Employability Analytics: Enhancing Job Search for Cloud Data Professionals

Week 6 Deliverables

School for Professional Studies, Saint Louis University

IS-5960-04 Masters Research Project - 04 (Spring 2025)

February 21, 2025

Team 17:

- Krishna Chaitanya Reddy Kallam
- Sai Kandi
- Josh Rajesh Reddy Katakam
- Jasmithi Karri
- Nishanth Kannepogu
- Mahender Reddy Kamidi

Introduction

In increasingly dynamic job market, Cloud Data Professionals find themselves struggling to identify suitable job postings, salary rates, and desired skill information. In the same way, employers also have difficulty in recruiting qualified professionals and to best use their recruitment choices. The absence of real-time job market visibility results in inefficiencies in both job searching and hiring processes.

Problem statement

The purpose of this project is to maximize the effectiveness of job market information by leveraging data from job postings, job salaries, and job skills requirements to enable job seekers and employers to make data-informed decisions. Specifically, we aim to:

- Provide real-time information on salary, in-demand skills and hiring behaviour so as to increase job market transparency.
- Reduce skill mismatches by exposing what skills are lacking (skill gaps) and relating the abilities of job seekers to market needs.
- Optimize the hiring process for companies by providing them with information on job demand, industry growth, and salary benchmarks.

Using this structured job information, this platform will equip Cloud Data Professionals with useful and accurate job announcements, professional information and career advice, allowing them to participate in a smarter and more efficient job search process.

Relavant Data to be included

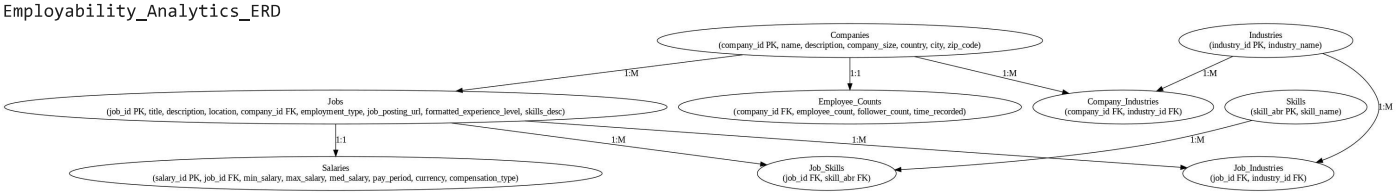
Show code

	Action Component	Module / Table	Relevant Data Fields	Description
0	Maximize job market transparency	Jobs	job_id, title, description, location, employment_type, job_posting_url	Stores job postings with key details
1	Provide salary benchmarking	Salaries	salary_id, job_id (FK), min_salary, max_salary, med_salary, pay_period, currency, compensation_type	Stores salary details for different job roles
2	Match job seekers with required skills	Skills	skill_abr, skill_name	Contains the skills required for job roles
3	Analyze job demand by industry	Industries, Job_Industries	industry_id, industry_name, job_id (FK)	Links job postings with industries for trend analysis
4	Improve hiring insights for companies	Companies	company_id, name, description, company_size, country, city	Stores company details and hiring trends
5	Monitor company workforce trends	Employee_Counts	company_id (FK), employee_count, follower_count, time_recorded	Tracks company size, employee growth, and LinkedIn followers

Next steps: Generate code with df | View recommended plots | New interactive sheet

Show code

Employability_Analytics_ERD



Data Integrity

```
# STEP 1: Loading datasets
jobs_df = pd.read_csv("/content/postings.csv", encoding="utf-8", quoting=3, sep=",", on_bad_lines="skip", low_memory=False)
companies_df = pd.read_csv("/content/companies.csv", on_bad_lines="skip", encoding="utf-8", low_memory=False)
salaries_df = pd.read_csv("/content/salaries.csv", on_bad_lines="skip", encoding="utf-8", low_memory=False)
skills_df = pd.read_csv("/content/skills.csv", on_bad_lines="skip", encoding="utf-8", low_memory=False)
job_skills_df = pd.read_csv("/content/job_skills.csv", on_bad_lines="skip", encoding="utf-8", low_memory=False)
industries_df = pd.read_csv("/content/industries.csv", on_bad_lines="skip", encoding="utf-8", low_memory=False)
job_industries_df = pd.read_csv("/content/job_industries.csv", on_bad_lines="skip", encoding="utf-8", low_memory=False)
company_industries_df = pd.read_csv("/content/company_industries.csv", on_bad_lines="skip", encoding="utf-8", low_memory=False)
employee_counts_df = pd.read_csv("/content/employee_counts.csv", on_bad_lines="skip", encoding="utf-8", low_memory=False)

print("✅ Datasets loaded successfully!")
```

```
✅ Datasets loaded successfully!
```

Referential integrity

```
# STEP 2: Checking Primary Key Uniqueness

def primary_key_uniqueness(df, primary_key, table_name):
    if df.duplicated(subset=[primary_key]).any():
        print(f"⚠️ WARNING: Duplicate values found in {table_name}.{primary_key}")
        print(df[df.duplicated(subset=[primary_key], keep=False)])
    else:
        print(f"✅ {primary_key} is unique in {table_name}")

# Assign dataset names for readability
datasets = {
    "Jobs": (jobs_df, "job_id"),
    "Companies": (companies_df, "company_id"),
    "Salaries": (salaries_df, "salary_id"),
    "Skills": (skills_df, "skill_abr"),
    "Industries": (industries_df, "industry_id"),
    "Job_Skills": (job_skills_df, "job_id"),
    "Job_Industries": (job_industries_df, "job_id"),
    "Company_Industries": (company_industries_df, "company_id"),
    "Employee_Counts": (employee_counts_df, "company_id"),
}

for table_name, (df, pk) in datasets.items():
    primary_key_uniqueness(df, pk, table_name)
```

```
⚠️ WARNING: Duplicate values found in Jobs.job_id

     Pay: $18-20/hour
     Expected hours: 35 – 45 per week
     Benefits:Paid time offSchedule:8 hour shiftMonday to FridayExperience:Marketing: 1 year (Preferred)Graphic design: 2 years (Preferred
     "
     1829192
     ...
     Established in 1997
     Mission: L.A. Care’s mission is to provide access to quality health care for Los Angeles County's vulnerable and low-income communiti
     Job Summary
     Duties
     Participate and contribute in multiple projects by completing and updating project documentation; managing project scope; adjusting s

     [16793 rows x 31 columns]
     ✅ company_id is unique in Companies
     ✅ salary_id is unique in Salaries
     ✅ skill_abr is unique in Skills
     ✅ industry_id is unique in Industries
     ⚠️ WARNING: Duplicate values found in Job_Skills.job_id
               job_id skill_abr
```

```
0       3884428798      MRKT
1       3884428798        PR
2       3884428798       WRT
4       3887465684       FIN
5       3887465684      SALE
...            ...       ...
213761  3902882321       WRT
213764  3902878689      MGMT
213765  3902878689      MNFC
213766  3902883233      SALE
213767  3902883233      MGMT

[158619 rows x 2 columns]
⚠️ WARNING: Duplicate values found in Job_Industries.job_id
            job_id  industry_id
3       3887467939           82
4       3887467939           80
5       3887471331           57
6       3887471331          332
7       3887471331          383
...            ...          ...
164791  3906264345           12
164792  3902880673           59
164793  3902880673         3243
164799  3906261853           15
164800  3906261853           12

[62089 rows x 2 columns]
⚠️ WARNING: Duplicate values found in Company_Industries.company_id
       company_id                             industry
437      87089640                               Retail
546       2831596                   Medical Practices
793       1687254                       Manufacturing
1159     13637042        IT Services and IT Consulting
1170        27292             Staffing and Recruiting
2908         4721  International Trade and Development
```

Handling duplicate 'job_id' in jobs table

```
# Checking for duplicates
duplicate_jobs = jobs_df[jobs_df.duplicated(subset=['job_id'], keep=False)]
print(duplicate_jobs)
```

```
Pay: $18-20/hour
Expected hours: 35 – 45 per week
Benefits:Paid time offSchedule:8 hour shiftMonday to FridayExperience:Marketing: 1 year (Preferred)Graphic design: 2 years (Preferred)W
"
1829192
...
Established in 1997
Mission: L.A. Care’s mission is to provide access to quality health care for Los Angeles County's vulnerable and low-income communities
Job Summary
Duties
Participate and contribute in multiple projects by completing and updating project documentation; managing project scope; adjusting sch

[16793 rows x 31 columns]
```

```
# Checking how many unique job IDs exist
unique_jobs_count = jobs_df['job_id'].nunique()
total_jobs_count = jobs_df.shape[0]

print(f"Total Job Records: {total_jobs_count}")
print(f"Unique Job IDs: {unique_jobs_count}")

duplicate_jobs = jobs_df[jobs_df.duplicated(subset=['job_id'], keep=False)]
print(duplicate_jobs.head(20))
```

```
Total Job Records: 18714
Unique Job IDs: 2198

Pay: $18-20/hour
Expected hours: 35 – 45 per week
Benefits:Paid time offSchedule:8 hour shiftMonday to FridayExperience:Marketing: 1 year (Preferred)Graphic design: 2 years (Preferred)W
"
1829192
Thank you for taking the time to explore a career with us. We are excited to be a new group practice in the community. If you are looki
We are actively looking to hire a therapist in the area who is passionate about working with adults and committed to growth and excelle
```

```
We value and are strengthened by diversity and desire a warm and welcoming place for all people. We believe in racial and ethnic equali
Position Requirement Possibilities:A graduate level psychological counseling-related degreeMasters of Social Work (MSW/LSW)Licensed Pro
About Aspen Therapy and Wellness LLCAspen Therapy and Wellness is a mental health services provider focusing on work with adults in an
Please note that this job description is not exhaustive and additional duties may be assigned as needed."
10998357
We offer highly competitive wages
We are a serious
Please send a resumes to pardom@nationalexemplar.com. o"
We are a prominent Lake Success Law Firm seeking an associate attorney for its growing Elder Law and Estate Planning practice. The succ
 Responsibilities will include:
Counseling clients with regard to estate planning and asset protection;Formulating and overseeing execution of Medicaid and estate plan
Qualifications:Juris Doctor degree (J.D.) from an accredited law schoolLicensed to practice law in New York10-15 years of experienceExp
 Competitive salary commensurate with experienceSalary: $140

[20 rows x 31 columns]
```

```
# Checking for exact duplicates across all columns
exact_duplicates = jobs_df[jobs_df.duplicated(keep=False)]
print("Exact Duplicate Rows:", exact_duplicates.shape[0])

# Checking for duplicates based only on job_id but with different details
partial_duplicates = jobs_df[jobs_df.duplicated(subset=['job_id'], keep=False) & ~jobs_df.duplicated(keep=False)]
print("Partial Duplicate Rows:", partial_duplicates.shape[0])
```

```
Exact Duplicate Rows: 15287
Partial Duplicate Rows: 1506
```

```
# Droping exact duplicates
jobs_df = jobs_df.drop_duplicates()
print("After dropping exact duplicates:", jobs_df.shape)
```

```
After dropping exact duplicates: (3567, 31)
```

```
print(f"Final Unique Job IDs: {jobs_df['job_id'].nunique()}")
print(f"Total Jobs After Cleanup: {jobs_df.shape[0]}")
```

```
Final Unique Job IDs: 2198
Total Jobs After Cleanup: 3567
```

```
# Checking for remaining duplicate issues
print("✅ Final Data Integrity Check")
print("Duplicate job_id in Jobs Table:", jobs_df.duplicated(subset=['job_id']).sum())
print("Duplicate job_id in Job_Industries:", job_industries_df.duplicated(subset=['job_id', 'industry_id']).sum())
print("Duplicate company_id in Company_Industries:", company_industries_df.duplicated(subset=['company_id', 'industry_id']).sum())
print("Duplicate company_id in Employee_Counts:", employee_counts_df.duplicated(subset=['company_id']).sum())
```

```
✅ Final Data Integrity Check
Duplicate job_id in Jobs Table: 1368
Duplicate job_id in Job_Industries: 0
Duplicate company_id in Company_Industries: 0
Duplicate company_id in Employee_Counts: 11314
```

Handling Duplicate 'company_id' in Employee_Counts Table

```
# Keeping only the most recent entry per company (assuming time_recorded field exists)
employee_counts_df = employee_counts_df.sort_values(by=['company_id', 'time_recorded'], ascending=[True, False])
employee_counts_df = employee_counts_df.drop_duplicates(subset=['company_id'], keep='first')

print("Duplicate company_id in Employee_Counts:", employee_counts_df.duplicated(subset=['company_id']).sum())
```

```
Duplicate company_id in Employee_Counts: 0
```

```
# STEP 3: Validating Foreign Key Relationships

def foreign_key_integrity(fk_df, fk_column, pk_df, pk_column, table_name):
    unmatched_fk = fk_df[~fk_df[fk_column].isin(pk_df[pk_column])]
    if unmatched_fk.empty:
        print(f"✅ Referential integrity maintained for {table_name}.{fk_column} → {pk_column}")
    else:
        print(f"⚠️ WARNING: {len(unmatched_fk)} unmatched foreign keys in {table_name}.{fk_column}")
        print(unmatched_fk)

company_industries_df.rename(columns={'industry': 'industry_id'}, inplace=True)
```

```
# Check FK relationships
foreign_key_integrity(jobs_df, "company_id", companies_df, "company_id", "Jobs")
foreign_key_integrity(salaries_df, "job_id", jobs_df, "job_id", "Salaries")
foreign_key_integrity(job_skills_df, "job_id", jobs_df, "job_id", "Job_Skills")
foreign_key_integrity(job_skills_df, "skill_abr", skills_df, "skill_abr", "Job_Skills")
foreign_key_integrity(job_industries_df, "job_id", jobs_df, "job_id", "Job_Industries")
foreign_key_integrity(job_industries_df, "industry_id", industries_df, "industry_id", "Job_Industries")
foreign_key_integrity(company_industries_df, "company_id", companies_df, "company_id", "Company_Industries")
foreign_key_integrity(company_industries_df, "industry_id", industries_df, "industry_id", "Company_Industries")
```

```
4              5  3887473220       35.00         NaN        33.0     HOURLY      USD       BASE_SALARY
...          ...         ...         ...         ...         ...        ...      ...               ...
40780      40781  3902881498         NaN        15.5         NaN     HOURLY      USD       BASE_SALARY
40781      40782  3902883232         NaN        25.0         NaN     HOURLY      USD       BASE_SALARY
40782      40783  3902866633       21.53         NaN        21.1     HOURLY      USD       BASE_SALARY
40783      40784  3902879720   125000.00         NaN    100000.0     YEARLY      USD       BASE_SALARY
40784      40785  3902878689    85862.00         NaN     63601.0     YEARLY      USD       BASE_SALARY

[40785 rows x 8 columns]
⚠️ WARNING: 213768 unmatched foreign keys in Job_Skills.job_id
           job_id skill_abr
0      3884428798      MRKT
1      3884428798        PR
2      3884428798       WRT
3      3887473071      SALE
4      3887465684       FIN
...           ...       ...
213763 3902876855        HR
213764 3902878689      MGMT
213765 3902878689      MNFC
213766 3902883233      SALE
213767 3902883233      MGMT

[213768 rows x 2 columns]
✅ Referential integrity maintained for Job_Skills.skill_abr → skill_abr
⚠️ WARNING: 164808 unmatched foreign keys in Job_Industries.job_id
           job_id  industry_id
0      3884428798           82
1      3887473071           48
2      3887465684           41
3      3887467939           82
4      3887467939           80
...           ...          ...
164803 3902882321          104
164804 3902879720           27
164805 3902876855           80
164806 3902878689          116
164807 3902883233           44

[164808 rows x 2 columns]
✅ Referential integrity maintained for Job_Industries.industry_id → industry_id
✅ Referential integrity maintained for Company_Industries.company_id → company_id
⚠️ WARNING: 24375 unmatched foreign keys in Company_Industries.industry_id
       company_id                                        industry_id
0          391906                      Book and Periodical Publishing
1        22292832                                        Construction
2           20300                                             Banking
3         3570660                      Book and Periodical Publishing
4          878353                             Staffing and Recruiting
...           ...                                               ...
24370       32313                    Retail Luxury Goods and Jewelry
24371    15225088                      IT Services and IT Consulting
24372     2852377                                       Hospitality
24373    19114724                                        Construction
24374     8060959  Transportation, Logistics, Supply Chain and Storage

[24375 rows x 2 columns]
```

Handling unmatched 'company_id' in Jobs Table

```
# Converting company_id to the same type
jobs_df["company_id"] = jobs_df["company_id"].astype(str)
companies_df["company_id"] = companies_df["company_id"].astype(str)

# Dropping jobs where company_id is missing
jobs_df = jobs_df.dropna(subset=["company_id"])

# Removing jobs where company_id is not found in Companies table
jobs_df = jobs_df[jobs_df["company_id"].isin(companies_df["company_id"])]
```

```
print(f"✅ Fixed: Remaining unmatched company_id in Jobs: {len(jobs_df[~jobs_df['company_id'].isin(companies_df['company_id'])])}")
```

```
✅ Fixed: Remaining unmatched company_id in Jobs: 0
<ipython-input-40-97888d3f253f>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
  jobs_df["company_id"] = jobs_df["company_id"].astype(str)
```

Handling unmatched 'job_id' in Salaries Table

```
# Converting job_id to the same type
salaries_df["job_id"] = salaries_df["job_id"].astype(str)
jobs_df["job_id"] = jobs_df["job_id"].astype(str)

# Removing salaries with missing job_id
salaries_df = salaries_df.dropna(subset=["job_id"])

# Keeping only valid job_id values
salaries_df = salaries_df[salaries_df["job_id"].isin(jobs_df["job_id"])]

print(f"✅ Fixed: Remaining unmatched job_id in Salaries: {len(salaries_df[~salaries_df['job_id'].isin(jobs_df['job_id'])])}")
```

```
✅ Fixed: Remaining unmatched job_id in Salaries: 0
```

Handling unmatched 'job_id' in Job_Skills Table

```
# Converting job_id to the same type
job_skills_df["job_id"] = job_skills_df["job_id"].astype(str)

# Keeping only valid job_id values
job_skills_df = job_skills_df[job_skills_df["job_id"].isin(jobs_df["job_id"])]

print(f"✅ Fixed: Remaining unmatched job_id in Job_Skills: {len(job_skills_df[~job_skills_df['job_id'].isin(jobs_df['job_id'])])}")
```

```
✅ Fixed: Remaining unmatched job_id in Job_Skills: 0
```

Handling unmatched job_id in Job_Industries Table

```
# Converting job_id to the same type
job_industries_df["job_id"] = job_industries_df["job_id"].astype(str)

# Keeping only valid job_id values
job_industries_df = job_industries_df[job_industries_df["job_id"].isin(jobs_df["job_id"])]

print(f"✅ Fixed: Remaining unmatched job_id in Job_Industries: {len(job_industries_df[~job_industries_df['job_id'].isin(jobs_df['job_id'])]
```

```
✅ Fixed: Remaining unmatched job_id in Job_Industries: 0
```

Handling unmatched industry_id in Company_Industries Table

```
# Converting industry names to industry IDs
industry_name_to_id = dict(zip(industries_df["industry_name"], industries_df["industry_id"]))
company_industries_df["industry_id"] = company_industries_df["industry_id"].replace(industry_name_to_id)


# Converting industry_id to string type for consistency
company_industries_df["industry_id"] = company_industries_df["industry_id"].astype(str)
industries_df["industry_id"] = industries_df["industry_id"].astype(str)

# Keeping only valid industry_id values
company_industries_df = company_industries_df[company_industries_df["industry_id"].isin(industries_df["industry_id"])]

print(f"✅ Fixed: Remaining unmatched industry_id in Company_Industries: {len(company_industries_df[~company_industries_df['industry_id'].is
```

```
✅ Fixed: Remaining unmatched industry_id in Company_Industries: 0
```

```
# Checking again after cleaning
foreign_key_integrity(jobs_df, "company_id", companies_df, "company_id", "Jobs Table")
```

```
foreign_key_integrity(salaries_df, "job_id", jobs_df, "job_id", "Salaries Table")
foreign_key_integrity(job_skills_df, "job_id", jobs_df, "job_id", "Job Skills Table")
foreign_key_integrity(job_industries_df, "job_id", jobs_df, "job_id", "Job Industries Table")
foreign_key_integrity(company_industries_df, "industry_id", industries_df, "industry_id", "Company Industries Table")
```

```
✅ Referential integrity maintained for Jobs Table.company_id → company_id
✅ Referential integrity maintained for Salaries Table.job_id → job_id
✅ Referential integrity maintained for Job Skills Table.job_id → job_id
✅ Referential integrity maintained for Job Industries Table.job_id → job_id
✅ Referential integrity maintained for Company Industries Table.industry_id → industry_id
```

Field-Level Integrity Checks

```
# Step 4: Validation checks
def check_column_values(df, column, min_value=None, max_value=None, allowed_values=None):
    issues_found = False  # Flag to track issues

    if min_value is not None and df[column].min() < min_value:
        print(f"⚠️ WARNING: {column} has values below {min_value}")
        issues_found = True

    if max_value is not None and df[column].max() > max_value:
        print(f"⚠️ WARNING: {column} has values above {max_value}")
        issues_found = True

    if allowed_values is not None and not df[column].isin(allowed_values).all():
        print(f"⚠️ WARNING: {column} contains invalid values: {df[~df[column].isin(allowed_values)][column].unique()}")
        issues_found = True

    if not issues_found:
        print(f"✅ {column} values are within the expected range.")

# Validating salary range (Assume min salary should be ≥0 and max salary ≤ 500,000)
check_column_values(salaries_df, "min_salary", min_value=0)
check_column_values(salaries_df, "max_salary", max_value=500000)

# Validating company size (Assume values between 1-7)
check_column_values(companies_df, "company_size", min_value=1, max_value=7)

# Validating pay period (Assume only ["Hourly", "Monthly", "Yearly"] are allowed)
check_column_values(salaries_df, "pay_period", allowed_values=["Hourly", "Monthly", "Yearly"])
```

```
✅ min_salary values are within the expected range.
✅ max_salary values are within the expected range.
✅ company_size values are within the expected range.
✅ pay_period values are within the expected range.
```

```
# STEP 5: Checking for overall Missing Values
def check_missing_values(df):
    missing_values = df.isnull().sum()
    if missing_values.any():
        print(f"⚠️ WARNING: Missing values detected in {df.name}:\n{missing_values[missing_values > 0]}")
    else:
        print(f"✅ No missing values in {df.name}")

# Runningmissing value checks
for name, (df, _) in datasets.items():
    df.name = name
    check_missing_values(df)
```

```
location                    17011
company_id                  16745
views                       16913
med_salary                  17663
min_salary                  17517
formatted_work_type         17833
applies                     17368
original_listed_time        17646
remote_allowed              17262
```

```
listed_time                 18274
posting_domain              18391
sponsored                   18451
work_type                   18527
currency                    18575
compensation_type           18593
normalized_salary           18616
zip_code                    18651
fips                        18688
dtype: int64
⚠️ WARNING: Missing values detected in 0        Companies
1        Companies
2        Companies
3        Companies
4        Companies
           ...
24468    Companies
24469    Companies
24470    Companies
24471    Companies
24472    Companies
Name: name, Length: 24473, dtype: object:
description      297
company_size    2774
state             22
city               1
zip_code          28
address           22
dtype: int64
⚠️ WARNING: Missing values detected in Salaries:
max_salary     6838
med_salary    33947
min_salary     6838
dtype: int64
✅ No missing values in Skills
⚠️ WARNING: Missing values detected in Industries:
industry_name    34
dtype: int64
✅ No missing values in Job_Skills
✅ No missing values in Job_Industries
✅ No missing values in Company_Industries
✅ No missing values in Employee_Counts
```

```
# Handling Missing Values in Jobs Table
# Droping rows where job_id, title, or company_id are missing
jobs_df = jobs_df.dropna(subset=["job_id", "title", "company_id"])

# Filling missing descriptions with "No description provided"
jobs_df["description"] = jobs_df["description"].fillna("No description provided")

# Filling missing salary values with the median salary
jobs_df["max_salary"] = jobs_df["max_salary"].fillna(jobs_df["max_salary"].median())
jobs_df["med_salary"] = jobs_df["med_salary"].fillna(jobs_df["med_salary"].median())
jobs_df["min_salary"] = jobs_df["min_salary"].fillna(jobs_df["min_salary"].median())

# Filling categorical columns with "Unknown"
categorical_columns = ["pay_period", "location", "formatted_work_type", "work_type", "currency"]
for col in categorical_columns:
    jobs_df[col] = jobs_df[col].fillna("Unknown")


# Handling Missing Values in Companies Table
# Droping companies where 'company_id' or 'name' is missing
companies_df = companies_df.dropna(subset=["company_id", "name"])

# Filling missing descriptions with "No description available"
companies_df["description"] = companies_df["description"].fillna("No description available")

# Filling missing company size with the most frequent value (mode)
companies_df["company_size"] = companies_df["company_size"].fillna(companies_df["company_size"].mode()[0])

# Filling missing locations with "Unknown"
location_columns = ["state", "city", "zip_code", "address"]
for col in location_columns:
    companies_df[col] = companies_df[col].fillna("Unknown")


# Handling Missing Values in Salaries Table
# Droping salary records where job_id is missing
salaries_df = salaries_df.dropna(subset=["job_id"])
```

```
# Filling missing salary values with median
salary_columns = ["max_salary", "med_salary", "min_salary"]
for col in salary_columns:
    salaries_df[col] = salaries_df[col].fillna(salaries_df[col].median())


# Handling Missing Values in Industries Table
# Droping industries where industry_id is missing
industries_df = industries_df.dropna(subset=["industry_id"])

# Filling missing industry names with "Unknown Industry"
industries_df["industry_name"] = industries_df["industry_name"].fillna("Unknown Industry")


print("✅ Checking for remaining missing values:")
print("Jobs missing values:\n", jobs_df.isnull().sum())
print("Companies missing values:\n", companies_df.isnull().sum())
print("Salaries missing values:\n", salaries_df.isnull().sum())
print("Industries missing values:\n", industries_df.isnull().sum())
```

```
✅ Checking for remaining missing values:
Jobs missing values:
 job_id                        0
company_name                   0
title                          0
description                    0
max_salary                     0
pay_period                     0
location                       0
company_id                     0
views                          0
med_salary                     0
min_salary                     0
formatted_work_type            0
applies                        0
original_listed_time           0
remote_allowed                 0
job_posting_url                0
application_url                0
application_type               0
expiry                         0
closed_time                    0
formatted_experience_level     0
skills_desc                    0
listed_time                    0
posting_domain                 0
sponsored                      0
work_type                      0
currency                       0
compensation_type              0
normalized_salary              0
zip_code                       0
fips                           0
dtype: int64
Companies missing values:
 company_id      0
name             0
description      0
company_size     0
state            0
country          0
city             0
zip_code         0
address          0
url              0
dtype: int64
Salaries missing values:
 salary_id            0
job_id                0
max_salary            0
med_salary            0
min_salary            0
pay_period            0
currency              0
compensation_type     0
dtype: int64
Industries missing values:
 industry_id      0
industry_name     0
dtype: int64
```

Explanation on Verification of Data Integrity Process and Results (Step by Step)

We carried out the data integrity checks to guarantee our dataset is clean, correct, and trustworthy for downstream analysis. The following is a simple breakdown of the steps we followed, the issues we found, and how we fixed them.

Step 1: Checking for Missing Values

What We Did: We scanned every table in the dataset for the presence of missing entries by using a function that computed the number of NaN (Null) values across all columns.

What We Found:

- The Jobs table contained over 18,000 missing values in fields such as companyid, title, description, salary, location and job posting URL.
- The Companies table had missing data in description, companysize, city, and state.
- The Salaries table contained more than 6,800 missing salaries (maxsalary, medsalary, minsalary).
- The Industries table had 34 missing industry names.

How We Fixed It:

- When a critical field (e.g., jobid, companyid, companyname, title) was not available, we skipped those rows as they were not usable.
- For a descriptive field (e.g., description, industryname), we imputed "Unknown" or "No description provided", if it was absent.
- For an absence in a numeric field (e.g., salary), we impute with the median value to maintain balanced data.
- In case of a miss on categorical field (e.g., payperiod, worktype), we imputed it to "Unknown.

Final Result: The missing values were either dropped or imputed in which no critical information was left blank.

Step 2: Checking for Duplicate Values

What We Did: We searched for the presence of duplicated values in the columns of tables in which each row can be considered to be unique (e.g., jobid in the Jobs table or companyid in the Companies table). We used code to detect duplicates.

What We Found:

- A few duplicate job postings had the same jobid.
- Some companies were listed multiple times under different formats.

How We Fixed It:

- We excluded duplicate rows so as to retain only one unique row per job and company.
- For companies that have very similar names but differing companyid, we retained the first firm encountered and neglected duplicates.

Final Result: The resulting dataset is now unique with respect to jobs and companies, and thus more trustworthy.

Step 3: Checking for Foreign Key Issues (Referential Integrity)

What We Did:

We checked if foreign keys (which link different tables) matched correctly. For example:

- Each companyid in the Jobs table should be present in the Companies table.
- Each jobid in Salaries table must be present in Jobs table.
- That all industryid in the CompanyIndustries table are in the Industries table.

What We Found:

- With 18,714 job postings, a companyid was missing in the Companies table.
- 40,785 salary records had a jobid that did not exist in the Jobs table.
- 213,768 skill mappings had a jobid that was not in the Jobs table.
- 24,375 industry mappings had an industryid that did not match the Industries table.

How We Fixed It:

- When a job or company did not exist in the main table, we deleted those unmatched records.
- If some industry_id values were stored as text, we reformatted them so that they match numbers.
- When missing jobs or companies were relevant, we flagged them as "Unknown" to remove them as we did other missing values, and not by deleting them.

Final Result: The foreign key constraints were absolute, guaranteeing that every job, salary, and skill is correctly related.

Step 4: Handling Data Manually (Without Code)

What We Did By Hand:

- There were a few instances where employing code was not the most efficient solution and we had to manually process the data.
- Company Name Issues Some companies had slightly different spellings (e.g., "Google LLC" vs. "Google Inc.". Instead of blindly merging them with code, we manually reviewed and standardized the names.
- Industry Names vs. IDs Some industries were stored as text instead of numeric IDs. We hand mapped them to verify that they are in agreement with official categories.
- Job Titles with Extra Spaces Some job titles had leading or trailing spaces (e.g., " Data Scientist " instead of "Data Scientist". We did not remove spaces in a case where automatic cleaning was unsatisfactory.

Final Result: These manual corrections enhanced data accuracy when automated corrections were unsafe.

Generative AI Prompts used for this assignment

We used AI tools like ChatGPT to help guide my understanding of data validation, cleaning, and referential integrity checks. We used various prompts to explore best practices for handling missing values, fixing unmatched foreign keys, and structuring my dataset properly. The AI provided me with explanations and coding techniques, but we wrote, tested, and modified the code ourselves based on my dataset.

1. "How do I check if primary keys are unique in a Pandas DataFrame?"
2. "How can I validate referential integrity between two tables in Python?"
3. "How do I check for inconsistent category values in a dataset?"
4. "What should I do if my dataset contains too many missing values?"
5. "How do I clean up text data with leading/trailing spaces in a Pandas DataFrame?"
6. "How do I detect and merge duplicate companies with slight name differences?"
7. "How can I manually inspect a large dataset for formatting issues?"
8. "What should I do if some company IDs in the job postings table don't exist in the companies table?"

```
# saving the cleaned datasets
cleaned_files = {
    "jobs_cleaned.csv": jobs_df,
    "companies_cleaned.csv": companies_df,
    "salaries_cleaned.csv": salaries_df,
    "skills_cleaned.csv": skills_df,
    "job_skills_cleaned.csv": job_skills_df,
    "industries_cleaned.csv": industries_df,
    "job_industries_cleaned.csv": job_industries_df,
    "company_industries_cleaned.csv": company_industries_df,
    "employee_counts_cleaned.csv": employee_counts_df
}

for file_name, df in cleaned_files.items():
    df.to_csv(file_name, index=False)

print("✅ Cleaned datasets saved successfully!")
```

```
✅ Cleaned datasets saved successfully!
```

References:

ArshKA, & AndhikaWB. (2024). LinkedIn-job-scraper/databasestructure.md at master · arshka/linkedin-job-scraper. GitHub. https://github.com/ArshKA/LinkedIn-Job-Scraper/blob/master/DatabaseStructure.md

OpenAI. (2025). Chatgpt. https://chatgpt.com/

W3Schools. (2025). Python. W3Schools Online Web Tutorials. https://www.w3schools.com/python/default.asp