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_EXPERIENC
#df.select("SALARY", "SALARY_FROM", "SALARY_TO", "MIN_YEARS_EXPERIENCE", "MAX_YEARS_EXPERIENC
```

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
                       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",

#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])), "
    .when(col("EDUCATION_LEVELS_NAME").rlike("|".join([f"(?i){deg}" for deg in higher_deg])))
    .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").pri

# 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").t
display(pdf4.head())
```

```
# Jittering and trimming
df_pd["MAX_EXPERIENCE_JITTER"] = df_pd["MAX_YEARS_EXPERIENCE"] + np.random.uniform(-0.25, 0.2
df_pd["AVERAGE_SALARY_JITTER"] = df_pd["Average_Salary"] + np.random.uniform(-2500, 2500, siz
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_GROU
    ).toPandas()

df_pd.head()

# Jittering and trimming
df_pd["MAX_EXPERIENCE_JITTER"] = df_pd["MAX_YEARS_EXPERIENCE"] + np.random.uniform(-0.15, 0.1
df_pd["AVERAGE_SALARY_JITTER"] = df_pd["Average_Salary"] + np.random.uniform(-1000, 1000, siz
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()
```