Data Analysis

Comprehensive Data Cleaning & Exploratory Analysis of Job Market Trends

Mahira Ayub

Ava Godsy

Joshua Lawrence

Table of contents

# 1. Exploratory Data Analysis

## 1.1 Importing Dataset Using Pandas

import pandas as pd  
  
# Load the dataset  
data = pd.read\_csv('data\lightcast\_job\_postings.csv')

<>:4: SyntaxWarning:  
  
invalid escape sequence '\l'  
  
<>:4: SyntaxWarning:  
  
invalid escape sequence '\l'  
  
C:\Users\avago\AppData\Local\Temp\ipykernel\_4856\3126061157.py:4: SyntaxWarning:  
  
invalid escape sequence '\l'  
  
C:\Users\avago\AppData\Local\Temp\ipykernel\_4856\3126061157.py:4: DtypeWarning:  
  
Columns (19,30) have mixed types. Specify dtype option on import or set low\_memory=False.

## 1.2 Dropping Unncessary Columns

* **Which columns are irrelevant or redundant?**  
  ID, URL, ACTIVE\_URLS, DUPLICATES, LAST\_UPDATED TIMESTAMP are irrelevant or redundant columns. They are mostly used for internal tracking and don’t contribute to the actual analysis of jobs, industries, and occupations.
* **Why are we removing multiple versions of NAICS/SOC codes?**  
  The dataset contains multiple versions of industry NAICS and Occupational SOC which can be risky to keep, as there is risk of duplications and inconsistent groupings.
* **How will this improve analysis?**  
  This will improve the analysis by enhancing the efficiency of the data; by having smaller datasets it will be easier to process and run data. It will also improve consistency because by having only one version of the data the risk of duplication will be low.

columns\_to\_drop = [  
'LAST\_UPDATED\_DATE', 'LAST\_UPDATED\_TIMESTAMP', 'DUPLICATES', 'EXPIRED', 'ACTIVE\_SOURCES\_INFO', 'TITLE\_RAW','BODY',  
'COMPANY\_NAME', 'COMPANY\_RAW', 'COMPANY\_IS\_STAFFING', 'EDUCATION\_LEVELS', 'EDUCATION\_LEVELS\_NAME', 'MIN\_EDULEVELS',  
'MIN\_EDULEVELS\_NAME', 'MAX\_EDULEVELS', 'MAX\_EDULEVELS\_NAME', 'MIN\_YEARS\_EXPERIENCE', 'MAX\_YEARS\_EXPERIENCE', 'IS\_INTERNSHIP',  
'ORIGINAL\_PAY\_PERIOD', 'SALARY\_TO', 'SALARY\_FROM', 'COUNTY\_OUTGOING', 'COUNTY\_NAME\_OUTGOING', 'COUNTY\_INCOMING', 'COUNTY\_NAME\_INCOMING',  
'MSA\_OUTGOING', 'MSA\_NAME\_OUTGOING','MSA\_INCOMING', 'MSA\_NAME\_INCOMING', 'NAICS2', 'NAICS2\_NAME', 'NAICS3', 'NAICS3\_NAME', 'NAICS4','NAICS4\_NAME',  
'NAICS5', 'NAICS5\_NAME', 'NAICS6', 'NAICS6\_NAME', 'ONET', 'ONET\_NAME', 'ONET\_2019', 'ONET\_2019\_NAME', 'CIP6', 'CIP6\_NAME', 'CIP4', 'CIP4\_NAME',  
'CIP2', 'CIP2\_NAME', 'SOC\_2021\_2', 'SOC\_2021\_2\_NAME', 'SOC\_2021\_3', 'SOC\_2021\_3\_NAME', 'SOC\_2021\_4', 'SOC\_2021\_4\_NAME', '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', 'SOC\_2', 'SOC\_2\_NAME', 'SOC\_3', 'SOC\_3\_NAME', 'SOC\_4', 'SOC\_4\_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',  
'DURATION', 'SOURCE\_TYPES', 'SOURCES', 'URL', 'ACTIVE\_URLS', 'MODELED\_EXPIRED', 'MODELED\_DURATION', 'LOT\_CAREER\_AREA', 'LOT\_CAREER\_AREA\_NAME',  
'LOT\_OCCUPATION', 'LOT\_OCCUPATION\_NAME', 'LOT\_SPECIALIZED\_OCCUPATION', 'LOT\_SPECIALIZED\_OCCUPATION\_NAME', 'LOT\_OCCUPATION\_GROUP', 'LIGHTCAST\_SECTORS',  
'LIGHTCAST\_SECTORS\_NAME'  
]  
data.drop(columns=columns\_to\_drop, inplace=True)

## 1.3 Handling Missing Values

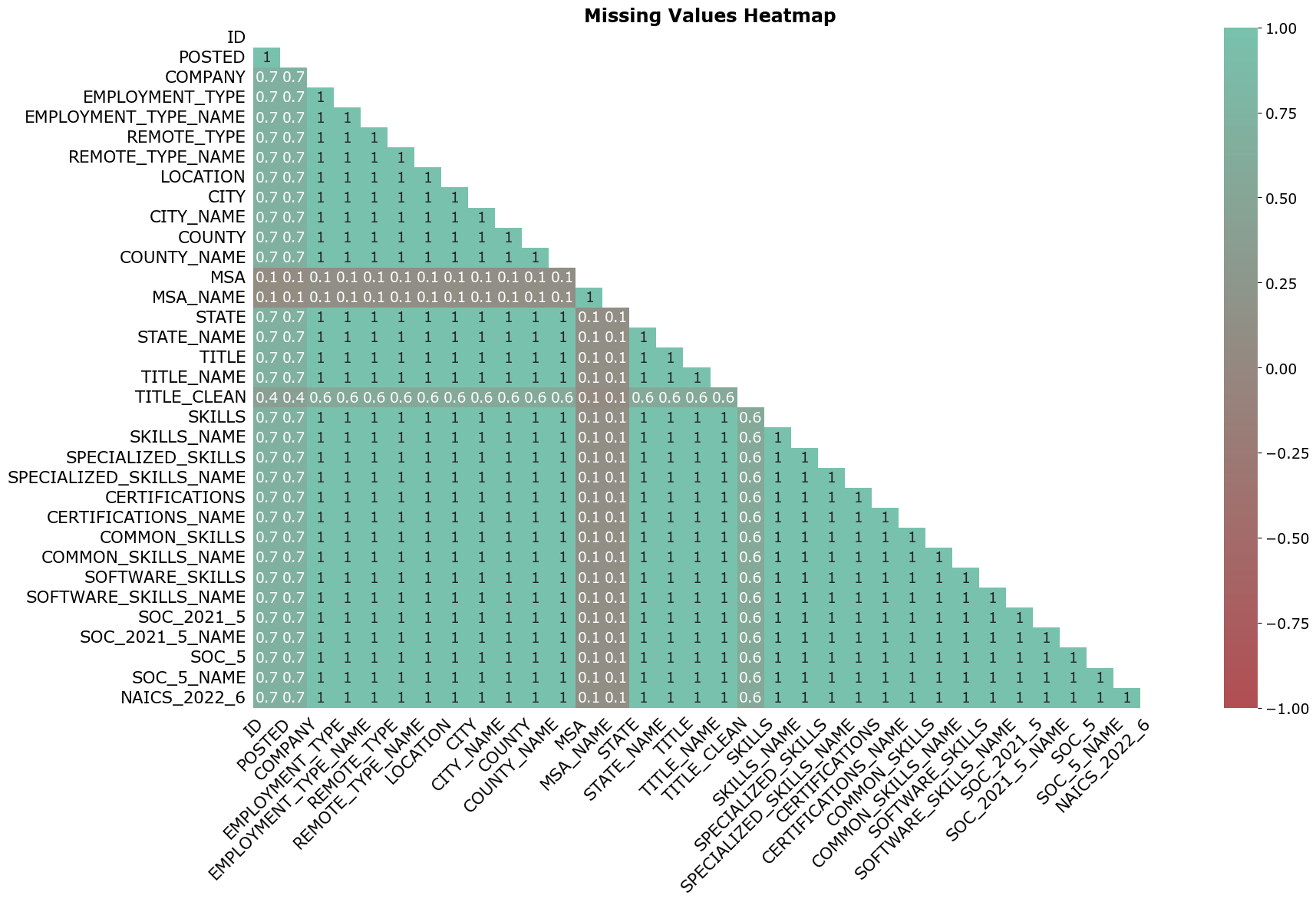
* **Answer the question: How should missing values be handled?**
  + Numerical fields (e.g., Salary) are filled with the median.
  + Categorical fields (e.g., Industry) are replaced with “Unknown”.
  + Columns with >50% missing values are dropped.

import missingno as msno  
import matplotlib.pyplot as plt  
  
  
  
# Fill missing values  
data["SALARY"].fillna(data["SALARY"].median(), inplace=True)  
data["NAICS\_2022\_6\_NAME"].fillna("Unknown", inplace=True)  
data["REMOTE\_TYPE\_NAME"].fillna("None, inplace=True")  
  
data.rename(columns={'NAICS\_2022\_6\_NAME': 'INDUSTRY'}, inplace=True)

C:\Users\avago\AppData\Local\Temp\ipykernel\_4856\1830881447.py:7: 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.  
  
  
  
C:\Users\avago\AppData\Local\Temp\ipykernel\_4856\1830881447.py:8: 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.

data.dropna(thresh=len(data) \* 0.5, axis=1, inplace=True)  
  
# Visualize missing data  
import matplotlib as mpl  
from matplotlib.colors import LinearSegmentedColormap  
  
# Set global font settings  
mpl.rcParams['font.family'] = 'Verdana'  
mpl.rcParams['font.size'] = 14  
mpl.rcParams['text.color'] = 'black'  
mpl.rcParams['axes.labelcolor'] = 'black'  
mpl.rcParams['xtick.color'] = 'black'  
mpl.rcParams['ytick.color'] = 'black'  
  
# Define custom green-to-red colormap  
custom\_cmap = LinearSegmentedColormap.from\_list(  
 'custom\_green\_red', ['#B14E53', '#78C2AD'], N=256  
)  
  
# Create the heatmap  
fig = plt.figure(figsize=(10, 6))  
ax = msno.heatmap(data, cmap=custom\_cmap)  
  
# Set the title  
plt.title("Missing Values Heatmap", fontsize=18, fontweight='bold')  
  
plt.show()  
plt.savefig("figures/missing\_values.png")

<Figure size 960x576 with 0 Axes>



<Figure size 480x384 with 0 Axes>

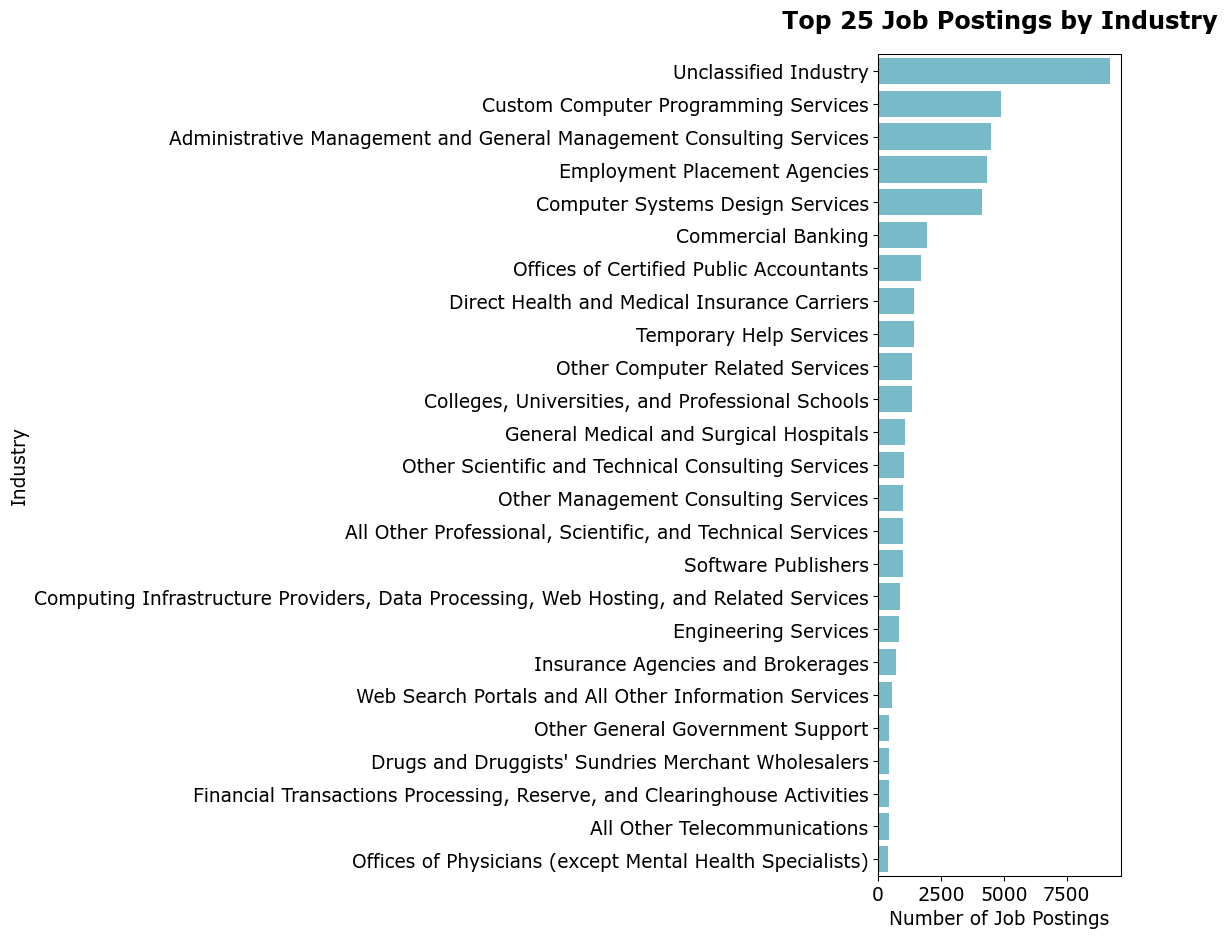
The heatmap shows a few missing values in the dataset. In this dataset most fields cluster near 1. The value 1 (dark blue) means the two columns have missing values together. 0.0 value (white) suggest there is no relationship.

## 1.4 Remove Duplicates

data = data.drop\_duplicates(subset=["TITLE", "COMPANY", "LOCATION", "POSTED"], keep="first")

## 1.5 Job Postings by Industry

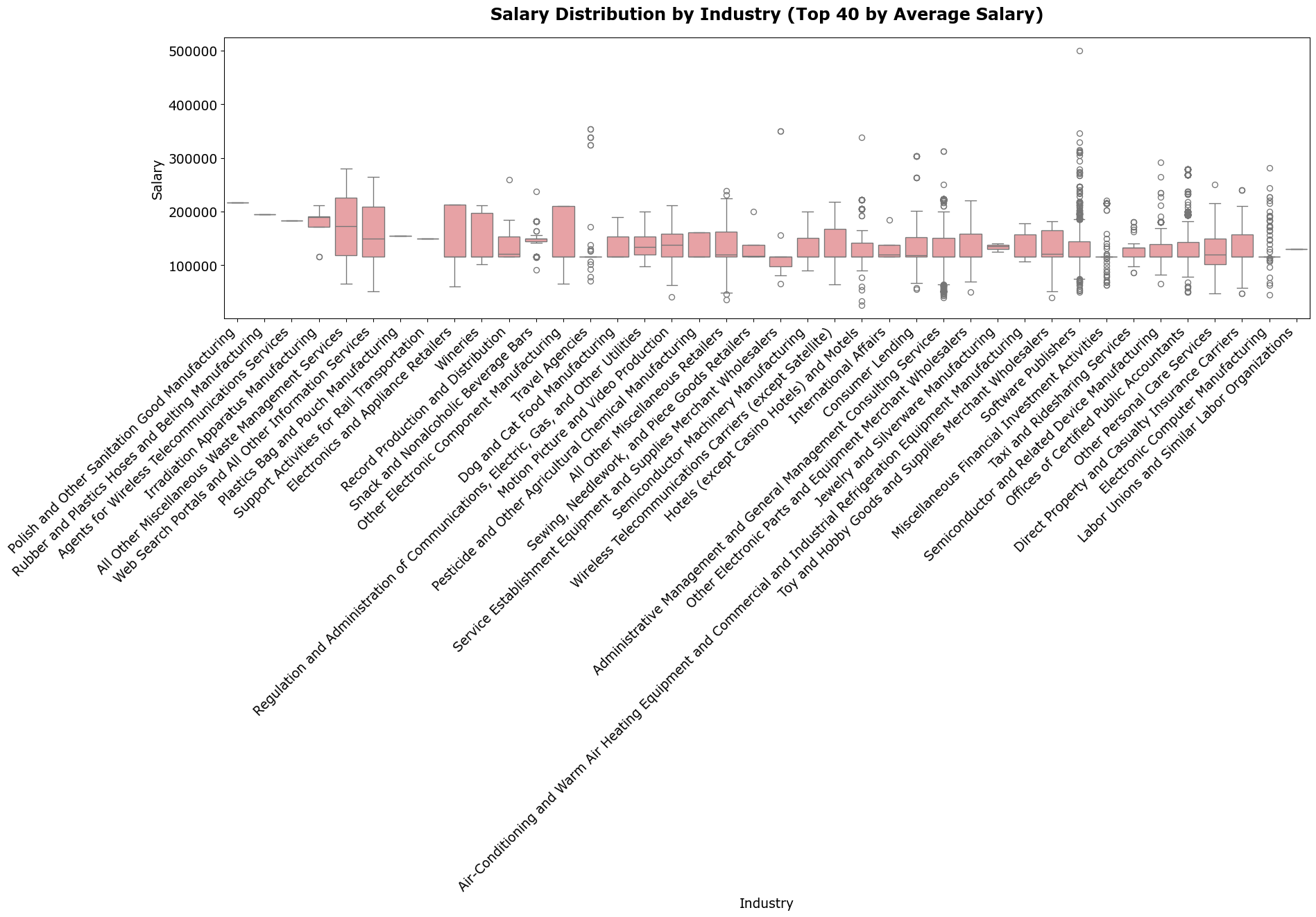
import seaborn as sns  
import matplotlib.pyplot as plt  
import matplotlib as mpl  
  
# Set global font family to Verdana  
mpl.rcParams['font.family'] = 'Verdana'  
  
# Compute posting count per industry  
data["posting\_count"] = data["ID"].groupby(data["INDUSTRY"]).transform("count")  
  
# Summarize and sort top 25 industries  
industry\_summary = data.groupby("INDUSTRY")["posting\_count"].first().sort\_values(ascending=False).head(25)  
  
# Plot  
plt.figure(figsize=(12, 10))  
sns.barplot(  
 x=industry\_summary.values,  
 y=industry\_summary.index,  
 orient='h',  
 color='#6CC3D5'  
)  
  
# Title and labels with updated font sizes  
plt.title("Top 25 Job Postings by Industry", fontsize=18, fontweight='bold', pad=20)  
plt.xlabel("Number of Job Postings", fontsize=14)  
plt.ylabel("Industry", fontsize=14)  
  
plt.tight\_layout()  
plt.show()  
plt.savefig("figures/top\_postings.png")



<Figure size 480x384 with 0 Axes>

## 1.6 Salary Distribution by Industry

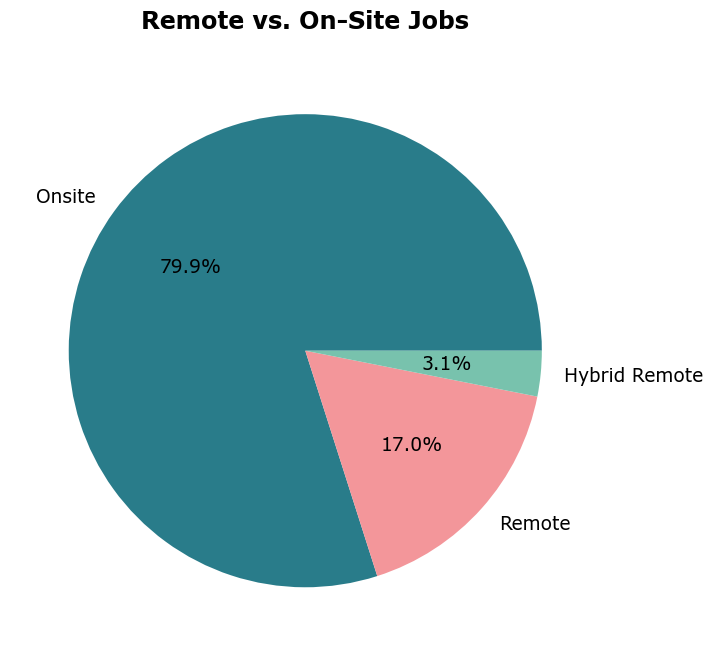
import seaborn as sns  
import matplotlib.pyplot as plt  
import matplotlib as mpl  
  
# Set global font family to Verdana  
mpl.rcParams['font.family'] = 'Verdana'  
  
# Calculate average industry salary  
data["AVERAGE\_INDUSTRY\_SALARY"] = data["SALARY"].groupby(data["INDUSTRY"]).transform("mean").round()  
  
# Get top 40 industries by average salary  
top\_40\_industries = data.groupby("INDUSTRY")["AVERAGE\_INDUSTRY\_SALARY"].first().sort\_values(ascending=False).head(40).index  
filtered\_data = data[data["INDUSTRY"].isin(top\_40\_industries)]  
  
# Create figure with taller height  
plt.figure(figsize=(20, 14)) # Increased height from 10 to 14  
  
# Create boxplot with custom color  
sns.boxplot(  
 data=filtered\_data,  
 x="INDUSTRY",  
 y="SALARY",  
 order=top\_40\_industries,  
 color='#F3969A' # Custom box color  
)  
  
# Title and labels with updated font sizes  
plt.title("Salary Distribution by Industry (Top 40 by Average Salary)", fontsize=18, fontweight='bold', pad=20)  
plt.xlabel("Industry", fontsize=14)  
plt.ylabel("Salary", fontsize=14)  
  
# Rotate x-tick labels for readability  
plt.xticks(rotation=45, ha='right')  
  
plt.tight\_layout()  
plt.show()  
  
plt.savefig("figures/salary\_dist.png")



<Figure size 480x384 with 0 Axes>

## 1.7 Remote vs. On-Site Jobs

import matplotlib.pyplot as plt  
import matplotlib as mpl  
  
# Set global font family to Verdana  
mpl.rcParams['font.family'] = 'Verdana'  
  
# Clean and group remote types  
remote\_grouped = data["REMOTE\_TYPE\_NAME"].fillna("Onsite").replace({  
 "[None]": "Onsite",   
 "Not Remote": "Onsite",  
 "None": "Onsite"  
})  
  
# Count values  
remote\_counts = remote\_grouped.value\_counts()  
  
# Define colors matching labels (make sure the order matches remote\_counts.index)  
color\_map = {  
 "Onsite": "#297C8A",  
 "Hybrid Remote": "#78C2AD",  
 "Remote": "#F3969A"  
}  
  
# Get colors for the pie chart slices in the correct order, fallback to gray if missing  
colors = [color\_map.get(label, "#cccccc") for label in remote\_counts.index]  
  
# Plot pie chart  
plt.figure(figsize=(8, 8))  
wedges, texts, autotexts = plt.pie(  
 remote\_counts.values,  
 labels=remote\_counts.index,  
 autopct='%1.1f%%',  
 colors=colors,  
 textprops={'fontsize': 14, 'color': 'black'}  
)  
  
# Title with larger font  
plt.title("Remote vs. On-Site Jobs", fontsize=18, fontweight='bold', pad=20)  
  
# Adjust autotext (percentage text) font size  
for autotext in autotexts:  
 autotext.set\_fontsize(14)  
  
plt.show()  
plt.savefig("figures/remote\_onsite.png")



<Figure size 480x384 with 0 Axes>

## 1.8 Why these visualizations were chosen

* **Bar Chart: Job Postings by Industry**
  + Makes it easy to compare job postings across different industries.
  + Provides a clear ranking that is simple to interpret.
* **Boxplot: Salary Distribution by Industry**
  + Shows medians, outliers, and summarizes salary distributions.
  + Allows for comparisons between industries and helps detect salary variability.
* **Pie Chart: Job Location Types**
  + Provides a clear visual breakdown of remote versus on-site jobs within the dataset.

## 1.9 Key insights from each graph

* **Bar Chart: Top 25 Industries by Job-Posting Volume**
  + The distribution is skewed, showing demand concentrated in technology and professional services.
  + A large portion of “unclassified industry” job postings suggests many jobs are not mapped to any industry.
* **Boxplot: Salary Distribution by Industry**
  + Salaries across the 40 industries skew high, with medians generally in the $120k–$170k range.
  + Industries with higher medians and greater dispersions include software/semiconductors and consulting.
  + Several industries have wide IQRs and many upper outliers, while some have less variability, indicating standardized pay.
* **Pie Chart: Job Location Types**
  + 17% of jobs are remote, 3.1% are hybrid, and 1.6% are not remote.
  + A large number of job postings do not specify whether the job is remote or on-site.