Kmeans Clustering Setup

Final Report

Kelly, Sabrina, Makenzie

October 15, 2025

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', 'MAX\_YEARS\_EXPERIENCE', 'DURATION']

/var/folders/7j/ct705g296ls7nrjh30h9pyg40000gn/T/ipykernel\_59115/3257818786.py:20: DtypeWarning:  
  
Columns (19,30) have mixed types. Specify dtype option on import or set low\_memory=False.

# 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}

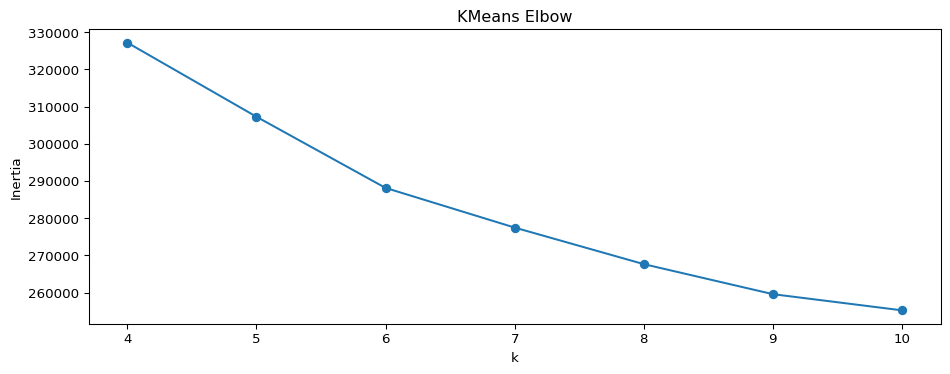
# Preprocessing

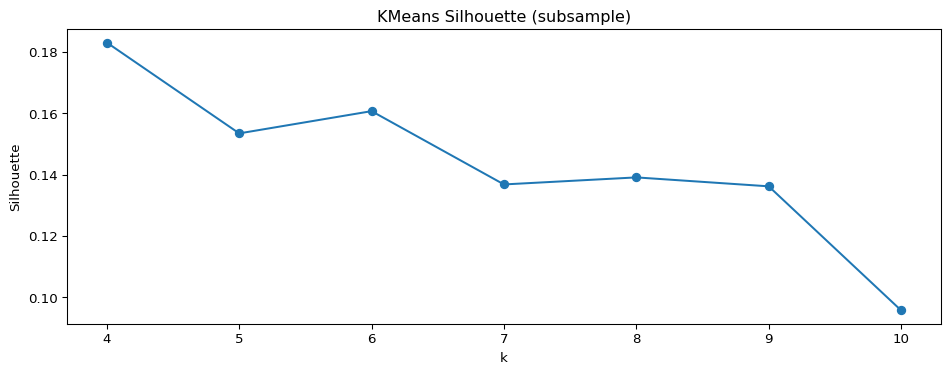
text\_pipe = Pipeline([  
 ("clean", FunctionTransformer(\_clean\_text\_input, validate=False)),  
 ("tfidf", TfidfVectorizer(  
 lowercase=True,  
 max\_features=40\_000,  
 ngram\_range=(1, 2),  
 min\_df=5  
 )),  
])  
  
cat\_pipe = Pipeline([  
 ("imputer", SimpleImputer(strategy="most\_frequent")),  
 ("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: (72498, 7752)

# 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

# Final fit

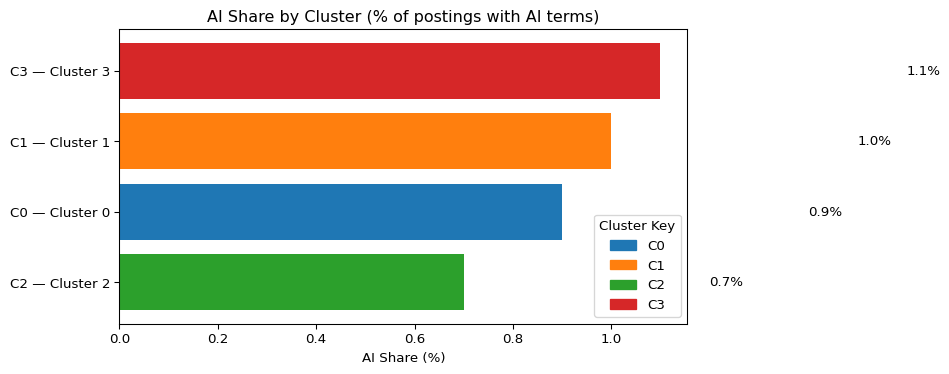
from sklearn.decomposition import TruncatedSVD # used later  
  
kmeans = KMeans(n\_clusters=best\_k, n\_init=20, random\_state=RANDOM\_STATE)  
labels = kmeans.fit\_predict(X)  
  
# Build a compact frame with inputs + cluster id  
df\_clusters = df[[text\_col] + cat\_cols + num\_cols].copy()  
df\_clusters["cluster"] = 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")

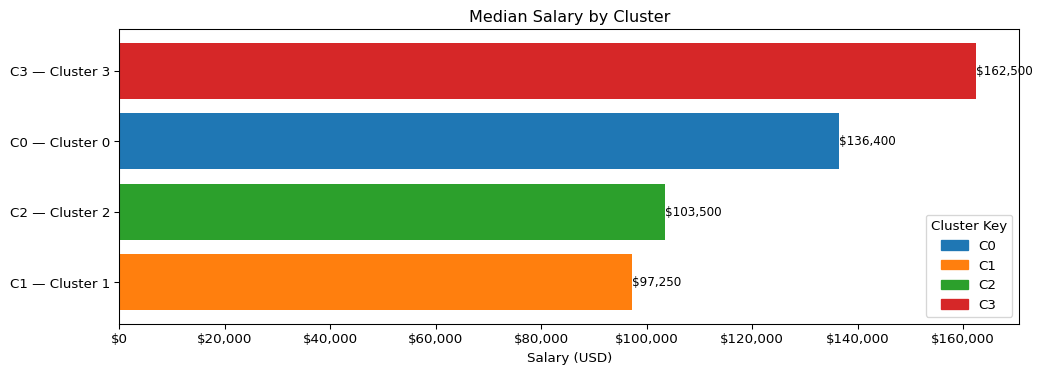
Cluster sizes:  
 cluster  
0 1673  
1 44869  
2 12657  
3 13299  
Name: count, dtype: int64  
Top terms file saved -> output/cluster\_top\_terms.txt

# 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\_59115/2483979144.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 | 1673 | 0.9 | 136400.0 |
| 1 | 1 | Cluster 1 | 44869 | 1.0 | 97250.0 |
| 2 | 2 | Cluster 2 | 12657 | 0.7 | 103500.0 |
| 3 | 3 | Cluster 3 | 13299 | 1.1 | 162500.0 |

# 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): 0.3385352006257032

# 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).

# 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: -0.001 | Homogeneity: 0.038 | Completeness: 0.000 | V-measure: 0.000  
  
Cluster → majority reference label:  
  
Overall purity: 0.999

# 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):

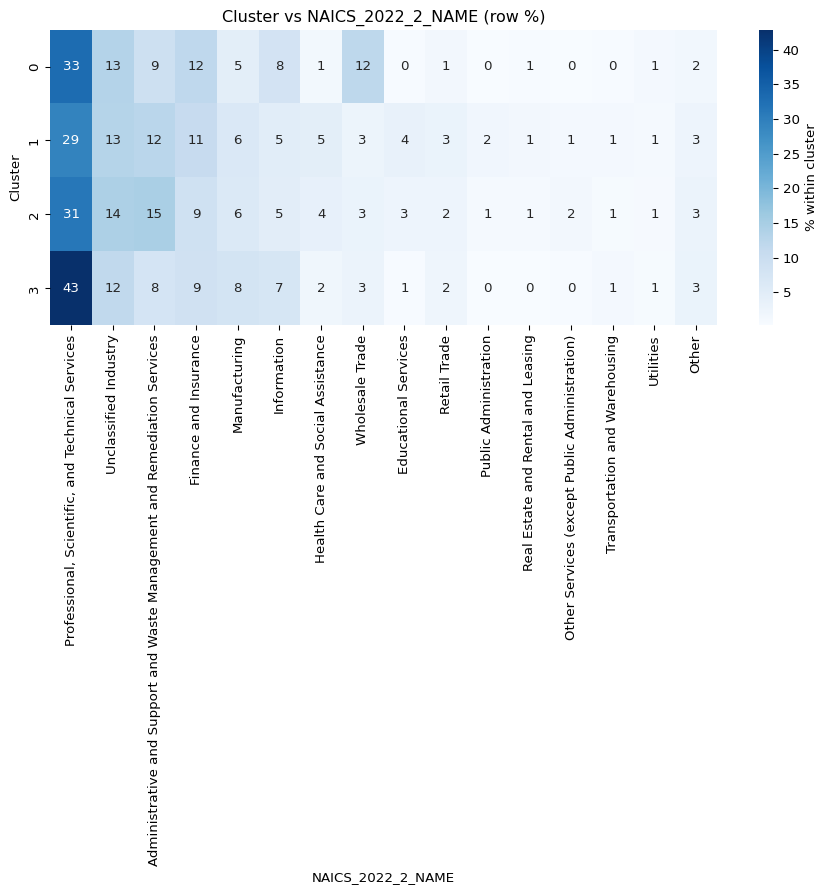
|  | column | non\_null\_share | n\_unique\_nonnull | top5 |
| --- | --- | --- | --- | --- |
| 0 | NAICS\_2022\_6\_NAME | 0.999393 | 814 | [Unclassified Industry, Custom Computer Progra... |
| 1 | NAICS\_2022\_4\_NAME | 0.999393 | 294 | [Computer Systems Design and Related Services,... |
| 2 | NAICS\_2022\_2\_NAME | 0.999393 | 21 | [Professional, Scientific, and Technical Servi... |
| 3 | ONET\_NAME | 0.999393 | 1 | [Business Intelligence Analysts] |
| 4 | ONET\_2019\_NAME | 0.999393 | 1 | [Business Intelligence Analysts] |
| 5 | SOC\_2021\_5\_NAME | 0.999393 | 1 | [Data Scientists] |
| 6 | SOC\_2021\_3\_NAME | 0.999393 | 1 | [Mathematical Science Occupations] |
| 7 | SOC\_2021\_2\_NAME | 0.999393 | 1 | [Computer and Mathematical Occupations] |

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

# 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\_59115/2086638697.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.

# 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.