recommended jobs:
      Position       Company           City  State_Name Employment_Type  \
84041   server      Sushi Kai       Milpitas  California       Part-Time
84073   server   Ajisen Ramen      San Mateo  California       Part-Time
84016   server  Ajito Izakaya      Cupertino  California       Part-Time
74233   server           Luce  San Francisco  California       Part-Time

                                         Job_Description
84041  we are located in milpitas if you are energeti...
84073  ajisen ramen located right inside downtown san...
84016  we are opening very soon hiring motivated wait...
74233  we are one of the most popular american restau...

c:\Users\FLEX 5\anaconda03\envs\learn-env\Lib\site-packages\sklearn\base.
py:493: UserWarning: X does not have valid feature names, but NearestNeig
hbors was fitted with feature names
  warnings.warn(
c:\Users\FLEX 5\anaconda03\envs\learn-env\Lib\site-packages\sklearn\base.
py:493: UserWarning: X does not have valid feature names, but NearestNeig
hbors was fitted with feature names
  warnings.warn(
```

In [150]:

```
# Evaluating  the model using a sample job index
def evaluate_model(job_index, model, tfidf_df, df, top_n=4):


    recommended_jobs = get_recommendations(job_index, tfidf_df, combined_j
    actual_position = df.iloc[job_index]['Position']
    recommended_positions = recommended_jobs['Position'].values

    # Calculating precision and recall
    true_positives = sum([1 for pos in recommended_positions if pos == act
    precision = true_positives / top_n
    recall = true_positives / sum(df['Position'] == actual_position)

    return precision, recall

# Example evaluation
precision, recall = evaluate_model(job_index, knn, tfidf_df, combined_jobs
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

```
Precision: 0.4
Recall: 0.03389830508474576

c:\Users\FLEX 5\anaconda03\envs\learn-env\Lib\site-packages\sklearn\base.
py:493: UserWarning: X does not have valid feature names, but NearestNeig
hbors was fitted with feature names
  warnings.warn(
```

Interpretation:

Precision = 0.4:

A precision of 0.4 means that 40% of the jobs recommended were relevant (i.e., had the same job position as the original job).

This suggests that the model does a decent job at selecting relevant jobs, but there is still room for improvement to make the recommendations more focused.

Recall = 0.0339 (or about 3.39%):

Recall represents the ratio of correctly recommended jobs to the total number of relevant jobs in the entire dataset.

A recall of 0.0339 means that only about 3.39% of the total relevant jobs were recommended.

The recall is quite low, which means the model is not covering all possible relevant jobs well.

Conclusion

The original model performs better at recommending jobs