All Analyses

Table of contents

# 1. The Next Normal: AI-Driven Analytics in Action

### 1.0.1 Affiliated With: Boston University-MET

#### 1.0.1.1 Report Written By:

Panyang Xiang

Pratham Kabra

Pranjul Garg

Binderiya Dugersuren

 Top Skills

 AI Skills Now a Must

 Tech & Finance Leading Hiring

 AI Gives Analysts an Edge

### 1.0.2 Goal of the Project:

In this research project, we wanted explore how the fields of Business Analytics, Data Science, and Machine Learning are evolving in 2025. With industries rapidly embracing AI technologies, understanding hiring trends and skill demands has become essential for students and professionals alike.

**Our Research Question:**

* What are the most in-demand skills for data science, business analytics, and ML roles?
* Have job descriptions evolved in 2024 to require more AI/ML expertise?
* What industries are hiring the most data scientists and why?
* What is the career outlook for business analytics professionals?

To answer these questions, we analyzed a large dataset of real job postings sourced from Lightcast. We applied data cleaning, exploratory analysis, skill extraction, and machine learning classification techniques to identify emerging trends and skill gaps.

#### 1.0.2.1 Why This Matters?

The rise of artificial intelligence (AI) and automation is not only transforming industries, it is changing the very skills required to succeed in the job market. Recent research shows that 86% of workers express concerns about AI-driven job displacement [@samuels2024ai], while businesses simultaneously seek employees who can work alongside AI tools and leverage data-driven insights to create value [@gartner2024ai].

As companies invest heavily in artificial intelligence technologies with an estimated $2.5 million average investment per organization in 2024 [@gartner2024ai] the demand for advanced technical skills continues to rise. Skills such as machine learning, Python programming, cloud computing, and data visualization have shifted from being optional to essential in many job descriptions. This trend reflects a broader transformation in workforce needs, where companies are not just adopting AI tools but are actively restructuring roles to emphasize data-driven decision making and technical proficiency. Professionals who are able to integrate AI and analytics into business processes will be better positioned for leadership and innovation roles in the years ahead.

The Future of Jobs Report 2025 by the World Economic Forum highlights that technological change, AI integration, and the green transition are reshaping global labor markets. Employers expect AI, big data, and automation to be the most transformative forces, with 60% of surveyed organizations anticipating these technologies to significantly impact their business models by 2030. As a result, roles requiring skills in AI, data analysis, cybersecurity, and technological literacy are projected to grow rapidly, while clerical and manual jobs face decline. Importantly, companies are prioritizing workforce reskilling, with 70% planning to train employees to meet emerging demands, and 63% identifying skill gaps as a major barrier to business transformation. These trends align with our findings, which show a clear advantage for business analytics and data science professionals who develop AI and advanced data skills to stay competitive in a rapidly evolving job market [@wef2025futurejobs].

Far from replacing all jobs, AI is reshaping roles to focus on higher value, decision driven tasks [@beverlywhite2024; @nigelrichardson2024]. As a result, professionals who upskill in AI, machine learning, and data analytics are better positioned for career advancement in a future where AI-human collaboration is key. Our project investigates these trends through real-world job posting analysis, revealing how demand for skills is shifting and where opportunities are growing.

## 1.1 Key Trends Shaping Data Science and Analytics Careers

### 1.1.1 Top Skills

Our analysis found that **Python, Machine Learning, and Cloud Computing** consistently rank among the top-requested skills in job postings. Employers are seeking candidates who can not only analyze data but also deploy models and build scalable solutions. Python remains the foundational language across roles, while machine learning capabilities and cloud platform expertise such as AWS or Azure offer clear competitive advantages.

### 1.1.2 AI Skills Now a Must

Compared to prior years, 2025 job postings show a notable shift: companies are explicitly requesting skills like Artificial Intelligence, Machine Learning, and Deep Learning. This highlights how AI technologies are no longer “nice-to-have” but are becoming core to business operations. Candidates without exposure to AI tools or methods risk being overlooked even for traditionally non-technical analytics roles.

### 1.1.3 Tech & Finance Leading Hiring

Our industry breakdown shows that technology and finance companies are the heaviest recruiters of data science talent. Tech firms are driving innovation through AI products, while finance companies leverage predictive analytics for risk management and investment strategies. These sectors offer strong opportunities, but they also expect candidates to have technical depth combined with business problem-solving skills.

### 1.1.4 AI Gives Analysts an Edge

Business analytics continues to grow across industries, but the professionals who can blend classic analytics with AI-driven insights are positioned for the best opportunities. Companies increasingly value analysts who can not just interpret historical data, but also build predictive models and optimize decisions using machine learning. Upskilling in AI and data science is no longer optional for career advancement in this field.

Back to top ↑

# 2. Cleaning

import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
import missingno as msno  
import plotly.express as px  
import numpy as np  
import plotly.graph\_objects as go  
  
df = pd.read\_parquet("data/lightcast.parquet")  
  
  
columns\_to\_keep = [  
 'COMPANY', 'LOCATION', 'POSTED', 'MIN\_EDULEVELS\_NAME', 'MAX\_EDULEVELS\_NAME',  
 'MIN\_YEARS\_EXPERIENCE', 'MAX\_YEARS\_EXPERIENCE', 'TITLE', 'SKILLS',  
 'SPECIALIZED\_SKILLS', 'CERTIFICATIONS', 'COMMON\_SKILLS', 'SOFTWARE\_SKILLS',  
 'SOC\_2021\_4\_NAME', 'NAICS\_2022\_6', 'NAICS2\_NAME', 'REMOTE\_TYPE\_NAME',  
 'SALARY', 'TITLE\_NAME', 'SKILLS\_NAME', 'SPECIALIZED\_SKILLS\_NAME', 'BODY'  
]  
eda\_data = df[columns\_to\_keep]

missing\_matrix = eda\_data.isnull().astype(int)  
corr = missing\_matrix.corr().round(2)  
  
mask = np.triu(np.ones(corr.shape), k=1).astype(bool)  
masked\_corr = corr.mask(mask)  
  
text\_labels = masked\_corr.astype(str)  
text\_labels[masked\_corr.isna()] = ""  
  
# plot  
fig = go.Figure(data=go.Heatmap(  
 z=masked\_corr.values,  
 x=masked\_corr.columns,  
 y=masked\_corr.index,  
 text=text\_labels.values,  
 texttemplate="%{text}",  
 colorscale="Blues",  
 colorbar=dict(title="Missing Corr"),  
 zmin=0,  
 zmax=1,  
 hoverinfo='skip'  
))  
  
fig.update\_layout(  
 title="Clean Triangle Missing Value Correlation Heatmap",  
 xaxis\_tickangle=45,  
 width=850,  
 height=600,  
 margin=dict(t=50, l=80, r=50, b=80),  
 font=dict(size=8),  
 plot\_bgcolor='white'  
)  
  
fig.update\_yaxes(autorange="reversed")  
  
fig.write\_html(  
 'figures/missing\_corr\_heatmap.html',  
 include\_plotlyjs='cdn', # lightweight HTML  
 full\_html=False # so you can embed easily  
)

This triangle heatmap visualizes the correlation of missing values between different columns in the dataset. Each square represents how often two columns are missing together, with darker blue indicating a stronger relationship. Most of the values are very high (close to 1.0), suggesting that when one column is missing, others are often missing too — especially among skill-related fields like SKILLS, SPECIALIZED\_SKILLS, and SOFTWARE\_SKILLS, which are likely part of the same job posting metadata.

This pattern indicates that missingness is not random, but structured — possibly due to differences in how job descriptions are recorded across roles or industries. For example, a job with no software skill tags might also lack common skills or NAICS codes, hinting at data input gaps rather than actual job content differences. Recognizing these correlations is helpful for choosing imputation strategies or deciding whether to drop certain rows or columns entirely during preprocessing.

if "SALARY" in eda\_data.columns:  
 eda\_data["SALARY"].fillna(eda\_data["SALARY"].median(), inplace=True)  
else:  
 print("Warning: 'SALARY' column not found in dataframe!")  
  
if "COMPANY" in eda\_data.columns:  
 eda\_data["COMPANY"].fillna("Unknown", inplace=True)  
else:  
 print("Warning: 'COMPANY' column not found in dataframe!")  
  
 # Fill numeric columns with mean  
num\_cols = eda\_data.select\_dtypes(include='number').columns  
for col in num\_cols:  
 if eda\_data[col].isnull().sum() > 0:  
 eda\_data[col].fillna(eda\_data[col].mean(), inplace=True)  
  
# Fill categorical columns with mode  
cat\_cols = eda\_data.select\_dtypes(include='object').columns  
for col in cat\_cols:  
 if eda\_data[col].isnull().sum() > 0:  
 eda\_data[col].fillna(eda\_data[col].mode()[0], inplace=True)  
  
eda\_data.dropna(thresh=len(eda\_data) \* 0.5, axis=1, inplace=True)  
  
  
# delete duplicates  
eda\_data = eda\_data.drop\_duplicates(subset=["TITLE", "COMPANY", "LOCATION", "POSTED","BODY"])  
eda\_data['BODY'] = eda\_data['BODY'].str.slice(0, 1000)  
eda\_data['BODY'] = eda\_data['BODY'].astype(str)  
eda\_data['COMPANY'] = eda\_data['COMPANY'].astype(str)

import pandas as pd  
eda\_data.to\_parquet('data/eda.parquet', engine='pyarrow', compression='gzip')

# 3. Exploratory Data Analytics

import pandas as pd  
eda = pd.read\_parquet("data/eda.parquet")

# identifying data analyst jobs by keyword searching  
keywords = ['Data Analyst', 'Business Analyst', 'Data Engineering', 'Deep Learning',  
 'Data Science', 'Data Analysis','Data Analytics', 'Market Research Analyst'   
 'LLM', 'Language Model', 'NLP', 'Natural Language Processing',  
 'Computer Vision', 'Business Intelligence Analyst', 'Quantitative Analyst', 'Operations Analyst']  
  
match = lambda col: eda[col].str.contains('|'.join(keywords), case=False, na=False)  
  
eda['DATA\_ANALYST\_JOB'] = match('TITLE\_NAME') \  
 | match('SKILLS\_NAME') \  
 | match('SPECIALIZED\_SKILLS\_NAME')   
eda['DATA\_ANALYST\_JOB'].value\_counts()

DATA\_ANALYST\_JOB  
False 38212  
True 33042  
Name: count, dtype: int64

import plotly.graph\_objects as go  
from plotly.subplots import make\_subplots  
  
df\_grouped = (  
 eda  
 .groupby(['DATA\_ANALYST\_JOB','NAICS2\_NAME'])  
 .size()  
 .reset\_index(name='Job\_Count')  
)  
  
short\_names = {  
 'Professional, Scientific, and Technical Services': 'Prof. Services',  
 'Administrative and Support and Waste Management and Remediation Services': 'Admin & Waste Mgmt',  
 'Health Care and Social Assistance': 'Healthcare',  
 'Finance and Insurance': 'Finance',  
 'Information': 'Info Tech',  
 'Educational Services': 'Education',  
 'Manufacturing': 'Manufacturing',  
 'Retail Trade': 'Retail',  
 'Accommodation and Food Services': 'Hospitality',  
 'Other Services (except Public Administration)': 'Other Services'  
}  
df\_grouped['Industry'] = df\_grouped['NAICS2\_NAME'].map(short\_names).fillna(df\_grouped['NAICS2\_NAME'])  
df\_grouped['Job\_Type'] = df\_grouped['DATA\_ANALYST\_JOB'].map({True:'True', False:'False'})  
  
pivot = (  
 df\_grouped  
 .pivot\_table(index='Industry', columns='Job\_Type', values='Job\_Count', fill\_value=0)  
 .reset\_index()  
)  
industries = pivot['Industry'].tolist()  
y\_true = pivot['True'].tolist()  
y\_false = pivot['False'].tolist()  
  
  
# 2) Build a 2-row subplot: bar on top, table below  
  
fig = make\_subplots(  
 rows=2, cols=1,  
 row\_heights=[0.70, 0.30], # give a bit more room to the table  
 specs=[[{"type":"bar"}],[{"type":"table"}]],  
 vertical\_spacing=0.12 # more space between bar and table  
)  
  
colors = {'True': '#FFE5E5', 'False': '#FF6B6B'}  
  
fig.add\_trace(  
 go.Bar(  
 x=industries, y=y\_true, name='True',  
 marker=dict(color=colors['True'], line=dict(color='#A81D1D', width=1)),  
 text=y\_true, textposition='outside'  
 ),  
 row=1, col=1  
)  
fig.add\_trace(  
 go.Bar(  
 x=industries, y=y\_false, name='False',  
 marker=dict(color=colors['False'], line=dict(color='#A81D1D', width=1)),  
 text=y\_false, textposition='outside'  
 ),  
 row=1, col=1  
)  
  
  
  
# 3) Slider steps: 0 → 8 000 in 200s  
  
steps = []  
for val in range(0, 8001, 200):  
 steps.append(dict(  
 label=str(val),  
 method="update",  
 args=[  
 {"y": [  
 [v if v>=val else 0 for v in y\_true],  
 [v if v>=val else 0 for v in y\_false]  
 ]},  
 {"title": f"Min Jobs ≥ {val:,}"}  
 ]  
 ))  
  
  
# 4) Final layout tweaks  
  
fig.update\_layout(  
 # lift slider above everything  
 sliders=[dict(  
 active=0,  
 currentvalue={"prefix":"Min Jobs: "},  
 pad={"b":0},  
 x=0.05,  
 y=1.05, # move slider way above the plot area  
 xanchor="left",  
 yanchor="bottom",  
 len=0.7,  
 font=dict(color='#A81D1D'),  
 steps=steps  
 )],  
  
 title=dict(  
 text="Data & Business Analytics Job Trends",  
 font=dict(size=24, color='#A81D1D'),  
 x=0.5,  
 y=0.95, # drop the title just below the slider  
 xanchor="center",  
 yanchor="top"  
 ),  
  
 width=1100, height=850,  
 margin=dict(l=60, r=60, t=180, b=200), # extra top & bottom margin  
  
 plot\_bgcolor='white',  
 paper\_bgcolor='white',  
  
 xaxis=dict(  
 title="Industry",  
 title\_font=dict(size=16, color='#A81D1D'),  
 tickmode='array',  
 tickvals=list(range(len(industries))),  
 ticktext=industries,  
 tickangle=-30,  
 tickfont=dict(size=11, color='#333'),  
 showline=True, linecolor='#A81D1D'  
 ),  
 yaxis=dict(  
 title="Number of Jobs",  
 title\_font=dict(size=16, color='#A81D1D'),  
 tickfont=dict(size=11, color='#333'),  
 gridcolor='rgba(200,200,200,0.3)',  
 showline=True, linecolor='#A81D1D',  
 range=[0, max(max(y\_true),max(y\_false))\*1.2]  
 ),  
  
 legend=dict(  
 title="Data Analyst Job",  
 title\_font=dict(color='#A81D1D'),  
 font=dict(size=12),  
 x=0.95, y=0.95  
 ),  
  
 bargap=0.2  
)  
  
fig.write\_html(  
 "figures/edaplot1.html",  
 include\_plotlyjs="cdn", # Use CDN to load Plotly JS  
 full\_html=False # Only include the plot div  
)

import plotly.express as px  
import pandas as pd  
  
# Prepare the data  
df = eda.copy()  
  
# Define analytics jobs (Data Analyst + Business Analyst)  
def classify\_analytics\_job(row):  
 if row['DATA\_ANALYST\_JOB']:  
 return True  
 title = str(row['TITLE\_NAME']).lower() if 'TITLE\_NAME' in row else str(row['TITLE']).lower()  
 if 'business analyst' in title:  
 return True  
 return False  
  
df['IS\_ANALYTICS\_JOB'] = df.apply(classify\_analytics\_job, axis=1)  
df['Job\_Category'] = df['IS\_ANALYTICS\_JOB'].map({True: 'Analytics Job', False: 'Non-Analytics Job'})  
  
# Create the box plot  
fig = px.box(df,   
 x='REMOTE\_TYPE\_NAME',   
 y='SALARY',   
 color='Job\_Category',  
 title='Salary Distribution by Remote Type for Analytics vs Non-Analytics Jobs',  
 labels={'REMOTE\_TYPE\_NAME': 'Remote Type', 'SALARY': 'Salary ($)', 'Job\_Category': 'Job Category'},  
 color\_discrete\_map={'Analytics Job': '#FF6B6B', 'Non-Analytics Job': '#4ECDC4'})  
  
# Beautify the layout with a red-white theme (no gradients)  
fig.update\_layout(  
 width=900,  
 height=600,  
 plot\_bgcolor='#FFFFFF', # Plain white background  
 paper\_bgcolor='#FFFFFF', # Plain white background  
 font=dict(family="Inter, sans-serif", size=14, color="#2D3748"),  
 title=dict(  
 font=dict(size=24, color="#FF6B6B"), # Red title for theme  
 x=0.5,  
 xanchor="center",  
 y=0.95,  
 yanchor="top"  
 ),  
 xaxis=dict(  
 title="Remote Type",  
 title\_font=dict(size=16),  
 tickfont=dict(size=12),  
 gridcolor="#E2E8F0",  
 linecolor="#2D3748",  
 linewidth=2,  
 showline=True  
 ),  
 yaxis=dict(  
 title="Salary ($)",  
 title\_font=dict(size=16),  
 tickfont=dict(size=12),  
 gridcolor="#E2E8F0",  
 linecolor="#2D3748",  
 linewidth=2,  
 showline=True,  
 showgrid=True,  
 zeroline=False  
 ),  
 legend=dict(  
 title="Job Category",  
 font=dict(size=13),  
 bgcolor="#FFFFFF",  
 bordercolor="#FF6B6B", # Red border for theme  
 borderwidth=1,  
 x=1.02,  
 y=0.5,  
 xanchor="left",  
 yanchor="middle"  
 ),  
 hovermode="closest",  
 hoverlabel=dict(  
 bgcolor="#FFFFFF",  
 font\_size=12,  
 font\_family="Inter, sans-serif",  
 font\_color="#2D3748",  
 bordercolor="#FF6B6B" # Red border for hover  
 )  
)  
  
fig.write\_html(  
 "figures/edaplot2.html",  
 include\_plotlyjs="cdn", # Use CDN to load Plotly JS  
 full\_html=False # Only include the plot div  
)

import plotly.express as px  
import pandas as pd  
  
# Prepare the data  
df = eda.copy()  
  
# Define analytics jobs (Data Analyst + Business Analyst)  
def classify\_analytics\_job(row):  
 if row['DATA\_ANALYST\_JOB']:  
 return True  
 title = str(row['TITLE\_NAME']).lower() if 'TITLE\_NAME' in row else str(row['TITLE']).lower()  
 if 'business analyst' in title:  
 return True  
 return False  
  
df['IS\_ANALYTICS\_JOB'] = df.apply(classify\_analytics\_job, axis=1)  
df['Job\_Category'] = df['IS\_ANALYTICS\_JOB'].map({True: 'Analytics Job', False: 'Non-Analytics Job'})  
  
# Group by industry and job category  
df\_grouped = df.groupby(['NAICS2\_NAME', 'IS\_ANALYTICS\_JOB']).size().reset\_index(name='Job\_Count')  
df\_grouped['Job\_Category'] = df\_grouped['IS\_ANALYTICS\_JOB'].map({True: 'Analytics Job', False: 'Non-Analytics Job'})  
  
# Shorten industry names for better readability  
short\_map = {  
 'Professional, Scientific, and Technical Services': 'Prof. Services',  
 'Administrative and Support and Waste Management and Remediation Services': 'Admin & Waste Mgmt',  
 'Health Care and Social Assistance': 'Healthcare',  
 'Finance and Insurance': 'Finance',  
 'Information': 'Info Tech',  
 'Educational Services': 'Education',  
 'Manufacturing': 'Manufacturing',  
 'Retail Trade': 'Retail',  
 'Accommodation and Food Services': 'Hospitality',  
 'Other Services (except Public Administration)': 'Other Services'  
}  
df\_grouped['Industry'] = df\_grouped['NAICS2\_NAME'].map(short\_map).fillna(df\_grouped['NAICS2\_NAME'])  
  
# Create the stacked bar chart  
fig = px.bar(df\_grouped,   
 x='Industry',   
 y='Job\_Count',   
 color='Job\_Category',  
 title='Top Industries Hiring Analytics Jobs',  
 labels={'Industry': 'Industry', 'Job\_Count': 'Number of Jobs', 'Job\_Category': 'Job Category'},  
 barmode='stack',  
 color\_discrete\_map={'Analytics Job': '#FF6B6B', 'Non-Analytics Job': '#4ECDC4'})  
  
# Beautify the layout with a red-white theme (no gradients)  
fig.update\_layout(  
 width=1000,  
 height=600,  
 plot\_bgcolor='#FFFFFF', # Plain white background  
 paper\_bgcolor='#FFFFFF', # Plain white background  
 font=dict(family="Inter, sans-serif", size=14, color="#2D3748"),  
 title=dict(  
 font=dict(size=24, color="#FF6B6B"), # Red title for theme  
 x=0.5,  
 xanchor="center",  
 y=0.95,  
 yanchor="top"  
 ),  
 xaxis=dict(  
 title="Industry",  
 title\_font=dict(size=16),  
 tickfont=dict(size=12),  
 tickangle=-45,  
 gridcolor="#E2E8F0",  
 linecolor="#2D3748",  
 linewidth=2,  
 showline=True  
 ),  
 yaxis=dict(  
 title="Number of Jobs",  
 title\_font=dict(size=16),  
 tickfont=dict(size=12),  
 gridcolor="#E2E8F0",  
 linecolor="#2D3748",  
 linewidth=2,  
 showline=True,  
 showgrid=True,  
 zeroline=False  
 ),  
 legend=dict(  
 title="Job Category",  
 font=dict(size=13),  
 bgcolor="#FFFFFF",  
 bordercolor="#FF6B6B", # Red border for theme  
 borderwidth=1,  
 x=1.02,  
 y=0.5,  
 xanchor="left",  
 yanchor="middle"  
 ),  
 hovermode="closest",  
 hoverlabel=dict(  
 bgcolor="#FFFFFF",  
 font\_size=12,  
 font\_family="Inter, sans-serif",  
 font\_color="#2D3748",  
 bordercolor="#FF6B6B" # Red border for hover  
 )  
)  
  
fig.write\_html(  
 "figures/edaplot3.html",  
 include\_plotlyjs="cdn", # Use CDN to load Plotly JS  
 full\_html=False # Only include the plot div  
)

import plotly.express as px  
import pandas as pd  
  
# Prepare the data  
df = eda.copy()  
  
# Define analytics jobs (Data Analyst + Business Analyst)  
def classify\_analytics\_job(row):  
 if row['DATA\_ANALYST\_JOB']:  
 return True  
 title = str(row['TITLE\_NAME']).lower() if 'TITLE\_NAME' in row else str(row['TITLE']).lower()  
 if 'business analyst' in title:  
 return True  
 return False  
  
df['IS\_ANALYTICS\_JOB'] = df.apply(classify\_analytics\_job, axis=1)  
df['Job\_Category'] = df['IS\_ANALYTICS\_JOB'].map({True: 'Analytics Job', False: 'Non-Analytics Job'})  
  
# Calculate average years of experience  
df['Avg\_Years\_Experience'] = (df['MIN\_YEARS\_EXPERIENCE'] + df['MAX\_YEARS\_EXPERIENCE']) / 2  
  
# Clean the data (remove rows with missing salary or experience)  
df = df.dropna(subset=['Avg\_Years\_Experience', 'SALARY'])  
  
# Create the scatter plot with trend line  
fig = px.scatter(df,   
 x='Avg\_Years\_Experience',   
 y='SALARY',   
 color='Job\_Category',  
 trendline='ols', # Add trend line (ordinary least squares)  
 title='Experience Requirements vs Salary for Analytics Jobs',  
 labels={'Avg\_Years\_Experience': 'Average Years of Experience', 'SALARY': 'Salary ($)', 'Job\_Category': 'Job Category'},  
 color\_discrete\_map={'Analytics Job': '#FF6B6B', 'Non-Analytics Job': '#4ECDC4'})  
  
# Beautify the layout with a red-white theme (no gradients)  
fig.update\_layout(  
 width=900,  
 height=600,  
 plot\_bgcolor='#FFFFFF', # Plain white background  
 paper\_bgcolor='#FFFFFF', # Plain white background  
 font=dict(family="Inter, sans-serif", size=14, color="#2D3748"),  
 title=dict(  
 font=dict(size=24, color="#FF6B6B"), # Red title for theme  
 x=0.5,  
 xanchor="center",  
 y=0.95,  
 yanchor="top"  
 ),  
 xaxis=dict(  
 title="Average Years of Experience",  
 title\_font=dict(size=16),  
 tickfont=dict(size=12),  
 gridcolor="#E2E8F0",  
 linecolor="#2D3748",  
 linewidth=2,  
 showline=True,  
 showgrid=True,  
 zeroline=False  
 ),  
 yaxis=dict(  
 title="Salary ($)",  
 title\_font=dict(size=16),  
 tickfont=dict(size=12),  
 gridcolor="#E2E8F0",  
 linecolor="#2D3748",  
 linewidth=2,  
 showline=True,  
 showgrid=True,  
 zeroline=False  
 ),  
 legend=dict(  
 title="Job Category",  
 font=dict(size=13),  
 bgcolor="#FFFFFF",  
 bordercolor="#FF6B6B", # Red border for theme  
 borderwidth=1,  
 x=1.02,  
 y=0.5,  
 xanchor="left",  
 yanchor="middle"  
 ),  
 hovermode="closest",  
 hoverlabel=dict(  
 bgcolor="#FFFFFF",  
 font\_size=12,  
 font\_family="Inter, sans-serif",  
 font\_color="#2D3748",  
 bordercolor="#FF6B6B" # Red border for hover  
 )  
)  
  
# Customize scatter points  
fig.update\_traces(  
 marker=dict(  
 size=8,  
 opacity=0.7,  
 line=dict(width=1, color="#2D3748")  
 )  
)  
  
fig.write\_html(  
 "figures/edaplot4.html",  
 include\_plotlyjs="cdn", # Use CDN to load Plotly JS  
 full\_html=False # Only include the plot div  
)

import plotly.graph\_objects as go  
import pandas as pd  
  
# Prepare the data  
df = eda.copy()  
  
# Define analytics jobs (Data Analyst + Business Analyst)  
def classify\_analytics\_job(row):  
 if row['DATA\_ANALYST\_JOB']:  
 return True  
 title = str(row['TITLE\_NAME']).lower() if 'TITLE\_NAME' in row else str(row['TITLE']).lower()  
 if 'business analyst' in title:  
 return True  
 return False  
  
df['IS\_ANALYTICS\_JOB'] = df.apply(classify\_analytics\_job, axis=1)  
df['Job\_Category'] = df['IS\_ANALYTICS\_JOB'].map({True: 'Analytics Job', False: 'Non-Analytics Job'})  
  
# Filter for Analytics jobs only  
df\_analytics = df[df['IS\_ANALYTICS\_JOB']].copy()  
  
# Clean the data (remove rows with missing industry)  
df\_analytics = df\_analytics.dropna(subset=['NAICS2\_NAME'])  
  
# Group by job category and industry to get job counts  
df\_grouped = df\_analytics.groupby(['Job\_Category', 'NAICS2\_NAME']).size().reset\_index(name='Job\_Count')  
  
# Shorten industry names for better readability  
short\_map = {  
 'Professional, Scientific, and Technical Services': 'Prof. Services',  
 'Administrative and Support and Waste Management and Remediation Services': 'Admin & Waste Mgmt',  
 'Health Care and Social Assistance': 'Healthcare',  
 'Finance and Insurance': 'Finance',  
 'Information': 'Info Tech',  
 'Educational Services': 'Education',  
 'Manufacturing': 'Manufacturing',  
 'Retail Trade': 'Retail',  
 'Accommodation and Food Services': 'Hospitality',  
 'Other Services (except Public Administration)': 'Other Services'  
}  
df\_grouped['NAICS2\_NAME'] = df\_grouped['NAICS2\_NAME'].map(short\_map).fillna(df\_grouped['NAICS2\_NAME'])  
  
# Prepare data for Sankey Diagram  
# Create a list of unique labels (nodes)  
labels = list(df\_grouped['Job\_Category'].unique()) + list(df\_grouped['NAICS2\_NAME'].unique())  
  
# Create source and target indices  
source = [labels.index(job\_cat) for job\_cat in df\_grouped['Job\_Category']]  
target = [labels.index(industry) for industry in df\_grouped['NAICS2\_NAME']]  
value = df\_grouped['Job\_Count'].tolist()  
  
# Create the Sankey Diagram  
fig = go.Figure(data=[go.Sankey(  
 node=dict(  
 pad=15,  
 thickness=20,  
 line=dict(color="#2D3748", width=0.5),  
 label=labels,  
 color="#FF6B6B" # Red nodes for the theme  
 ),  
 link=dict(  
 source=source,  
 target=target,  
 value=value,  
 color="rgba(255, 107, 107, 0.5)" # Semi-transparent red links  
 )  
)])  
  
# Beautify the layout with a red-white theme (no gradients)  
fig.update\_layout(  
 width=900,  
 height=600,  
 plot\_bgcolor='#FFFFFF', # Plain white background  
 paper\_bgcolor='#FFFFFF', # Plain white background  
 font=dict(family="Inter, sans-serif", size=14, color="#2D3748"),  
 title=dict(  
 text='Distribution of Analytics Job Postings by Industry',  
 font=dict(size=24, color="#FF6B6B"), # Red title for theme  
 x=0.5,  
 xanchor="center",  
 y=0.95,  
 yanchor="top"  
 ),  
 margin=dict(l=20, r=20, t=80, b=20),  
)  
fig.write\_html(  
 "figures/edaplot5.html",  
 include\_plotlyjs="cdn", # Use CDN to load Plotly JS  
 full\_html=False # Only include the plot div  
)

# 4. Skill Gap Analysis

import pandas as pd  
df = pd.read\_parquet("data/eda.parquet")

## 4.1 Group 11 Skill

import pandas as pd  
  
skills\_data = {  
 "Name": ["Binderiya", "Pranjul", "Pratham", "Panyang"],  
 "Python": [4, 4, 5, 3],  
 "SQL": [4, 4, 5, 4],  
 "Machine Learning": [2, 3, 2, 2],  
 "PySpark": [3, 3, 3, 3],  
 "Excel": [4, 5, 5, 4],  
 "Data Visualization": [5, 5, 3, 3],  
 "Power Bi/ Tableau": [4, 5, 3, 4],  
 "Version Control Git": [4, 4, 3, 3],  
 "ETL/Data pipeline": [3, 2, 1, 2],  
 "Communication": [4, 4, 5, 3],  
 "Project Management": [5, 5, 5, 3],  
 "Cloud Computing": [4, 4, 2, 2]  
}  
  
df\_skills = pd.DataFrame(skills\_data)  
df\_skills.set\_index("Name", inplace=True)  
df\_skills

|  | Python | SQL | Machine Learning | PySpark | Excel | Data Visualization | Power Bi/ Tableau | Version Control Git | ETL/Data pipeline | Communication | Project Management | Cloud Computing |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name |  |  |  |  |  |  |  |  |  |  |  |  |
| Binderiya | 4 | 4 | 2 | 3 | 4 | 5 | 4 | 4 | 3 | 4 | 5 | 4 |
| Pranjul | 4 | 4 | 3 | 3 | 5 | 5 | 5 | 4 | 2 | 4 | 5 | 4 |
| Pratham | 5 | 5 | 2 | 3 | 5 | 3 | 3 | 3 | 1 | 5 | 5 | 2 |
| Panyang | 3 | 4 | 2 | 3 | 4 | 3 | 4 | 3 | 2 | 3 | 3 | 2 |

import plotly.express as px  
  
fig = px.imshow(  
 df\_skills,  
 text\_auto=True,   
 color\_continuous\_scale="YlGnBu",  
 aspect="auto"   
)  
  
fig.update\_layout(  
 title="Team Skill Levels Heatmap",  
 xaxis\_title="Skills",  
 yaxis\_title="Team Members",  
 width=700,  
 height=400,  
 margin=dict(l=50, r=20, t=50, b=50)  
)  
  
fig.write\_html(  
 "figures/skill\_gap\_plot1.html",  
 include\_plotlyjs="cdn",  
 full\_html=False  
)

import plotly.graph\_objects as go  
from IPython.display import IFrame  
fig = go.Figure()  
  
for name in df\_skills.index:  
 values = df\_skills.loc[name].tolist()  
 values += values[:1] # close the loop  
 fig.add\_trace(go.Scatterpolar(  
 r=values,  
 theta=df\_skills.columns.tolist() + [df\_skills.columns[0]],  
 fill='toself',  
 name=name  
 ))  
  
fig.update\_layout(  
 polar=dict(radialaxis=dict(visible=True, range=[0, 5])),  
 showlegend=True,  
 title='Team Skills Radar Chart'  
)  
  
fig.write\_html(  
 "figures/skill\_gap\_plot2.html",  
 include\_plotlyjs="cdn",  
 full\_html=False  
)

### 4.1.1 Interactive Radar Chart

From this radar chart visualization we can see that our team has a lot of room for improvement for skills like PySpark and Machine Learning. Also we can see that not a lot of our team mates are confident in their skills in Cloud Computing and ETL.

## 4.2 Top Skills

keywords = ['Data Analyst', 'Business Analyst', 'Data Engineering', 'Deep Learning',  
 'Data Science', 'Data Analysis','Data Analytics', 'Market Research Analyst'   
 'LLM', 'Language Model', 'NLP', 'Natural Language Processing',  
 'Computer Vision', 'Business Intelligence Analyst', 'Quantitative Analyst', 'Operations Analyst']  
  
match = lambda col: df[col].str.contains('|'.join(keywords), case=False, na=False)  
  
df['DATA\_ANALYST\_JOB'] = match('TITLE\_NAME') \  
 | match('SKILLS\_NAME') \  
 | match('SPECIALIZED\_SKILLS\_NAME')   
df['DATA\_ANALYST\_JOB'].value\_counts()

DATA\_ANALYST\_JOB  
False 38212  
True 33042  
Name: count, dtype: int64

import ast  
import pandas as pd  
import matplotlib.pyplot as plt  
import plotly.express as px  
  
# Safely apply literal\_eval only to non-null values  
df['SKILLS'] = df['SKILLS\_NAME'].apply(lambda x: ast.literal\_eval(x) if pd.notnull(x) else [])  
  
  
data\_skills = df[df['DATA\_ANALYST\_JOB']]['SKILLS'].explode().value\_counts().reset\_index()  
data\_skills.columns = ['Skill', 'Count']  
  
fig = px.bar(data\_skills, x='Skill', y='Count',  
 title="Top Skills",  
 labels={'Skill': 'Skill Name', 'Count': 'Frequency'},  
 color='Skill')  
df\_skills.index = df\_skills.index.str.strip()

from collections import defaultdict  
  
# Lowercase everything  
team\_skills = [s.lower().strip() for s in df\_skills.columns]  
job\_demand\_raw = data\_skills.copy()  
job\_demand\_raw['Skill'] = job\_demand\_raw['Skill'].str.lower().str.strip()  
  
# New dict to map cleaned team skill to total count from job postings  
skill\_demand\_map = defaultdict(int)  
  
for \_, row in job\_demand\_raw.iterrows():  
 skill\_in\_posting = row['Skill']  
 count = row['Count']  
 for team\_skill in team\_skills:  
 if team\_skill in skill\_in\_posting:  
 skill\_demand\_map[team\_skill] += count

team\_skills = [s.strip().lower() for s in df\_skills.columns]  
print("Team skills:", team\_skills)  
print(job\_demand\_raw['Skill'].head(10).tolist())  
for skill\_text in job\_demand\_raw['Skill'].head(10):  
 for team\_skill in team\_skills:  
 if team\_skill in skill\_text:  
 print(f" '{team\_skill}' found in: '{skill\_text}'")

Team skills: ['python', 'sql', 'machine learning', 'pyspark', 'excel', 'data visualization', 'power bi/ tableau', 'version control git', 'etl/data pipeline', 'communication', 'project management', 'cloud computing']  
['data analysis', 'sql (programming language)', 'communication', 'management', 'python (programming language)', 'tableau (business intelligence software)', 'dashboard', 'computer science', 'problem solving', 'power bi']  
 'sql' found in: 'sql (programming language)'  
 'communication' found in: 'communication'  
 'python' found in: 'python (programming language)'

for \_, row in job\_demand\_raw.iterrows():  
 skill\_text = row['Skill']  
 count = row['Count']  
 for team\_skill in team\_skills:  
 if team\_skill in skill\_text: # no regex, just substring  
 skill\_demand\_map[team\_skill] += count  
  
job\_demand = pd.Series(skill\_demand\_map)

job\_demand = pd.Series(skill\_demand\_map)  
job\_demand.name = "Count"  
team\_avg = df\_skills.mean()  
team\_avg.index = team\_avg.index.str.strip().str.lower()   
# Now match only overlapping skills  
common\_skills = job\_demand.index.intersection(team\_avg.index)  
team\_avg = team\_avg[common\_skills]  
job\_demand = job\_demand[common\_skills]  
  
# Normalize job demand  
job\_demand\_normalized = 5 \* (job\_demand / job\_demand.max())  
job\_demand\_normalized.name = "Job Demand (Normalized)"  
  
# Combine  
comparison\_df = pd.concat([team\_avg, job\_demand\_normalized], axis=1)  
comparison\_df.columns = ["Team Average Skill", "Job Demand (Normalized)"]  
comparison\_df["Skill Gap"] = comparison\_df["Job Demand (Normalized)"] - comparison\_df["Team Average Skill"]  
comparison\_df.sort\_values("Skill Gap", ascending=False, inplace=True)  
  
comparison\_df

|  | Team Average Skill | Job Demand (Normalized) | Skill Gap |
| --- | --- | --- | --- |
| communication | 4.00 | 5.000000 | 1.000000 |
| sql | 4.25 | 4.959112 | 0.709112 |
| machine learning | 2.25 | 0.869894 | -1.380106 |
| python | 4.00 | 2.283191 | -1.716809 |
| data visualization | 4.00 | 1.520015 | -2.479985 |
| excel | 4.50 | 1.959357 | -2.540643 |
| cloud computing | 3.00 | 0.266590 | -2.733410 |
| pyspark | 3.00 | 0.104878 | -2.895122 |
| project management | 4.50 | 1.530850 | -2.969150 |

comparison\_df = comparison\_df.reset\_index().rename(columns={"index": "Skill"})

import plotly.express as px  
  
fig = px.bar(  
 comparison\_df,  
 x='Skill',  
 y='Skill Gap',  
 color='Skill Gap',  
 color\_continuous\_scale='RdBu\_r',  
 title='Skill Gaps: Job Market Expectations vs. Team Capability',  
 labels={'Skill Gap': 'Gap (Job Demand - Team Skill)', 'Skill': 'Skill'},  
)  
  
fig.add\_hline(y=0, line\_dash='dash')  
fig.update\_layout(  
 xaxis\_tickangle=-45,  
 yaxis\_title='Gap (Positive = Market expects more)',  
 font=dict(size=13),  
 height=500,  
 plot\_bgcolor='white',  
)  
fig.write\_html(  
 "figures/skill\_gap\_plot3.html",  
 include\_plotlyjs="cdn",  
 full\_html=False  
)

Our skill gap analysis compares the team’s average skill levels with normalized job market expectations. The “Skill Gap” metric is calculated based on the frequency of each skill’s appearance in job postings, rather than a direct assessment of skill proficiency required by employers. As such, while skills like SQL and Python appear to have smaller gaps, this may partly reflect the fact that they are mentioned slightly less often in the postings relative to emerging areas like cloud computing or machine learning. Nevertheless, SQL and Python remain critical foundational skills that the job market consistently expects from candidates.

The results indicate that the team meets or slightly exceeds market expectations in communication and SQL. However, there are notable gaps in technical domains such as machine learning, cloud computing, PySpark, data visualization, and project management. These areas should be prioritized for upskilling to align better with market demand.

# 5. Random Forest Classification for ML/Data Science Requirement

#### 5.0.0.1 Loading the Dataset

import pandas as pd  
df = pd.read\_parquet("data/eda.parquet", engine='pyarrow')

ml\_keywords = ["machine learning", "data science", "ai", "artificial intelligence", "deep learning", "data scientist"]  
  
def requires\_ml(skills):  
 if pd.isnull(skills):  
 return 0  
 skills = skills.lower()  
 return int(any(kw in skills for kw in ml\_keywords))  
  
df["REQUIRES\_ML"] = df["SKILLS\_NAME"].apply(requires\_ml)

features = ["TITLE", "SOC\_2021\_4\_NAME", "NAICS2\_NAME", "MIN\_EDULEVELS\_NAME", "MIN\_YEARS\_EXPERIENCE"]  
target = "REQUIRES\_ML"  
  
df = df[features + [target, 'BODY']].dropna()

from sklearn.preprocessing import LabelEncoder  
  
df\_encoded = df.copy()  
label\_encoders = {}  
  
for col in features:  
 if df\_encoded[col].dtype == "object":  
 le = LabelEncoder()  
 df\_encoded[col] = le.fit\_transform(df\_encoded[col])  
 label\_encoders[col] = le

from sklearn.model\_selection import train\_test\_split  
  
X = df\_encoded[features]  
y = df\_encoded[target]  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report  
  
rf = RandomForestClassifier(n\_estimators=100, random\_state=42)  
rf.fit(X\_train, y\_train)  
  
y\_pred = rf.predict(X\_test)  
print(classification\_report(y\_test, y\_pred))

precision recall f1-score support  
  
 0 0.76 0.77 0.77 7933  
 1 0.71 0.70 0.70 6318  
  
 accuracy 0.74 14251  
 macro avg 0.74 0.73 0.73 14251  
weighted avg 0.74 0.74 0.74 14251

The Random Forest model achieved an overall accuracy of 74% when classifying whether a role requires ML skills. The precision and recall for class 0, non-ML, roles were slightly higher with precision: 0.76 and recall: 0.77 compared to class 1, ML roles, where both precision and recall were around 0.70. This indicates the model performs reasonably well but is slightly better at identifying non-ML roles than ML roles. Overall, the model shows decent predictive power, but there is some room for improvement, especially in detecting ML-related positions.

import plotly.express as px  
fig = px.bar(  
 x=rf.feature\_importances\_,  
 y=features,  
 orientation='h',  
 labels={'x': 'Importance', 'y': 'Feature'},  
 title='Feature Importance – ML Role Classification'  
)  
  
fig.update\_layout(  
 yaxis=dict(categoryorder='total ascending'),  
 margin=dict(l=100, r=20, t=50, b=20),  
 height=500,  
 template='plotly\_white'  
)  
  
fig.write\_html(  
 'figures/rm\_model\_plot1.html',  
 include\_plotlyjs='cdn',  
 full\_html=False  
)

This bar chart displays the feature importance scores from a random forest model predicting whether a job role involves ML/Data Science. The most influential feature by far in the model is the job title (TITLE), which has a significantly higher importance than all other variables. Secondary contributors include industry classification (NAICS2\_NAME) and minimum years of experience, where education level and SOC code had relatively low influence on the model’s prediction. This is suggesting that the job title alone carries strong predictive power for identifying ML-related roles.

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.feature\_extraction.text import TfidfVectorizer  
  
# Cleaned job descriptions  
df['BODY\_clean'] = df['BODY'].fillna("").str.lower()  
  
# Target  
y = df['REQUIRES\_ML'] # this should be a binary 1/0 column  
  
# TF-IDF vectorization  
tfidf = TfidfVectorizer(max\_features=5000, stop\_words='english')  
X = tfidf.fit\_transform(df['BODY\_clean'])

from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y)  
  
model = RandomForestClassifier(random\_state=42)  
model.fit(X\_train, y\_train)  
  
y\_pred = model.predict(X\_test)  
print(classification\_report(y\_test, y\_pred))

precision recall f1-score support  
  
 0 0.79 0.87 0.82 9991  
 1 0.80 0.70 0.75 7823  
  
 accuracy 0.79 17814  
 macro avg 0.80 0.78 0.79 17814  
weighted avg 0.79 0.79 0.79 17814

In this model, we incorporated the job description text by applying TF-IDF vectorization to extract the most important words from each posting. Using these features, the Random Forest model achieved a strong accuracy of 79%, a notable improvement compared to the previous model that relied only on structured fields like job titles and industries. The model shows a higher recall 87% for non-ML roles but a lower recall of 69% for ML-related roles, suggesting it is better at identifying traditional roles than detecting specialized data science positions. Overall, adding the job description significantly enhanced the model’s ability to capture complex signals related to AI/ML requirements across different industries.

import numpy as np  
importances = model.feature\_importances\_  
top\_idx = np.argsort(importances)[-20:]  
top\_words = tfidf.get\_feature\_names\_out()[top\_idx]  
top\_importances = importances[top\_idx]  
  
fig = px.bar(  
 x=top\_importances,  
 y=top\_words,  
 orientation='h',  
 labels={'x': 'Importance', 'y': 'Word'},  
 title='Top 20 TF-IDF Words for ML Role Classification'  
)  
  
fig.update\_layout(  
 yaxis={'categoryorder':'total ascending'},  
 margin=dict(l=120, r=20, t=60, b=20),  
 width=800, height=600, template='plotly\_white'  
)  
  
fig.write\_html(  
 'figures/rm\_model\_plot2.html',  
 include\_plotlyjs='cdn',  
 full\_html=False  
)

This bar chart shows the top words contributing to the classification of job roles as Machine Learning (ML)related based on job description data. Surprisingly, the most influential words are “attention,” “chain,” and “supply”, which could be an indication of overlap with supply chain roles or reflect noise in the model. More expected terms like “machine,” “learning,” “python,” “AI,” and “analytics” also appear, reinforcing that relevant technical language still plays a role in identifying ML-related positions. The presence of general words like “strong” or “communication” suggests that not all influential terms are strictly technical.

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay  
import numpy as np  
import plotly.figure\_factory as ff  
labels = [str(lbl) for lbl in model.classes\_]  
  
cm = confusion\_matrix(y\_test, y\_pred)  
labels = [str(c) for c in model.classes\_]   
  
fig = ff.create\_annotated\_heatmap(  
 z=cm,  
 x=labels,  
 y=labels,  
 colorscale='Blues',  
 showscale=True,  
 annotation\_text=cm,  
 hoverinfo='z'  
)  
  
fig.update\_layout(  
 title='Confusion Matrix – ML Role Classification',  
 xaxis\_title='Predicted Label',  
 yaxis\_title='Actual Label',  
 xaxis=dict(tickmode='array', tickvals=list(range(len(labels))), ticktext=labels),  
 yaxis=dict(tickmode='array', tickvals=list(range(len(labels))), ticktext=labels),  
 width=700,  
 height=600,  
 template='plotly\_white',  
 margin=dict(l=80, r=20, t=60, b=80)  
)  
  
fig.write\_html(  
 "figures/rm\_model\_plot3.html",  
 include\_plotlyjs='cdn',  
 full\_html=False  
)

The confusion matrix illustrates that while the model performs well overall, it is particularly strong at identifying non-ML roles, class 0, but faces more difficulty correctly predicting ML-related roles, class 1, reflected in a higher number of false negatives. Nevertheless, the integration of the unstructured data meaningfully improved classification performance.

Structured features such as TITLE, SOC\_2021\_4\_NAME, NAICS2\_NAME, MIN\_EDULEVELS\_NAME, and MIN\_YEARS\_EXPERIENCE were chosen based on domain relevance, these fields reflect the role’s function, industry, required education, and experience level, all of which can signal ML-related requirements. Additionally, we included the job description BODY text, applying TF-IDF vectorization to extract key terms. This allowed the model to learn from nuanced language patterns within postings. Feature importance and performance metrics confirm that both structured metadata and text data contribute meaningfully to classification accuracy.

# 6. Analytics Model

from sklearn.cluster import KMeans  
from sklearn.preprocessing import StandardScaler  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.preprocessing import LabelEncoder  
from sklearn.metrics import adjusted\_rand\_score  
eda = pd.read\_parquet("data/eda.parquet")

features = eda[['SALARY', 'MAX\_YEARS\_EXPERIENCE', 'MIN\_YEARS\_EXPERIENCE']].copy()  
  
for col in ['MAX\_YEARS\_EXPERIENCE', 'MIN\_YEARS\_EXPERIENCE', 'SALARY']:  
 features[col] = pd.to\_numeric(features[col], errors='coerce')  
  
features = features.dropna()  
  
scaler = StandardScaler()  
X = scaler.fit\_transform(features)  
  
kmeans = KMeans(n\_clusters=4, random\_state=688)  
eda.loc[features.index, 'Cluster'] = kmeans.fit\_predict(X)  
  
true\_labels = eda.loc[features.index, 'SOC\_2021\_4\_NAME']  
true\_labels\_encoded = LabelEncoder().fit\_transform(true\_labels)  
  
ari = adjusted\_rand\_score(true\_labels\_encoded, eda.loc[features.index, 'Cluster'])

import plotly.express as px  
import plotly.graph\_objects as go  
from IPython.display import HTML  
  
# 1) Build the DataFrame  
df\_plot = features.copy()  
df\_plot['Cluster'] = eda.loc[features.index, 'Cluster']  
  
# 2) Compute centroids in original units  
centroids = kmeans.cluster\_centers\_  
centroids\_x = centroids[:, 0] \* X.std(axis=0)[0] + X.mean(axis=0)[0]  
centroids\_y = centroids[:, 1] \* X.std(axis=0)[1] + X.mean(axis=0)[1]  
  
# 3) Create an interactive Plotly Figure  
fig = px.scatter(  
 df\_plot,  
 x='SALARY',  
 y='MAX\_YEARS\_EXPERIENCE',  
 color='Cluster',  
 title="KMeans Clustering by Salary and Max Years Experience",  
 labels={  
 'SALARY': 'Salary',  
 'MAX\_YEARS\_EXPERIENCE': 'Max Years Experience',  
 'Cluster': 'Cluster'  
 },  
 width=600,  
 height=400,  
)  
  
# 4) Add centroid traces  
fig.add\_trace(  
 go.Scatter(  
 x=centroids\_x,  
 y=centroids\_y,  
 mode='markers',  
 marker=dict(symbol='x', size=8, color='black', line=dict(width=2, color='white')),  
 name='Centroids'  
 )  
)  
  
fig.write\_html(  
 "figures/analytics\_plot1.html",  
 include\_plotlyjs="cdn",  
 full\_html=True  
)  
fig

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

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

Here we have 4 cluster groups. Group 0, which represent as green have lower salary, mostly under 150k, and max years experience in 2-5 years, it is likely Likely junior to mid-level employees with moderate pay. Group 1 with orange, has medium to high salary, wide range from $100k–$500k and with narrow range ~3 years, they are suggests specialized or high-paying roles with short experience — possibly fast-track promotions or high-demand fields. cluster 2 are low salary and experience from 0-4 years, they are clearly entry level employee. cluster 3 has medium salary, mostly under 200k with higher experiences, like 6-13 eyars. They probably are senior professionals with more experience but not the highest salaries.

import pandas as pd  
from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error, r2\_score  
import plotly.graph\_objects as go  
  
# Prepare features & target  
features = eda[['MIN\_YEARS\_EXPERIENCE', 'MAX\_YEARS\_EXPERIENCE']].apply(pd.to\_numeric, errors='coerce')  
features = features.dropna()  
X = features  
y = eda.loc[X.index, 'SALARY']  
  
# Train/test split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=688)  
  
# Fit model & predict  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
  
# Metrics (optional, but handy)  
mse = mean\_squared\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
print(f"MSE: {mse:.2f}, R²: {r2:.3f}")  
  
# Define min/max for the identity line  
min\_val = y\_test.min()  
max\_val = y\_test.max()

MSE: 797238619.53, R²: 0.095

fig = go.Figure([  
 go.Scatter(  
 x=y\_test,  
 y=y\_pred,  
 mode='markers',  
 marker=dict(color='skyblue', opacity=0.6),  
 name='Predicted vs Actual'  
 ),  
 go.Scatter(  
 x=[min\_val, max\_val],  
 y=[min\_val, max\_val],  
 mode='lines',  
 line=dict(color='red', dash='dash'),  
 name='Ideal Fit'  
 )  
])  
  
fig.update\_layout(  
 title='Actual vs Predicted Salary (Multiple Regression)',  
 xaxis\_title='Actual Salary',  
 yaxis\_title='Predicted Salary',  
 width=600,  
 height=400,  
 template='plotly\_white'  
)  
  
  
fig.write\_html(  
 'figures/analytics\_plot2.html',  
 include\_plotlyjs='cdn',  
 full\_html=False  
)  
fig

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

This plot shows the Actual vs. Predicted Salary using a multiple linear regression model. The blue dots represent individual predictions, and the red dashed line is the ideal line where predicted = actual. Since most points lie very close to the red line, it means your model predicts salary very accurately, with minimal error and strong linear fit — likely reflected in a high R² score near 1.0.