Assignment 03

Julio Vargas

September 21, 2025

from pyspark.sql import SparkSession  
from pyspark.sql import functions as F  
from pyspark.sql.functions import col, split, explode, regexp\_replace, transform, when  
from pyspark.sql.functions import col, monotonically\_increasing\_id  
from pyspark.sql.types import StructType # to/from JSON  
  
import json  
import re  
import numpy as np  
import pandas as pd  
  
import plotly.express as px  
import plotly.io as pio  
import plotly.graph\_objects as go  
  
  
np.random.seed(30) # set a fixed seed for reproducibility  
pio.renderers.default = "vscode+notebook" #  
# Initialize Spark Session  
spark = SparkSession.builder.appName("JobPostingsAnalysis").getOrCreate()  
# Load schema from JSON file  
with open("data/schema\_lightcast.json") as f:  
 schema = StructType.fromJson(json.load(f))  
  
# Load Data  
df = (spark.read  
 .option("header", "true")  
 .option("inferSchema", "false")  
 .schema(schema) # saved schema  
 .option("multiLine", "true")  
 .option("escape", "\"")  
 .csv("data/lightcast\_job\_postings.csv")  
 )  
  
df.createOrReplaceTempView("job\_postings")  
# Show Schema and Sample Data  
#df.printSchema()   
df.show(5)  
df.count()

# 1. 1.1 Casting salary and experience columns

### 1.0.1 1.1 Computing medians

from pyspark.sql.functions import col, regexp\_replace, trim  
# 1.1 Casting salary and experience columns  
  
  
df = df.withColumn("SALARY", col("SALARY").cast("float")) \  
 .withColumn("SALARY\_FROM", col("SALARY\_FROM").cast("float")) \  
 .withColumn("SALARY\_TO", col("SALARY\_TO").cast("float")) \  
 .withColumn("MIN\_YEARS\_EXPERIENCE", col("MIN\_YEARS\_EXPERIENCE").cast("float"))\  
 .withColumn("MAX\_YEARS\_EXPERIENCE", col("MAX\_YEARS\_EXPERIENCE").cast("float"))  
#df.select("SALARY", "SALARY\_FROM", "SALARY\_TO", "MIN\_YEARS\_EXPERIENCE", "MAX\_YEARS\_EXPERIENCE").printSchema()  
#df.select("SALARY", "SALARY\_FROM", "SALARY\_TO", "MIN\_YEARS\_EXPERIENCE", "MAX\_YEARS\_EXPERIENCE").show(5)

### 1.0.2 1.2 Computing medians

# 1.2 Computing medians  
def compute\_median(sdf, col\_name):  
 q = sdf.approxQuantile(col\_name, [0.5], 0.01) #50 percentile 1% error  
 return q[0] if q else None  
  
median\_from = compute\_median(df, "SALARY\_FROM")  
median\_to = compute\_median(df, "SALARY\_TO")  
median\_salary = compute\_median(df, "SALARY")

# 1.2 Output  
#the median\_from, median\_to , median\_salary respectively are:  
  
print("- Median SALARY\_FROM: $" + str(median\_from))  
print("- Median SALARY\_TO: $" + str(median\_to))  
print("- Median SALARY: $" + str(median\_salary))

# 1.3 Imputing missing salaries  
df = df.fillna({  
 "SALARY\_FROM": median\_from,  
 "SALARY\_TO": median\_to,  
 "SALARY": median\_salary  
})  
  
# 1.3 Add new column Average\_Salary  
df = df.withColumn("Average\_Salary", (col("SALARY\_FROM") + col("SALARY\_TO")) / 2)  
  
export\_cols = ["Average\_Salary","SALARY","EDUCATION\_LEVELS\_NAME","REMOTE\_TYPE\_NAME",  
 "MAX\_YEARS\_EXPERIENCE","LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME"]  
  
# 1.3 Output  
df.select(\*export\_cols).show(5, truncate=False)

#1.4 Cleaning Education column  
#remove the \n and \r  
df1 = df.withColumn("EDUCATION\_LEVELS\_NAME",  
 trim(  
 regexp\_replace(regexp\_replace(col("EDUCATION\_LEVELS\_NAME"),r"\n|\r", ""), #remove \n and \r  
 r"\[\s+\"", "[\"" ) #remove spaces.  
 )  
)  
# 1.4 Output  
df1.select(\*export\_cols).show(5, truncate=False)

#1.5 Exporting Cleaned Data  
# Export to CSV  
df\_selected=df1.select(\*export\_cols)  
pdf = df\_selected.toPandas()  
pdf.to\_csv("data/lightcast\_cleaned.csv", index=False)  
  
print("Data cleaning complete. Rows retained:", len(pdf))

# 2. 2.0 TEMPLATE

import plotly.graph\_objects as go  
import plotly.io as pio  
  
pio.templates["nike"] = go.layout.Template(  
 # LAYOUT  
 layout = {  
 # Fonts and colors  
 'title': {  
 'font': {'family': 'HelveticaNeue-CondensedBold, Helvetica, Sans-serif',  
 'size': 30,  
 'color': '#13007c'}   
 },  
 'font': {'family': 'Helvetica Neue, Helvetica, Sans-serif',  
 'size': 16,  
 'color': '#3b3b3b'},   
  
 'colorway': ['#fffb00', '#e010fc'],   
 # Adding others  
 'hovermode': 'x unified',  
 'plot\_bgcolor': '#E5ECF6',  
 'paper\_bgcolor': "#FFFFFF",  
   
 },  
 # DATA  
 data = {  
 # Default style applied to all bar charts  
 'bar': [go.Bar(  
 texttemplate = '%{value:$.2s}',  
 textposition = 'outside',  
 textfont = {'family': 'Helvetica Neue, Helvetica, Sans-serif',  
 'size': 20,  
 'color': '#ff6874'} # FFFFFF  
 )]  
 }  
)

## 2.1 2.1 Salary Distribution by Industry and Employment Type

#your code for first query  
import pandas as pd  
import polars as pl  
from IPython.display import display, HTML  
  
#2.2 Filter the dataset - Remove records where salary is missing or zero.  
df\_valid\_salaries = df.filter(df["SALARY"] > 0).select("NAICS2\_NAME","EMPLOYMENT\_TYPE\_NAME", "SALARY")  
  
#2.2 output - convert to pandas  
pdf = df\_valid\_salaries.toPandas()  
print("Data cleaning complete. Rows retained:", len(pdf))  
  
#2.3 Aggregate data - NAICS industry codes, employment type and compute salary distribution.  
  
# Clean employment type names for better readability  
pdf["EMPLOYMENT\_TYPE\_NAME"] = (pdf["EMPLOYMENT\_TYPE\_NAME"].astype(str)  
 .str.replace(r"[^\x00-\x7F]+", "", regex=True)  
)  
  
#2.3 output  
median\_salaries\_naics = pdf.groupby("NAICS2\_NAME")["SALARY"].median()  
median\_salaries\_employee = pdf.groupby("EMPLOYMENT\_TYPE\_NAME")["SALARY"].median()  
display(median\_salaries\_naics.to\_frame().head())  
display(median\_salaries\_employee.to\_frame().head())

#2.4 Visualize results box plot   
# X-axis = NAICS2\_NAME || Y-axis = SALARY\_FROM || Group by EMPLOYMENT\_TYPE\_NAME.  
pdf = df.select("NAICS2\_NAME", "SALARY").toPandas()  
fig = px.box(pdf, x="NAICS2\_NAME", y="SALARY", title="Salary Distribution by Industry",  
 color\_discrete\_sequence=["#EF553B"])  
 # add nike template  
#fig.update\_layout(template="nike")  
  
#fig.update\_xaxes(tickangle=45)  
  
fig.update\_layout(  
 template="nike",  
 height=700,  
 xaxis=dict(  
 title=dict(text="NAICS2\_NAME", standoff=40),   
 tickangle=45,  
 tickfont=dict(size=10),  
 automargin=True  
 ),  
 yaxis=dict(title=dict(text="Salary")),  
 margin=dict(b=150)   
   
)  
  
  
fig.show()

# 3. 3 Salary Analysis by ONET Occupation Type (Bubble Chart)

import plotly.express as px  
#3.1 Analyze how salaries differ across LOT\_OCCUPATION\_NAME occupation types.  
#ONET\_NAME CHANGE TO LOT\_OCCUPATION\_NAME  
  
#Aggregate Data  
  
salary\_analysis = spark.sql("""  
 SELECT  
 LOT\_OCCUPATION\_NAME AS Occupation\_name,   
 PERCENTILE(SALARY, 0.5) AS Median\_Salary,  
 COUNT(\*) AS Job\_Postings  
 FROM job\_postings  
 WHERE LOT\_OCCUPATION\_NAME IS NOT NULL  
 GROUP BY LOT\_OCCUPATION\_NAME  
 ORDER BY Job\_Postings DESC  
 LIMIT 10  
""") #the result only has 6 results and a null, limit to 10 is not necessary  
  
salary\_pd = salary\_analysis.toPandas()  
display(salary\_pd.head())  
  
#Simple plot to Analyze  
figa = px.scatter(  
 salary\_pd,  
 x="Occupation\_name",  
 y="Median\_Salary",  
 size="Job\_Postings",  
 title="Salary Analysis by Occupation",  
 color="Occupation\_name"  
)  
figa.update\_xaxes(tickangle=45, automargin=True)  
figa.show()  
  
#3.2 Visualize results bubble chart  
import plotly.express as px  
  
fig = px.scatter(  
 salary\_pd,  
 x="Occupation\_name",  
 y="Median\_Salary",  
 size="Job\_Postings",  
 title="Salary Analysis by LOT Occupation Type (Bubble Chart)",  
 labels={  
 "Occupation\_name": "LOT Occupation",  
 "Median\_Salary": "Median Salary",  
 "Job\_Postings": "Number of Job Postings"  
 },  
 hover\_name="Occupation\_name",  
 size\_max=60,  
 width=900,  
 height=600,  
 color="Job\_Postings",  
 color\_continuous\_scale="Plasma"  
)  
#customize layout  
fig.update\_layout(  
   
 height=700,  
 font\_family="Arial",  
 font\_size=14,  
 title\_font\_size=25,  
 title\_font\_color="#13007c",  
 font\_color="#2e2e2e",   
 xaxis\_title="LOT Occupation",  
 yaxis\_title="Median Salary",  
 plot\_bgcolor="#FAFDFF",  
 xaxis=dict(  
 tickangle=-45,  
 showline=True,  
 linecolor="#444"  
 ),  
 yaxis=dict(  
 showline=True,  
 linecolor="black"  
 )  
)  
  
  
fig.show()  
fig.write\_image("output/Q3.svg", width=1000, height=600, scale=1)

# 4. 4 Salary by Education Level

# Defining education level groupings  
lower\_deg = ["Bachelor's", "Associate", "GED", "No Education Listed", "High school"]  
higher\_deg = ["Master's degree", "PhD or professional degree"]  
  
# Adding new column EDU\_GROUP   
df = df.withColumn(  
 "EDU\_GROUP",  
 when(col("EDUCATION\_LEVELS\_NAME").rlike("|".join([f"(?i){deg}" for deg in lower\_deg])), "Bachelor's or lower")  
 .when(col("EDUCATION\_LEVELS\_NAME").rlike("|".join([f"(?i){deg}" for deg in higher\_deg])), "Master's or PhD")  
 .otherwise("Other")  
)  
  
# Modyfying/Casting necessary columns to float  
df = df.withColumn("MAX\_YEARS\_EXPERIENCE", col("MAX\_YEARS\_EXPERIENCE").cast("float"))  
df = df.withColumn("Average\_Salary", col("Average\_Salary").cast("float"))  
  
# df.select("MAX\_YEARS\_EXPERIENCE","Average\_Salary","EDU\_GROUP","EDUCATION\_LEVELS\_NAME").printSchema() #check schema changes  
  
# print(df.count()) #Total 72,498 after 8074  
  
# Filtering for non-null and positive values  
df = df.filter(  
 col("MAX\_YEARS\_EXPERIENCE").isNotNull() & col("Average\_Salary").isNotNull() &  
 (col("MAX\_YEARS\_EXPERIENCE") > 0) & (col("Average\_Salary") > 0)  
)  
  
# Filtering for just the two EDU\_GROUP groups  
df\_filtered = df.filter(col("EDU\_GROUP").isin("Bachelor's or lower", "Master's or PhD"))  
  
# Converting to Pandas for plotting  
df\_pd = df\_filtered.toPandas()  
pdf4=df.select("MAX\_YEARS\_EXPERIENCE","Average\_Salary","EDU\_GROUP","EDUCATION\_LEVELS\_NAME").toPandas()  
display(pdf4.head())

# Jittering and trimming  
df\_pd["MAX\_EXPERIENCE\_JITTER"] = df\_pd["MAX\_YEARS\_EXPERIENCE"] + np.random.uniform(-0.25, 0.25, size=len(df\_pd))  
df\_pd["AVERAGE\_SALARY\_JITTER"] = df\_pd["Average\_Salary"] + np.random.uniform(-2500, 2500, size=len(df\_pd))  
df\_pd = df\_pd.round(2)  
  
# Remove outlier higher than 399K  
df\_pd = df\_pd[df\_pd["AVERAGE\_SALARY\_JITTER"] <= 399000]  
  
df\_pd.head()

#jittering and triming  
# Plot four groups  
fig1 = px.scatter(  
 df\_pd,  
 x="MAX\_EXPERIENCE\_JITTER",  
 y="AVERAGE\_SALARY\_JITTER",  
 color="EDU\_GROUP",  
 hover\_data=["LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME"],  
 title="<b>Experience vs Salary by Education Level</b>",  
 opacity=1.0, #0.7  
 color\_discrete\_sequence=[  
 "#636EFA", # Blue  
 "#EF553B", # Red  
 "#00CC96", # Green  
 "#AB63FA" # Purple  
 ]  
)  
  
fig1.update\_traces(  
 marker=dict(size=10, line=dict(width=1, color="black"))  
)  
  
fig1.update\_layout(  
 plot\_bgcolor="#fcfcf0", # light grey chart background  
 paper\_bgcolor="#f5d9b2", # soft blue frame  
 font=dict(family="Segoe UI", size=14, color="#2b2b2b"),  
 title\_font=dict(size=22, color="#4b3832"),  
 xaxis\_title="Years of Experience",  
 yaxis\_title="Average Salary (USD)",  
 legend\_title="Education Group",  
 hoverlabel=dict(bgcolor="white", font\_size=13, font\_family="Arial"),  
 margin=dict(t=70, b=60, l=60, r=60),  
 xaxis=dict(  
 gridcolor="#e0e0e0",  
 tickmode="linear",  
 dtick=1 # show every integer year clearly  
 ),  
 yaxis=dict(gridcolor="#cccccc")  
)  
  
fig1.show()  
fig1.write\_html("output/q\_1a\_Experience\_vs\_Salary\_by\_Education\_Level.html")

# 5. 5 Salary by Remote Work Type

from pyspark.sql.functions import when, col, trim  
  
#5.1 Split into three groups based on REMOTE\_TYPE\_NAME  
df = 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")  
)  
  
#print(df.count())  
  
#5.1 Filter valid values  
df = df.filter(  
 col("MAX\_YEARS\_EXPERIENCE").isNotNull() & col("Average\_Salary").isNotNull() &  
 (col("MAX\_YEARS\_EXPERIENCE") > 0) & (col("Average\_Salary") > 0)  
)  
  
#5.1 Pandas  
df\_pd = df.select(  
 "MAX\_YEARS\_EXPERIENCE","Average\_Salary","LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME","REMOTE\_GROUP"  
 ).toPandas()  
  
df\_pd.head()  
  
# Jittering and trimming  
df\_pd["MAX\_EXPERIENCE\_JITTER"] = df\_pd["MAX\_YEARS\_EXPERIENCE"] + np.random.uniform(-0.15, 0.15, size=len(df\_pd))  
df\_pd["AVERAGE\_SALARY\_JITTER"] = df\_pd["Average\_Salary"] + np.random.uniform(-1000, 1000, size=len(df\_pd))  
df\_pd = df\_pd.round(2)  
  
# Remove outlier higher than 399K  
df\_pd = df\_pd[df\_pd["AVERAGE\_SALARY\_JITTER"] <= 399000]

# Plot four groups  
fig5 = px.scatter(  
 df\_pd,  
 x="MAX\_EXPERIENCE\_JITTER",  
 y="AVERAGE\_SALARY\_JITTER",  
 color="REMOTE\_GROUP",  
 hover\_data=["LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME"],  
 title="<b>Experience vs Salary by Remote Work Type </b>",  
 opacity=1.0, #0.7  
 color\_discrete\_sequence=[  
 "#636EFA", # Blue  
 "#EF553B", # Red  
 "#00CC96", # Green  
 "#AB63FA" # Purple  
 ]  
)  
  
fig5.update\_traces(  
 marker=dict(size=10, line=dict(width=1, color="black"))  
)  
  
fig5.update\_layout(  
 plot\_bgcolor="#fcfcf0", # light grey chart background  
 paper\_bgcolor="#f5d9b2", # soft blue frame  
 font=dict(family="Segoe UI", size=14, color="#2b2b2b"),  
 title\_font=dict(size=22, color="#4b3832"),  
 xaxis\_title="Years of Experience",  
 yaxis\_title="Average Salary (USD)",  
 legend\_title="Education Group",  
 hoverlabel=dict(bgcolor="white", font\_size=13, font\_family="Arial"),  
 margin=dict(t=70, b=60, l=60, r=60),  
 xaxis=dict(  
 gridcolor="#e0e0e0",  
 tickmode="linear",  
 dtick=1 # show every integer year clearly  
 ),  
 yaxis=dict(gridcolor="#cccccc")  
)  
  
fig5.show()