Data Analysis

Comprehensive Data Cleaning & Exploratory Analysis of Job Market Trends

Connor Coulter

Wei Wang

Balqis Bevi Abdul Hannan Kanaga

# Import Data

import pandas as pd, numpy as np, os  
  
# --- Load dataset safely ---  
CSV\_PATHS = ["data/lightcast\_job\_postings.csv", "lightcast\_job\_postings.csv"]  
csv\_path = next((p for p in CSV\_PATHS if os.path.exists(p)), None)  
if not csv\_path:  
 raise FileNotFoundError("⚠️ lightcast\_job\_postings.csv not found")  
  
df = pd.read\_csv(csv\_path, low\_memory=False)  
print("✅ Loaded dataset:", df.shape)  
  
# --- Ensure key derived columns exist ---  
# Industry display  
if "NAICS\_2022\_6\_NAME" in df.columns:  
 df["INDUSTRY\_DISPLAY"] = df["NAICS\_2022\_6\_NAME"]  
elif "INDUSTRY" in df.columns:  
 df["INDUSTRY\_DISPLAY"] = df["INDUSTRY"]  
else:  
 df["INDUSTRY\_DISPLAY"] = "Unknown"  
  
# Salary display  
salary\_candidates = ["SALARY","SALARY\_MEDIAN","SALARY\_MID","SALARY\_ANNUAL","PAY\_RATE"]  
found\_salary = next((c for c in salary\_candidates if c in df.columns), None)  
if found\_salary:  
 df["SALARY\_DISPLAY"] = pd.to\_numeric(df[found\_salary], errors="coerce")  
 df["SALARY\_DISPLAY"].fillna(df["SALARY\_DISPLAY"].median(), inplace=True)  
else:  
 df["SALARY\_DISPLAY"] = np.nan  
  
print("Derived columns:", {  
 "INDUSTRY\_DISPLAY": df["INDUSTRY\_DISPLAY"].notna().sum(),  
 "SALARY\_DISPLAY": df["SALARY\_DISPLAY"].notna().sum()  
})

✅ Loaded dataset: (72498, 131)  
Derived columns: {'INDUSTRY\_DISPLAY': np.int64(72454), 'SALARY\_DISPLAY': np.int64(72498)}

/var/folders/g7/sfc5tly50013vn\_cy1c842180000gn/T/ipykernel\_17312/1469011101.py:26: FutureWarning:  
  
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 work 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(value) instead, to perform the operation inplace on the original object.

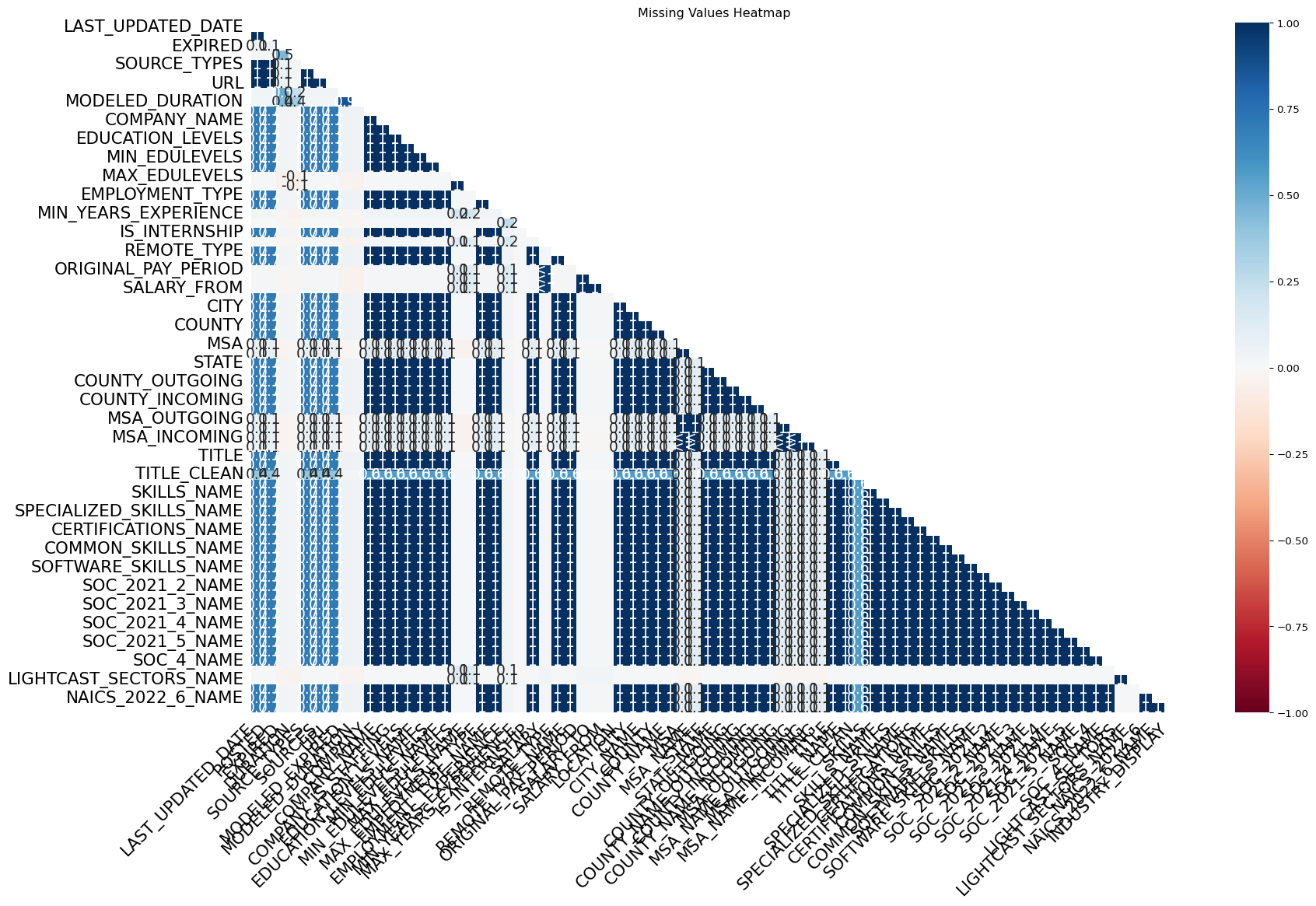
print(“Derived non-null:”, { “INDUSTRY\_DISPLAY”: df[“INDUSTRY\_DISPLAY”].notna().sum(), “SALARY\_DISPLAY”: df[“SALARY\_DISPLAY”].notna().sum() })

::: {.cell execution\_count=2}  
``` {.python .cell-code}  
# Data Cleaning & Preprocessing  
# Drop Unnecessary Columns  
columns\_to\_drop = [  
 "ID","LAST\_UPDATED\_TIMESTAMP","DUPLICATES","ACTIVE\_URLS","ACTIVE\_SOURCES\_INFO",  
 "TITLE\_RAW","BODY","COMPANY\_RAW",  
 "NAICS2","NAICS2\_NAME","NAICS3","NAICS3\_NAME","NAICS4","NAICS4\_NAME",  
 "NAICS5","NAICS5\_NAME","NAICS6","NAICS6\_NAME",  
 "NAICS\_2022\_2","NAICS\_2022\_2\_NAME","NAICS\_2022\_3","NAICS\_2022\_3\_NAME",  
 "NAICS\_2022\_4","NAICS\_2022\_4\_NAME","NAICS\_2022\_5","NAICS\_2022\_5\_NAME",  
 "SOC\_2","SOC\_2\_NAME","SOC\_3","SOC\_3\_NAME","SOC\_5","SOC\_5\_NAME",  
 "CIP2","CIP2\_NAME","CIP4","CIP4\_NAME","CIP6","CIP6\_NAME",  
 "LOT\_CAREER\_AREA","LOT\_CAREER\_AREA\_NAME","LOT\_OCCUPATION","LOT\_OCCUPATION\_NAME",  
 "LOT\_SPECIALIZED\_OCCUPATION","LOT\_SPECIALIZED\_OCCUPATION\_NAME",  
 "LOT\_OCCUPATION\_GROUP","LOT\_OCCUPATION\_GROUP\_NAME",  
 "LOT\_V6\_SPECIALIZED\_OCCUPATION","LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME",  
 "LOT\_V6\_OCCUPATION","LOT\_V6\_OCCUPATION\_NAME","LOT\_V6\_OCCUPATION\_GROUP",  
 "LOT\_V6\_OCCUPATION\_GROUP\_NAME","LOT\_V6\_CAREER\_AREA","LOT\_V6\_CAREER\_AREA\_NAME",  
 "ONET","ONET\_NAME","ONET\_2019","ONET\_2019\_NAME"  
]  
drop\_existing = [c for c in columns\_to\_drop if c in df.columns]  
df.drop(columns=drop\_existing, inplace=True)  
print("Remaining columns (first 30):", list(df.columns)[:30])

Remaining columns (first 30): ['LAST\_UPDATED\_DATE', 'POSTED', 'EXPIRED', 'DURATION', 'SOURCE\_TYPES', 'SOURCES', 'URL', 'MODELED\_EXPIRED', 'MODELED\_DURATION', 'COMPANY', 'COMPANY\_NAME', 'COMPANY\_IS\_STAFFING', 'EDUCATION\_LEVELS', 'EDUCATION\_LEVELS\_NAME', 'MIN\_EDULEVELS', 'MIN\_EDULEVELS\_NAME', 'MAX\_EDULEVELS', 'MAX\_EDULEVELS\_NAME', 'EMPLOYMENT\_TYPE', 'EMPLOYMENT\_TYPE\_NAME', 'MIN\_YEARS\_EXPERIENCE', 'MAX\_YEARS\_EXPERIENCE', 'IS\_INTERNSHIP', 'SALARY', 'REMOTE\_TYPE', 'REMOTE\_TYPE\_NAME', 'ORIGINAL\_PAY\_PERIOD', 'SALARY\_TO', 'SALARY\_FROM', 'LOCATION']

:::

# Handle Missing Values  
import missingno as msno, matplotlib.pyplot as plt  
  
msno.heatmap(df)  
plt.title("Missing Values Heatmap")  
plt.show()  
  
df.dropna(thresh=len(df) \* 0.5, axis=1, inplace=True)  
  
if "SALARY\_DISPLAY" in df.columns:  
 df["SALARY\_DISPLAY"].fillna(df["SALARY\_DISPLAY"].median(), inplace=True)  
  
for col in df.select\_dtypes(include="object").columns:  
 df[col].fillna("Unknown", inplace=True)  
  
# Remove Duplicates  
subset\_cols = [c for c in ["TITLE","COMPANY\_NAME","LOCATION","POSTED"] if c in df.columns]  
if subset\_cols:  
 before = len(df)  
 df.drop\_duplicates(subset=subset\_cols, keep="first", inplace=True)  
 print(f"Removed {before - len(df)} duplicates using {subset\_cols}")  
  
# Exploratory Data Analysis (EDA)  
# Job Postings by Industry (Top 15)  
import plotly.express as px  
  
counts = (  
 df["INDUSTRY\_DISPLAY"]  
 .value\_counts(dropna=False)  
 .head(15)  
 .reset\_index(name="Count")  
 .rename(columns={"index": "Industry"})  
 .sort\_values("Count")  
)  
fig1 = px.bar(  
 counts, x="Count", y="INDUSTRY\_DISPLAY", orientation="h",  
 title="Top 15 Industries by Number of Job Postings"  
)  
fig1.show()



Removed 3300 duplicates using ['TITLE', 'COMPANY\_NAME', 'LOCATION', 'POSTED']

/var/folders/g7/sfc5tly50013vn\_cy1c842180000gn/T/ipykernel\_17312/2976636553.py:11: FutureWarning:  
  
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 work 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(value) instead, to perform the operation inplace on the original object.  
  
  
  
/var/folders/g7/sfc5tly50013vn\_cy1c842180000gn/T/ipykernel\_17312/2976636553.py:14: FutureWarning:  
  
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 work 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(value) instead, to perform the operation inplace on the original object.

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

# Salary Distribution by Industry (Top 15)  
sdf = df[["INDUSTRY\_DISPLAY","SALARY\_DISPLAY"]].copy()  
sdf = sdf.dropna()  
sdf = sdf[sdf["SALARY\_DISPLAY"] > 0]  
  
top\_industries = sdf["INDUSTRY\_DISPLAY"].value\_counts().head(15).index  
sdf = sdf[sdf["INDUSTRY\_DISPLAY"].isin(top\_industries)]  
  
fig2 = px.box(  
 sdf, x="INDUSTRY\_DISPLAY", y="SALARY\_DISPLAY",  
 title="Salary Distribution by Industry (Top 15)",  
 points=False  
)  
fig2.update\_layout(xaxis\_tickangle=-45)  
fig2.show()  
  
# Remote vs. On-Site Jobs  
if "REMOTE\_TYPE\_NAME" in df.columns:  
 rc = df["REMOTE\_TYPE\_NAME"].value\_counts().reset\_index()  
 rc.columns = ["Remote Type","Count"]  
 fig3 = px.pie(  
 rc, names="Remote Type", values="Count",  
 title="Remote vs. On-Site Job Distribution"  
 )  
 fig3.show()

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

# EDA: Rationale & Insights

## Job Postings by Industry

Why: Highlights sectors where demand is concentrated, showing which industries are actively hiring. Key Insights: The top three industries by job postings are Temporary Help Services, Miscellaneous Ambulatory Health Care Services, and Semiconductor and Related Device Manufacturing.

## Salary Distribution by Industry

Why: Shows where negotiation power exists and highlights industries paying well. Key Insights: Automotive Parts and Accessories Retailers show a wide range (negotiation potential), while Barber Shops show a narrow range (little negotiation).

## Remote vs. On-Site Jobs

Why: Workplace flexibility is a major factor in today’s job market. Key Insights: Most postings (78.3%) don’t specify remote status. About 17% are remote, 3.1% hybrid, and 1.6% explicitly not remote.