AI vs. Non-AI Careers

Final Report

Kelly, Sabrina, Makenzie

October 15, 2025

# 1. Introduction

We live in a time when artificial intelligence (AI) is changing the way people work. Across industries, tasks that were once done by humans are being automated, while new opportunities are appearing that require different skills. In 2024, this topic is especially important because AI tools are no longer limited to technology companies, they are now widely used in healthcare, education, energy, retail, and many other sectors. Understanding how AI affects the job market helps prepare for shifts in employment, salary trends, and the creation of new professions.

# 2. Research Rationale

We study this topic because the impact of AI on jobs is both a challenge and an opportunity for today’s workforce. Trends show that AI is displacing routine work while creating space for new, high skill roles in areas such as AI ethics, automation operations, and data science. In 2024, it is crucial to examine these changes so job seekers know which skills to build and how to adapt to market demand. We expect to find that AI continues to reshape traditional roles but also opens new careers that combine human creativity, problem solving, and technical knowledge.

# 3. Literature Review

Studies highlight that AI often leads to job shifting rather than complete job loss, with productivity gains linked to long-term job creation when training and policy support are present George ((2024)). At the same time, some occupations face real displacement risk where tasks are repetitive and routine, which increases the pressure to reskill Ansari and Ansari ((2024)).

Other scholarship points to the rise of new professions designed for an AI-driven economy examples include AI Ethics Specialists, Smart Grid Engineers, and Telemedicine Coordinators which did not exist a decade ago and are now growing rapidly Ejjami ((2024)). Recent work also shows the job market is being restructured as AI adoption shapes both the decline of certain roles and the emergence of opportunities that require continuous upskilling. Together, this literature suggests AI is transforming, not eliminating, the world of work, and future outcomes depend on how well workers, firms, and governments manage the transition.

## 3.1 References

# 4. Exploratory Data Analysis

# 5. Loading Libraries and Data

from pyspark.sql import SparkSession  
import pandas as pd  
import matplotlib.pyplot as plt  
  
raw\_df = pd.read\_csv("data/lightcast\_job\_postings.csv")  
#raw\_df.columns.tolist()

/var/folders/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/1289692877.py:5: DtypeWarning:  
  
Columns (19,30) have mixed types. Specify dtype option on import or set low\_memory=False.

# 6. Cleaning Data

columns\_to\_drop = [  
 "ID", "URL", "ACTIVE\_URLS", "DUPLICATES", "LAST\_UPDATED\_TIMESTAMP",  
 "NAICS2", "NAICS3", "NAICS4", "NAICS5", "NAICS6",  
 "SOC\_2", "SOC\_3", "SOC\_5"  
]  
raw\_df.drop(columns=columns\_to\_drop, inplace=True)  
  
# Fill missing values  
raw\_df["SALARY"].fillna(raw\_df["SALARY"].median(), inplace=True)  
raw\_df["NAICS\_2022\_6"].fillna("Unknown", inplace=True)  
  
# Drop columns with >50% missing values  
raw\_df.dropna(thresh=len(raw\_df) \* 0.5, axis=1, inplace=True)  
  
raw\_df = raw\_df.drop\_duplicates(subset=["TITLE", "COMPANY", "LOCATION", "POSTED"], keep="first")  
  
#raw\_df.head()

/var/folders/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/2470702404.py:9: 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/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/2470702404.py:10: 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/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/2470702404.py:10: FutureWarning:  
  
Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'Unknown' has dtype incompatible with float64, please explicitly cast to a compatible dtype first.

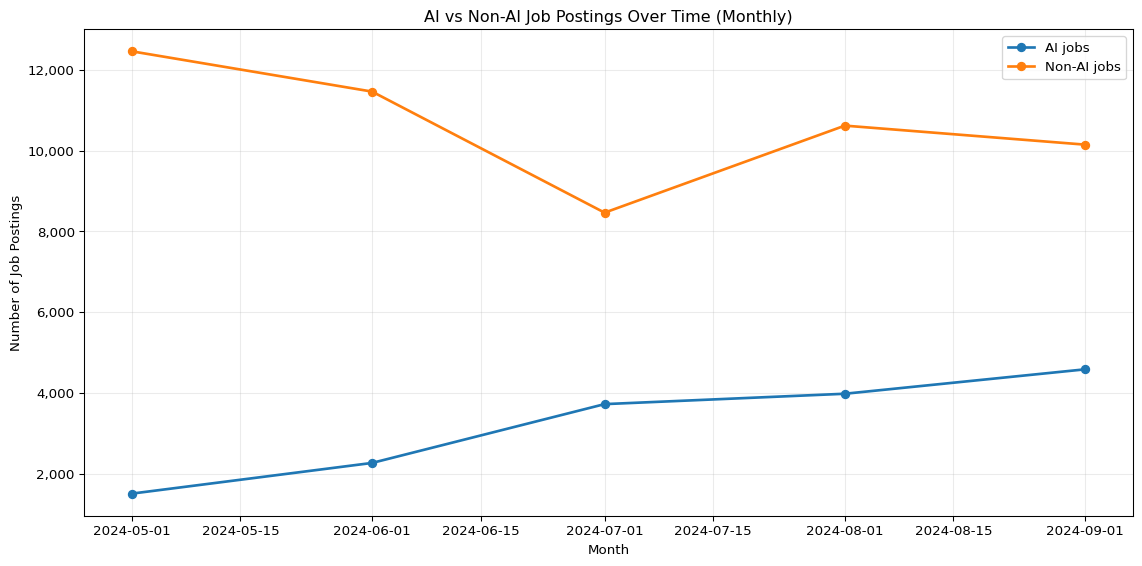
# 7. Plot setup for AI vs Non-AI job posting count

import os, re  
import pandas as pd  
  
# Kelly’s cleaned dataframe must exist  
assert "raw\_df" in globals(), "raw\_df must exist (Kelly’s cleaned dataframe)."  
df = raw\_df.copy()  
  
# Parse date -> month  
if "POSTED" not in df.columns:  
 raise ValueError("Expected a POSTED column in raw\_df.")  
df["POSTED\_DT"] = pd.to\_datetime(df["POSTED"], errors="coerce")  
df = df.dropna(subset=["POSTED\_DT"])  
df["month"] = df["POSTED\_DT"].dt.to\_period("M").dt.to\_timestamp()  
  
# Combine likely text fields  
candidate\_text\_cols = [  
 "TITLE","TITLE\_CLEAN","TITLE\_NAME","BODY",  
 "SKILLS","SKILLS\_NAME",  
 "SPECIALIZED\_SKILLS","SPECIALIZED\_SKILLS\_NAME",  
 "SOFTWARE\_SKILLS","SOFTWARE\_SKILLS\_NAME",  
 "COMMON\_SKILLS","COMMON\_SKILLS\_NAME",  
 "CERTIFICATIONS\_NAME",  
]  
text\_cols = [c for c in candidate\_text\_cols if c in df.columns]  
if not text\_cols:  
 text\_cols = ["TITLE"] if "TITLE" in df.columns else []  
text\_series = (  
 df[text\_cols].astype(str).agg(" ".join, axis=1).str.lower()  
 if text\_cols else pd.Series([""] \* len(df), index=df.index)  
)  
  
# FIXED list (no cut-off strings)  
ai\_terms = [  
 "artificial intelligence","ai","machine learning","deep learning",  
 "neural network","nlp","natural language","computer vision",  
 "reinforcement learning","generative ai","llm","gpt","chatgpt",  
 "transformer","bert","prompt engineer","prompt engineering"  
]  
  
# Word-boundary pattern so we don't match 'retail' for 'ai'  
ai\_pattern = re.compile(  
 r"\b(?:"  
 + "|".join(re.escape(t) for t in ai\_terms)  
 + r")\b",  
 flags=re.IGNORECASE,  
)  
  
df["IS\_AI"] = text\_series.str.contains(ai\_pattern, na=False)  
  
# Monthly counts  
monthly = (  
 df.groupby(["month","IS\_AI"])  
 .size()  
 .unstack(fill\_value=0)  
 .rename(columns={True: "AI", False: "Non-AI"})  
 .sort\_index()  
)  
  
monthly\_plt = monthly.copy()  
monthly\_plt.head(12)

| IS\_AI | Non-AI | AI |
| --- | --- | --- |
| month |  |  |
| 2024-05-01 | 12460 | 1503 |
| 2024-06-01 | 11462 | 2263 |
| 2024-07-01 | 8463 | 3720 |
| 2024-08-01 | 10619 | 3977 |
| 2024-09-01 | 10148 | 4582 |

# 8. Job Postings for AI vs Non-AI Jobs

import os  
import matplotlib.pyplot as plt  
from matplotlib.ticker import FuncFormatter  
  
assert "monthly\_plt" in globals(), "Run the setup chunk first."  
  
os.makedirs("output", exist\_ok=True)  
  
plt.figure(figsize=(12, 6))  
plt.plot(monthly\_plt.index, monthly\_plt["AI"], marker="o", linewidth=2, label="AI jobs")  
plt.plot(monthly\_plt.index, monthly\_plt["Non-AI"], marker="o", linewidth=2, label="Non-AI jobs")  
plt.title("AI vs Non-AI Job Postings Over Time (Monthly)")  
plt.xlabel("Month")  
plt.ylabel("Number of Job Postings")  
plt.gca().yaxis.set\_major\_formatter(FuncFormatter(lambda x, \_: f"{int(x):,}"))  
plt.grid(True, alpha=0.25)  
plt.legend()  
plt.tight\_layout()  
plt.savefig("output/ai\_vs\_nonai\_over\_time.png", dpi=200, bbox\_inches="tight")  
plt.show()



The AI VS Non-AI Jobs graph hows two clear trends:

* AI jobs have been increasing uninterruptedly each month from May to September 2024. This suggests growing demand for AI-related roles.
* Non-AI jobs started much higher but declined between May and July before slightly recovering in August. By September, they still remained lower than at the start.

In summary, there are still more Non-AI jobs than AI jobs in total; however, the number of available AI jobs is increasing rapidly. This transition indicates the shifting of the job market towards the AI-based roles.

# 9. Prep for monthly AI counts

import os, re  
import numpy as np  
import pandas as pd  
  
assert "raw\_df" in globals(), "raw\_df must exist."  
  
df = raw\_df.copy()  
  
# ---- Dates -> month ----  
df["POSTED\_DT"] = pd.to\_datetime(df["POSTED"], errors="coerce")  
df = df.dropna(subset=["POSTED\_DT"])  
df["month"] = df["POSTED\_DT"].dt.to\_period("M").dt.to\_timestamp()  
  
# ---- AI detector (reuse if already present) ----  
if "IS\_AI" not in df.columns:  
 ai\_terms = [  
 "artificial intelligence","ai","machine learning","deep learning",  
 "neural network","nlp","natural language","computer vision",  
 "reinforcement learning","generative ai","llm","gpt","chatgpt",  
 "transformer","bert","prompt engineer","prompt engineering"  
 ]  
 text\_cols = [c for c in [  
 "TITLE","TITLE\_CLEAN","BODY",  
 "SKILLS","SKILLS\_NAME",  
 "SPECIALIZED\_SKILLS","SPECIALIZED\_SKILLS\_NAME",  
 "SOFTWARE\_SKILLS","SOFTWARE\_SKILLS\_NAME",  
 "COMMON\_SKILLS","COMMON\_SKILLS\_NAME",  
 "CERTIFICATIONS\_NAME"  
 ] if c in df.columns]  
 combined = (df[text\_cols].astype(str).agg(" ".join, axis=1).str.lower()  
 if text\_cols else pd.Series([""], index=df.index))  
 pattern = re.compile(r"\b(?:%s)\b" % "|".join(re.escape(t) for t in ai\_terms), re.I)  
 df["IS\_AI"] = combined.str.contains(pattern, na=False)  
  
# ---- Pick the best available industry label ----  
ind\_candidates = [  
 "NAICS\_2022\_6\_NAME","NAICS6\_NAME",  
 "NAICS\_2022\_4\_NAME","NAICS4\_NAME",  
 "NAICS\_2022\_2\_NAME","NAICS2\_NAME",  
 "NAICS\_2022\_6","NAICS6"  
]  
IND\_COL = next((c for c in ind\_candidates if c in df.columns), None)  
if IND\_COL is None:  
 raise ValueError("No NAICS/industry name/code columns found.")  
  
# ---- Monthly AI counts per industry ----  
ai = df[df["IS\_AI"]].copy()  
ai\_monthly = (  
 ai.groupby([IND\_COL, "month"])  
 .size()  
 .reset\_index(name="count")  
)  
  
# ---- Compute growth: (last 3-mo avg - first 3-mo avg) / first 3-mo avg ----  
def growth\_row(g):  
 g = g.sort\_values("month")  
 k = min(3, len(g))  
 first = g["count"].iloc[:k].mean()  
 last = g["count"].iloc[-k:].mean()  
 total = g["count"].sum()  
 if k < 2 or first == 0:  
 return pd.Series({"growth\_pct": np.nan, "first\_avg": first, "last\_avg": last, "months": len(g), "total": total})  
 return pd.Series({"growth\_pct": (last - first) / first \* 100.0,  
 "first\_avg": first, "last\_avg": last,  
 "months": len(g), "total": total})  
  
growth\_df = ai\_monthly.groupby(IND\_COL).apply(growth\_row).reset\_index()  
  
# filter out tiny-volume industries to avoid wild % changes  
MIN\_TOTAL = 30  
growth\_df = growth\_df[growth\_df["total"] >= MIN\_TOTAL].dropna(subset=["growth\_pct"])  
  
# pick top movers (adjust top\_n)  
top\_n = 12  
growth\_top = growth\_df.sort\_values("growth\_pct", ascending=False).head(top\_n)  
  
# Save a copy if want a table in the doc  
growth\_top\_rounded = growth\_top.copy()  
growth\_top\_rounded["growth\_pct"] = growth\_top\_rounded["growth\_pct"].round(1)  
growth\_top\_rounded.head(top\_n)

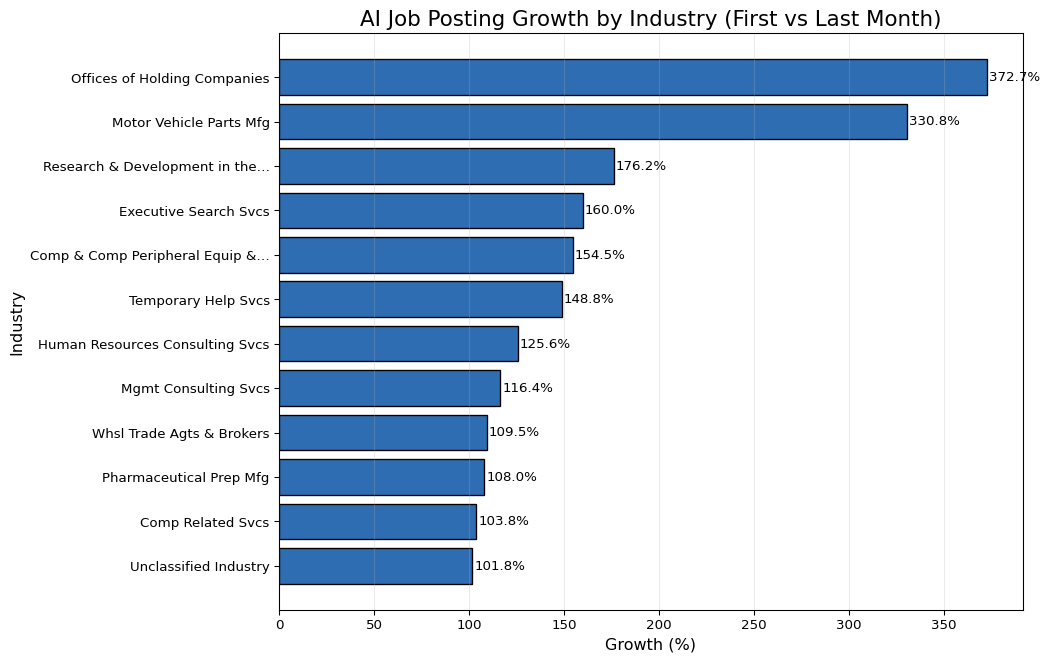
/var/folders/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/672499099.py:67: DeprecationWarning:  
  
DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include\_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.

|  | NAICS\_2022\_6\_NAME | growth\_pct | first\_avg | last\_avg | months | total |
| --- | --- | --- | --- | --- | --- | --- |
| 260 | Offices of Other Holding Companies | 372.7 | 3.666667 | 17.333333 | 5.0 | 56.0 |
| 300 | Other Motor Vehicle Parts Manufacturing | 330.8 | 4.333333 | 18.666667 | 5.0 | 68.0 |
| 358 | Research and Development in the Physical, Engi... | 176.2 | 14.000000 | 38.666667 | 5.0 | 133.0 |
| 137 | Executive Search Services | 160.0 | 6.666667 | 17.333333 | 5.0 | 59.0 |
| 80 | Computer and Computer Peripheral Equipment and... | 154.5 | 3.666667 | 9.333333 | 5.0 | 33.0 |
| 406 | Temporary Help Services | 148.8 | 42.333333 | 105.333333 | 5.0 | 352.0 |
| 189 | Human Resources Consulting Services | 125.6 | 13.000000 | 29.333333 | 5.0 | 96.0 |
| 296 | Other Management Consulting Services | 116.4 | 46.666667 | 101.000000 | 5.0 | 337.0 |
| 427 | Wholesale Trade Agents and Brokers | 109.5 | 7.000000 | 14.666667 | 5.0 | 52.0 |
| 325 | Pharmaceutical Preparation Manufacturing | 108.0 | 8.333333 | 17.333333 | 5.0 | 62.0 |
| 280 | Other Computer Related Services | 103.8 | 70.666667 | 144.000000 | 5.0 | 509.0 |
| 415 | Unclassified Industry | 101.8 | 280.666667 | 566.333333 | 5.0 | 2025.0 |

In this table, we can see AI job postings growing across many industries. A few sectors jump fast but are small, like Other Holding Companies and Motor Vehicle Parts, and sound signals, but the totals are low, so that they may swing. The strongest, faster growth in real volume is in Temporary Help Services, Other Management Consulting, and Other Computer-Related Services; their monthly averages more than doubled. R&D and Executive Search are up too, with mid totals. There is also a growing group called “Unclassified.” This group includes various categories, so we need to use this data carefully.

# 10. AI-Driven Job Growth by Industry

# --- Short, readable labels + plot & save (Option B) -------------------------  
import os, re  
from textwrap import shorten  
import matplotlib.pyplot as plt  
  
# Use existing industry column if defined; otherwise default:  
IND\_COL = IND\_COL if "IND\_COL" in globals() else "NAICS\_2022\_6\_NAME"  
  
def short\_label(s: str) -> str:  
 s = str(s)  
 # Common abbreviations to keep labels compact but clear  
 s = re.sub(r'(?i)\bservices?\b', 'Svcs', s)  
 s = re.sub(r'(?i)\bmanufacturing\b', 'Mfg', s)  
 s = re.sub(r'(?i)\bpreparation\b', 'Prep', s)  
 s = re.sub(r'(?i)\bmanagement\b', 'Mgmt', s)  
 s = re.sub(r'(?i)\bcomputer\b', 'Comp', s)  
 s = re.sub(r'(?i)\bequipment\b', 'Equip', s)  
 s = re.sub(r'(?i)\bwholesale\b', 'Whsl', s)  
 s = re.sub(r'(?i)\bagents?\b', 'Agts', s)  
 s = re.sub(r'(?i)\bscientific\b', 'Sci', s)  
 s = re.sub(r'(?i)\bengineering\b', 'Eng', s)  
 s = re.sub(r'(?i)\band\b', '&', s)  
 s = re.sub(r'(?i)\bother\b\s\*', '', s) # drop leading "Other"  
 # Final safe trim  
 return shorten(s.strip(), width=32, placeholder='…')  
  
plot\_df = growth\_top.copy()  
plot\_df["label"] = plot\_df[IND\_COL].astype(str).map(short\_label)  
  
fig, ax = plt.subplots(figsize=(11, 7))  
bars = ax.barh(plot\_df["label"], plot\_df["growth\_pct"], color="#2F6DB3", edgecolor="black")  
ax.invert\_yaxis() # biggest at top  
  
ax.set\_title("AI Job Posting Growth by Industry (First vs Last Month)", fontsize=16)  
ax.set\_xlabel("Growth (%)", fontsize=12)  
ax.set\_ylabel("Industry", fontsize=12)  
ax.grid(axis="x", alpha=0.25)  
  
# Value labels  
for b in bars:  
 w = b.get\_width()  
 ax.text(  
 w + (1 if w >= 0 else -1),  
 b.get\_y() + b.get\_height() / 2,  
 f"{w:.1f}%",  
 va="center",  
 ha="left" if w >= 0 else "right",  
 fontsize=10,  
 )  
  
plt.tight\_layout()  
os.makedirs("output", exist\_ok=True)  
plt.savefig("output/ai\_industry\_growth\_shortlabels.png", dpi=200, bbox\_inches="tight")  
plt.show()



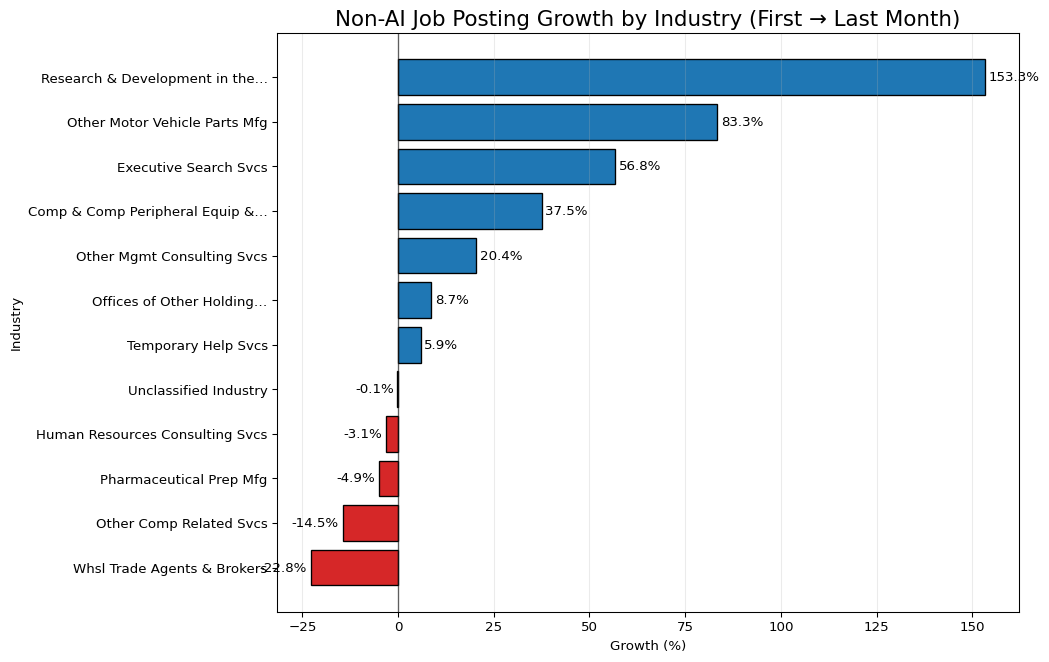
The chart shows that AI job postings are increasing quickly in many industries. The largest increases are in Offices of Holding Companies (about 373%) and Motor Vehicle Parts Manufacturing (about 331%). We also see significant growth in R&D (physical/engineering) (about 176%), Executive Search (about 160%), Computer & Peripheral Equipment (about 155%), and Temporary Help Services (about 149%). Other sectors like HR Consulting, Management Consulting, Wholesale Trade Agents, Pharmaceutical Preparation, Other Computer Related, and Unclassified categories enlarged by slightly more than 100% all together. The growth is mainly distributed among the different sectors, while only some industries exhibit the highest increases.

# 11. Non-AI Job Posting Growth by Industry

import os, re  
import pandas as pd  
from textwrap import shorten  
  
assert "raw\_df" in globals(), "raw\_df must exist (cleaned dataframe)."  
df = raw\_df.copy()  
  
# Pick an industry-name column that exists  
IND\_COL = next((c for c in [  
 "NAICS\_2022\_6\_NAME","NAICS6\_NAME","NAICS\_2022\_4\_NAME","NAICS4\_NAME",  
 "NAICS\_2022\_2\_NAME","NAICS2\_NAME"  
] if c in df.columns), None)  
assert IND\_COL, "No industry/NAICS name column found."  
  
TITLE\_COL = "TITLE\_CLEAN" if "TITLE\_CLEAN" in df.columns else "TITLE"  
assert TITLE\_COL in df.columns, "Need a TITLE or TITLE\_CLEAN column."  
  
# Dates → month  
df["POSTED\_DT"] = pd.to\_datetime(df["POSTED"], errors="coerce")  
df = df.dropna(subset=["POSTED\_DT", IND\_COL])  
df["month"] = df["POSTED\_DT"].dt.to\_period("M").dt.to\_timestamp()  
  
# Detect AI vs non-AI via title keywords  
AI\_TERMS = [  
 "artificial intelligence", r"\bAI\b", "machine learning", r"\bML\b",  
 "deep learning", r"\bLLM\b", r"\bNLP\b", "computer vision",  
 "generative", "chatgpt", r"gpt-\d+", "transformer", "bert",  
 "prompt engineer", "reinforcement learning"  
]  
ai\_pat = re.compile("|".join(AI\_TERMS), flags=re.IGNORECASE)  
if "is\_ai" not in df.columns:  
 df["is\_ai"] = df[TITLE\_COL].astype(str).str.contains(ai\_pat, na=False)  
  
# Use industries from AI-growth visual if available; otherwise pick top 10 by AI postings last month  
if "growth\_top" in globals():  
 target\_inds = [str(x) for x in growth\_top[IND\_COL].dropna().tolist()]  
else:  
 last\_m = df["month"].max()  
 ai\_last = (df[df["is\_ai"] & (df["month"] == last\_m)]  
 .groupby(IND\_COL).size().sort\_values(ascending=False).head(10))  
 target\_inds = ai\_last.index.astype(str).tolist()  
  
# Build non-AI growth (% first→last month) for those industries  
non\_ai = df[(~df["is\_ai"]) & (df[IND\_COL].astype(str).isin(target\_inds))].copy()  
monthly = (non\_ai.groupby([IND\_COL, "month"]).size().reset\_index(name="count"))  
  
first\_last = (monthly.sort\_values("month")  
 .groupby(IND\_COL, as\_index=False)  
 .agg(first=("count","first"), last=("count","last")))  
first\_last = first\_last[first\_last["first"] > 0].copy()  
first\_last["growth\_pct"] = ((first\_last["last"] - first\_last["first"])  
 / first\_last["first"]) \* 100.0  
first\_last = first\_last[first\_last[IND\_COL].astype(str).isin(target\_inds)]  
  
def short\_label(s: str) -> str:  
 s = str(s)  
 s = re.sub(r'(?i)\bservices?\b', 'Svcs', s)  
 s = re.sub(r'(?i)\bmanufacturing\b', 'Mfg', s)  
 s = re.sub(r'(?i)\bmanagement\b', 'Mgmt', s)  
 s = re.sub(r'(?i)\bpreparation\b', 'Prep', s)  
 s = re.sub(r'(?i)\bcomputer\b', 'Comp', s)  
 s = re.sub(r'(?i)\bequipment\b', 'Equip', s)  
 s = re.sub(r'(?i)\bwholesale\b', 'Whsl', s)  
 s = re.sub(r'(?i)\band\b', '&', s)  
 s = re.sub(r'\s+', ' ', s).strip()  
 return shorten(s, width=32, placeholder='…')  
  
non\_ai\_growth\_plot\_df = (first\_last  
 .sort\_values("growth\_pct", ascending=False)  
 .assign(label=lambda d: d[IND\_COL].map(short\_label)))  
  
os.makedirs("output", exist\_ok=True)  
non\_ai\_growth\_plot\_df.head(10) # preview table

|  | NAICS\_2022\_6\_NAME | first | last | growth\_pct | label |
| --- | --- | --- | --- | --- | --- |
| 8 | Research and Development in the Physical, Engi... | 45 | 114 | 153.333333 | Research & Development in the… |
| 6 | Other Motor Vehicle Parts Manufacturing | 12 | 22 | 83.333333 | Other Motor Vehicle Parts Mfg |
| 1 | Executive Search Services | 37 | 58 | 56.756757 | Executive Search Svcs |
| 0 | Computer and Computer Peripheral Equipment and... | 16 | 22 | 37.500000 | Comp & Comp Peripheral Equip &… |
| 5 | Other Management Consulting Services | 162 | 195 | 20.370370 | Other Mgmt Consulting Svcs |
| 3 | Offices of Other Holding Companies | 23 | 25 | 8.695652 | Offices of Other Holding… |
| 9 | Temporary Help Services | 272 | 288 | 5.882353 | Temporary Help Svcs |
| 10 | Unclassified Industry | 2028 | 2025 | -0.147929 | Unclassified Industry |
| 2 | Human Resources Consulting Services | 32 | 31 | -3.125000 | Human Resources Consulting Svcs |
| 7 | Pharmaceutical Preparation Manufacturing | 61 | 58 | -4.918033 | Pharmaceutical Prep Mfg |

import os  
import matplotlib.pyplot as plt  
  
# If prep didn't run in this kernel, rebuild quickly  
if "non\_ai\_growth\_plot\_df" not in globals():  
 # Re-run a minimal rebuild using the same logic  
 assert "raw\_df" in globals(), "raw\_df is required to rebuild plot data."  
 # execute the prep cell above; keeping this here for resilience:  
 # (We simply import the name if it exists; otherwise advise to run prep.)  
 raise AssertionError("Run the prep chunk first to create non\_ai\_growth\_plot\_df.")  
  
dfp = non\_ai\_growth\_plot\_df.copy()  
  
if dfp.empty:  
 print("non\_ai\_growth\_plot\_df is empty—nothing to plot. "  
 "Check that selected industries have non-AI rows in the first/last month.")  
else:  
 os.makedirs("output", exist\_ok=True)  
 colors = dfp["growth\_pct"].ge(0).map({True: "#1f77b4", False: "#d62728"}).to\_numpy()  
  
 fig, ax = plt.subplots(figsize=(11, 7))  
 bars = ax.barh(dfp["label"], dfp["growth\_pct"], color=colors, edgecolor="black")  
  
 ax.axvline(0, color="black", linewidth=1, alpha=0.6)  
 ax.invert\_yaxis()  
 ax.set\_title("Non-AI Job Posting Growth by Industry (First → Last Month)", fontsize=16)  
 ax.set\_xlabel("Growth (%)")  
 ax.set\_ylabel("Industry")  
 ax.grid(axis="x", alpha=0.25)  
  
 for b, v in zip(bars, dfp["growth\_pct"]):  
 ax.text(  
 v + (1 if v >= 0 else -1),  
 b.get\_y() + b.get\_height() / 2,  
 f"{v:.1f}%",  
 va="center",  
 ha="left" if v >= 0 else "right",  
 fontsize=10,  
 )  
  
 plt.tight\_layout()  
 plt.savefig("output/non\_ai\_growth\_pct\_in\_ai\_growth\_industries\_colored.png",  
 dpi=200, bbox\_inches="tight")  
 plt.show()



The Non-AI job posting growth chart shows the following:

* The biggest gains are in Research & Development (~153%) and Motor Vehicle Parts (~83%). Executive Search (~57%) and Computer/Peripheral Equipment (~38%) also grew well. Management Consulting (~20%) is up, while Offices of Holding Companies (~9%) and Temporary Help (~6%) show small growth.
* A few areas are slipping: Wholesale Trade Agents & Brokers (~-23%), Other Computer-Related Services (~-15%), Pharmaceutical Prep (~-5%), and HR Consulting (~-3%) declined; Unclassified is flat.

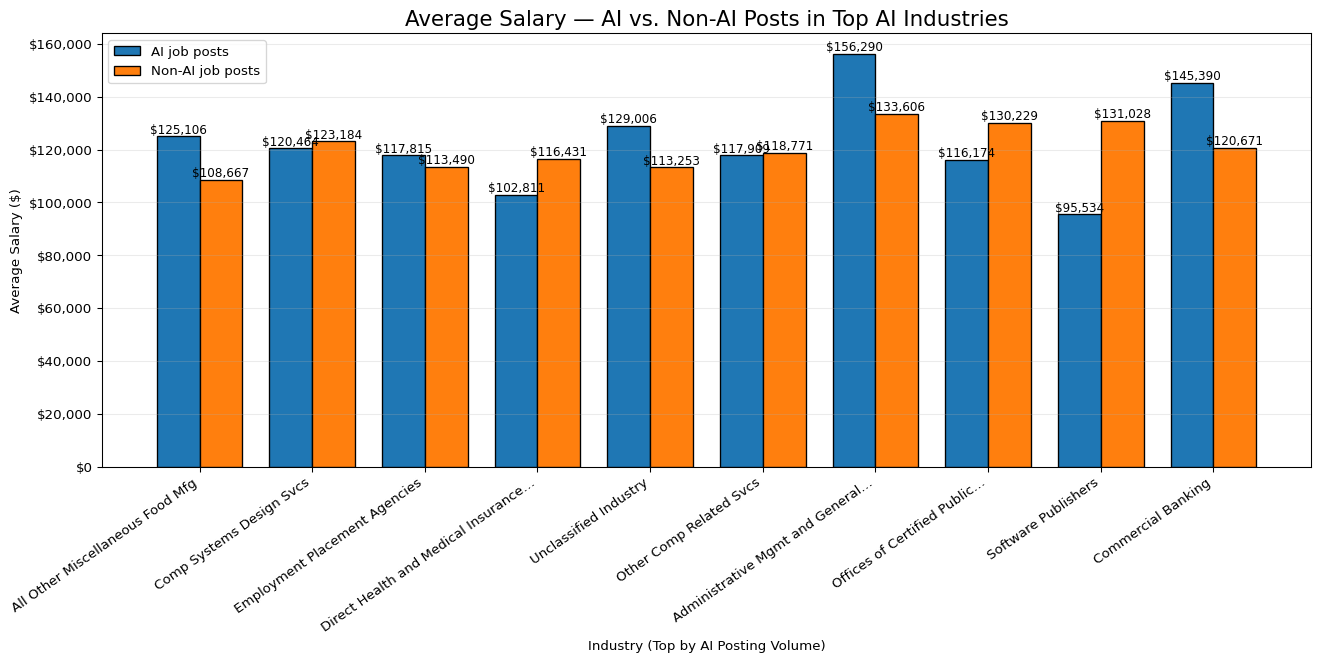
This means for us that non-AI roles are still growing, just not as fast as AI roles. The non-AI opportunities we want more of should be targeted to R&D labs, automotive suppliers, executive search firms, and hardware/peripherals. Outreach to these fields will guarantee more non-AI opportunities. The areas of wholesale, general computer, and pharma-related services, HR consulting would be where one could select with more discrimination since the demand for these practices is on the decline.

# 12. AI-powered roles vs non-ai roles salary prep

import os, re  
import numpy as np  
import pandas as pd  
  
assert "raw\_df" in globals(), "raw\_df (cleaned dataframe) must exist."  
  
df = raw\_df.copy()  
  
# --- Pick columns robustly ---  
TITLE\_COL = "TITLE\_CLEAN" if "TITLE\_CLEAN" in df.columns else "TITLE"  
assert TITLE\_COL in df.columns, "Need a TITLE or TITLE\_CLEAN column."  
  
IND\_COL = next((c for c in [  
 "NAICS\_2022\_6\_NAME","NAICS6\_NAME","NAICS\_2022\_4\_NAME","NAICS4\_NAME",  
 "NAICS\_2022\_2\_NAME","NAICS2\_NAME","LIGHTCAST\_SECTORS\_NAME"  
] if c in df.columns), None)  
assert IND\_COL, "No industry/NAICS name column found."  
  
# Salary: prefer SALARY; otherwise average of FROM/TO if present  
sal = pd.to\_numeric(df.get("SALARY", np.nan), errors="coerce")  
if sal.isna().all() and {"SALARY\_FROM","SALARY\_TO"} <= set(df.columns):  
 s\_from = pd.to\_numeric(df["SALARY\_FROM"], errors="coerce")  
 s\_to = pd.to\_numeric(df["SALARY\_TO"], errors="coerce")  
 sal = (s\_from + s\_to) / 2  
df["salary\_num"] = pd.to\_numeric(sal, errors="coerce")  
  
# keep positive salaries only  
df = df[df["salary\_num"] > 0].copy()  
  
# Optional: cap extreme outliers so a few posts don't dominate the mean  
q\_low, q\_hi = df["salary\_num"].quantile([0.01, 0.99])  
df["salary\_num"] = df["salary\_num"].clip(q\_low, q\_hi)  
  
# --- Tag AI vs non-AI using title keywords ---  
AI\_TERMS = [  
 "artificial intelligence", r"\bAI\b", "machine learning", r"\bML\b",  
 "deep learning", r"\bLLM\b", r"\bNLP\b", "computer vision",  
 "generative", "chatgpt", r"gpt-\d+", "transformer", r"\bbert\b",  
 "prompt engineer", "reinforcement learning", "data scientist", "ai engineer"  
]  
ai\_pat = re.compile("|".join(AI\_TERMS), flags=re.IGNORECASE)  
df["is\_ai"] = df[TITLE\_COL].astype(str).str.contains(ai\_pat, na=False)  
  
# --- Find top industries by AI posting count (choose N)  
N = 10  
ai\_counts = (df[df["is\_ai"]]  
 .groupby(IND\_COL, dropna=True)  
 .size()  
 .sort\_values(ascending=False)  
 .head(N)  
 .rename("ai\_posts")  
 .reset\_index())  
top\_inds = ai\_counts[IND\_COL].astype(str).tolist()  
  
# --- Compute average salary (AI vs non-AI) for those industries  
subset = df[df[IND\_COL].astype(str).isin(top\_inds)].copy()  
grp = subset.groupby([IND\_COL, "is\_ai"])["salary\_num"].agg(["mean", "count"]).reset\_index()  
  
ai\_part = grp[grp["is\_ai"]].rename(columns={"mean":"avg\_ai\_salary", "count":"ai\_postings"})  
non\_part = grp[~grp["is\_ai"]].rename(columns={"mean":"avg\_nonai\_salary", "count":"non\_ai\_postings"})  
  
ai\_vs\_trad\_industry\_salary = (  
 ai\_counts[[IND\_COL, "ai\_posts"]] # preserves AI-based ordering  
 .merge(ai\_part[[IND\_COL, "avg\_ai\_salary", "ai\_postings"]], on=IND\_COL, how="left")  
 .merge(non\_part[[IND\_COL, "avg\_nonai\_salary", "non\_ai\_postings"]], on=IND\_COL, how="left")  
)  
  
# Clean up & preview  
ai\_vs\_trad\_industry\_salary = ai\_vs\_trad\_industry\_salary.fillna(0)  
os.makedirs("output", exist\_ok=True)  
ai\_vs\_trad\_industry\_salary.head(N)

|  | NAICS\_2022\_6\_NAME | ai\_posts | avg\_ai\_salary | ai\_postings | avg\_nonai\_salary | non\_ai\_postings |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | All Other Miscellaneous Food Manufacturing | 124 | 125106.451613 | 124 | 108667.836735 | 49 |
| 1 | Computer Systems Design Services | 81 | 120464.666667 | 81 | 123184.301252 | 4073 |
| 2 | Employment Placement Agencies | 72 | 117815.555556 | 72 | 113490.867107 | 4244 |
| 3 | Direct Health and Medical Insurance Carriers | 67 | 102811.940299 | 67 | 116431.746753 | 1386 |
| 4 | Unclassified Industry | 57 | 129006.070175 | 57 | 113253.334609 | 9148 |
| 5 | Other Computer Related Services | 53 | 117909.245283 | 53 | 118771.586707 | 1309 |
| 6 | Administrative Management and General Manageme... | 52 | 156290.384615 | 52 | 133606.945132 | 4447 |
| 7 | Offices of Certified Public Accountants | 48 | 116174.354167 | 48 | 130229.120365 | 1645 |
| 8 | Software Publishers | 35 | 95534.285714 | 35 | 131028.541624 | 973 |
| 9 | Commercial Banking | 34 | 145390.588235 | 34 | 120671.148515 | 1919 |

import os, re  
import matplotlib.pyplot as plt  
from textwrap import shorten  
from matplotlib.ticker import FuncFormatter  
  
assert "ai\_vs\_trad\_industry\_salary" in globals(), "Run the prep chunk first."  
  
dfp = ai\_vs\_trad\_industry\_salary.copy()  
if dfp.empty:  
 print("No data to plot after filtering. Check AI detection or industry column.")  
else:  
 # shorten very long industry labels  
 def short\_label(s: str) -> str:  
 s = str(s)  
 s = re.sub(r'(?i)\bservices?\b', 'Svcs', s)  
 s = re.sub(r'(?i)\bmanufacturing\b', 'Mfg', s)  
 s = re.sub(r'(?i)\bmanagement\b', 'Mgmt', s)  
 s = re.sub(r'(?i)\bcomputer\b', 'Comp', s)  
 s = re.sub(r'(?i)\bequipment\b', 'Equip', s)  
 s = re.sub(r'\s+', ' ', s).strip()  
 return shorten(s, width=36, placeholder='…')  
  
 dfp["label"] = dfp[IND\_COL].map(short\_label)  
  
 x = range(len(dfp))  
 w = 0.38  
  
 fig, ax = plt.subplots(figsize=(14, 7))  
 b1 = ax.bar([i - w/2 for i in x], dfp["avg\_ai\_salary"], width=w, label="AI job posts", edgecolor="black")  
 b2 = ax.bar([i + w/2 for i in x], dfp["avg\_nonai\_salary"], width=w, label="Non-AI job posts", edgecolor="black")  
  
 ax.set\_title("Average Salary — AI vs. Non-AI Posts in Top AI Industries", fontsize=16)  
 ax.set\_xlabel("Industry (Top by AI Posting Volume)")  
 ax.set\_ylabel("Average Salary ($)")  
 ax.set\_xticks(list(x))  
 ax.set\_xticklabels(dfp["label"], rotation=35, ha="right")  
 ax.yaxis.set\_major\_formatter(FuncFormatter(lambda v, pos: f"${int(v):,}"))  
 ax.grid(axis="y", alpha=0.25)  
 ax.legend()  
  
 # annotate bars  
 for bars in (b1, b2):  
 for p in bars:  
 h = p.get\_height()  
 if h > 0:  
 ax.text(p.get\_x()+p.get\_width()/2, h, f"${int(h):,}",  
 ha="center", va="bottom", fontsize=9)  
  
 plt.tight\_layout()  
 os.makedirs("output", exist\_ok=True)  
 plt.savefig("output/ai\_vs\_nonai\_avg\_salary\_by\_industry.png", dpi=200, bbox\_inches="tight")  
 plt.show()



In this graph, we compared pay for AI and non-AI roles across industries. In most of these sectors, AI jobs pay more; significant gaps show up in Administrative Management & General Services, Commercial Banking, Unclassified, and Food Manufacturing. A few places flip the pattern: Software Publishers, Offices of Certified Public Accountants, Health/Medical Insurance, and Computer Systems Design show higher pay for non-AI roles (or pay that is almost the same in Other Computer-Related Services). For our search, we should target AI roles in industries with clear premium pay, and be careful in the few sectors where AI pay trails non-AI; those roles may be more junior or support.

# 13. Market Takeaways from our EDA

AI does not affect every field the same. From our EDA, the fields that use AI consulting, computer services, R&D, and parts of banking show rising postings and often higher pay, so jobs look safer there. In contrast, areas with flat or falling non-AI postings, wholesale trade agents, some general computer services, pharma prep, and HR consulting look less secure unless they adopt AI tools. Routine, repeat tasks are most at risk, while roles that mix tech and people skills (analysis, client work, project work) look safer.

Looking across industries, we see growth where AI is being put to work: other computer-related services, management consulting, temporary help services, R&D, and commercial banking. Meanwhile, we see possible displacement or slowdown on the non-AI side in wholesale trade agents, parts of computer services, pharma prep, and HR consulting. In short, sectors leaning into AI grow; those that don’t may shrink or re-scope roles.

Compared with traditional paths like mechanical engineering, farming, and retail, AI-powered roles are growing faster and usually pay more, according to our charts. Traditional fields still matter, but many tasks there now use AI (e.g., predictive maintenance in mechanical, precision ag in farming, demand forecasting in retail). People who add data and AI skills to their core trade tend to get better options and higher pay.

Finally, we see new job titles emerging: AI/ML Engineer, MLOps/AI Ops, AI Product Manager, Prompt Engineer, Data/ML Analyst, AI Solutions Consultant, and AI Ethics/Governance, along with more analytics-heavy PM and consulting roles. Overall, where AI is adopted, jobs shift toward analysis, tools, and delivery, and hiring grows. Where AI is ignored, roles risk shrinking. Our best move is to build AI and delivery skills (CS basics, operations, project management) and target the growing sectors.

# 14. Skill Gap Analysis

# 15. Loading Libraries and Data

from pyspark.sql import SparkSession  
import pandas as pd  
import matplotlib.pyplot as plt  
  
raw\_df = pd.read\_csv("data/lightcast\_job\_postings.csv")  
#raw\_df.columns.tolist()

/var/folders/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/1289692877.py:5: DtypeWarning:  
  
Columns (19,30) have mixed types. Specify dtype option on import or set low\_memory=False.

# 16. Cleaning Data

columns\_to\_drop = [  
 "ID", "URL", "ACTIVE\_URLS", "DUPLICATES", "LAST\_UPDATED\_TIMESTAMP",  
 "NAICS2", "NAICS3", "NAICS4", "NAICS5", "NAICS6",  
 "SOC\_2", "SOC\_3", "SOC\_5"  
]  
raw\_df.drop(columns=columns\_to\_drop, inplace=True)  
  
# Fill missing values  
raw\_df["SALARY"].fillna(raw\_df["SALARY"].median(), inplace=True)  
raw\_df["NAICS\_2022\_6"].fillna("Unknown", inplace=True)  
  
# Drop columns with >50% missing values  
raw\_df.dropna(thresh=len(raw\_df) \* 0.5, axis=1, inplace=True)  
  
raw\_df = raw\_df.drop\_duplicates(subset=["TITLE", "COMPANY", "LOCATION", "POSTED"], keep="first")  
  
#raw\_df.head()

/var/folders/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/2470702404.py:9: 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/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/2470702404.py:10: 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/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/2470702404.py:10: FutureWarning:  
  
Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'Unknown' has dtype incompatible with float64, please explicitly cast to a compatible dtype first.

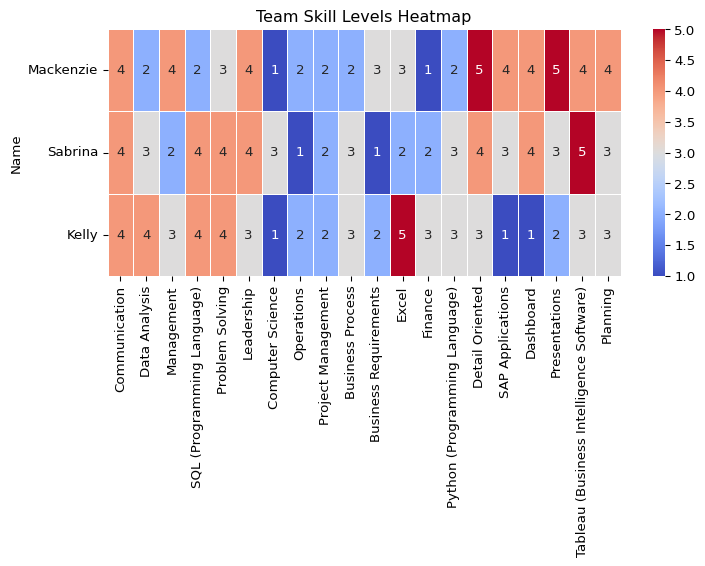
# 17. Create a team-based skill dataframe

Use Scale (1-5) to indicate proficiency levels for each team Member:

* 1 = Beginner
* 2 = Basic knowledge
* 3 = Intermediate
* 4 = Advanced
* 5 = Expert

**Note:** We build the team skill dataframe based on our skills and the most in-demand IT skills and set 1-5 levels for each team member. This allows us to compare side by side in the comparation analysis below.

import pandas as pd  
  
skills\_data = {  
 "Name": ["Mackenzie", "Sabrina", "Kelly"],  
 "Communication": [4, 4, 4],  
 "Data Analysis": [2, 3, 4],  
 "Management": [4, 2, 3],  
 "SQL (Programming Language)": [2, 4, 4],  
 "Problem Solving": [3, 4, 4],  
 "Leadership": [4, 4, 3],  
 "Computer Science": [1, 3, 1],  
 "Operations": [2, 1, 2],  
 "Project Management": [2, 2, 2],  
 "Business Process": [2, 3, 3],  
 "Business Requirements": [3, 1, 2],  
 "Excel": [3, 2, 5],  
 "Finance": [1, 2, 3],  
 "Python (Programming Language)": [2, 3, 3],  
 "Detail Oriented": [5, 4, 3],  
 "SAP Applications": [4, 3, 1],  
 "Dashboard": [4, 4, 1],  
 "Presentations": [5, 3, 2],  
 "Tableau (Business Intelligence Software)": [4, 5, 3],  
 "Planning": [4, 3, 3]  
}  
  
df\_skills = pd.DataFrame(skills\_data)  
df\_skills.set\_index("Name", inplace=True)  
df\_skills  
  
#--- Plot -----  
  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(8, 6))  
sns.heatmap(df\_skills, annot=True, cmap="coolwarm", linewidths=0.5, vmin=1, vmax=5)  
plt.title("Team Skill Levels Heatmap")  
plt.yticks(rotation=0) # keep names horizontal  
plt.tight\_layout()  
plt.savefig("output/team\_skills\_heatmap.png", dpi=300)  
plt.show()  
plt.close()



The team skills levels headmap demostrated we are strogest in communication and problem solving. We also solid in SQL, dashboard /Tableau, and finance. Our weak spots are Computer Science, operations, project management, and writing clear business requiments and process steps.

## 17.1 Top strengths per person\*\*

top\_strengths = df\_skills.apply(lambda row: row[row == row.max()].index.tolist(), axis=1)  
top\_strengths.to\_frame(name="Top Strength Skills")

|  | Top Strength Skills |
| --- | --- |
| Name |  |
| Mackenzie | [Detail Oriented, Presentations] |
| Sabrina | [Tableau (Business Intelligence Software)] |
| Kelly | [Excel] |

## 17.2 Team averages by skill

team\_avg = df\_skills.mean().sort\_values(ascending=False)  
team\_avg.to\_frame(name="Team Average (1–5)")

|  | Team Average (1–5) |
| --- | --- |
| Communication | 4.000000 |
| Tableau (Business Intelligence Software) | 4.000000 |
| Detail Oriented | 4.000000 |
| Problem Solving | 3.666667 |
| Leadership | 3.666667 |
| Excel | 3.333333 |
| Presentations | 3.333333 |
| Planning | 3.333333 |
| SQL (Programming Language) | 3.333333 |
| Data Analysis | 3.000000 |
| Dashboard | 3.000000 |
| Management | 3.000000 |
| Business Process | 2.666667 |
| Python (Programming Language) | 2.666667 |
| SAP Applications | 2.666667 |
| Project Management | 2.000000 |
| Finance | 2.000000 |
| Business Requirements | 2.000000 |
| Operations | 1.666667 |
| Computer Science | 1.666667 |

# 18. Compare team skills to industry requirements

## 18.1 Extract the Most In-Demand Skills from IT Job Postings

from collections import Counter  
import ast  
import pandas as pd  
  
# Parse SKILLS\_NAME into Python lists  
raw\_df["SKILLS\_NAME"] = raw\_df["SKILLS\_NAME"].apply(  
 lambda x: ast.literal\_eval(x) if isinstance(x, str) and x.strip().startswith("[") else (x if isinstance(x, list) else [])  
)  
  
# rename things so similar ones match and stay consistent  
  
alias\_map = {  
 "SQL": "SQL (Programming Language)",  
 "Sql": "SQL (Programming Language)",  
 "MS Excel": "Excel",  
 "Microsoft Excel": "Excel",  
 "PowerBI": "Power BI",  
}  
def canon\_skill(s: str) -> str:  
 s = s.strip()  
 return alias\_map.get(s, s)  
  
# 3) Combines and count  
all\_skills = [canon\_skill(s) for sublist in raw\_df["SKILLS\_NAME"] for s in (sublist if isinstance(sublist, list) else []) if isinstance(s, str)]  
skill\_counts = Counter([s for s in all\_skills if s])  
  
# 4) Top Skills   
top\_skills = [skill for skill, \_ in skill\_counts.most\_common(20)]  
print("Top skills (dataset):", top\_skills)

Top skills (dataset): ['Communication', 'Data Analysis', 'Management', 'SQL (Programming Language)', 'Problem Solving', 'Leadership', 'Computer Science', 'Operations', 'Project Management', 'Business Process', 'Business Requirements', 'Excel', 'Finance', 'Python (Programming Language)', 'Detail Oriented', 'SAP Applications', 'Dashboard', 'Presentations', 'Tableau (Business Intelligence Software)', 'Planning']

## 18.2 Industry expertise demand (data-driven 1–5)

In this section, We built a simple market target for each top skill using what employers write in job postings. First, we joined the job title and description so we could read the text. If a post listed MIN\_YEARS\_EXPERIENCE, we used it; if not, we pulled numbers like “3+ years” from the text and mapped years to a 1–5 level (with a small bump so “3 years” sits in the middle of a 3–5 range). Next, we read seniority words in the title (junior vs. senior/lead/manager) and skill phrases in the text (basic, intermediate, advanced, expert) and turned those into levels too. For each posting and skill, we combined the three signals, years (50%), seniority (30%), and phrases (20%) to get one score. Finally, we averaged those scores across all postings for each skill to get a 1–5 industry target, which we used to compare with our team’s ratings.

import re  
import numpy as np  
  
# Build text fields if they exist; otherwise empty strings  
title\_col = "TITLE" if "TITLE" in raw\_df.columns else None  
body\_col = "BODY" if "BODY" in raw\_df.columns else None  
  
text\_title = raw\_df[title\_col].astype(str).str.lower() if title\_col else ""  
text\_body = raw\_df[body\_col].astype(str).str.lower() if body\_col else ""  
text\_all = (text\_title + " " + text\_body).astype(str).str.strip()  
  
# Prefer MIN\_YEARS\_EXPERIENCE if present; else parse "3+ years" from text  
min\_col = "MIN\_YEARS\_EXPERIENCE" if "MIN\_YEARS\_EXPERIENCE" in raw\_df.columns else None  
min\_years = pd.to\_numeric(raw\_df[min\_col], errors="coerce") if min\_col else pd.Series(np.nan, index=raw\_df.index)  
  
def extract\_years\_from\_text(text):  
 if not isinstance(text, str): return np.nan  
 m = re.search(r'(\d+)\s\*\+?\s\*(?:years?|yrs)\s+(?:of\s+)?experience', text, flags=re.I)  
 return float(m.group(1)) if m else np.nan  
  
years\_from\_text = text\_all.apply(extract\_years\_from\_text)  
years\_req = min\_years.where(min\_years.notna(), years\_from\_text)  
  
def years\_to\_level\_from\_min(y):  
 if pd.isna(y): return np.nan  
 y = float(y)  
 base = 1 if y < 1 else 2 if y < 2 else 3 if y < 4 else 4 if y < 6 else 5  
 return min(5, base + 0.3) # small bump so "3 years" ≈ mid of typical 3–5  
  
years\_level = years\_req.apply(years\_to\_level\_from\_min)  
  
def seniority\_from\_title(t):  
 if not isinstance(t, str): return np.nan  
 if re.search(r'\b(intern|junior|jr|entry)\b', t): return 2  
 if re.search(r'\b(senior|sr|lead|principal|architect)\b', t): return 5  
 if re.search(r'\b(manager|director|head)\b', t): return 5  
 return 3  
  
seniority\_level = text\_title.apply(seniority\_from\_title) if title\_col else pd.Series(np.nan, index=raw\_df.index)  
  
PHRASE\_LEVELS = [  
 (r'\b(expert|expertise|mastery|guru)\b', 5),  
 (r'\b(advanced|in-depth|strong|proficient|hands-on|solid)\b', 4),  
 (r'\b(intermediate|working knowledge)\b', 3),  
 (r'\b(basic|knowledge of|familiarity)\b', 2),  
]  
def phrase\_level(text):  
 if not isinstance(text, str): return np.nan  
 lvl = np.nan  
 for pat, v in PHRASE\_LEVELS:  
 if re.search(pat, text):  
 lvl = v if pd.isna(lvl) else max(lvl, v)  
 return lvl  
  
phrase\_level\_series = text\_all.apply(phrase\_level)  
  
# Explode one row per (posting, skill), keep only Top 20 skills  
exploded = pd.DataFrame({  
 "SKILLS\_LIST": raw\_df["SKILLS\_NAME"],  
 "years\_level": years\_level,  
 "seniority\_level": seniority\_level,  
 "phrase\_level": phrase\_level\_series  
}).explode("SKILLS\_LIST")  
  
exploded["SKILL"] = exploded["SKILLS\_LIST"].astype(str).apply(canon\_skill)  
exploded = exploded[exploded["SKILL"].isin(top\_skills)]  
  
# Combine signals → expected level per row  
w\_years, w\_seniority, w\_phrase = 0.5, 0.3, 0.2  
def combine\_levels(row):  
 vals, wts = [], []  
 if pd.notna(row["years\_level"]): vals.append(row["years\_level"]); wts.append(w\_years)  
 if pd.notna(row["seniority\_level"]): vals.append(row["seniority\_level"]); wts.append(w\_seniority)  
 if pd.notna(row["phrase\_level"]): vals.append(row["phrase\_level"]); wts.append(w\_phrase)  
 if not vals: return 3.0  
 return float(np.average(vals, weights=wts))  
  
exploded["EXPECTED\_LEVEL"] = exploded.apply(combine\_levels, axis=1)  
  
# Final per-skill target (1–5)  
expected\_per\_skill = (  
 exploded.groupby("SKILL")["EXPECTED\_LEVEL"]  
 .mean()  
 .clip(1,5)  
 .round(2)  
 .reindex(top\_skills)  
)  
expected\_per\_skill.name = "Target (Data-Driven)"  
expected\_per\_skill

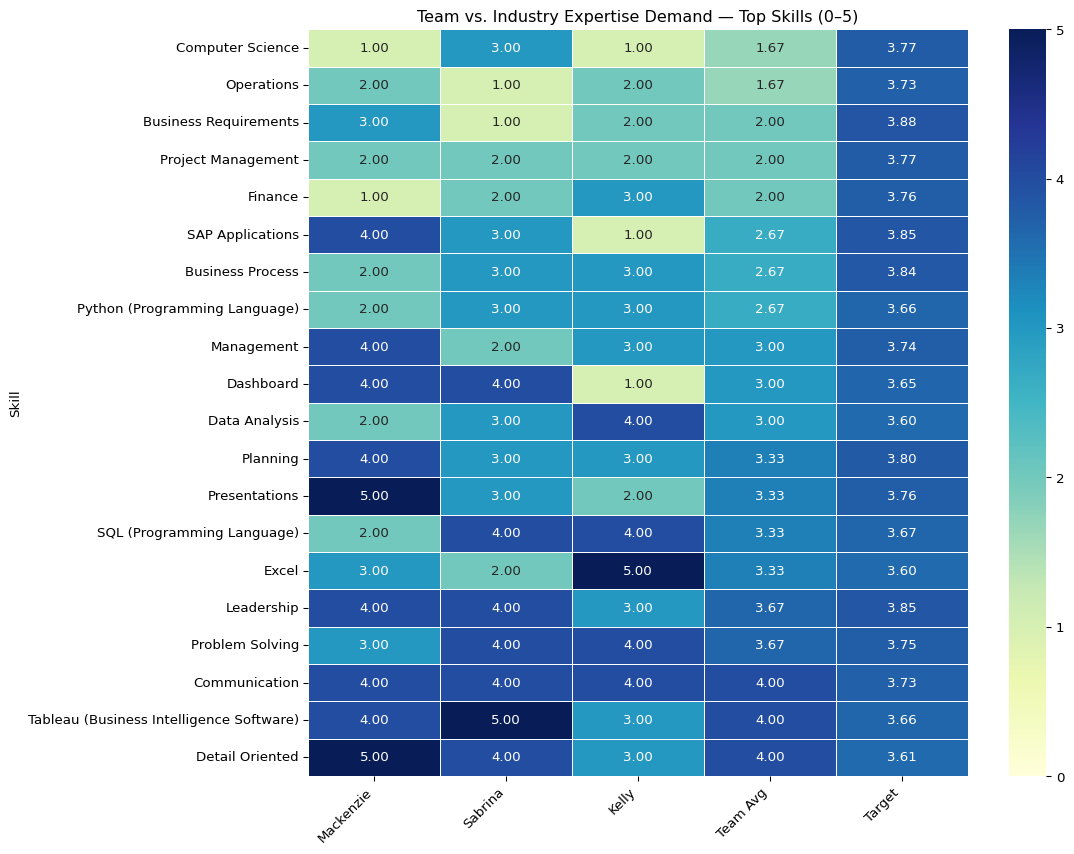
SKILL  
Communication 3.73  
Data Analysis 3.60  
Management 3.74  
SQL (Programming Language) 3.67  
Problem Solving 3.75  
Leadership 3.85  
Computer Science 3.77  
Operations 3.73  
Project Management 3.77  
Business Process 3.84  
Business Requirements 3.88  
Excel 3.60  
Finance 3.76  
Python (Programming Language) 3.66  
Detail Oriented 3.61  
SAP Applications 3.85  
Dashboard 3.65  
Presentations 3.76  
Tableau (Business Intelligence Software) 3.66  
Planning 3.80  
Name: Target (Data-Driven), dtype: float64

## 18.3 Team Skills Vs. Industry Requiments

import pandas as pd  
import numpy as np  
  
# --- 0) Preconditions: you already have ---  
# df\_skills (index=Name, columns include your top\_skills)  
# top\_skills (list of skills to compare)  
# expected\_per\_skill (Series: index=skills, values=industry target 1–5)  
  
# --- 1) Ensure apples-to-apples (only top\_skills) ---  
df\_team\_top10 = (  
 df\_skills.reindex(columns=top\_skills, fill\_value=0)  
 .apply(pd.to\_numeric, errors="coerce")  
 .fillna(0).clip(0,5)  
)  
  
# --- 2) Build tidy comparison table ---  
team\_avg = df\_team\_top10.mean(axis=0)  
target = expected\_per\_skill.reindex(top\_skills).astype(float)  
gap = (target - team\_avg)  
  
# Order columns: Skill, each person…, Team Avg, Target, Gap  
members = df\_team\_top10.index.tolist()  
rows = []  
for skill in top\_skills:  
 row = {  
 "Skill": skill,  
 \*\*{name: float(df\_team\_top10.loc[name, skill]) for name in members},  
 "Team Avg": round(float(team\_avg[skill]), 2),  
 "Target": round(float(target[skill]), 2) if pd.notna(target[skill]) else np.nan,  
 "Gap (Target−Avg)": round(float(gap[skill]), 2) if pd.notna(gap[skill]) else np.nan,  
 }  
 rows.append(row)  
  
comparison\_table = pd.DataFrame(rows)  
  
# Sort by biggest gap (needs first)  
comparison\_table = comparison\_table.sort\_values("Gap (Target−Avg)", ascending=False, na\_position="last").reset\_index(drop=True)  
  
# Nice rounding for member columns too  
for name in members:  
 comparison\_table[name] = comparison\_table[name].round(2)  
  
comparison\_table

|  | Skill | Mackenzie | Sabrina | Kelly | Team Avg | Target | Gap (Target−Avg) |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Computer Science | 1.0 | 3.0 | 1.0 | 1.67 | 3.77 | 2.10 |
| 1 | Operations | 2.0 | 1.0 | 2.0 | 1.67 | 3.73 | 2.06 |
| 2 | Business Requirements | 3.0 | 1.0 | 2.0 | 2.00 | 3.88 | 1.88 |
| 3 | Project Management | 2.0 | 2.0 | 2.0 | 2.00 | 3.77 | 1.77 |
| 4 | Finance | 1.0 | 2.0 | 3.0 | 2.00 | 3.76 | 1.76 |
| 5 | SAP Applications | 4.0 | 3.0 | 1.0 | 2.67 | 3.85 | 1.18 |
| 6 | Business Process | 2.0 | 3.0 | 3.0 | 2.67 | 3.84 | 1.17 |
| 7 | Python (Programming Language) | 2.0 | 3.0 | 3.0 | 2.67 | 3.66 | 0.99 |
| 8 | Management | 4.0 | 2.0 | 3.0 | 3.00 | 3.74 | 0.74 |
| 9 | Dashboard | 4.0 | 4.0 | 1.0 | 3.00 | 3.65 | 0.65 |
| 10 | Data Analysis | 2.0 | 3.0 | 4.0 | 3.00 | 3.60 | 0.60 |
| 11 | Planning | 4.0 | 3.0 | 3.0 | 3.33 | 3.80 | 0.47 |
| 12 | Presentations | 5.0 | 3.0 | 2.0 | 3.33 | 3.76 | 0.43 |
| 13 | SQL (Programming Language) | 2.0 | 4.0 | 4.0 | 3.33 | 3.67 | 0.34 |
| 14 | Excel | 3.0 | 2.0 | 5.0 | 3.33 | 3.60 | 0.27 |
| 15 | Leadership | 4.0 | 4.0 | 3.0 | 3.67 | 3.85 | 0.18 |
| 16 | Problem Solving | 3.0 | 4.0 | 4.0 | 3.67 | 3.75 | 0.08 |
| 17 | Communication | 4.0 | 4.0 | 4.0 | 4.00 | 3.73 | -0.27 |
| 18 | Tableau (Business Intelligence Software) | 4.0 | 5.0 | 3.0 | 4.00 | 3.66 | -0.34 |
| 19 | Detail Oriented | 5.0 | 4.0 | 3.0 | 4.00 | 3.61 | -0.39 |

import os  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Pick which columns to plot (only numbers!)  
members = df\_team\_top10.index.tolist()   
numeric\_cols = members + ["Team Avg", "Target"]   
  
# Make a numeric matrix indexed by Skill  
heatmap\_df = (  
 comparison\_table  
 .set\_index("Skill")[numeric\_cols]  
 .apply(pd.to\_numeric, errors="coerce")  
)  
  
# 3) Plot  
os.makedirs("output", exist\_ok=True)  
plt.figure(figsize=(12, max(6, 0.45\*len(heatmap\_df)))) # grow height if many skills  
sns.heatmap(  
 heatmap\_df, annot=True, fmt=".2f",  
 cmap="YlGnBu", linewidths=0.5, vmin=0, vmax=5  
)  
plt.title("Team vs. Industry Expertise Demand — Top Skills (0–5)")  
plt.xticks(rotation=45, ha="right")  
plt.yticks(rotation=0)  
plt.tight\_layout()  
plt.savefig("output/team\_vs\_industry\_expertise\_heatmap.png", dpi=300)  
plt.show()  
plt.close()



The team vs. industry headmad compares our skills to the market targets (bottom row). We meet or beat the target in communication and Tableau and are close on problem solving, Excel, and detail-oriented work. We are below the target in computer science, operations, project management, business requirements/process, leadership, finance, Python, and SQL. Our focus should be to raise those areas to about 3.5 or 4 with short courses and practice. Overall, we explain and show insights well, but we need stronger basics and delivery skills to match the market.

## 18.4 Gaps + Market Adjusted Priorities

# Unweighted gap (Target − Team Avg)  
team\_avg = df\_team\_top10.mean()  
gap = expected\_per\_skill - team\_avg  
  
gap\_table = (  
 pd.DataFrame({  
 "Avg Team Level": team\_avg.round(2),  
 "Target (Data-Driven)": expected\_per\_skill.round(2),  
 "Gap (Target − Avg)": gap.round(2)  
 })  
 .sort\_values("Gap (Target − Avg)", ascending=False)  
)  
gap\_table

|  | Avg Team Level | Target (Data-Driven) | Gap (Target − Avg) |
| --- | --- | --- | --- |
| Computer Science | 1.67 | 3.77 | 2.10 |
| Operations | 1.67 | 3.73 | 2.06 |
| Business Requirements | 2.00 | 3.88 | 1.88 |
| Project Management | 2.00 | 3.77 | 1.77 |
| Finance | 2.00 | 3.76 | 1.76 |
| SAP Applications | 2.67 | 3.85 | 1.18 |
| Business Process | 2.67 | 3.84 | 1.17 |
| Python (Programming Language) | 2.67 | 3.66 | 0.99 |
| Management | 3.00 | 3.74 | 0.74 |
| Dashboard | 3.00 | 3.65 | 0.65 |
| Data Analysis | 3.00 | 3.60 | 0.60 |
| Planning | 3.33 | 3.80 | 0.47 |
| Presentations | 3.33 | 3.76 | 0.43 |
| SQL (Programming Language) | 3.33 | 3.67 | 0.34 |
| Excel | 3.33 | 3.60 | 0.27 |
| Leadership | 3.67 | 3.85 | 0.18 |
| Problem Solving | 3.67 | 3.75 | 0.08 |
| Communication | 4.00 | 3.73 | -0.27 |
| Tableau (Business Intelligence Software) | 4.00 | 3.66 | -0.34 |
| Detail Oriented | 4.00 | 3.61 | -0.39 |

The gaps table shows where we sit below the market target. Positive numbers mean we need to improve; negative numbers mean we already meet or beat the target. Our biggest gaps are in computer science, operations, business requirements, project management, and finance. Smaller gaps show up in SAP, business process, and Python. We already match or exceed the market in communication, Tableau, and being detail-oriented.

To set priorities, we adjust each gap by how common the skill is in job postings (market-adjusted priority). Using that, we should first focus on computer science and operations, then business requirements and project management, followed by finance/SAP. This plan lets us close the largest, most market-relevant gaps first while keeping our strengths sharp.

# 19. Improvement Plan

To improve our IT skills, we can focus on computer science basics, business requirements, project management, and operations. Mackenzie can prioritize computer science, then operations, finance, and project management. Sabrina will focus on project management, business requirements, and operations. Finally, Kelly can focus on computer science, then business requirements, operations, and SAP basics. We can keep our strengths, such as communication, Tableau, SQL, and problem-solving, refreshed with light weekly practice.

Additionally, we could use the following resources to improve our skills: Harvard CS50x for an introduction to computer science (a free course) and LeetCode for practice. For the programming languages SQL and Python, we can use Coursera and DataCamp courses, and LeetCode for programming exercises, which are really helpful for interview practice. For Project Management and operations, we can take the Google Project Management on Coursera. For business requirements, we can take the Managing Requirements for the Business Analysis course on Pluralsight. For finance, we can use Khan Academy’s accounting and financial statements modules. For SAP, we can use SAP Learning’s free fundamentals. For Excel, we can follow Microsoft Learn’s Excel path to reach level 3+.

Finally, we can address our skill gaps with a simple schedule. We can meet weekly or every other week and rotate knowledge guides for each session. If someone is great at project management or another skill, they will share their knowledge and tips. We will work on small, practical projects where we code together, review each other’s work, and discuss what we find. We can keep a shared document and a simple dashboard using a template to track each person’s goals, progress, and next steps for each area that needs improvement. After each session, we will note what we learned, what held us back, and what went well to keep improving.

# 20. K-Means Clustering

# 21. Kmeans Clustering Setup

import os, re  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.compose import ColumnTransformer  
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import OneHotEncoder, StandardScaler, FunctionTransformer  
from sklearn.impute import SimpleImputer  
from sklearn.cluster import KMeans  
from sklearn.metrics import silhouette\_score  
  
os.makedirs("output", exist\_ok=True)  
  
# Use your cleaned frame if present; else load CSV  
try:  
 df = raw\_df.copy()  
except NameError:  
 df = pd.read\_csv("data/lightcast\_job\_postings.csv")  
  
# Pick a title/text column robustly  
for c in ["TITLE\_CLEAN", "TITLE", "TITLE\_NAME", "TITLE\_RAW"]:  
 if c in df.columns:  
 text\_col = c  
 break  
else:  
 raise ValueError("No title column found (TITLE\_CLEAN/TITLE/TITLE\_NAME/TITLE\_RAW).")  
  
# Coerce useful numerics if present  
for c in ["SALARY", "SALARY\_FROM", "SALARY\_TO", "MIN\_YEARS\_EXPERIENCE", "MAX\_YEARS\_EXPERIENCE", "DURATION"]:  
 if c in df.columns:  
 df[c] = pd.to\_numeric(df[c], errors="coerce")  
  
# Candidate categoricals (kept if present & not too wide)  
candidate\_cat = [  
 "REMOTE\_TYPE\_NAME", "STATE\_NAME", "EMPLOYMENT\_TYPE\_NAME",  
 "COMPANY\_IS\_STAFFING", "NAICS\_2022\_6\_NAME", "ONET\_NAME", "SOC\_2021\_5\_NAME"  
]  
cat\_cols = [c for c in candidate\_cat if c in df.columns]  
cat\_cols = [c for c in cat\_cols if df[c].nunique(dropna=True) <= 200]  
  
num\_cols = [c for c in ["SALARY", "MIN\_YEARS\_EXPERIENCE", "MAX\_YEARS\_EXPERIENCE", "DURATION"] if c in df.columns]  
  
print("Using columns:")  
print(" text\_col:", text\_col)  
print(" cat\_cols:", cat\_cols)  
print(" num\_cols:", num\_cols)  
  
RANDOM\_STATE = 42

Using columns:  
 text\_col: TITLE\_CLEAN  
 cat\_cols: ['REMOTE\_TYPE\_NAME', 'STATE\_NAME', 'EMPLOYMENT\_TYPE\_NAME', 'COMPANY\_IS\_STAFFING', 'ONET\_NAME', 'SOC\_2021\_5\_NAME']  
 num\_cols: ['SALARY', 'MIN\_YEARS\_EXPERIENCE', 'DURATION']

# 22. Helpers

def \_clean\_text\_input(x):  
 """  
 Accept Series, 1-col DataFrame, or numpy array from ColumnTransformer  
 and return a plain Python list[str] with NaNs -> "".  
 """  
 if isinstance(x, pd.Series):  
 s = x  
 elif isinstance(x, pd.DataFrame):  
 s = x.iloc[:, 0]  
 elif isinstance(x, np.ndarray):  
 s = pd.Series(x.ravel())  
 else:  
 s = pd.Series(x)  
 s = s.astype("string").fillna("")  
 return s.tolist()  
  
# Handle OneHotEncoder API difference across sklearn versions  
try:  
 \_ = OneHotEncoder(sparse\_output=True)  
 \_OHE\_KW = {"sparse\_output": True}  
except TypeError:  
 \_OHE\_KW = {"sparse": True}

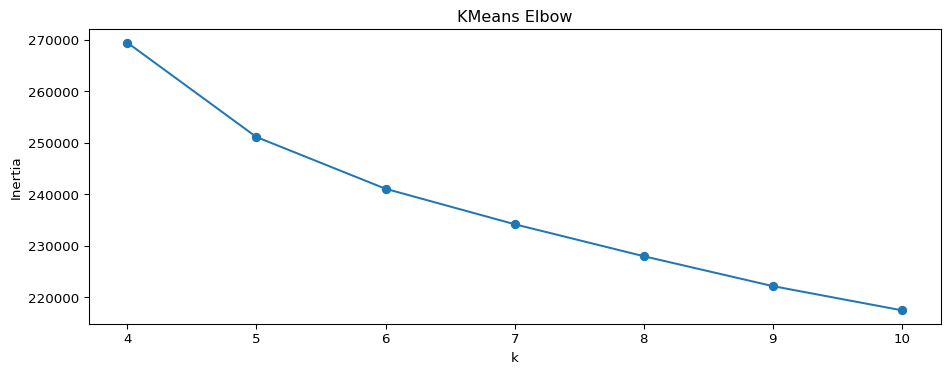
# 23. Preprocessing

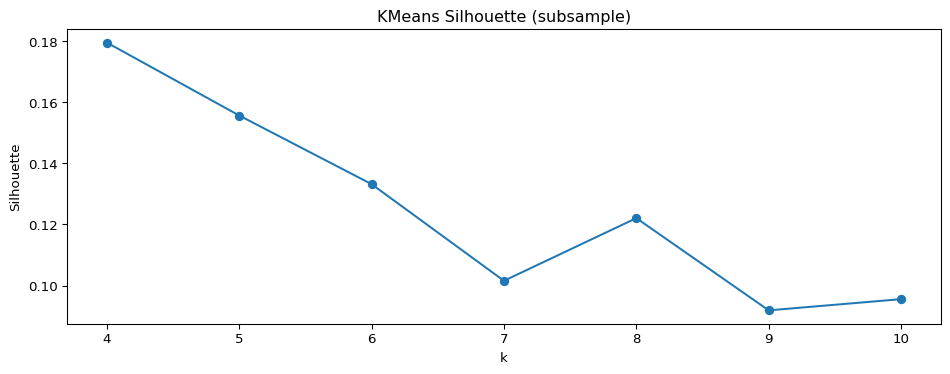
text\_pipe = Pipeline([  
 ("clean", FunctionTransformer(\_clean\_text\_input, validate=False)),  
 ("tfidf", TfidfVectorizer(  
 lowercase=True,  
 max\_features=40\_000,  
 ngram\_range=(1, 2),  
 min\_df=5  
 )),  
])  
  
cat\_pipe = Pipeline([  
 ("imputer", SimpleImputer(strategy="most\_frequent")),  
 ("ohe", OneHotEncoder(handle\_unknown="ignore", \*\*\_OHE\_KW)),  
])  
  
num\_pipe = Pipeline([  
 ("imputer", SimpleImputer(strategy="median")),  
 ("scaler", StandardScaler(with\_mean=False)),  
])  
  
pre = ColumnTransformer(  
 transformers=[  
 ("txt", text\_pipe, [text\_col]), # pass 2-D slice  
 ("cat", cat\_pipe, cat\_cols),  
 ("num", num\_pipe, num\_cols),  
 ],  
 sparse\_threshold=1.0  
)  
  
X = pre.fit\_transform(df)  
print("Feature matrix shape:", X.shape)

Feature matrix shape: (69198, 7548)

# 24. Model Selection

k\_values = list(range(4, 11))  
inertias, sils = [], []  
  
# Subsample for silhouette if very large  
if X.shape[0] > 8000:  
 rng = np.random.default\_rng(RANDOM\_STATE)  
 idx = rng.choice(X.shape[0], size=8000, replace=False)  
 X\_sil = X[idx]  
else:  
 X\_sil = X  
  
for k in k\_values:  
 km = KMeans(n\_clusters=k, n\_init=20, random\_state=RANDOM\_STATE)  
 km.fit(X)  
 inertias.append(km.inertia\_)  
 sils.append(silhouette\_score(X\_sil, km.predict(X\_sil), metric="euclidean"))  
  
plt.figure(figsize=(10,4))  
plt.plot(k\_values, inertias, marker="o")  
plt.xlabel("k"); plt.ylabel("Inertia"); plt.title("KMeans Elbow")  
plt.tight\_layout(); plt.savefig("output/kmeans\_elbow.png", dpi=150); plt.show()  
  
plt.figure(figsize=(10,4))  
plt.plot(k\_values, sils, marker="o")  
plt.xlabel("k"); plt.ylabel("Silhouette"); plt.title("KMeans Silhouette (subsample)")  
plt.tight\_layout(); plt.savefig("output/kmeans\_silhouette.png", dpi=150); plt.show()  
  
best\_k = int(k\_values[int(np.argmax(sils))])  
print("Chosen k:", best\_k)





Chosen k: 4

# 25. Final fit

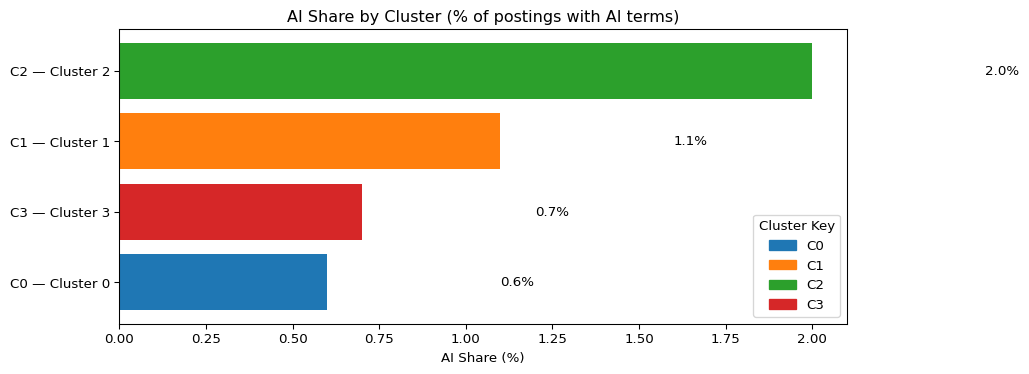
from sklearn.decomposition import TruncatedSVD # used later  
  
kmeans = KMeans(n\_clusters=best\_k, n\_init=20, random\_state=RANDOM\_STATE)  
labels = kmeans.fit\_predict(X)  
  
# Build a compact frame with inputs + cluster id  
df\_clusters = df[[text\_col] + cat\_cols + num\_cols].copy()  
df\_clusters["cluster"] = labels  
  
# Save outputs for reuse  
df\_clusters.to\_csv("output/cluster\_assignments.csv", index=False)  
  
sizes = df\_clusters["cluster"].value\_counts().sort\_index()  
print("Cluster sizes:\n", sizes)  
sizes.to\_csv("output/cluster\_sizes.csv")  
  
# --- Top TF-IDF terms per cluster (text portion) ---  
tfidf = pre.named\_transformers\_["txt"].named\_steps["tfidf"]  
terms = np.array(tfidf.get\_feature\_names\_out())  
text\_only = tfidf.transform(\_clean\_text\_input(df[text\_col]))  
  
top\_n = 15  
top\_terms = {}  
for c in range(best\_k):  
 mask = (labels == c)  
 if mask.sum() == 0:  
 top\_terms[c] = []  
 continue  
 centroid = text\_only[mask].mean(axis=0)  
 centroid = np.asarray(centroid).ravel()  
 idx = np.argsort(centroid)[::-1][:top\_n]  
 top\_terms[c] = terms[idx].tolist()  
  
with open("output/cluster\_top\_terms.txt", "w") as f:  
 for c in range(best\_k):  
 f.write(f"Cluster {c} — top terms:\n")  
 f.write(", ".join(top\_terms[c]) + "\n\n")  
  
print("Top terms file saved -> output/cluster\_top\_terms.txt")

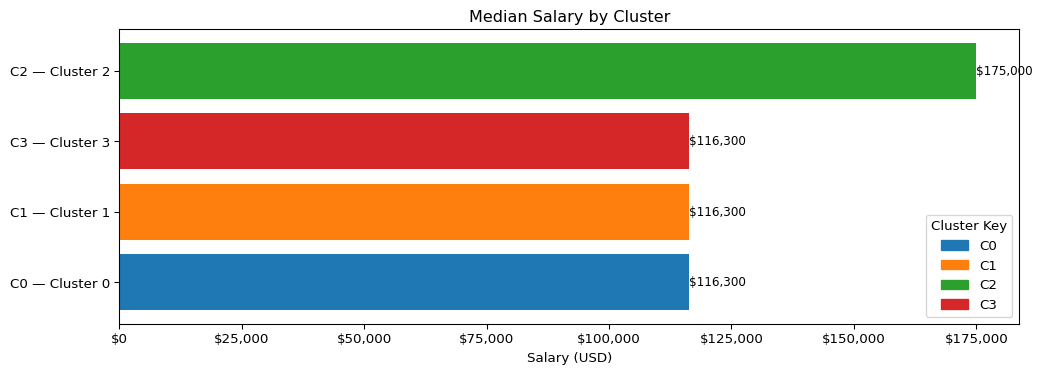
Cluster sizes:  
 cluster  
0 9190  
1 42435  
2 5550  
3 12023  
Name: count, dtype: int64  
Top terms file saved -> output/cluster\_top\_terms.txt

# 26. Cluster summary charts

import matplotlib.patches as mpatches  
from matplotlib.ticker import FuncFormatter  
  
# Build dfc in-memory from assignments  
dfc = df\_clusters.copy()  
  
# Detect AI terms on TITLE  
TITLE\_COL = "TITLE\_CLEAN" if "TITLE\_CLEAN" in dfc.columns else "TITLE"  
assert TITLE\_COL in dfc.columns, "Need a TITLE or TITLE\_CLEAN column in dfc."  
  
AI\_TERMS = [  
 r"\bAI\b", r"\bML\b", r"\bLLM\b", r"\bNLP\b",  
 "artificial intelligence", "machine learning", "deep learning",  
 "computer vision", "generative", "gen ai", "chatgpt", r"gpt-\d+",  
 "transformer", "bert", "prompt engineer", "reinforcement learning"  
]  
ai\_pat = re.compile("|".join(AI\_TERMS), flags=re.IGNORECASE)  
if "is\_ai" not in dfc.columns:  
 dfc["is\_ai"] = dfc[TITLE\_COL].astype(str).str.contains(ai\_pat, na=False)  
  
# Colors per cluster id  
cluster\_colors = {c: plt.cm.get\_cmap("tab10")(i % 10) for i, c in enumerate(sorted(dfc["cluster"].unique()))}  
  
# Human-friendly names (fallback to id if not mapped)  
cluster\_names = {i: f"Cluster {i}" for i in sorted(dfc["cluster"].unique())}  
dfc["cluster\_name"] = dfc["cluster"].map(cluster\_names)  
  
# Aggregate stats  
stats = (  
 dfc.groupby("cluster", as\_index=False)  
 .agg(postings=("cluster","size"),  
 ai\_share=("is\_ai","mean"),  
 median\_salary=("SALARY","median"))  
)  
stats["cluster\_name"] = stats["cluster"].map(cluster\_names)  
stats["label"] = stats.apply(lambda r: f"C{int(r.cluster)} — {r.cluster\_name}", axis=1)  
stats["ai\_share\_pct"] = (stats["ai\_share"] \* 100).round(1)  
stats["color"] = stats["cluster"].map(cluster\_colors)  
  
# Legend/key (reusable)  
legend\_handles = [  
 mpatches.Patch(color=cluster\_colors[c], label=f"C{c}")  
 for c in sorted(stats["cluster"].unique())  
]  
  
# Plot 1: AI share by cluster  
dfp = stats.sort\_values("ai\_share\_pct", ascending=True)  
plt.figure(figsize=(11, 4))  
plt.barh(dfp["label"], dfp["ai\_share\_pct"], color=dfp["color"])  
plt.title("AI Share by Cluster (% of postings with AI terms)")  
plt.xlabel("AI Share (%)")  
for y, v in enumerate(dfp["ai\_share\_pct"]):  
 plt.text(v + 0.5, y, f"{v:.1f}%", va="center")  
plt.legend(handles=legend\_handles, title="Cluster Key", loc="lower right")  
plt.tight\_layout()  
plt.savefig("output/cluster\_ai\_share.png", dpi=200, bbox\_inches="tight")  
plt.show()  
  
# Plot 2: Median salary by cluster  
dfp = stats.sort\_values("median\_salary", ascending=True)  
plt.figure(figsize=(11, 4))  
plt.barh(dfp["label"], dfp["median\_salary"], color=dfp["color"])  
plt.title("Median Salary by Cluster")  
plt.xlabel("Salary (USD)")  
plt.gca().xaxis.set\_major\_formatter(FuncFormatter(lambda x, pos: f"${int(x):,}"))  
for y, v in enumerate(dfp["median\_salary"]):  
 if pd.notnull(v):  
 plt.text(v, y, f"${int(v):,}", va="center", ha="left", fontsize=9)  
plt.legend(handles=legend\_handles, title="Cluster Key", loc="lower right")  
plt.tight\_layout()  
plt.savefig("output/cluster\_median\_salary.png", dpi=200, bbox\_inches="tight")  
plt.show()  
  
# Save tabular summary  
stats\_out = stats[["cluster","cluster\_name","postings","ai\_share\_pct","median\_salary"]].sort\_values("cluster")  
stats\_out.to\_csv("output/cluster\_summary.csv", index=False)  
stats\_out.head(10)

/var/folders/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/2483979144.py:22: MatplotlibDeprecationWarning:  
  
The get\_cmap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get\_cmap()`` or ``pyplot.get\_cmap()`` instead.





|  | cluster | cluster\_name | postings | ai\_share\_pct | median\_salary |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | Cluster 0 | 9190 | 0.6 | 116300.0 |
| 1 | 1 | Cluster 1 | 42435 | 1.1 | 116300.0 |
| 2 | 2 | Cluster 2 | 5550 | 2.0 | 175000.0 |
| 3 | 3 | Cluster 3 | 12023 | 0.7 | 116300.0 |

# 27. SVD 2D

from sklearn.decomposition import TruncatedSVD  
  
assert "X" in globals(), "Feature matrix X missing."  
svd = TruncatedSVD(n\_components=2, random\_state=RANDOM\_STATE)  
XY = svd.fit\_transform(X)  
print("Explained variance (2 comps):", svd.explained\_variance\_ratio\_.sum())

Explained variance (2 comps): 0.3754399133742323

# 28. Single Cluster Scatter

plt.figure(figsize=(9,6))  
for c in sorted(np.unique(labels)):  
 m = (labels == c)  
 plt.scatter(XY[m,0], XY[m,1], s=8, alpha=0.5,  
 color=cluster\_colors[c], label=f"C{c}")  
plt.title("KMeans clusters (2-D SVD embedding)")  
plt.xlabel("SVD 1"); plt.ylabel("SVD 2")  
plt.legend(markerscale=2, frameon=True)  
plt.tight\_layout()  
plt.savefig("output/kmeans\_svd\_scatter.png", dpi=180, bbox\_inches="tight")  
plt.show()



Under the recommendation of the Kmeans Elbow and Silhouette measures, four clear segments have emerged. C0 (EA / SAP–Oracle Consulting, Sr) comprises enterprise solution owners and senior consultants focused on ERP/CRM integrations, domain architecture, and delivery roadmaps. C1 (Data / BI Analysts) is the high‑volume analytics backbone handling reporting, dashboards, and KPI/ad‑hoc analysis at mid‑career compensation. C2 (Enterprise / Cloud Architects) is the premium niche cluster with principals and leads who own cloud platforms, reliability/security, and cross‑team technical direction, and therefore command the highest pay. C3 (Data / BI Analysts, consulting tilt) mirrors C1’s skills but skews toward consulting and remote work and shows the highest AI‑keyword incidence, reflecting applied‑AI enablement inside analytics teams. Overall, analyst demand drives scale (C1/C3), enterprise solutioning provides the integration bench (C0), and cross‑platform leadership remains scarce and premium (C2).

# 29. Reference table

from sklearn.metrics import (  
 adjusted\_rand\_score, normalized\_mutual\_info\_score,  
 homogeneity\_score, completeness\_score, v\_measure\_score  
)  
  
# Choose a reasonable reference column   
REF\_COL = next((c for c in [  
 "SOC\_2021\_3\_NAME","SOC\_2021\_2\_NAME","SOC\_2021\_5\_NAME",  
 "NAICS\_2022\_4\_NAME","NAICS\_2022\_2\_NAME","NAICS\_2022\_6\_NAME",  
 "ONET\_NAME","ONET\_2019\_NAME"  
] if c in df.columns), None)  
assert REF\_COL, "No SOC/NAICS/ONET label column found."  
  
df\_clusters = df\_clusters.copy()  
df\_clusters["ref\_label"] = df[REF\_COL].astype("string").fillna("Unknown").values  
  
y\_true = df\_clusters["ref\_label"].astype(str).values  
y\_pred = df\_clusters["cluster"].astype(int).values  
  
nmi = normalized\_mutual\_info\_score(y\_true, y\_pred)  
ari = adjusted\_rand\_score(y\_true, y\_pred)  
hom = homogeneity\_score(y\_true, y\_pred)  
comp = completeness\_score(y\_true, y\_pred)  
vms = v\_measure\_score(y\_true, y\_pred)  
  
print(f"NMI: {nmi:.3f} | ARI: {ari:.3f} | Homogeneity: {hom:.3f} | Completeness: {comp:.3f} | V-measure: {vms:.3f}")  
  
# Majority label per cluster + purity  
ct = pd.crosstab(df\_clusters["cluster"], df\_clusters["ref\_label"])  
cluster\_major = ct.idxmax(axis=1).rename("majority\_label")  
cluster\_hits = ct.max(axis=1)  
purity = cluster\_hits.sum() / ct.values.sum()  
  
summary = (  
 pd.concat([cluster\_major, cluster\_hits.rename("majority\_count"),  
 ct.sum(axis=1).rename("cluster\_size")], axis=1)  
 .assign(majority\_share=lambda d: d["majority\_count"] / d["cluster\_size"])  
 .sort\_index()  
)  
  
print("\nCluster → majority reference label:")  
summary.head(best\_k)  
print(f"\nOverall purity: {purity:.3f}")  
  
summary.to\_csv("output/cluster\_majority\_label\_summary.csv")  
ct.to\_csv("output/cluster\_label\_crosstab.csv")

NMI: 0.000 | ARI: -0.000 | Homogeneity: 0.026 | Completeness: 0.000 | V-measure: 0.000  
  
Cluster → majority reference label:  
  
Overall purity: 1.000

# 30. Heatmap prep

import pandas as pd  
  
# Quick diagnostics for label visibility  
cands = [  
 "NAICS\_2022\_6\_NAME","NAICS\_2022\_4\_NAME","NAICS\_2022\_2\_NAME",  
 "ONET\_NAME","ONET\_2019\_NAME",  
 "SOC\_2021\_5\_NAME","SOC\_2021\_3\_NAME","SOC\_2021\_2\_NAME",  
]  
diag = []  
for c in cands:  
 if c in df.columns:  
 s = df[c].astype("string")  
 diag.append({  
 "column": c,  
 "non\_null\_share": s.notna().mean(),  
 "n\_unique\_nonnull": s.dropna().nunique(),  
 "top5": s.value\_counts(dropna=True).head(5).index.tolist()  
 })  
diag\_df = pd.DataFrame(diag).sort\_values(["n\_unique\_nonnull","non\_null\_share"], ascending=[False, False])  
print("Label candidates (more uniques is better):")  
display(diag\_df)  
  
# --- Auto-pick with stronger uniqueness requirement ---  
# Prefer columns with decent coverage and at least 5–40 distinct values  
viable = diag\_df[(diag\_df["non\_null\_share"] >= 0.40) & (diag\_df["n\_unique\_nonnull"].between(5, 40))]  
if len(viable):  
 REF\_COL = viable.iloc[0]["column"]  
else:  
 # Fallback  
 REF\_COL = diag\_df.iloc[0]["column"]  
  
  
print("Using reference label column:", REF\_COL)

Label candidates (more uniques is better):

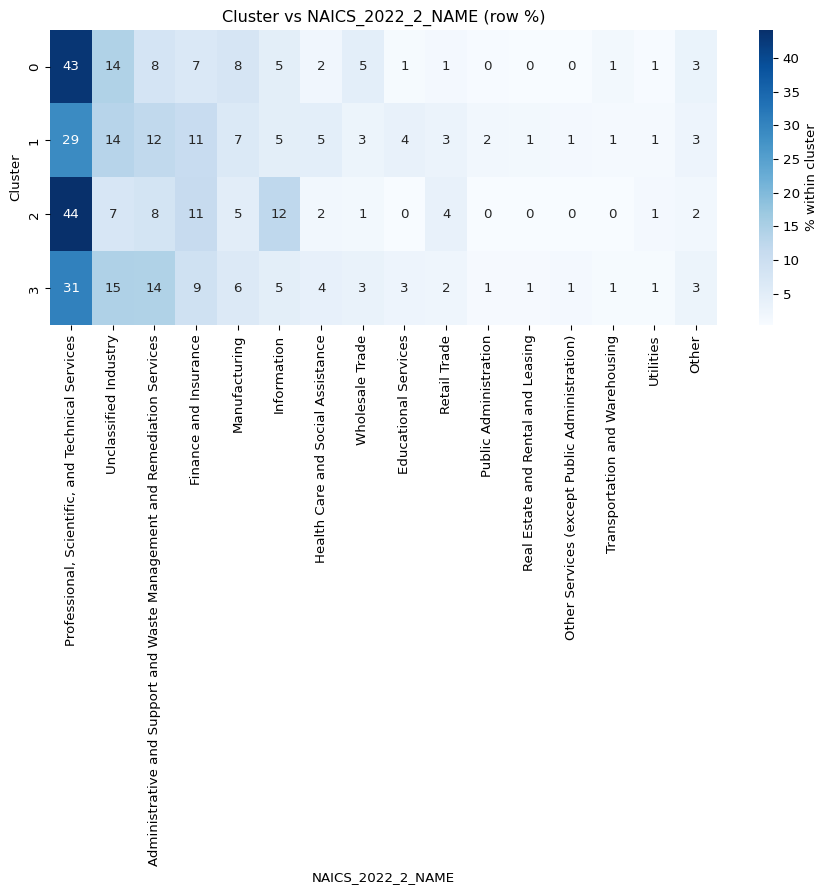
|  | column | non\_null\_share | n\_unique\_nonnull | top5 |
| --- | --- | --- | --- | --- |
| 0 | NAICS\_2022\_6\_NAME | 0.999754 | 814 | [Unclassified Industry, Custom Computer Progra... |
| 1 | NAICS\_2022\_4\_NAME | 0.999754 | 294 | [Computer Systems Design and Related Services,... |
| 2 | NAICS\_2022\_2\_NAME | 0.999754 | 21 | [Professional, Scientific, and Technical Servi... |
| 3 | ONET\_NAME | 0.999754 | 1 | [Business Intelligence Analysts] |
| 4 | ONET\_2019\_NAME | 0.999754 | 1 | [Business Intelligence Analysts] |
| 5 | SOC\_2021\_5\_NAME | 0.999754 | 1 | [Data Scientists] |
| 6 | SOC\_2021\_3\_NAME | 0.999754 | 1 | [Mathematical Science Occupations] |
| 7 | SOC\_2021\_2\_NAME | 0.999754 | 1 | [Computer and Mathematical Occupations] |

Using reference label column: NAICS\_2022\_2\_NAME

# 31. Heatmap plot

import numpy as np, pandas as pd, os  
import matplotlib.pyplot as plt  
try:  
 import seaborn as sns  
 use\_sns = True  
except Exception:  
 use\_sns = False  
  
os.makedirs("output", exist\_ok=True)  
  
# Base clustered frame  
if "dfc" in globals():  
 base = dfc.copy()  
elif "df\_clusters" in globals():  
 base = df\_clusters.copy()  
else:  
 raise AssertionError("Run the KMeans chunk so dfc/df\_clusters exist.")  
  
# Attach chosen reference label (aligned by row order)  
base["ref\_label"] = df[REF\_COL].astype("string").fillna("Unknown").values  
  
# Crosstab clusters x labels  
ct = pd.crosstab(base["cluster"], base["ref\_label"])  
  
# Drop 'Unknown' column if present  
if "Unknown" in ct.columns:  
 ct = ct.drop(columns=["Unknown"])  
  
# Keep top N labels overall (bucket the rest)  
TOPN = 15  
if ct.shape[1] > TOPN:  
 keep = ct.sum(axis=0).sort\_values(ascending=False).head(TOPN).index  
 ct\_reduced = ct[keep].copy()  
 ct\_reduced["Other"] = ct.drop(columns=keep).sum(axis=1)  
else:  
 ct\_reduced = ct  
  
# Remove empty rows (just in case)  
ct\_reduced = ct\_reduced.loc[ct\_reduced.sum(axis=1) > 0]  
  
# Convert to row percentages  
pct = ct\_reduced.div(ct\_reduced.sum(axis=1), axis=0) \* 100  
  
  
if pct.shape[1] <= 1:  
 print(f" {REF\_COL} doesn’t have enough diversity after cleaning. "  
 f"Try overriding REF\_COL to something like NAICS\_2022\_4\_NAME or ONET\_NAME.")  
else:  
 plt.figure(figsize=(max(10, 0.7\*pct.shape[1]), max(4, 0.6\*pct.shape[0])))  
 if use\_sns:  
 ax = sns.heatmap(pct, cmap="Blues", annot=True, fmt=".0f",  
 cbar\_kws={"label": "% within cluster"})  
 else:  
 im = plt.imshow(pct.values, aspect="auto", cmap="Blues")  
 plt.colorbar(im, label="% within cluster")  
 plt.xticks(range(pct.shape[1]), pct.columns, rotation=45, ha="right")  
 plt.yticks(range(pct.shape[0]), pct.index)  
 ax = plt.gca()  
  
 plt.title(f"Cluster vs {REF\_COL} (row %)")  
 plt.xlabel(REF\_COL); plt.ylabel("Cluster")  
 plt.tight\_layout()  
 plt.savefig("output/cluster\_vs\_reference\_heatmap.png", dpi=200, bbox\_inches="tight")  
 plt.show()

/var/folders/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_54294/2086638697.py:62: UserWarning:  
  
Tight layout not applied. The bottom and top margins cannot be made large enough to accommodate all Axes decorations.



This figure is a heatmap of job‑posting shares by industry, displayed as row percentages. The leftmost column— that includes Professional, Scientific & Technical Services, shows the darkest shading overall, meaning a large portion of postings in each row come from that industry, while lighter columns represent industries that account for a smaller share. The Profesional, Scientific, and Technical Services industry does seem to be the hottest job market based off of this dataset, however, this does not mean that it is necessarily driven by AI alone.

# 32. Kmeans Summary

The results point to a job market organized around four role families with one noticeably premium niche, and they suggest that compensation and hiring dynamics are driven more by seniority and enterprise scope than by AI keywords. Cluster C1 (Analyst & BI Core) supplies most of the volume (≈56%), centered on titles like data analyst, business analyst, and BI/analytics; it anchors day‑to‑day decision support and sits near the mid‑$110k median. Cluster C3 (Data Analyst & BI — Mixed/Remote, AI‑skewed) is a smaller analyst cohort with a stronger remote profile and the highest AI‑keyword incidence; it reflects applied‑AI enablement inside analytics teams and also prices around the mid‑$110k band. Cluster C0 (Enterprise Architecture & Solutions) blends architect/enterprise/SAP‑Oracle/consultant language and tilts toward cross‑system solution ownership—ERP/CRM modernization, integration roadmaps, and domain architecture—with compensation likewise clustering near the mid‑$110k range. Finally, Cluster C2 (Senior Enterprise/Cloud Leadership) is the small but premium niche (~7%) that stands apart both visually and economically, with a roughly $175k median in the provided charts; employers reward systems ownership, architectural accountability, reliability/security considerations, and cross‑team leadership, even when titles don’t carry explicit AI keywords. Meanwhile, AI‑tagged titles are increasing from a small base and are unevenly distributed—most common in C3 and least common in C2, yet the salary premium does not follow that same AI gradient, reinforcing that pay aligns with role family and enterprise scope rather than AI labeling. Industry composition across all clusters is anchored in Professional, Scientific & Technical Services, with steady contributions from Administrative & Support, Finance & Insurance, and Manufacturing. The fastest AI growth is occurring outside of pure software—holding companies, motor‑vehicle parts, R&D services, executive search, and temporary help which signals diffusion of AI demand into corporate centers and industrial supply chains. This suggests that an increased use of applied AI in operational workforces. Altogether, the evidence suggests a hybrid hiring mix: steady growth of applied‑AI roles embedded in analytics workflows (C1/C3), alongside continued scarcity and premium pricing for cross‑functional platform leaders (C2) who can translate strategy into architecture and delivery, with C0 providing the enterprise solutions bench that stitches systems and governance together. Although AI is a key component within this analysis, it does not necessarily mean that AI is overtaking the job industry and ‘destroying the job market’, but rather remodeling the job market and how we use technology within the workplace. AI is steadily contributing to industries such as Administrative & Support, Finance & Insurance, and Manufacturing, however, premium job postings with the highest salaries seem to still value human judgement and ability.

# 33. Logistic Regression

from pyspark.sql import SparkSession, Row  
from pyspark.sql.functions import col, lit, concat\_ws, lower, regexp\_replace, when, trim, mean as \_mean  
from pyspark.sql.types import DoubleType  
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler, MinMaxScaler  
from pyspark.ml.classification import LogisticRegression  
from pyspark.ml.evaluation import BinaryClassificationEvaluator  
from pyspark.ml import Pipeline  
from pyspark.ml.functions import vector\_to\_array  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
# --- Start Spark ---  
spark = SparkSession.builder.appName("AI\_Job\_Comparison").getOrCreate()  
  
# --- Load data ---  
df = (spark.read.option("header", "true")  
 .option("inferSchema", "true")  
 .option("multiLine", "true")  
 .option("escape", "\"")  
 .csv("data/lightcast\_job\_postings.csv"))  
  
# --- Create text features for AI labeling ---  
text\_cols = ["TITLE\_RAW", "TITLE\_CLEAN", "BODY", "SKILLS\_NAME",  
 "COMMON\_SKILLS\_NAME", "SPECIALIZED\_SKILLS\_NAME",   
 "SOFTWARE\_SKILLS\_NAME"]  
  
df = df.fillna("")  
df = df.withColumn("text\_features",  
 concat\_ws(" ", \*[lower(regexp\_replace(col(c), r"[^a-zA-Z0-9 ]", "")) for c in text\_cols]))  
  
# --- AI job label ---  
ai\_keywords = [  
 r"\bAI\b", r"\bML\b", r"\bLLM\b", r"\bNLP\b",  
 "artificial intelligence", "machine learning", "deep learning",  
 "computer vision", "generative", "gen ai", "chatgpt", r"gpt-\d+",  
 "transformer", "bert", "prompt engineer", "reinforcement learning"  
]  
ai\_pattern = "|".join([f"(?i){k}" for k in ai\_keywords])  
df = df.withColumn("requires\_ai", when(col("text\_features").rlike(ai\_pattern), lit(1)).otherwise(lit(0)))  
  
# --- Create average salary ---  
df = df.withColumn(  
 "SALARY\_AVG",  
 when(col("SALARY") > 0, col("SALARY"))  
 .when(col("SALARY\_FROM").isNotNull() & col("SALARY\_TO").isNotNull(),  
 (col("SALARY\_FROM") + col("SALARY\_TO")) / 2)  
 .when(col("SALARY\_FROM").isNotNull(), col("SALARY\_FROM"))  
 .when(col("SALARY\_TO").isNotNull(), col("SALARY\_TO"))  
 .otherwise(None)  
)  
  
# --- Handle education level ---  
df = df.withColumn("MIN\_EDULEVELS", when(col("MIN\_EDULEVELS") == 99, 0).otherwise(col("MIN\_EDULEVELS")))  
  
# --- Select relevant features ---  
model\_df = df.select(  
 "requires\_ai",  
 "SALARY\_AVG",  
 "MIN\_YEARS\_EXPERIENCE",  
 "MIN\_EDULEVELS",  
 "STATE\_NAME",  
 "REMOTE\_TYPE"  
)

WARNING: Using incubator modules: jdk.incubator.vector  
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties  
25/10/15 11:39:09 WARN Utils: Your hostname, Kellys-MacBook-Air.local, resolves to a loopback address: 127.0.0.1; using 192.168.1.6 instead (on interface en0)  
25/10/15 11:39:09 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address  
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
25/10/15 11:39:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
[Stage 1:> (0 + 1) / 1]

# --- Handle numeric nulls ---  
numeric\_cols = ["SALARY\_AVG", "MIN\_YEARS\_EXPERIENCE", "MIN\_EDULEVELS"]  
for c in numeric\_cols:  
 model\_df = model\_df.withColumn(c, col(c).cast(DoubleType()))  
 mean\_val = model\_df.select(\_mean(col(c))).first()[0]  
 model\_df = model\_df.na.fill({c: mean\_val})  
  
# --- Handle categorical nulls ---  
categorical\_cols = ["STATE\_NAME", "REMOTE\_TYPE"]  
model\_df = model\_df.filter(  
 (trim(col("STATE\_NAME")) != "") &  
 (trim(col("REMOTE\_TYPE")) != "") &  
 col("STATE\_NAME").isNotNull() &  
 col("REMOTE\_TYPE").isNotNull()  
)  
  
# --- Encode categorical variables ---  
indexers = [StringIndexer(inputCol=c, outputCol=f"{c}\_IDX", handleInvalid="keep") for c in categorical\_cols]  
encoders = [OneHotEncoder(inputCol=f"{c}\_IDX", outputCol=f"{c}\_VEC") for c in categorical\_cols]  
  
# --- Assemble numeric features ---  
numeric\_assembler = VectorAssembler(inputCols=numeric\_cols, outputCol="numeric\_features")  
  
# --- Scale numeric features to 0-1 ---  
scaler = MinMaxScaler(inputCol="numeric\_features", outputCol="numeric\_scaled")  
  
# --- Assemble final feature vector (scaled numeric + categorical vectors) ---  
final\_assembler = VectorAssembler(  
 inputCols=["numeric\_scaled"] + [f"{c}\_VEC" for c in categorical\_cols],  
 outputCol="features"  
)  
  
# --- Logistic Regression classifier ---  
from pyspark.ml.classification import LogisticRegression  
lr = LogisticRegression(featuresCol="features", labelCol="requires\_ai", maxIter=50, regParam=0.0, elasticNetParam=0.0)  
  
# --- Build pipeline ---  
pipeline = Pipeline(stages=indexers + encoders + [numeric\_assembler, scaler, final\_assembler, lr])  
  
# --- Split train/test and fit ---  
train\_df, test\_df = model\_df.randomSplit([0.8, 0.2], seed=42)  
lr\_model = pipeline.fit(train\_df)

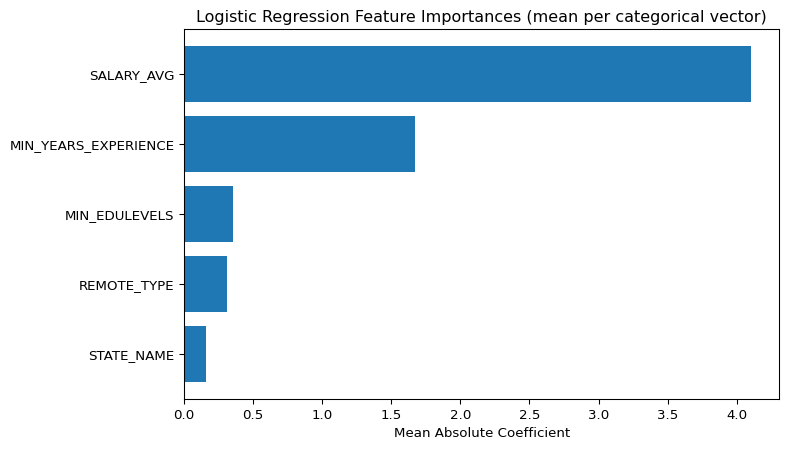
[Stage 2:> (0 + 1) / 1] [Stage 5:> (0 + 1) / 1] [Stage 8:> (0 + 1) / 1] [Stage 11:> (0 + 1) / 1] [Stage 17:> (0 + 1) / 1] [Stage 23:> (0 + 1) / 1] 25/10/15 11:40:57 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.  
[Stage 26:> (0 + 1) / 1] 25/10/15 11:41:31 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS  
[Stage 27:> (0 + 1) / 1]

# --- Predict & evaluate ---  
predictions = lr\_model.transform(test\_df)  
evaluator = BinaryClassificationEvaluator(labelCol="requires\_ai", metricName="areaUnderROC")  
roc\_auc = evaluator.evaluate(predictions)  
print(f"ROC-AUC: {roc\_auc:.3f}")  
  
# --- Extract logistic regression stage ---  
lr\_stage = lr\_model.stages[-1]  
  
# LogisticRegressionModel in Spark stores coefficients in a dense vector  
coefficients = lr\_stage.coefficients.toArray()  
  
# --- Compute feature sizes ---  
# Number of numeric features  
num\_numeric = len(numeric\_cols)  
# One-hot vector sizes for categorical features  
features\_vec = lr\_model.transform(test\_df).select("features").first()[0]  
vector\_size = len(features\_vec)  
num\_categorical = len(categorical\_cols)  
categorical\_vector\_sizes = [(vector\_size - num\_numeric)//num\_categorical]\*num\_categorical  
  
# --- Aggregate coefficients by feature ---  
feature\_sizes = [1]\*num\_numeric + categorical\_vector\_sizes  
feature\_names = numeric\_cols + categorical\_cols  
  
agg\_importances = []  
start = 0  
for size in feature\_sizes:  
 # Take mean of absolute values of coefficients for categorical features  
 agg\_importances.append(np.mean(np.abs(coefficients[start:start+size])))  
 start += size  
  
# --- Create DataFrame & plot ---  
importance\_df = pd.DataFrame({  
 "Feature": feature\_names,  
 "Importance": agg\_importances  
}).sort\_values("Importance", ascending=False)  
  
plt.figure(figsize=(8,5))  
plt.barh(importance\_df["Feature"], importance\_df["Importance"])  
plt.xlabel("Mean Absolute Coefficient")  
plt.title("Logistic Regression Feature Importances (mean per categorical vector)")  
plt.gca().invert\_yaxis()  
plt.savefig("output/logistic\_regression.png", dpi=200, bbox\_inches="tight")  
plt.show()

[Stage 38:> (0 + 1) / 1]

ROC-AUC: 0.623

[Stage 47:> (0 + 1) / 1]



ROC-AUC was used instead of accuracy because it is more suited for binary classifications such as AI vs non-AI.

# --- Create sample job listings ---  
sample\_jobs = [  
 Row(MIN\_YEARS\_EXPERIENCE=10.0, SALARY\_AVG=300000.0, STATE\_NAME="Massachusetts", MIN\_EDULEVELS=3.0, REMOTE\_TYPE="Remote"),  
 Row(MIN\_YEARS\_EXPERIENCE=5.0, SALARY\_AVG=75000.0, STATE\_NAME="Texas", MIN\_EDULEVELS=0.0, REMOTE\_TYPE="Onsite"),  
 Row(MIN\_YEARS\_EXPERIENCE=10.0, SALARY\_AVG=120000.0, STATE\_NAME="Minnesota", MIN\_EDULEVELS=2.0, REMOTE\_TYPE="Hybrid"),  
 Row(MIN\_YEARS\_EXPERIENCE=15.0, SALARY\_AVG=150000.0, STATE\_NAME="New York", MIN\_EDULEVELS=2.0, REMOTE\_TYPE="Remote"),  
]  
  
sample\_df = spark.createDataFrame(sample\_jobs)  
  
# Ensure numeric columns are DoubleType  
for c in ["MIN\_YEARS\_EXPERIENCE", "SALARY\_AVG", "MIN\_EDULEVELS"]:  
 sample\_df = sample\_df.withColumn(c, col(c).cast(DoubleType()))  
  
# --- Transform samples through the trained pipeline ---  
predictions = lr\_model.transform(sample\_df)  
  
# --- Extract probability of AI class ---  
predictions\_array = predictions.withColumn("prob\_array", vector\_to\_array(col("probability")))  
  
# --- Show results ---  
results = predictions\_array.select(  
 "MIN\_YEARS\_EXPERIENCE",  
 "SALARY\_AVG",  
 "STATE\_NAME",  
 "MIN\_EDULEVELS",  
 "REMOTE\_TYPE",  
 col("prob\_array")[1].alias("AI\_prob"),  
 "prediction"  
)  
  
results.show(truncate=False)

+--------------------+----------+-------------+-------------+-----------+-------------------+----------+  
|MIN\_YEARS\_EXPERIENCE|SALARY\_AVG|STATE\_NAME |MIN\_EDULEVELS|REMOTE\_TYPE|AI\_prob |prediction|  
+--------------------+----------+-------------+-------------+-----------+-------------------+----------+  
|10.0 |300000.0 |Massachusetts|3.0 |Remote |0.5243945602165612 |1.0 |  
|5.0 |75000.0 |Texas |0.0 |Onsite |0.23680753160930024|0.0 |  
|10.0 |120000.0 |Minnesota |2.0 |Hybrid |0.3289598642581484 |0.0 |  
|15.0 |150000.0 |New York |2.0 |Remote |0.49757478511466824|0.0 |  
+--------------------+----------+-------------+-------------+-----------+-------------------+----------+

# 34. Logistic Regression Summary

This analysis skimmed the job listings for keywords to distinguish AI from non-AI job postings and used a logistic regression model trained on five features: minimum years of experience, average salary, state, minimum education level, and remote work type. These features were chosen because AI roles typically require more experience, offer higher compensation, demand advanced degrees, and are concentrated in specific regions, while remote flexibility can influence applicant reach. The model revealed that years of experience and salary were the most influential factors, followed by state, education, and remote type, reflecting the expected patterns of AI job characteristics.

In order to create the model, we encoded the categorical variables and used MinMaxScaler to size the continuous variables to a 0-1 scale, which allowed the training model to weigh each variable equally. After obtaining the model data, we then charted the feature importance, making sure to take the aggregate coefficients to avoid over-reporting the importance of the encoded variables. The model reveals that salary is the most important feature when determining whether or not a role involves AI. When using the model against a sample data set, it predicted that the $300,000 salary job in Massachusetts with 10 years of minimum experience would involve AI, but the $150,000 salary job in New York with 15 minimum years of experience would not.

In order to refine its predictive capabilities, this model could benefit from some fine-tuning; right now it is evaluating probability and deeming any data with over 50% chance as involving AI, but if we were to get more granular with the data, the ideal threshold to use in this process may be closer to 35% or 40%. Additionally, given the ever-changing landscape of AI and the continued trend of AI becoming commonplace in many jobs, that threshold may continue to lower with time.

# 35. Conclusion

Overall, the EDA shows AI roles are growing across many industries and usually pay more, especially in computer services, consulting, R&D, and parts of banking. From the skill-gap work, we’re strong in communication, problem-solving, SQL/Tableau, and finance basics. Still, we’re light on computer-science basics, operations, project management, business requirements, and some SAP. In contrast, some non-AI service areas are flat or down. K-means says the market clusters into: (1) core analyst/BI roles (most volume), (2) analyst/BI with more AI and remote, (3) enterprise architecture/solutions, and (4) senior cloud/enterprise leadership (small but high pay). Pay tracks seniority and scope, not just “AI” labels. The random-forest check lines up: years of experience and salary are the strongest signals for AI-type roles, with education and location also mattering. Put together, the market rewards applied AI inside analytics teams now, and enterprise/platform leadership later

# References

Ansari, A., and A. Ansari. (2024): “[Consequences of AI induced job displacement](https://ijbat.com/index.php/IJBAT/article/view/18),” *International Journal of Business, Analytics, and Technology*, 2, 4–19.

Ejjami, R. (2024): “[Emerging professions in the age of AI across multiple sectors](https://jngr5.com/public/blog/Emerging%20Professions.pdf),” *International Journal for Multidisciplinary Research (IJFMR)*, 6, 1–32.

George, A. S. (2024): “[Artificial intelligence and the future of work: Job shifting not job loss](https://doi.org/10.5281/zenodo.10936489),” *Partners Universal Innovative Research Publication*, 2, 17–30.