ML Methods

Final Report

Pranathi Nagasai Andhe

Emily Sundberg

today

title: “Gender Disparities in Hiring & Political Influence” author: “Pranathi Nagasai & Emily Sundberg” date: today format: html: theme: cosmo toc: true toc-depth: 2 code-fold: true —

This research investigates the intersection of gender-based employment patterns and political geography in the United States.

## Research Questions

### 1. How do hiring patterns differ for men vs. women across industries?

We analyze gender representation and hiring trends across multiple sectors:

* **Technology Sector**: Software development, AI/ML, data science
* **Healthcare**: Medical professionals, administrative roles
* **Finance**: Banking, investment, fintech
* **Manufacturing**: Production, engineering, management
* **Service Industries**: Retail, hospitality, education

### 2. Do gender-based employment disparities vary between red and blue states?

Examining state-level political affiliations and their relationship with:

* **Red States (Conservative-leaning)**
  + Gender hiring ratios
  + Workplace policies
  + Wage structures
* **Blue States (Liberal-leaning)**
  + Gender hiring ratios
  + Workplace policies
  + Wage structures
* **Swing States**: Comparative analysis of mixed political environments

### 3. Are women more underrepresented in AI fields in conservative states vs. liberal states?

Focused investigation on the technology sector, specifically:

* **AI and Machine Learning roles**
  + Data scientists
  + ML engineers
  + AI researchers
* **Women’s representation comparison**:
  + Conservative states (red states)
  + Liberal states (blue states)
  + National averages
* **Factors analyzed**:
  + Educational pipeline differences
  + Company culture and policies
  + State-level STEM initiatives
  + Industry concentration by state

### 4. How do wage gaps compare between gender and political affiliations?

Comprehensive wage analysis examining:

* **Gender wage gaps** across:
  + Red states
  + Blue states
  + Swing states
* **Industry-specific wage disparities**:
  + Tech and AI fields
  + Healthcare
  + Finance
  + Manufacturing
* **Controlling for**:
  + Experience level
  + Education
  + Job title/role
  + Company size
  + Cost of living adjustments
* **Political affiliation impact**:
  + State minimum wage policies
  + Equal pay legislation
  + Workplace protection laws

## Expected Findings

We expect this analysis will reveal that political geography influences gender-based employment outcomes, particularly in emerging high-wage sectors such as AI and technology.

# Abstract

We analyze gender disparities in hiring, AI participation, and wages across U.S. industries and states, and situate these patterns in a global context. Drawing on recent federal statistics (Labor Statistics (2025)), U.S. pay-gap trackers (University Women (2025a); Women’s Policy Research (2024)), and international reports (Economic Co-operation and Development (2024); Organization (2023); Organization (2025); Forum (2025)), we find persistent occupational segregation, widening U.S. annual earnings gaps since 2022, and continued underrepresentation of women in AI-intensive roles. State-level disparities correlate with policy environments such as pay-transparency and salary-history bans (University Women (2025b)), though industry composition and family structure are important confounders. Conservative/market-based perspectives emphasize occupational choice, hours, and career continuity as key mechanisms and warn that some transparency policies may compress wages (Institute (2021); Foundation (2024); Cullen and Pakzad-Hurson (2021); Cullen (2023); Mas (2014)). We integrate both views and outline implications for job seekers selecting sectors, geographies, and employers.

# Executive Summary

* U.S. employment remains gender-segregated: women ≈47% of workers but overrepresented in health/education and underrepresented in construction, engineering, and many tech roles (Labor Statistics (2025)).
* Every U.S. state has a pay gap; gaps tend to be smaller where transparency and equal-pay policies are stronger, though composition matters (University Women (2025b)).
* U.S. annual earnings gap widened after 2022 (82.7¢ in 2023; ≈80.9¢ in 2024 among full-time year-round), while hourly measures show ~85% in 2024 (Women’s Policy Research (2024); Center (2025)).
* Women remain 22–30% of the global AI workforce and are more exposed to AI-driven task change in clerical/admin roles (Economic Co-operation and Development (2024); Organization (2023); Organization (2025)).
* Globally, parity stands at 68.8% and may take ~123 years at current pace (Forum (2025)).
* Market-oriented analyses attribute much of the unadjusted gap to hours, occupation, and career continuity and note possible wage compression from transparency (Institute (2021); Foundation (2024); Cullen and Pakzad-Hurson (2021); Cullen (2023); Mas (2014)).
* Implications: build AI-complementary skills, target transparent employers and supportive states/metros, and use posted pay bands as inputs to evidence-based negotiation.

# Introduction

Gender continues to shape labor-market outcomes in the United States and worldwide. In 2024–2025, women’s representation varies sharply across industries, wage gaps persist, and AI both creates opportunities and raises exposure risks. We examine four questions: (1) how hiring patterns differ for men and women across industries; (2) whether disparities vary between red and blue states; (3) whether women are more underrepresented in AI fields; and (4) how wage gaps compare by gender and political affiliations. We synthesize high-quality, recent statistics and research to inform job-seeker strategy.

# Qualitative Research Method

We triangulate multiple sources: U.S. Bureau of Labor Statistics Current Population Survey (CPS) 2024 annual averages for occupational distributions (Labor Statistics (2025)); AAUW 2025 national and state pay-gap indicators (University Women (2025a); University Women (2025b)); IWPR 2024 fact sheets on annual earnings gaps (Women’s Policy Research (2024)); Pew Research Center 2025 hourly pay-gap analysis (Center (2025)); OECD 2024 policy brief on AI and women (Economic Co-operation and Development (2024)); International Labour Organization reports on generative-AI exposure (2023; 2025 update) (Organization (2023); Organization (2025)); and the WEF 2025 Global Gender Gap Report (Forum (2025)). To incorporate conservative/market perspectives, we review AEI and Heritage commentary (Institute (2021); Foundation (2024)) and research on equilibrium effects of pay transparency (Cullen and Pakzad-Hurson (2021); Cullen (2023); Mas (2014)).

# Hiring Patterns Across Industries (U.S. and Global)

U.S. employment remains gender-segregated. CPS 2024 annual averages indicate women comprise roughly 47% of total employment but are more concentrated in health care, education, and service roles, with lower shares in construction, engineering, and portions of tech (Labor Statistics (2025)). Internationally, the World Economic Forum (2025) estimates overall global gender parity at 68.8%, with economic participation parity at about 60–61%, implying persistent cross-country segmentation (Forum (2025)). Market-oriented analyses argue that part of observed differences in outcomes reflect hours worked, occupation mix, and career continuity rather than like-for-like pay differences (Institute (2021); Foundation (2024)).

# State Politics and Gender Disparities (Red vs. Blue)

AAUW’s 2025 analysis shows that every U.S. state has a gender pay gap, with substantial dispersion across states (University Women (2025b)). Cross-state differences correlate with policy adoption such as salary-history bans and pay-transparency requirements, which are more prevalent in many blue states (University Women (2025b)). However, composition matters: industry mix (e.g., energy and construction), unionization, urbanization, and childcare access vary across states and can generate red–blue patterns without ideology being the sole driver. Earlier peer-reviewed work associates state liberalism with narrower gaps, but causality remains difficult to establish (Maume (2015)). Recent reporting suggests that post-pandemic return-to-office mandates have reduced flexibility and may contribute to widening national gaps, though these effects likely differ by state and sector (Post (2025)).

# Women in AI Fields (U.S. & Global) and Political Context

Women remain underrepresented in AI and tech roles. The OECD documents lower female representation in AI-exposed professional occupations and constrained access to AI tools (Economic Co-operation and Development (2024)). The ILO shows that clerical and administrative tasks—female-heavy—are highly exposed to generative-AI transformation in high-income countries; a 2025 refinement confirms the asymmetric exposure (Organization (2023); Organization (2025)). Direct state-by-state measures of female AI participation are limited. It is therefore premature to assert causality from political ideology to AI underrepresentation without merging employer-level AI job postings and hires with state policy and industry controls. Nonetheless, differences in STEM pipelines, childcare, higher education, and transparency regimes plausibly contribute to cross-state variation (Organization (2023); Organization (2025); Economic Co-operation and Development (2024)).

# Wage Gaps and Political Affiliation

On annual full-time, year-round earnings, IWPR reports a deterioration from 2022 to 2023 (82.7¢) and news coverage indicates about 80.9¢ in 2024—the lowest since 2016 (Women’s Policy Research (2024); Newsweek (2025)). By contrast, Pew Research Center’s hourly series shows women earned about 85% of men’s hourly pay in 2024 when combining full- and part-time workers (Center (2025)). Adjusted gaps shrink after controlling for occupation, hours, and experience but do not disappear (Institute (2021); Center (2025)). Policy can narrow gaps: pay-transparency laws are associated with smaller within-firm gaps but may compress overall wages or slow wage growth according to equilibrium analyses (Cullen and Pakzad-Hurson (2021); Cullen (2023); Mas (2014)).

# Implications for Job Seekers (2025)

* **Sector choice:** Target underrepresented, higher-growth fields such as data, AI, and engineering while building verifiable skills, certifications, and portfolios (Labor Statistics (2025); Economic Co-operation and Development (2024)).
* **Geography:** Favor states and metros with pay-transparency requirements and supportive care infrastructure while benchmarking offers with state snapshots (University Women (2025b)).
* **AI resilience:** Develop AI-complementary skills to hedge exposure in clerical/admin roles and to compete for AI-adjacent, higher-pay tracks (Organization (2023); Economic Co-operation and Development (2024)).
* **Employer screening:** Prefer organizations with posted pay bands, career-progression transparency, and flexible/hybrid policies as RTO mandates may widen disparities (Post (2025)).
* **Negotiation:** Use posted ranges as inputs, not anchors, and negotiate based on documented contributions; be aware of transparency’s potential compression effects (Cullen and Pakzad-Hurson (2021); Cullen (2023); Mas (2014)).

# Limitations

Causal attribution of political ideology to gender disparities is challenging due to confounding by industry mix, demographics, and local cost structures. AI participation statistics with state-gender granularity remain sparse. International comparisons depend on differing definitions of occupations, pay, and employment. Transparency policy effects vary by market and occupation; equilibrium responses may offset some intended benefits.

# Conclusion

Gender disparities in hiring, AI participation, and pay persist across the U.S. and globally. State policies and employer practices shape observed gaps, but composition and choice also matter. A pragmatic job-search strategy in 2025 combines sector targeting, AI-adjacent upskilling, careful geography selection, and screening for transparent, flexible employers. Continuous measurement using CPS updates, AAUW/IWPR dashboards, and international benchmarks will be essential for tracking progress.

title: “Data Cleaning” format: html: code-overflow: wrap code-fold: true toc : false execute: echo: true eval: false freeze: auto —

## Load the dataset

This code initializes a PySpark environment to load and explore a dataset of job postings. It begins by importing and starting a Spark session named “JobPostingsAnalysis”, then reads a CSV file (lightcast\_job\_postings.csv) into a Spark DataFrame with headers, schema inference, and support for multi-line fields. The DataFrame is registered as a temporary SQL view called “job\_postings” to enable SQL-style queries. Finally, it performs a basic diagnostic check by printing the schema and previewing the first five rows of data—steps that are intended for local debugging and should be commented out when rendering the final submission.

import pandas as pd  
import plotly.express as px  
import plotly.io as pio  
from pyspark.sql import SparkSession  
import re  
import numpy as np  
import plotly.graph\_objects as go  
from pyspark.sql.functions import col, split, explode, regexp\_replace, transform, when  
from pyspark.sql import functions as F  
from pyspark.sql.functions import col, monotonically\_increasing\_id  
  
np.random.seed(42)  
  
pio.renderers.default = "notebook"  
  
spark = SparkSession.builder.appName("LightcastData").getOrCreate()  
  
jobs\_df = spark.read.option("header", "true").option("inferSchema", "true").option("multiLine","true").option("escape", "\"").csv("./data/lightcast\_job\_postings.csv")  
jobs\_df.createOrReplaceTempView("job\_postings")  
  
elections\_df = spark.read.option("header", "true").option("inferSchema", "true").option("multiLine","true").option("escape", "\"").csv("./data/2024\_election\_results.csv")  
elections\_df.createOrReplaceTempView("election\_results")  
  
#print("---This is Diagnostic check, No need to print it in the final doc---")  
  
#df.printSchema() # comment this line when rendering the submission  
#jobs\_df.show(5)  
#elections\_df.show(5)

## Data Cleaning

The following code cleans and standardizes the job postings dataset — ensuring proper data types, filling missing salaries, removing duplicates, categorizing remote types, and dropping overly sparse columns — to produce a clean, analysis-ready DataFrame.

# casting corrected variable type  
jobs\_df = jobs\_df.withColumn("SALARY\_FROM", col ("SALARY\_FROM").cast("float"))\  
 .withColumn("SALARY\_TO", col("SALARY\_TO").cast("float")) \  
 .withColumn("MAX\_YEARS\_EXPERIENCE", col("MAX\_YEARS\_EXPERIENCE").cast("float"))\  
 .withColumn("MIN\_YEARS\_EXPERIENCE", col("MIN\_YEARS\_EXPERIENCE").cast("float"))\  
 .withColumn("SALARY", col("SALARY").cast("float"))  
  
# Clean Up Columns  
jobs\_df = jobs\_df.withColumn("EDUCATION\_LEVELS\_NAME", regexp\_replace(col("EDUCATION\_LEVELS\_NAME"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("SOURCE\_TYPES", regexp\_replace(col("SOURCE\_TYPES"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("SOURCES", regexp\_replace(col("SOURCES"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("SKILLS", regexp\_replace(col("SKILLS"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("SKILLS\_NAME", regexp\_replace(col("SKILLS\_NAME"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("SPECIALIZED\_SKILLS\_NAME", regexp\_replace(col("SPECIALIZED\_SKILLS\_NAME"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("CERTIFICATIONS\_NAME", regexp\_replace(col("CERTIFICATIONS\_NAME"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("COMMON\_SKILLS\_NAME", regexp\_replace(col("COMMON\_SKILLS\_NAME"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("SOFTWARE\_SKILLS\_NAME", regexp\_replace(col("SOFTWARE\_SKILLS\_NAME"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("CIP6\_NAME", regexp\_replace(col("CIP6\_NAME"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("CIP4\_NAME", regexp\_replace(col("CIP4\_NAME"), "[\n\r]", ""))  
jobs\_df = jobs\_df.withColumn("CIP2\_NAME", regexp\_replace(col("CIP2\_NAME"), "[\n\r]", ""))  
  
  
# Compute and impute Median Salary  
def compute\_median(sdf, col\_name):  
 q = sdf.approxQuantile(col\_name, [0.5], 0.01)  
 return q[0] if q else None  
  
  
median\_from = compute\_median(jobs\_df, "SALARY\_FROM")  
median\_to = compute\_median(jobs\_df, "SALARY\_TO")  
median\_salary = compute\_median(jobs\_df, "SALARY")  
  
print("Medians:", median\_from, median\_to, median\_salary)  
  
jobs\_df = jobs\_df.fillna({  
 "SALARY\_FROM": median\_from,  
 "SALARY\_TO": median\_to,  
 "SALARY": median\_salary  
})  
  
from pyspark.sql.functions import col  
jobs\_df = jobs\_df.withColumn(  
 "MIDPOINT\_SALARY",  
 (col("SALARY\_TO") + col("SALARY\_FROM")) / 2  
)  
  
# Dropping unnecessary columns  
columns\_to\_drop = [  
 "ID", "URL", "ACTIVE\_URLS", "DUPLICATES", "LAST\_UPDATED\_TIMESTAMP","STATE","COUNTY\_OUTGOING","COUNTY\_INCOMMING","MSA\_OUTGOING","MSA\_INCOMING",  
 "NAICS2", "NAICS3", "NAICS4", "NAICS5", "NAICS6", "ONET","ONET\_2019","CIP6","CIP4","CIP2","SOC\_2021\_2","SOC\_2021\_3","SOC\_2021\_4","SOC\_2021\_5","SOC\_2", "SOC\_3", "SOC\_4","SOC\_5", "NAICS\_2022\_2","NAICS\_2022\_3","NAICS\_2022\_4","NAICS\_2022\_5","NAICS\_2022\_6","CITY","COUNTY","MSA","COUNTY\_INCOMING"  
]  
jobs\_df = jobs\_df.drop(\*columns\_to\_drop)  
  
# configuring remote work groups  
from pyspark.sql.functions import when, col, trim  
  
jobs\_df = jobs\_df.withColumn("REMOTE\_GROUP",  
 when(trim(col("REMOTE\_TYPE\_NAME"))== "Remote", "Remote")  
 .when(trim(col("REMOTE\_TYPE\_NAME"))== "Hybrid Remote", "Hybrid")  
 .when(trim(col("REMOTE\_TYPE\_NAME"))== "Not Remote", "Onsite")  
 .when(col("REMOTE\_TYPE\_NAME").isNull(), "Onsite")  
 .otherwise("Onsite")  
)  
  
# dropping any duplicate postings  
jobs\_df = jobs\_df.dropDuplicates(["TITLE", "COMPANY", "LOCATION", "POSTED"])  
  
# handling missing data  
from pyspark.sql.functions import col, when, sum as spark\_sum  
  
total\_rows = jobs\_df.count()  
missing\_threshold = total\_rows \* 0.5  
null\_counts = jobs\_df.select([  
 (spark\_sum(col(c).isNull().cast("int"))).alias(c) for c in jobs\_df.columns  
]).collect()[0].asDict()  
columns\_to\_keep = [c for c, nulls in null\_counts.items() if nulls <= missing\_threshold or c == "SALARY"]  
jobs\_df = jobs\_df.select(columns\_to\_keep)  
  
#jobs\_df.show(15)

This part of the script joins in another data frame that has the 2024 presidential election results by state. This allows us to use the states’ political affiliation as an attribute of the job posting.

from pyspark.sql import functions as F  
  
jobs\_df = jobs\_df.withColumn("STATE\_ABBREVIATION", F.trim(F.split(jobs\_df["COUNTY\_NAME"], ",").getItem(1)))  
  
jobs\_alias = jobs\_df.alias("jobs")  
elections\_alias = elections\_df.alias("elections")  
  
jobs\_df = jobs\_alias.join(  
 elections\_alias,  
 F.col("jobs.STATE\_ABBREVIATION") == F.col("elections.STATE"),  
 "left"  
)  
jobs\_df = jobs\_df.drop(F.col("elections.STATE"))  
  
jobs\_df = jobs\_df.withColumnRenamed("Affiliation", "AFFILIATION")  
  
#jobs\_df.show(15)

Now, this script selects only the columns we want to look at specifically

selected\_df = jobs\_df.select(  
 "EDUCATION\_LEVELS\_NAME",  
 "MIN\_EDULEVELS\_NAME",  
 "EMPLOYMENT\_TYPE\_NAME",  
 "MIN\_YEARS\_EXPERIENCE",  
 "SALARY\_TO",  
 "SALARY\_FROM",  
 "SALARY",  
 "CITY\_NAME",  
 "MSA\_NAME",  
 "STATE\_NAME",  
 "NAICS2\_NAME",  
 "NAICS3\_NAME",  
 "NAICS4\_NAME",  
 "NAICS5\_NAME",  
 "NAICS6\_NAME",  
 "SKILLS\_NAME",  
 "SPECIALIZED\_SKILLS\_NAME",  
 "CERTIFICATIONS\_NAME",  
 "COMMON\_SKILLS\_NAME",  
 "SOFTWARE\_SKILLS\_NAME",  
 "ONET\_NAME",  
 "LOT\_CAREER\_AREA\_NAME",  
 "LOT\_OCCUPATION\_NAME",  
 "LOT\_SPECIALIZED\_OCCUPATION\_NAME",  
 "LOT\_OCCUPATION\_GROUP\_NAME",  
 "LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME",  
 "LOT\_V6\_OCCUPATION\_NAME",  
 "LOT\_V6\_OCCUPATION\_GROUP\_NAME",  
 "LOT\_V6\_CAREER\_AREA\_NAME",  
 "SOC\_2\_NAME",  
 "SOC\_3\_NAME",  
 "SOC\_4\_NAME",  
 "SOC\_5\_NAME",  
 "REMOTE\_GROUP",  
 "STATE\_ABBREVIATION",  
 "AFFILIATION",  
 "MIDPOINT\_SALARY"  
)

Once we have the columns we want to look at, we create a heat map to show us the remaining missing values. We have already dealt with a lot of missing values earlier, but this will help us visualize what is left.

import pandas as pd  
from pyspark.sql.functions import col, sum as spark\_sum, when, trim, length  
import hvplot.pandas  
  
df\_sample\_viz = selected\_df.select(  
 "MIN\_YEARS\_EXPERIENCE",  
 "SALARY",  
 "MSA\_NAME",  
 "NAICS5\_NAME"  
)  
  
df\_sample = df\_sample\_viz.sample(fraction = .15, seed = 42).toPandas()  
  
missing\_mask = df\_sample.isnull()  
  
missing\_long = (  
 missing\_mask.reset\_index()  
 .melt(id\_vars = "index", var\_name = "column", value\_name = "is\_missing")  
)  
  
missing\_long["is\_missing"] = missing\_long["is\_missing"].astype(int)  
  
missing = missing\_long.hvplot.heatmap(  
 x="column",  
 y="index",  
 C = "is\_missing",  
 cmap = "Blues",  
 width = 900,  
 height = 500,  
 title = "Heatmap of Missing Values (15%)"  
).opts(xrotation=45)  
  
hvplot.save(missing, './output/missing\_heatmap.html')

As you can see above, the missing values are mainly in the columns for minimum years of experience, and MSA name. The following script cleans up some of the column values and replaces missing values with an appropriate substitute such as 0 or “unknown”.

from pyspark.sql.functions import countDistinct  
  
selected\_df.select([  
 countDistinct(c).alias(c+"\_nunique")  
 for c in selected\_df.columns  
]).show(truncate=False)  
  
# Education Levels  
  
selected\_df = selected\_df.withColumn(  
 "EDUCATION\_LEVELS\_NAME",  
 when(col("EDUCATION\_LEVELS\_NAME").isNull(), "No Education Listed")  
 .otherwise(col("EDUCATION\_LEVELS\_NAME"))  
)  
  
# Min Edu Levels  
  
selected\_df = selected\_df.withColumn(  
 "MIN\_EDULEVELS\_NAME",  
 when(col("MIN\_EDULEVELS\_NAME").isNull(), "No Education Listed")  
 .otherwise(col("MIN\_EDULEVELS\_NAME"))  
)  
  
# Employment Type Name  
  
selected\_df = selected\_df.withColumn(  
 "EMPLOYMENT\_TYPE\_NAME",  
 when(col("EMPLOYMENT\_TYPE\_NAME") == "Part-time / full-time","Flexible")  
 .when(col("EMPLOYMENT\_TYPE\_NAME") == "Part-time (â‰¤ 32 hours)","Part-Time")  
 .when(col("EMPLOYMENT\_TYPE\_NAME") == "Full-time (> 32 hours)","Full-Time")  
 .when(col("EMPLOYMENT\_TYPE\_NAME").isNull(), "Full-Time")  
 .otherwise(col("EMPLOYMENT\_TYPE\_NAME"))  
)  
  
# Min Years Experience  
selected\_df = selected\_df.withColumn(  
 "MIN\_YEARS\_EXPERIENCE",  
 when(col("MIN\_YEARS\_EXPERIENCE").isNull(), 0)  
 .otherwise(col("MIN\_YEARS\_EXPERIENCE"))  
)  
  
# Salary to  
selected\_df = selected\_df.withColumn(  
 "SALARY\_TO",  
 when(col("SALARY\_TO").isNull(), median\_to)  
 .otherwise(col("SALARY\_TO"))  
)  
  
# Salary from  
selected\_df = selected\_df.withColumn(  
 "SALARY\_FROM",  
 when(col("SALARY\_FROM").isNull(), median\_from)  
 .otherwise(col("SALARY\_FROM"))  
)  
  
# Salary   
selected\_df = selected\_df.withColumn(  
 "SALARY",  
 when(col("SALARY").isNull(), median\_salary)  
 .otherwise(col("SALARY"))  
)  
  
# City Name  
selected\_df = selected\_df.withColumn(  
 "CITY\_NAME",  
 when(col("CITY\_NAME").isNull(), "Unknown")  
 .otherwise(col("CITY\_NAME"))  
)  
  
# MSA Name  
selected\_df = selected\_df.withColumn(  
 "MSA\_NAME",  
 when(col("MSA\_NAME").isNull(), "Unknown")  
 .otherwise(col("MSA\_NAME"))  
)  
  
# State Name  
selected\_df = selected\_df.withColumn(  
 "STATE\_NAME",  
 when(col("STATE\_NAME").isNull(), "Unknown")  
 .otherwise(col("STATE\_NAME"))  
)  
  
# NAICS2\_NAME   
selected\_df = selected\_df.withColumn(  
 "NAICS2\_NAME",  
 when(col("NAICS2\_NAME").isNull(), "Unknown")  
 .otherwise(col("NAICS2\_NAME"))  
)  
  
# NAICS3\_NAME   
selected\_df = selected\_df.withColumn(  
 "NAICS3\_NAME",  
 when(col("NAICS3\_NAME").isNull(), "Unknown")  
 .otherwise(col("NAICS3\_NAME"))  
)  
  
# NAICS4\_NAME   
selected\_df = selected\_df.withColumn(  
 "NAICS4\_NAME",  
 when(col("NAICS4\_NAME").isNull(), "Unknown")  
 .otherwise(col("NAICS4\_NAME"))  
)  
  
# NAICS5\_NAME   
selected\_df = selected\_df.withColumn(  
 "NAICS5\_NAME",  
 when(col("NAICS5\_NAME").isNull(), "Unknown")  
 .otherwise(col("NAICS5\_NAME"))  
)  
  
# NAICS6\_NAME   
selected\_df = selected\_df.withColumn(  
 "NAICS6\_NAME",  
 when(col("NAICS6\_NAME").isNull(), "Unknown")  
 .otherwise(col("NAICS6\_NAME"))  
)  
  
#STATE ABBREVIATION  
selected\_df = selected\_df.withColumn(  
 "STATE\_ABBREVIATION",  
 when(col("STATE\_ABBREVIATION").isNull(), "Unknown")  
 .otherwise(col("STATE\_ABBREVIATION"))  
)

Finally, we have a clean dataset so we will convert it to a pandas dataframe and save it a csv.

pdf = selected\_df.toPandas()  
  
pdf.to\_csv("./data/lightcast\_cleaned.csv", index=False)  
  
pdf.head(15)  
  
print("Data Cleaning Complete. Rows retained:", len(pdf))

import pandas as pd  
import plotly.express as px  
import plotly.io as pio  
from pyspark.sql import SparkSession  
import re  
import numpy as np  
import plotly.graph\_objects as go  
from pyspark.sql.functions import col, split, explode, regexp\_replace, transform, when  
from pyspark.sql import functions as F  
from pyspark.sql.functions import col, monotonically\_increasing\_id  
  
np.random.seed(42)  
  
pio.renderers.default = "notebook"  
  
spark = SparkSession.builder.appName("LightcastCleanedData").getOrCreate()  
  
eda = spark.read.option("header", "true").option("inferSchema", "true").option("multiLine","true").option("escape", "\"").csv("./data/lightcast\_cleaned.csv")  
  
eda.show(15)

## Introduction

The following visualizations are based on the lightcast job postings data frame that was cleaned in the previous section. This analysis explores different facets of the data specifically related to the political affiliation of the states and the different job postings in each state. We also take a closer look at AI related jobs and the impact of political climate on salary.

## Exploring the Salary by State Political Affiliation

from pyspark.sql import functions as F  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
eda = eda.na.drop(subset=["AFFILIATION"])  
  
# Aggregate the Data   
salary\_by\_affiliation = (  
 eda.groupBy("AFFILIATION","NAICS2\_NAME")  
 .agg(  
 F.mean("SALARY").alias("avg\_salary"),  
 F.expr("percentile\_approx(SALARY, 0.5)").alias("median\_salary"),  
 F.count("\*").alias("count")  
 )  
 .orderBy("avg\_salary", ascending=False)  
)  
  
#salary\_by\_affiliation.show()  
  
pdf = salary\_by\_affiliation.toPandas()  
  
# visualize the data  
fig = px.bar(  
 pdf,  
 x="NAICS2\_NAME",  
 y="avg\_salary",  
 color="AFFILIATION",  
 barmode="group",  
 hover\_data=["median\_salary", "count"],   
 labels={  
 "NAICS2\_NAME": "Industry (NAICS2)",  
 "avg\_salary": "Average Salary",  
 "AFFILIATION": "Political Affiliation"  
 },  
 title="Average Salary by Industry and Political Affiliation"  
)  
  
fig.update\_layout(  
 xaxis\_tickangle=-45,  
 yaxis\_title="Average Salary",  
 xaxis\_title="Industry (NAICS2)",  
 legend\_title="Affiliation",  
 template="plotly\_white"  
)  
  
fig.show()  
  
fig.write\_html("./output/salary\_affiliation.html")

This data clearly shows that average salaries are slightly higher in Blue states vs Red States across nearly all NAICS2 categories. In red states, Professional, Scientific, and Technical Services has a slightly higher average salary.

## Exploring the minimum education level by political affiliation

import plotly.express as px  
  
# Aggregate the Data   
edulevel\_by\_affiliation = (  
 eda.groupBy("AFFILIATION","MIN\_EDULEVELS\_NAME")  
 .agg(  
 F.count("\*").alias("count"),  
 F.mean("SALARY").alias("avg\_salary")  
 )  
)  
  
edulevel\_by\_affiliation.show()  
  
pdf = edulevel\_by\_affiliation.toPandas()  
  
# visualize the data  
fig = px.bar(  
 pdf,  
 x="MIN\_EDULEVELS\_NAME",  
 y="count",  
 color="AFFILIATION",  
 barmode="group",  
 title="Education Level Requirements by Political Affiliation",  
 hover\_data=["avg\_salary"], # optional: show avg salary on hover  
 color\_discrete\_map={"Blue": "#1f77b4", "Red": "#d62728"}  
)  
  
fig.update\_layout(  
 xaxis\_title="Minimum Education Level",  
 yaxis\_title="Number of Postings",  
 template="plotly\_white"  
)  
  
fig.show()  
  
fig.write\_html("./output/education\_affiliation.html")

This graph shows us the comparison of job postings by political affiliation and minimum education level. As you can see, red states have far more job postings for lower education levels such as High School, Associate Degree, or Bachelor’s degree, and blue states have more postings requiring a Master’s Degree or higher.

fig = px.bar(  
 pdf,  
 x="MIN\_EDULEVELS\_NAME",  
 y="avg\_salary",  
 color="AFFILIATION",  
 barmode="group",  
 title="Average Salary by Education Level and Political Affiliation",  
 hover\_data=["count"],  
 color\_discrete\_map={"Blue": "#1f77b4", "Red": "#d62728"}  
)  
  
fig.update\_layout(  
 xaxis\_title="Minimum Education Level",  
 yaxis\_title="Average Salary (USD)",  
 template="plotly\_white"  
)  
  
fig.show()  
  
fig.write\_html("./output/salary\_education\_affiliation.html")

This figure tells us that despite red states having more job postings with education level requirements below a Master’s degree, the salaries in Blue states are higher for every single minimum education level. This could be attributed to a higher cost of living in most blue states.

## AI Jobs Analysis

from pyspark.sql import functions as F  
  
ai\_keywords = [  
 "artificial intelligence", "machine learning", "deep learning",  
 "neural network", "nlp", "natural language processing",  
 "computer vision", "data science", "data scientist",  
 "ai engineer", "ai research", "ml engineer",  
 "tensorflow", "pytorch", "keras", "hugging face", "openai", "scikit-learn"  
]  
pattern = "|".join([f"(?i){kw}" for kw in ai\_keywords])  
  
ai\_jobs = eda.filter(  
 F.col("SKILLS\_NAME").rlike(pattern) |  
 F.col("SPECIALIZED\_SKILLS\_NAME").rlike(pattern) |  
 F.col("SOFTWARE\_SKILLS\_NAME").rlike(pattern) |  
 F.col("COMMON\_SKILLS\_NAME").rlike(pattern) |  
 F.col("LOT\_OCCUPATION\_NAME").rlike(pattern) |  
 F.col("ONET\_NAME").rlike(pattern)  
)  
  
ai\_jobs.select("MIN\_EDULEVELS\_NAME","LOT\_OCCUPATION\_NAME", "SKILLS\_NAME", "SALARY", "STATE\_ABBREVIATION","AFFILIATION","NAICS2\_NAME","NAICS3\_NAME","NAICS4\_NAME","NAICS5\_NAME","NAICS6\_NAME").show(10, truncate=False)

import plotly.express as px  
  
# Aggregate the Data   
ai\_job\_analysis = (  
 ai\_jobs.groupBy("AFFILIATION","MIN\_EDULEVELS\_NAME")  
 .agg(  
 F.count("\*").alias("count"),  
 F.mean("SALARY").alias("avg\_salary")  
 )  
)  
  
ai\_job\_analysis.show()  
  
ai\_pdf = ai\_job\_analysis.toPandas()  
  
# visualize the data  
fig = px.bar(  
 ai\_pdf,  
 x="MIN\_EDULEVELS\_NAME",  
 y="count",  
 color="AFFILIATION",  
 barmode="group",  
 title="AI jobs Education Level Requirements by Political Affiliation",  
 hover\_data=["avg\_salary"],  
 color\_discrete\_map={"Blue": "#1f77b4", "Red": "#d62728"}  
)  
  
fig.update\_layout(  
 xaxis\_title="Minimum Education Level",  
 yaxis\_title="Number of Postings",  
 template="plotly\_white"  
)  
  
fig.show()  
  
fig.write\_html("./output/AI\_education\_affiliation.html")

This figure shows us the number of postings just with certain key words related to AI. It groups by education level and political affiliation. The graph tells us that the distribution of number of AI job postings across education level and affiliation mirrors that of the larger data set. Thus, AI jobs are not posted at any higher frequency across red or blue states than any other job.

fig = px.bar(  
 ai\_pdf,  
 x="MIN\_EDULEVELS\_NAME",  
 y="avg\_salary",  
 color="AFFILIATION",  
 barmode="group",  
 title="AI Jobs Average Salary by Education Level and Political Affiliation",  
 hover\_data=["count"],   
 color\_discrete\_map={"Blue": "#1f77b4", "Red": "#d62728"}  
)  
  
fig.update\_layout(  
 xaxis\_title="Minimum Education Level",  
 yaxis\_title="Average Salary (USD)",  
 template="plotly\_white"  
)  
  
fig.show()  
  
fig.write\_html("./output/AI\_salary\_education\_affiliation.html")

Similarly, in this figure, AI Jobs show salary distributions equivalent to that of jobs in other industries. Like the general analysis, blue states offer higher salaries for AI jobs across each of education levels. Again, this is likely due to the higher cost of living in most blue states.

state\_counts = (  
 ai\_jobs.groupBy("STATE\_ABBREVIATION", "AFFILIATION")  
 .agg(  
 F.count("\*").alias("count"),  
 F.mean("SALARY").alias("avg\_salary")  
 )  
 .orderBy(F.desc("count"))  
)  
state\_counts.show(10, truncate=False)  
  
state\_counts\_pd = state\_counts.toPandas()  
  
import plotly.express as px  
  
state\_counts\_pd\_sorted = state\_counts\_pd.sort\_values("count", ascending=True)  
  
fig = px.bar(  
 state\_counts\_pd\_sorted,  
 x="count",  
 y="STATE\_ABBREVIATION",  
 orientation="h",  
 color = "AFFILIATION",  
 color\_discrete\_map={  
 "Red": "red",  
 "Blue": "blue"  
 },  
 hover\_data=["avg\_salary"], # Show average salary on hover  
 title="Number of AI Job Postings by State and Affiliation",  
 labels={"count": "Job Postings", "STATE\_ABBREVIATION": "State"}  
)  
  
fig.update\_layout(  
 yaxis={'categoryorder':'total ascending'},  
 xaxis\_title="Number of Job Postings",  
 yaxis\_title="State",  
 legend\_title="Affiliation",  
 template="plotly\_white"  
)  
  
fig.show()  
  
fig.write\_html("./output/AI\_byState.html")

This graph clearly shows that of the top 5 states with the most number of AI job postings, 4/5 of them are blue states.

import plotly.express as px  
from pyspark.sql import functions as F  
  
ai\_naics = (ai\_jobs.filter(col("NAICS2\_NAME") =="Professional, Scientific, and Technical Services")  
)  
  
naics\_levels = ["NAICS4\_NAME", "NAICS5\_NAME", "NAICS6\_NAME"]  
  
ai\_naicscounts\_dfs = []  
for level in naics\_levels:  
 df = (  
 ai\_naics.groupBy(level)  
 .agg(F.count("\*").alias("count"))  
 .withColumnRenamed(level, "Industry")  
 .withColumn("NAICS\_Level", F.lit(level))  
 )  
 ai\_naicscounts\_dfs.append(df)  
  
combined\_ai\_naicscounts = ai\_naicscounts\_dfs[0]  
for df in ai\_naicscounts\_dfs[1:]:  
 combined\_ai\_naicscounts = combined\_ai\_naicscounts.union(df)  
  
combined\_ai\_naicscounts = combined\_ai\_naicscounts.orderBy(F.desc("count"))  
  
  
ai\_naics\_combined = combined\_ai\_naicscounts.toPandas()  
  
ai\_naics\_combined

fig = px.bar(  
 ai\_naics\_combined,  
 x="Industry",  
 y="count",  
 color = "NAICS\_Level",  
 barmode="group",  
 hover\_data=["NAICS\_Level", "count"],  
 title="Job Posting Counts Across NAICS Levels 2–6",  
 labels={"count": "Job Postings", "Industry": "Industry", "NAICS\_Level": "NAICS Level"}  
)  
  
# Rotate x-axis labels for readability  
fig.update\_layout(  
 xaxis\_tickangle=-45,  
 yaxis\_title="Number of Job Postings",  
 xaxis\_title="Industry",  
 template="plotly\_white"  
)  
  
fig.show()  
  
fig.write\_html("./output/AI\_industries.html")

This graph is showing us the number of job postings by NAICS. For AI jobs, the NAICS with the largest number of AI job postings. Computer Systems Design is predominate NAICS with consulting services coming in second.

#eda.select("NAICS2\_NAME").distinct().show(25, truncate = False)  
  
from pyspark.sql import functions as F  
  
gender\_jobs = spark.read.option("header", "true").option("inferSchema", "true").option("multiLine","true").option("escape", "\"").csv("./data/Gender\_Industries.csv")  
gender\_jobs.createOrReplaceTempView("gender\_industries")  
  
  
sorted\_gender\_jobs = (  
 gender\_jobs  
 .orderBy(F.desc("TOTAL\_NUMBER\_OF\_WOMEN"))  
 .select("NAICS\_NAME", "NUMBER\_OF\_PEOPLE","TOTAL\_NUMBER\_OF\_WOMEN", "PERCENT\_WOMEN")  
 .limit(10)  
)  
  
#sorted\_gender\_jobs.show()  
  
sorted\_gender\_jobs = sorted\_gender\_jobs.toPandas()

import plotly.graph\_objects as go  
  
fig = go.Figure()  
  
fig.add\_trace(go.Bar(  
 x=sorted\_gender\_jobs["NAICS\_NAME"],  
 y=sorted\_gender\_jobs["TOTAL\_NUMBER\_OF\_WOMEN"],  
 name="Number of Women",  
 yaxis="y1"  
))  
  
fig.add\_trace(go.Scatter(  
 x=sorted\_gender\_jobs["NAICS\_NAME"],  
 y=sorted\_gender\_jobs["PERCENT\_WOMEN"],  
 name="% Women",  
 yaxis="y2",  
 mode="lines+markers"  
))  
  
  
fig.update\_layout(  
 title="Top 10 Industries by Number of Women",  
 xaxis=dict(title="Industry (NAICS\_NAME)"),  
 yaxis=dict(title="Number of Women", side="left"),  
 yaxis2=dict(  
 title="% Women",  
 overlaying="y",  
 side="right"  
 ),  
 legend=dict(x=0.02, y=0.98),  
 template="plotly\_white",  
 xaxis\_tickangle=-45  
)  
  
fig.show()  
  
fig.write\_html("./output/gender\_and\_industry.html")

The above graph has two different important features. First, it shows the top 10 industries for women on the bar chart, sorted by the overall number of women according the U.S. Bureau of Labor Statistics Labor Statistics ((2025)). The second important feature is the line chart which shows the percentage of women in that field. As you can see, some of the most female dominated industries do not have very many women in them and some of the industries with the most women are heavily male dominated.

from pyspark.sql import functions as F  
import plotly.graph\_objects as go  
from pyspark.sql.functions import col  
  
gender\_jobs = gender\_jobs.withColumn("NUMBER\_OF\_PEOPLE", col ("NUMBER\_OF\_PEOPLE").cast("float"))  
  
sorted\_gender\_jobs = (  
 gender\_jobs  
 .orderBy(F.desc("NUMBER\_OF\_PEOPLE"))  
 .select("NAICS\_NAME", "NUMBER\_OF\_PEOPLE","TOTAL\_NUMBER\_OF\_WOMEN", "PERCENT\_WOMEN")  
 .limit(10)  
)  
  
sorted\_gender\_jobs.show()  
sorted\_gender\_jobs\_pd = sorted\_gender\_jobs.toPandas()  
  
  
fig = go.Figure()  
  
fig.add\_trace(go.Bar(  
 x=sorted\_gender\_jobs\_pd["NAICS\_NAME"],  
 y=sorted\_gender\_jobs\_pd["NUMBER\_OF\_PEOPLE"],  
 name="Number of People",  
 yaxis="y1"  
))  
  
fig.add\_trace(go.Scatter(  
 x=sorted\_gender\_jobs\_pd["NAICS\_NAME"],  
 y=sorted\_gender\_jobs\_pd["PERCENT\_WOMEN"],  
 name="% Women",  
 yaxis="y2",  
 mode="lines+markers"  
))  
  
fig.update\_layout(  
 title="Top 10 Industries by Number of People",  
 xaxis=dict(title="Industry (NAICS\_NAME)"),  
 yaxis=dict(title="Percent Female", side="left"),  
 yaxis2=dict(  
 title="% Women",  
 overlaying="y",  
 side="right"  
 ),  
 legend=dict(x=0.02, y=0.98),  
 template="plotly\_white",  
 xaxis\_tickangle=-45  
)  
  
fig.show()  
  
fig.write\_html("./output/industry\_gender\_gap.html")

The above figure shows the top 10 industries with the most number of employed individuals based on data from the U.S. Bureau of Labor Statistics Labor Statistics ((2025)). The blue bars show the number of people in those industries, and the red bars show the number of women in the industry.

from pyspark.sql import functions as F  
import plotly.graph\_objects as go  
from pyspark.sql.functions import col  
import plotly.express as px  
  
  
gender\_jobs\_pd = gender\_jobs.toPandas()  
  
ai\_gender = gender\_jobs\_pd[(gender\_jobs\_pd["NAICS\_NAME"] == "Computer and mathematical occupations") | (gender\_jobs\_pd["NAICS\_NAME"] == "Computer systems analysts")| (gender\_jobs\_pd["NAICS\_NAME"] == "Information security analysts") | (gender\_jobs\_pd["NAICS\_NAME"] == "Computer programmers") | (gender\_jobs\_pd["NAICS\_NAME"] == "Software developers") | (gender\_jobs\_pd["NAICS\_NAME"] == "Database administrators and architects") | (gender\_jobs\_pd["NAICS\_NAME"] == "Network and computer systems administrators") | (gender\_jobs\_pd["NAICS\_NAME"] == "Computer network architects") | (gender\_jobs\_pd["NAICS\_NAME"] == "Computer occupations, all other")]  
  
ai\_gender = ai\_gender.sort\_values(by = "PERCENT\_WOMEN", ascending = False)  
  
fig = go.Figure()  
  
fig.add\_trace(go.Bar(  
 x=ai\_gender["NAICS\_NAME"],  
 y=ai\_gender["NUMBER\_OF\_PEOPLE"],  
 name="Number of People",  
 yaxis="y1"  
))  
  
fig.add\_trace(go.Scatter(  
 x=ai\_gender["NAICS\_NAME"],  
 y=ai\_gender["PERCENT\_WOMEN"],  
 name="% Women",  
 yaxis="y2",  
 mode="lines+markers"  
))  
  
fig.update\_layout(  
 title="AI Industries and Female Representation",  
 xaxis=dict(title="Industry (NAICS\_NAME)"),  
 yaxis=dict(title="Percent Female", side="left"),  
 yaxis2=dict(  
 title="% Women",  
 overlaying="y",  
 side="right"  
 ),  
 legend=dict(x=0.02, y=0.98),  
 template="plotly\_white",  
 xaxis\_tickangle=-45  
)  
  
fig.show()  
  
fig.write\_html("./output/ai\_gender.html")

The above figure shows the representation of women in AI industries. It’s worth noting that not a single one is female dominated, the highest female representation coming from Computer Systems Analysis with 43% female. The majority of these industries are less than 30% female. This displays a clear underrepresentation of females in AI industries.

import pandas as pd  
import plotly.express as px  
import plotly.io as pio  
from pyspark.sql import SparkSession  
import re  
import numpy as np  
import plotly.graph\_objects as go  
from pyspark.sql.functions import col, split, explode, regexp\_replace, transform, when  
from pyspark.sql import functions as F  
from pyspark.sql.functions import col, monotonically\_increasing\_id  
  
np.random.seed(42)  
  
pio.renderers.default = "notebook"  
  
spark = SparkSession.builder.appName("LightcastData").getOrCreate()  
  
jobs\_df = spark.read.option("header", "true").option("inferSchema", "true").option("multiLine","true").option("escape", "\"").csv("./data/lightcast\_job\_postings.csv")  
jobs\_df.createOrReplaceTempView("job\_postings")  
  
#print("---This is Diagnostic check, No need to print it in the final doc---")  
  
#df.printSchema() # comment this line when rendering the submission  
#jobs\_df.show(5)  
#elections\_df.show(5)

import pandas as pd  
  
skills\_data = {  
 "Name": ["Emily","Pranathi"],  
 "Python": [2,1],  
 "Java" :[2,1],  
 "SQL": [2,1],  
 "Power BI": [4,4],  
 "Machine Learning": [1,1],  
 "Cloud Computing": [2,2]  
}  
  
df\_skills = pd.DataFrame(skills\_data)  
df\_skills.set\_index("Name", inplace=True)  
df\_skills

import seaborn as sns  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(8, 6))  
sns.heatmap(df\_skills, annot=True, cmap="RdYlGn", linewidths=0.5)  
plt.title("Team Skill Levels Heatmap")  
plt.show()

# view\_cols = [  
# "BODY",  
# "SKILLS\_NAME",  
# "SPECIALIZED\_SKILLS\_NAME",  
# "CERTIFICATIONS\_NAME",  
# "COMMON\_SKILLS\_NAME",  
# "SOFTWARE\_SKILLS\_NAME"  
# ]  
  
# for colname in view\_cols:  
# print(f"\n----{colname} ----")  
# jobs\_df.select(colname).distinct().show(60,truncate = False)

from collections import Counter  
  
skills\_pd = jobs\_df.select("SKILLS\_NAME").toPandas()  
  
top\_skills = ["Python","Java","Power BI","Machine Learning","Cloud Computing"]  
  
skill\_counts = Counter()  
for skill in top\_skills:  
 skill\_counts[skill] = skills\_pd['SKILLS\_NAME'].str.contains(skill, case=False, regex=True).sum()  
  
job\_skill\_counts = Counter(top\_skills)  
  
for skill in top\_skills:  
 if skill not in df\_skills.columns:  
 df\_skills[skill] = 0   
  
df\_skills.loc["Job Postings Count"] = [skill\_counts.get(skill, 0) for skill in df\_skills.columns]  
  
df\_skills

Based on our Skills Gap Analysis, we need to work on our Python skills. In order to be more competitive in the job market, we will finish the Python DataCamp and practice incorporating our new skills into our current roles.

import pandas as pd  
import numpy as np  
import plotly.express as px  
import plotly.graph\_objects as go  
import plotly.io as pio  
from plotly.subplots import make\_subplots  
  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import LabelEncoder, StandardScaler  
from sklearn.linear\_model import LinearRegression  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.cluster import KMeans  
from sklearn.metrics import mean\_squared\_error, r2\_score, silhouette\_score  
  
import warnings  
warnings.filterwarnings('ignore')  
  
# Set Plotly theme  
pio.templates.default = "plotly\_white"  
  
print("✓ All libraries loaded successfully!")

## Data Loading and Exploration

# Load lightcast job postings data  
df = pd.read\_csv('data/lightcast\_job\_postings.csv')  
  
print(f"Dataset Shape: {df.shape[0]:,} rows × {df.shape[1]} columns")  
print(f"\nFirst few rows:")  
df.head()

# Check data quality  
print("="\*80)  
print("DATA QUALITY ASSESSMENT")  
print("="\*80)  
  
# Key columns for analysis  
key\_columns = ['SALARY', 'STATE', 'TITLE', 'NAICS\_2022\_2', 'SOC\_2', 'ONET', 'LIGHTCAST\_SECTORS']  
  
info\_df = pd.DataFrame({  
 'Column': key\_columns,  
 'Missing': [df[col].isnull().sum() if col in df.columns else 'N/A' for col in key\_columns],  
 'Missing %': [f"{(df[col].isnull().sum() / len(df) \* 100):.2f}%" if col in df.columns else 'N/A' for col in key\_columns],  
 'Unique Values': [df[col].nunique() if col in df.columns else 'N/A' for col in key\_columns]  
})  
  
print(info\_df.to\_string(index=False))  
  
# Salary statistics  
if 'SALARY' in df.columns:  
 print(f"\nSalary Statistics:")  
 print(f" Mean: ${df['SALARY'].mean():,.2f}")  
 print(f" Median: ${df['SALARY'].median():,.2f}")  
 print(f" Std Dev: ${df['SALARY'].std():,.2f}")  
 print(f" Range: ${df['SALARY'].min():,.2f} - ${df['SALARY'].max():,.2f}")

## Data Preprocessing

# Create working copy  
df\_clean = df.copy()  
  
# Remove rows with missing salary or state  
required\_cols = ['SALARY', 'STATE']  
initial\_rows = len(df\_clean)  
df\_clean = df\_clean.dropna(subset=required\_cols)  
removed\_rows = initial\_rows - len(df\_clean)  
  
print(f"Removed {removed\_rows:,} rows with missing salary or state data")  
print(f"Final dataset: {len(df\_clean):,} rows")

## Political Leaning Classification (Fixed)  
  
#| label: fix-fips-and-political-leaning  
#| code-fold: true  
  
# FIPS code to state abbreviation mapping  
fips\_to\_state = {  
 1: 'AL', 2: 'AK', 4: 'AZ', 5: 'AR', 6: 'CA', 8: 'CO', 9: 'CT', 10: 'DE',  
 11: 'DC', 12: 'FL', 13: 'GA', 15: 'HI', 16: 'ID', 17: 'IL', 18: 'IN',  
 19: 'IA', 20: 'KS', 21: 'KY', 22: 'LA', 23: 'ME', 24: 'MD', 25: 'MA',  
 26: 'MI', 27: 'MN', 28: 'MS', 29: 'MO', 30: 'MT', 31: 'NE', 32: 'NV',  
 33: 'NH', 34: 'NJ', 35: 'NM', 36: 'NY', 37: 'NC', 38: 'ND', 39: 'OH',  
 40: 'OK', 41: 'OR', 42: 'PA', 44: 'RI', 45: 'SC', 46: 'SD', 47: 'TN',  
 48: 'TX', 49: 'UT', 50: 'VT', 51: 'VA', 53: 'WA', 54: 'WV', 55: 'WI',  
 56: 'WY', 72: 'PR'  
}  
  
# Convert FIPS codes to state abbreviations  
df\_clean['STATE\_ABBREV'] = df\_clean['STATE'].apply(  
 lambda x: fips\_to\_state.get(int(x), 'Unknown') if pd.notna(x) else 'Unknown'  
)  
  
# Political classifications  
red\_states = ['AL', 'AK', 'AR', 'FL', 'ID', 'IN', 'IA', 'KS', 'KY',   
 'LA', 'MS', 'MO', 'MT', 'NE', 'ND', 'OH', 'OK', 'SC',   
 'SD', 'TN', 'TX', 'UT', 'WV', 'WY']  
  
blue\_states = ['CA', 'CO', 'CT', 'DE', 'HI', 'IL', 'ME', 'MD', 'MA',   
 'MI', 'MN', 'NH', 'NJ', 'NM', 'NY', 'OR', 'PA', 'RI',   
 'VT', 'VA', 'WA', 'WI', 'DC']  
  
swing\_states = ['AZ', 'GA', 'NC', 'NV']  
  
def assign\_political\_leaning(state\_abbrev):  
 if pd.isna(state\_abbrev) or state\_abbrev == 'Unknown':  
 return 'Unknown'  
 state\_abbrev = str(state\_abbrev).upper()  
 if state\_abbrev in red\_states:  
 return 'Red'  
 elif state\_abbrev in blue\_states:  
 return 'Blue'  
 elif state\_abbrev in swing\_states:  
 return 'Swing'  
 else:  
 return 'Other'  
  
df\_clean['political\_leaning'] = df\_clean['STATE\_ABBREV'].apply(assign\_political\_leaning)  
  
print("="\*60)  
print("POLITICAL LEANING DISTRIBUTION (FIXED)")  
print("="\*60)  
print(df\_clean['political\_leaning'].value\_counts())  
print("\nPercentage:")  
print((df\_clean['political\_leaning'].value\_counts() / len(df\_clean) \* 100).round(2))

# Visualize political leaning distribution  
fig = px.pie(  
 values=df\_clean['political\_leaning'].value\_counts().values,  
 names=df\_clean['political\_leaning'].value\_counts().index,  
 title='Distribution of Jobs by Political Leaning of State',  
 hole=0.4,  
 color\_discrete\_map={'Red': '#FF6B6B', 'Blue': '#4ECDC4', 'Swing': '#FFD93D', 'Other': '#95A5A6'}  
)  
fig.update\_layout(template="plotly\_white", height=400)  
fig.show()  
  
fig.write\_html("./output/visualize\_political\_distribution.html")

## K-Means Clustering (unsupervised)

### Setup and Feature Engineering

# Determine which reference label to use (SOC, NAICS, or ONET)  
reference\_label = None  
for label in ['SOC\_2', 'NAICS\_2022\_2', 'ONET', 'LIGHTCAST\_SECTORS']:  
 if label in df\_clean.columns and df\_clean[label].notna().sum() > 0:  
 reference\_label = label  
 print(f"✓ Using {label} as reference label")  
 break  
  
if reference\_label is None:  
 print("No classification column found. Using TITLE as reference.")  
 reference\_label = 'TITLE'  
  
print(f"\nReference Label: {reference\_label}")  
print(f"Unique values: {df\_clean[reference\_label].nunique():,}")  
print(f"\nTop 10 {reference\_label} categories:")  
print(df\_clean[reference\_label].value\_counts().head(10))

# Prepare features for clustering  
df\_cluster = df\_clean.copy()  
  
print("="\*80)  
print("FEATURE ENGINEERING FOR CLUSTERING")  
print("="\*80)  
  
# Encode categorical variables  
encoders = {}  
categorical\_cols = ['STATE', 'TITLE', 'political\_leaning', 'LIGHTCAST\_SECTORS']  
  
print(f"\nEncoding categorical variables:")  
for col in categorical\_cols:  
 if col in df\_cluster.columns:  
 le = LabelEncoder()  
 df\_cluster[f'{col}\_encoded'] = le.fit\_transform(  
 df\_cluster[col].fillna('Unknown').astype(str)  
 )  
 encoders[col] = le  
 print(f" ✓ {col}: {df\_cluster[col].nunique()} unique values")  
  
# Select clustering features  
clustering\_features = ['SALARY']  
  
for col in categorical\_cols:  
 encoded\_col = f'{col}\_encoded'  
 if encoded\_col in df\_cluster.columns:  
 clustering\_features.append(encoded\_col)  
  
# Add years of experience if available  
if 'MIN\_YEARS\_EXPERIENCE' in df\_cluster.columns:  
 df\_cluster['MIN\_YEARS\_EXPERIENCE'] = pd.to\_numeric(  
 df\_cluster['MIN\_YEARS\_EXPERIENCE'], errors='coerce'  
 )  
 clustering\_features.append('MIN\_YEARS\_EXPERIENCE')  
 print(f" ✓ MIN\_YEARS\_EXPERIENCE: numeric feature")  
  
print(f"\n📊 Total Clustering Features: {len(clustering\_features)}")  
print("\nFeature List:")  
for i, feature in enumerate(clustering\_features, 1):  
 print(f" {i}. {feature}")  
  
# Prepare feature matrix  
X\_cluster = df\_cluster[clustering\_features].fillna(df\_cluster[clustering\_features].mean())  
print(f"\nFeature Matrix Shape: {X\_cluster.shape}")

### Determine Optimal K

# Standardize features  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X\_cluster)  
  
# Test different numbers of clusters  
K\_range = range(2, 11)  
inertias = []  
silhouette\_scores = []  
  
print("Testing different numbers of clusters...")  
print(f"{'k':<5} {'Inertia':<15} {'Silhouette Score'}")  
print("-" \* 40)  
  
for k in K\_range:  
 kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)  
 kmeans.fit(X\_scaled)  
 inertias.append(kmeans.inertia\_)  
 sil\_score = silhouette\_score(X\_scaled, kmeans.labels\_)  
 silhouette\_scores.append(sil\_score)  
 print(f"{k:<5} {kmeans.inertia\_:<15.2f} {sil\_score:.4f}")  
  
optimal\_k = list(K\_range)[silhouette\_scores.index(max(silhouette\_scores))]  
print(f"\n💡 Optimal k based on Silhouette Score: {optimal\_k}")

# Visualize elbow curve and silhouette scores  
fig = make\_subplots(  
 rows=1, cols=2,  
 subplot\_titles=('Elbow Method', 'Silhouette Score Method')  
)  
  
fig.add\_trace(  
 go.Scatter(x=list(K\_range), y=inertias, mode='lines+markers',   
 name='Inertia', line=dict(color='blue')),  
 row=1, col=1  
)  
  
fig.add\_trace(  
 go.Scatter(x=list(K\_range), y=silhouette\_scores, mode='lines+markers',   
 name='Silhouette', line=dict(color='orange')),  
 row=1, col=2  
)  
  
fig.update\_xaxes(title\_text="Number of Clusters (k)", row=1, col=1)  
fig.update\_xaxes(title\_text="Number of Clusters (k)", row=1, col=2)  
fig.update\_yaxes(title\_text="Inertia", row=1, col=1)  
fig.update\_yaxes(title\_text="Silhouette Score", row=1, col=2)  
  
fig.update\_layout(  
 height=400,   
 showlegend=False,   
 template="plotly\_white",  
 title\_text="Determining Optimal Number of Clusters"  
)  
fig.show()  
  
fig.write\_html("./output/elbow\_curve.html")

### Cluster Analysis

# Perform KMeans with k=5 (per assignment requirements)  
n\_clusters = 5  
  
print(f"Performing KMeans with k={n\_clusters} clusters...")  
kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42, n\_init=10)  
df\_cluster['cluster'] = kmeans.fit\_predict(X\_scaled)  
  
print(f"✓ Clustering complete!")  
print(f"\nCluster Distribution:")  
cluster\_counts = df\_cluster['cluster'].value\_counts().sort\_index()  
for cluster\_id, count in cluster\_counts.items():  
 pct = (count / len(df\_cluster)) \* 100  
 print(f" Cluster {cluster\_id}: {count:,} jobs ({pct:.1f}%)")

# Sample for performance (5000 points)  
sample\_size = min(5000, len(df\_cluster))  
df\_sample = df\_cluster.sample(sample\_size, random\_state=42)  
  
fig = px.scatter(  
 df\_sample,  
 x='SALARY',  
 y='STATE\_encoded',  
 color='cluster',  
 hover\_data=['TITLE', 'political\_leaning', reference\_label] if 'TITLE' in df\_sample.columns else None,  
 title=f'KMeans Clustering Results (k={n\_clusters}, n={sample\_size:,} sample)',  
 labels={'SALARY': 'Annual Salary ($)', 'STATE\_encoded': 'State (Encoded)', 'cluster': 'Cluster'},  
 color\_continuous\_scale='Viridis'  
)  
fig.update\_layout(template="plotly\_white", height=550, font=dict(family="Roboto", size=12))  
fig.show()  
  
fig.write\_html("./output/clusters.html")

**Key Findings:** - Cluster 0: Entry-level positions ($50-100k) - Cluster 4: Senior roles ($200k+) - Geographic clustering evident by state encoding

# Analyze cluster characteristics  
print("="\*80)  
print("CLUSTER PROFILES")  
print("="\*80)  
  
cluster\_profiles = df\_cluster.groupby('cluster').agg({  
 'SALARY': ['mean', 'median', 'std', 'min', 'max'],  
 'political\_leaning': lambda x: x.mode()[0] if len(x.mode()) > 0 else 'Mixed',  
 'cluster': 'count'  
})  
  
cluster\_profiles.columns = [  
 'Avg\_Salary', 'Median\_Salary', 'Salary\_StdDev', 'Min\_Salary', 'Max\_Salary',  
 'Dominant\_Political', 'Count'  
]  
  
cluster\_profiles = cluster\_profiles.round(2)  
print(cluster\_profiles)

# Visualize cluster salary profiles  
profile\_df = cluster\_profiles.reset\_index()  
  
fig = px.bar(  
 profile\_df,  
 x='cluster',  
 y='Avg\_Salary',  
 text='Count',  
 title='Average Salary by Cluster',  
 labels={'cluster': 'Cluster', 'Avg\_Salary': 'Average Salary ($)'},  
 color='Avg\_Salary',  
 color\_continuous\_scale='Viridis'  
)  
fig.update\_traces(texttemplate='n=%{text:,}', textposition='outside')  
fig.update\_layout(template="plotly\_white", height=450, font=dict(family="Roboto", size=12))  
fig.show()  
  
fig.write\_html("./output/cluster\_profiles.html")

# Compare clusters with reference labels  
print(f"\n{'='\*80}")  
print(f"CLUSTER vs {reference\_label.upper()} COMPARISON")  
print(f"{'='\*80}")  
  
for cluster\_id in sorted(df\_cluster['cluster'].unique()):  
 cluster\_data = df\_cluster[df\_cluster['cluster'] == cluster\_id]  
 top\_categories = cluster\_data[reference\_label].value\_counts().head(5)  
   
 print(f"\n📊 Cluster {cluster\_id}:")  
 print(f" Size: {len(cluster\_data):,} jobs")  
 print(f" Avg Salary: ${cluster\_data['SALARY'].mean():,.2f}")  
 if 'political\_leaning' in cluster\_data.columns:  
 print(f" Dominant Political: {cluster\_data['political\_leaning'].mode()[0] if len(cluster\_data['political\_leaning'].mode()) > 0 else 'Mixed'}")  
 print(f" Top 5 {reference\_label} categories:")  
 for category, count in top\_categories.items():  
 pct = (count / len(cluster\_data)) \* 100  
 print(f" • {category}: {count:,} ({pct:.1f}%)")

## Multiple Regression Analysis

# Prepare features for salary prediction  
df\_reg = df\_clean.copy()  
  
print("="\*80)  
print("FEATURE ENGINEERING FOR SALARY PREDICTION")  
print("="\*80)  
  
# Encode categorical variables  
le\_reg = {}  
categorical\_features = ['STATE', 'TITLE', 'political\_leaning', 'LIGHTCAST\_SECTORS']  
  
# Add SOC or NAICS if available  
if 'SOC\_2' in df\_reg.columns:  
 categorical\_features.append('SOC\_2')  
elif 'NAICS\_2022\_2' in df\_reg.columns:  
 categorical\_features.append('NAICS\_2022\_2')  
  
print(f"\nCategorical features to encode ({len(categorical\_features)}):")  
for col in categorical\_features:  
 if col in df\_reg.columns:  
 le = LabelEncoder()  
 df\_reg[f'{col}\_encoded'] = le.fit\_transform(  
 df\_reg[col].fillna('Unknown').astype(str)  
 )  
 le\_reg[col] = le  
 print(f" ✓ {col}: {df\_reg[col].nunique()} unique values")  
  
# Select features for regression  
feature\_cols = []  
for col in categorical\_features:  
 encoded\_col = f'{col}\_encoded'  
 if encoded\_col in df\_reg.columns:  
 feature\_cols.append(encoded\_col)  
  
# Add numerical features if available  
numeric\_features = ['MIN\_YEARS\_EXPERIENCE', 'MAX\_YEARS\_EXPERIENCE']  
for num\_feat in numeric\_features:  
 if num\_feat in df\_reg.columns:  
 df\_reg[num\_feat] = pd.to\_numeric(df\_reg[num\_feat], errors='coerce')  
 if df\_reg[num\_feat].notna().sum() > 0:  
 feature\_cols.append(num\_feat)  
 print(f" ✓ {num\_feat}: numeric feature added")  
  
print(f"\n Total Features for Salary Prediction: {len(feature\_cols)}")  
  
# Prepare X and y  
X = df\_reg[feature\_cols].fillna(df\_reg[feature\_cols].mean())  
y = df\_reg['SALARY']  
  
print(f"\n Dataset Statistics:")  
print(f" • Feature Matrix Shape: {X.shape}")  
print(f" • Salary Statistics:")  
print(f" - Mean: ${y.mean():,.2f}")  
print(f" - Median: ${y.median():,.2f}")  
print(f" - Std Dev: ${y.std():,.2f}")  
print(f" - Range: ${y.min():,.2f} - ${y.max():,.2f}")

Feature Selection Justification: The features were selected based on their theoretical and empirical relationship with salary:

STATE & Political Leaning: Geographic location and political climate influence cost of living and compensation policies TITLE: Job title is the primary indicator of role level and responsibility LIGHTCAST\_SECTORS: Industry sector determines baseline compensation structure SOC/NAICS: Occupation classification provides standardized job categorization Years of Experience: Direct correlation with salary progression (if available)

# Split data (70/30 as per assignment requirements)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.30, random\_state=42  
)  
  
print("="\*80)  
print("TRAIN/TEST SPLIT")  
print("="\*80)  
print(f"\nTraining Set: {X\_train.shape[0]:,} samples ({X\_train.shape[0]/len(X)\*100:.1f}%)")  
print(f"Test Set: {X\_test.shape[0]:,} samples ({X\_test.shape[0]/len(X)\*100:.1f}%)")  
print(f"\nSalary Distribution:")  
print(f" Training - Mean: ${y\_train.mean():,.2f}, Std: ${y\_train.std():,.2f}")  
print(f" Test - Mean: ${y\_test.mean():,.2f}, Std: ${y\_test.std():,.2f}")

print("="\*80)  
print("MODEL 1: MULTIPLE LINEAR REGRESSION")  
print("="\*80)  
  
# Train model  
lin\_reg = LinearRegression()  
lin\_reg.fit(X\_train, y\_train)  
  
# Make predictions  
y\_pred\_lin = lin\_reg.predict(X\_test)  
  
# Calculate evaluation metrics  
rmse\_lin = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_lin))  
r2\_lin = r2\_score(y\_test, y\_pred\_lin)  
mae\_lin = np.mean(np.abs(y\_test - y\_pred\_lin))  
  
print(f"\n📊 Model Performance:")  
print(f" • R² Score: {r2\_lin:.4f}")  
print(f" → Explains {r2\_lin\*100:.2f}% of salary variance")  
print(f" • RMSE: ${rmse\_lin:,.2f}")  
print(f" • MAE: ${mae\_lin:,.2f}")  
  
# Feature coefficients  
coef\_df = pd.DataFrame({  
 'Feature': feature\_cols,  
 'Coefficient': lin\_reg.coef\_  
}).sort\_values('Coefficient', key=abs, ascending=False)  
  
print(f"\n📈 Top 10 Most Influential Features:")  
print(coef\_df.head(10).to\_string(index=False))

# Visualize top 15 feature coefficients  
top\_features = coef\_df.head(15)  
  
fig = px.bar(  
 top\_features,  
 x='Coefficient',  
 y='Feature',  
 orientation='h',  
 title='Top 15 Feature Coefficients - Multiple Linear Regression',  
 labels={'Coefficient': 'Impact on Salary ($)', 'Feature': 'Feature'},  
 color='Coefficient',  
 color\_continuous\_scale='RdBu',  
 color\_continuous\_midpoint=0  
)  
  
fig.update\_layout(template="plotly\_white", height=500, font=dict(family="Roboto", size=12))  
fig.show()  
  
fig.write\_html("./output/linear\_coefficients.html")

print("="\*80)  
print("MODEL 2: RANDOM FOREST REGRESSION")  
print("="\*80)  
  
# Train model  
rf\_reg = RandomForestRegressor(  
 n\_estimators=100,  
 max\_depth=20,  
 min\_samples\_split=10,  
 min\_samples\_leaf=4,  
 random\_state=42,  
 n\_jobs=-1  
)  
  
print("Training Random Forest model...")  
rf\_reg.fit(X\_train, y\_train)  
print("✓ Training complete!")  
  
# Make predictions  
y\_pred\_rf = rf\_reg.predict(X\_test)  
  
# Calculate evaluation metrics  
rmse\_rf = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_rf))  
r2\_rf = r2\_score(y\_test, y\_pred\_rf)  
mae\_rf = np.mean(np.abs(y\_test - y\_pred\_rf))  
  
print(f"\n Model Performance:")  
print(f" • R² Score: {r2\_rf:.4f}")  
print(f" → Explains {r2\_rf\*100:.2f}% of salary variance")  
print(f" • RMSE: ${rmse\_rf:,.2f}")  
print(f" • MAE: ${mae\_rf:,.2f}")  
  
# Feature importance  
importance\_df = pd.DataFrame({  
 'Feature': feature\_cols,  
 'Importance': rf\_reg.feature\_importances\_  
}).sort\_values('Importance', ascending=False)  
  
print(f"\n Top 10 Most Important Features:")  
print(importance\_df.head(10).to\_string(index=False))  
  
# Calculate improvement safely  
if r2\_lin > 0:  
 improvement = ((r2\_rf - r2\_lin) / r2\_lin) \* 100  
 print(f"\n🚀 Random Forest improves R² by {improvement:.1f}% over Linear Regression")

# Visualize feature importance  
top\_features\_rf = importance\_df.head(15)  
  
fig = px.bar(  
 top\_features\_rf,  
 x='Importance',  
 y='Feature',  
 orientation='h',  
 title='Top 15 Feature Importance - Random Forest Regression',  
 labels={'Importance': 'Importance Score', 'Feature': 'Feature'},  
 color='Importance',  
 color\_continuous\_scale='Viridis'  
)  
  
fig.update\_layout(template="plotly\_white", height=500, font=dict(family="Roboto", size=12))  
fig.show()  
  
fig.write\_html("./output/feature\_importance.html")

print("="\*80)  
print("REGRESSION MODEL COMPARISON")  
print("="\*80)  
  
# Create comparison dataframe  
comparison\_df = pd.DataFrame({  
 'Model': ['Multiple Linear Regression', 'Random Forest Regression'],  
 'R² Score': [r2\_lin, r2\_rf],  
 'RMSE ($)': [rmse\_lin, rmse\_rf],  
 'MAE ($)': [mae\_lin, mae\_rf]  
})  
  
print("\n" + comparison\_df.to\_string(index=False))  
  
best\_model = comparison\_df.loc[comparison\_df['R² Score'].idxmax(), 'Model']  
best\_r2 = comparison\_df['R² Score'].max()  
  
print(f"\n🏆 Best Performing Model: {best\_model}")  
print(f" • Achieves R² of {best\_r2:.4f}")  
print(f" • Explains {best\_r2\*100:.2f}% of salary variance")

# Sample for performance  
sample\_size = min(2000, len(y\_test))  
indices = np.random.choice(len(y\_test), sample\_size, replace=False)  
  
comparison\_results = pd.DataFrame({  
 'Actual': y\_test.iloc[indices],  
 'Linear\_Regression': y\_pred\_lin[indices],  
 'Random\_Forest': y\_pred\_rf[indices]  
})  
  
fig = go.Figure()  
  
# Random Forest predictions  
fig.add\_trace(go.Scatter(  
 x=comparison\_results['Actual'],  
 y=comparison\_results['Random\_Forest'],  
 mode='markers',  
 name='Random Forest',  
 opacity=0.6,  
 marker=dict(size=5, color='blue')  
))  
  
# Linear Regression predictions  
fig.add\_trace(go.Scatter(  
 x=comparison\_results['Actual'],  
 y=comparison\_results['Linear\_Regression'],  
 mode='markers',  
 name='Linear Regression',  
 opacity=0.6,  
 marker=dict(size=5, color='red')  
))  
  
# Perfect prediction line  
min\_val = comparison\_results['Actual'].min()  
max\_val = comparison\_results['Actual'].max()  
  
fig.add\_trace(go.Scatter(  
 x=[min\_val, max\_val],  
 y=[min\_val, max\_val],  
 mode='lines',  
 name='Perfect Prediction',  
 line=dict(color='green', dash='dash', width=2)  
))  
  
fig.update\_layout(  
 title=f'Actual vs Predicted Salary - Model Comparison (n={sample\_size:,})',  
 xaxis\_title='Actual Salary ($)',  
 yaxis\_title='Predicted Salary ($)',  
 template="plotly\_white",  
 height=550,  
 hovermode='closest'  
)  
  
fig.show()  
  
fig.write\_html("./output/model\_comparison.html")

Center, P. R. (2025): “[Gender Pay Gap in u.s. Has Narrowed Slightly over 2 Decades](https://www.pewresearch.org/short-reads/2025/03/04/gender-pay-gap-in-us-has-narrowed-slightly-over-2-decades/),”

Cullen, Z. (2023): [Is Pay Transparency Good?](https://www.nber.org/system/files/working_papers/w31060/w31060.pdf),.

Cullen, Z., and B. Pakzad-Hurson. (2021): [Equilibrium Effects of Pay Transparency](https://www.nber.org/system/files/working_papers/w28903/revisions/w28903.rev0.pdf),.

Economic Co-operation, O. for, and Development. (2024): “[Algorithm and Eve: How AI Will Impact Women at Work](https://www.oecd.org/content/dam/oecd/en/publications/reports/2024/12/algorithm-and-eve_0e889c45/a1603510-en.pdf),”

Forum, W. E. (2025): [Global Gender Gap Report 2025](https://reports.weforum.org/docs/WEF_GGGR_2025.pdf),.

Foundation, H. (2024): “[Making Sense of the Wage Gap](https://www.heritage.org/jobs-and-labor/commentary/making-sense-the-wage-gap),”

Institute, A. E. (2021): “[Gender Pay Gap Commentary Archive](https://www.aei.org/),”

Labor Statistics, U. S. B. of. (2025): [Employed Persons by Detailed Occupation, Sex, Race, and Hispanic or Latino Ethnicity: Annual Averages 2024](https://www.bls.gov/cps/cpsaat11.htm),.

Mas, A. (2014): “[Does transparency lead to pay compression?](https://www.nber.org/system/files/working_papers/w20558/w20558.pdf)” *National Bureau of Economic Research Working Paper 20558*,.

Maume, D. J. (2015): “State liberalism, female supervisors, and the gender wage gap,” *Social Science Research*,.

Newsweek. (2025): “[America’s Gender Pay Gap Going in Wrong Direction](https://www.newsweek.com/americas-gender-pay-gap-going-wrong-direction-10739408),”

Organization, I. L. (2023): [Generative AI and Jobs: A Global Analysis of Potential Effects](https://www.ilo.org/sites/default/files/wcmsp5/groups/public/%40dgreports/%40inst/documents/publication/wcms_890761.pdf),.

---. (2025): [Generative AI and Jobs: A Refined Global Index of Occupational Exposure](https://www.ilo.org/publications/generative-ai-and-jobs-refined-global-index-occupational-exposure),.

Post, T. W. (2025): “[Women Are Taking Pay Cuts as Companies Mandate Return to Office](https://www.washingtonpost.com/business/2025/10/11/rto-mandates-gender-wage-gap/),”

University Women, A. A. of. (2025a): “[The Simple Truth about the Gender Pay Gap 2025](https://www.aauw.org/resources/research/simple-truth/),”

---. (2025b): “[Gender Pay Gap by State](https://www.aauw.org/resources/article/gender-pay-gap-by-state/),”

Women’s Policy Research, I. for. (2024): [Gender and Racial Wage Gaps Worsened in 2023 and Pay Equity Still Decades Away](https://iwpr.org/wp-content/uploads/2024/09/IWPR-National-Wage-Gap-Fact-Sheet-2024.pdf),.