Assignment 03

Wei Wang

September 20, 2025

# 1. Load the Dataset

from pyspark.sql import SparkSession  
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 monotonically\_increasing\_id  
  
np.random.seed(42)  
  
pio.renderers.default = "vscode+notebook+png"  
  
# Initialize Spark Session  
spark = SparkSession.builder.appName("LightcastData").getOrCreate()  
  
# Load Data  
df = spark.read.option("header", "true").option("inferSchema", "true").option("multiLine","true").option("escape", "\"").csv("data/lightcast\_job\_postings.csv")  
  
# Show Schema and Sample Data  
# print("---This is Diagnostic check, No need to print it in the final doc---")  
  
# df.printSchema() # comment this line when rendering the submission  
# df.show(5)

WARNING: Using incubator modules: jdk.incubator.vector  
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
25/09/22 02:31:55 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
25/09/22 02:31:57 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.  
[Stage 1:> (0 + 1) / 1]

# 2. Data Preparation

# Step 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"))  
  
# Step 2 Computing median for salary columns  
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(df,"SALARY\_FROM")  
median\_to = compute\_median(df,"SALARY\_TO")  
median\_salary = compute\_median(df,"SALARY")  
  
print("Medians:",median\_from, median\_to, median\_salary)  
  
# Step 3 Imputing missing salaries, but not experience  
df = df.fillna({  
 "SALARY\_FROM": median\_from,  
 "SALARY\_TO": median\_to,  
 "SALARY": median\_salary  
})  
  
# Step 4 Computing average salary  
df = df.withColumn("Average\_Salary",(col("SALARY\_FROM")+col("SALARY\_TO"))/2)  
  
# Step 5 Selecting required columns  
export\_cols = [  
 "EDUCATION\_LEVELS\_NAME",  
 "REMOTE\_TYPE\_NAME",  
 "MAX\_YEARS\_EXPERIENCE",  
 "Average\_Salary",  
 "SALARY",  
 "LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME"  
]  
df\_selected = df.select(\*export\_cols)  
  
# Step 6 Saving to CSV  
pdf = df\_selected.toPandas()  
pdf.to\_csv("data/lightcast\_cleaned.csv", index=False)  
  
print("Data cleaning complete. Rows retained:", len(pdf))

[Stage 2:> (0 + 1) / 1] [Stage 3:> (0 + 1) / 1] [Stage 4:> (0 + 1) / 1]

Medians: 87295.0 130042.0 115024.0

[Stage 5:> (0 + 1) / 1]

Data cleaning complete. Rows retained: 72498

# 3. Salary Distribution by Industry and Employment Type

* Compare salary variations across industries.
* **Filter the dataset**
  + Remove records where **salary is missing or zero**.
* **Aggregate Data**
  + Group by **NAICS industry codes**.
  + Group by **employment type** and compute salary distribution.
* **Visualize results**
  + Create a **box plot** where:
    - **X-axis** = NAICS2\_NAME
    - **Y-axis** = SALARY\_FROM
    - Group by EMPLOYMENT\_TYPE\_NAME.
  + Customize colors, fonts, and styles.

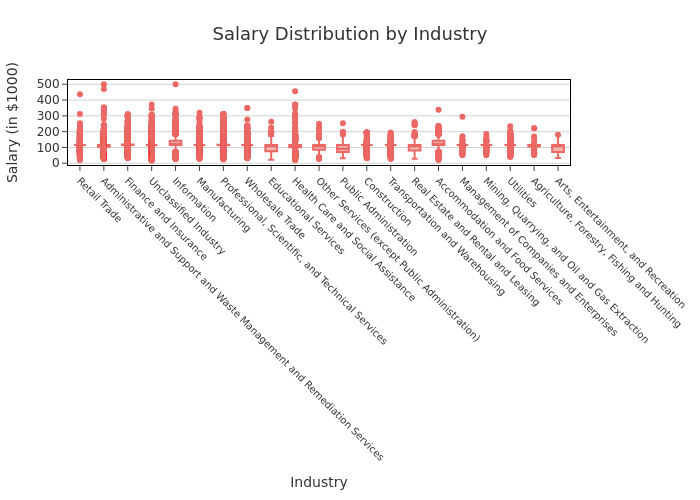
## 3.1 Salary Distribution by Industry

* **Explanation**: This graph reveals that the Information industry offers the highest median salaries (around $140K), while most other industries cluster between $100K and $120K with relatively similar distributions. It also highlights that all industries exhibit substantial salary ranges and numerous outliers, with some positions reaching $400K–$500K, indicating significant intra-industry variation in compensation.

# Filtering out missing or zero salary values  
pdf = df.filter(df["SALARY"] > 0).select("NAICS2\_NAME", "SALARY").toPandas()  
  
# Cleaning industry names  
pdf["NAICS2\_NAME"] = (  
 pdf["NAICS2\_NAME"]  
 .fillna("")  
 .apply(lambda x: re.sub(r"[^\x00-\x7F]+", "", x).strip())  
)  
pdf = pdf[pdf["NAICS2\_NAME"].str.len() > 0]  
  
# Converting salary to $1000 units  
pdf["SALARY"] = pdf["SALARY"] / 1000  
  
# Computing median salary by industry for sorting  
median\_salaries = pdf.groupby("NAICS2\_NAME")["SALARY"].median()  
sorted\_industries = median\_salaries.sort\_values(ascending=False).index  
  
# Applying sorted categories  
pdf["NAICS2\_NAME"] = pd.Categorical(  
 pdf["NAICS2\_NAME"],  
 categories=sorted\_industries,  
 ordered=True  
)  
  
# Creating box plot  
fig = px.box(  
 pdf,  
 x="NAICS2\_NAME",  
 y="SALARY",  
 points="outliers",  
 title="Salary Distribution by Industry",  
 labels={  
 "NAICS2\_NAME": "Industry",  
 "SALARY": "Salary (in $1000)"  
 },  
 color\_discrete\_sequence=["#eb6864"],  
 height=600  
)  
  
fig.update\_layout(  
 font=dict(family="Arial", size=12, color="#333333"),  
 plot\_bgcolor="white",  
 paper\_bgcolor="white",  
 title=dict(x=0.5, xanchor="center", font=dict(size=18)),  
 xaxis=dict(  
 tickangle=45,  
 tickfont=dict(size=10),  
 showgrid=False,  
 zeroline=False,  
 linecolor='black',  
 ticks='outside',  
 showline=True,  
 mirror=True  
 ),  
 yaxis=dict(  
 tick0=0,  
 dtick=100,  
 showgrid=True,  
 gridcolor='lightgray',  
 zeroline=False,  
 linecolor='black',  
 ticks='outside',  
 showline=True,  
 mirror=True  
 ),  
 margin=dict(l=60, r=40, t=80, b=200),  
 boxmode="group",  
 hovermode="x unified"  
)  
  
fig.show()  
fig.write\_image("output/Q2\_Industry\_BoxPlot.svg", width=3500, height=600, scale=1)

[Stage 6:> (0 + 1) / 1]

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



## 3.2 Salary Distribution by Employment Type

* **Explanation**: The box plot reveals that full-time employees tend to earn higher median salaries compared to part-time and mixed employment types. However, full-time roles also exhibit a wider salary range and more extreme outliers, indicating greater variability in compensation.

# Filtering out missing or zero salary values  
pdf = df.filter(df["SALARY"]>0).select("EMPLOYMENT\_TYPE\_NAME","SALARY").toPandas()  
  
# Cleaning employment type names for better readability  
pdf["EMPLOYMENT\_TYPE\_NAME"] = (  
 pdf["EMPLOYMENT\_TYPE\_NAME"]  
 .fillna("")  
 .apply(lambda x: re.sub(r"[^\x00-\x7F]+", "", x).strip())  
)  
pdf = pdf[pdf["EMPLOYMENT\_TYPE\_NAME"].str.len() > 0]  
  
# Converting salary to $1000 units  
pdf["SALARY"] = pdf["SALARY"] / 1000  
  
# Computing media salary for sorting  
median\_salaries = pdf.groupby("EMPLOYMENT\_TYPE\_NAME")["SALARY"].median()  
  
# Sorting employment types based on median salary  
sorted\_employment\_types = median\_salaries.sort\_values(ascending=False).index  
  
# Applying sorted categories  
pdf["EMPLOYMENT\_TYPE\_NAME"] = pd.Categorical(  
 pdf["EMPLOYMENT\_TYPE\_NAME"],  
 categories=sorted\_employment\_types,  
 ordered=True  
)  
  
# Creating box plot  
fig = px.box(  
 pdf,  
 x="EMPLOYMENT\_TYPE\_NAME",  
 y="SALARY",  
 points="outliers",  
 title="Salary Distribution by Employment Type",  
 labels={  
 "EMPLOYMENT\_TYPE\_NAME": "Employment Type",  
 "SALARY": "Salary (in $1000)"  
 },  
 color\_discrete\_sequence=["#eb6864"],  
 height=500  
)  
  
fig.update\_layout(  
 font=dict(family="Arial", size=14, color="#333333"),  
 plot\_bgcolor="white",  
 paper\_bgcolor="white",  
 title=dict(x=0.5, xanchor="center", font=dict(size=18)),  
 xaxis=dict(  
 showgrid=False,  
 zeroline=False,  
 linecolor='black',  
 ticks='outside',  
 showline=True,  
 mirror=True  
 ),  
 yaxis=dict(  
 showgrid=True,  
 gridcolor='lightgray',  
 zeroline=False,  
 linecolor='black',  
 ticks='outside',  
 showline=True,  
 mirror=True,  
 tick0=0,  
 dtick=50  
 ),  
 margin=dict(l=60, r=40, t=80, b=60),  
 boxmode="group",  
 hovermode="x unified"  
)  
  
fig.show()  
fig.write\_image("output/Q1\_EMPLOYMENT\_TYPE\_BoxPlot.svg", width=3000, height=500, scale=1)

[Stage 7:> (0 + 1) / 1]

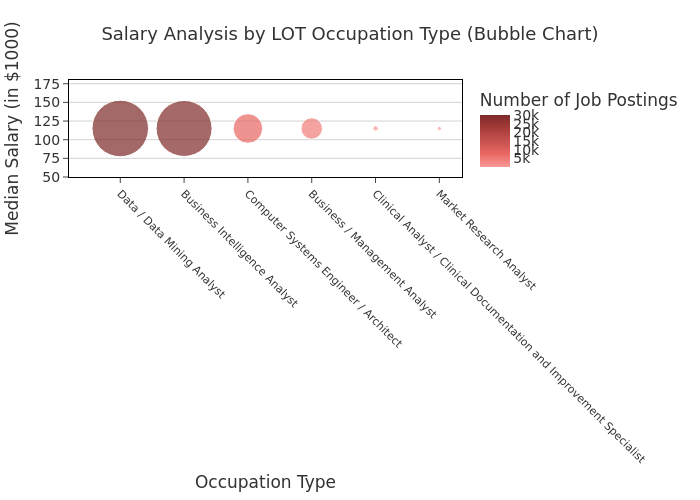


# 4. Salary Analysis by ONET Occupation Type (Bubble Chart)

* Analyze how salaries differ across ONET occupation types.
* **Aggregate Data**
  + Compute **median salary** for each occupation in the **ONET taxonomy**.
* **Visualize results**
  + Create a **bubble chart** where:
    - **X-axis** = ONET\_NAME
    - **Y-axis** = Median Salary
    - **Size** = Number of job postings
  + Apply custom colors and font styles.

# Spark SQL - Median salary and job count per LOT\_OCCUPATION\_NAME  
df.createOrReplaceTempView("Job\_Postings")  
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  
 GROUP BY LOT\_OCCUPATION\_NAME  
 ORDER BY Job\_Postings DESC  
 LIMIT 10  
""")  
  
# Converting to Pandas Data Frame  
salary\_pd = salary\_analysis.toPandas()  
  
# Converting salary to $1000 units  
salary\_pd["Median\_Salary"] = salary\_pd["Median\_Salary"] / 1000  
  
# Creating Bubble Chart  
custom\_coral\_scale = [  
 [0.0, "#f79a96"],  
 [0.25, "#eb6864"],  
 [0.5, "#c6524f"],  
 [0.75, "#a63b39"],  
 [1.0, "#7c2a29"]  
]  
fig = px.scatter(  
 salary\_pd,  
 x="Occupation\_Name",  
 y="Median\_Salary",  
 size="Job\_Postings",  
 color="Job\_Postings",  
 color\_continuous\_scale=custom\_coral\_scale,  
 title="Salary Analysis by LOT Occupation Type (Bubble Chart)",  
 labels={  
 "Occupation\_Name": "Occupation Type",  
 "Median\_Salary": "Median Salary (in $1000)",  
 "Job\_Postings": "Number of Job Postings"  
 },  
 hover\_name="Occupation\_Name",  
 width=1000,  
 height=500,  
 size\_max=40  
)  
  
fig.update\_layout(  
 font=dict(family="Arial", size=14, color="#333333"),  
 title=dict(x=0.5, xanchor="center", font=dict(size=18)),  
 plot\_bgcolor="white",  
 paper\_bgcolor="white",  
 xaxis=dict(  
 tickangle=45,  
 tickfont=dict(size=11),  
 showline=True,  
 linecolor='black',  
 title\_standoff=10,  
 showgrid=False,  
 zeroline=False,  
 ticks='outside',  
 mirror=True  
 ),  
 yaxis=dict(  
 tick0=0,  
 dtick=25,  
 range=[50, 180],  
 showline=True,  
 linecolor='black',  
 title\_standoff=10,  
 showgrid=True,  
 gridcolor='lightgray',  
 zeroline=False,  
 ticks='outside',  
 mirror=True  
 ),  
 coloraxis\_colorbar=dict(  
 tickvals=[5000, 10000, 15000, 20000, 25000, 30000],  
 ticktext=["5k", "10k", "15k", "20k", "25k", "30k"],  
 title="Number of Job Postings"  
 ),  
 margin=dict(l=60, r=40, t=80, b=140),  
 hovermode="closest"  
)  
  
fig.show()  
fig.write\_image("output/Q3\_BubbleChart\_V1.svg", width=1500, height=600, scale=1)

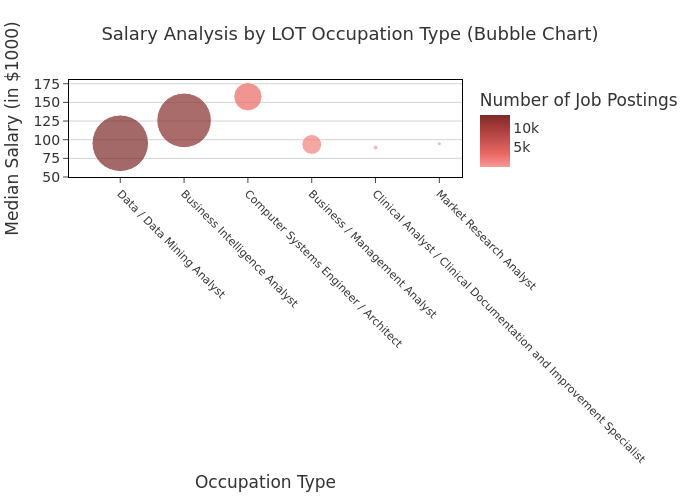
25/09/22 02:33:05 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.  
[Stage 8:> (0 + 1) / 1]



* **Explanation:** The bubble chart reveals that while job posting volumes vary widely across LOT occupation types, the median salaries appear uniformly clustered. This pattern is likely due to an earlier step where missing salary values were imputed with the overall median, flattening natural variation. Therefore, we take a further step to filter out the most common imputed value to improve accuracy.

# Spark SQL - Median salary and job count per LOT\_OCCUPATION\_NAME  
df.createOrReplaceTempView("Job\_Postings")  
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 SALARY IS NOT NULL   
 AND SALARY > 0   
 AND SALARY NOT IN (115024.0)   
 AND LOT\_OCCUPATION\_NAME IS NOT NULL  
 GROUP BY LOT\_OCCUPATION\_NAME  
 ORDER BY Job\_Postings DESC  
 LIMIT 10  
""")  
  
# Converting to Pandas Data Frame  
salary\_pd = salary\_analysis.toPandas()  
  
# Converting salary to $1000 units  
salary\_pd["Median\_Salary"] = salary\_pd["Median\_Salary"] / 1000  
  
# Creating Bubble Chart  
custom\_coral\_scale = [  
 [0.0, "#f79a96"],  
 [0.25, "#eb6864"],  
 [0.5, "#c6524f"],  
 [0.75, "#a63b39"],  
 [1.0, "#7c2a29"]  
]  
fig = px.scatter(  
 salary\_pd,  
 x="Occupation\_Name",  
 y="Median\_Salary",  
 size="Job\_Postings",  
 color="Job\_Postings",  
 color\_continuous\_scale=custom\_coral\_scale,  
 title="Salary Analysis by LOT Occupation Type (Bubble Chart)",  
 labels={  
 "Occupation\_Name": "Occupation Type",  
 "Median\_Salary": "Median Salary (in $1000)",  
 "Job\_Postings": "Number of Job Postings"  
 },  
 hover\_name="Occupation\_Name",  
 width=1000,  
 height=500,  
 size\_max=40  
)  
  
fig.update\_layout(  
 font=dict(family="Arial", size=14, color="#333333"),  
 title=dict(x=0.5, xanchor="center", font=dict(size=18)),  
 plot\_bgcolor="white",  
 paper\_bgcolor="white",  
 xaxis=dict(  
 tickangle=45,  
 tickfont=dict(size=11),  
 showline=True,  
 linecolor='black',  
 title\_standoff=10,  
 showgrid=False,  
 zeroline=False,  
 ticks='outside',  
 mirror=True  
 ),  
 yaxis=dict(  
 tick0=0,  
 dtick=25,  
 range=[50, 180],  
 showline=True,  
 linecolor='black',  
 title\_standoff=10,  
 showgrid=True,  
 gridcolor='lightgray',  
 zeroline=False,  
 ticks='outside',  
 mirror=True  
 ),  
 coloraxis\_colorbar=dict(  
 tickvals=[5000, 10000, 15000, 20000, 25000, 30000],  
 ticktext=["5k", "10k", "15k", "20k", "25k", "30k"],  
 title="Number of Job Postings"  
 ),  
 margin=dict(l=60, r=40, t=80, b=140),  
 hovermode="closest"  
)  
  
fig.show()  
fig.write\_image("output/Q3\_BubbleChart\_V2.svg", width=1500, height=600, scale=1)

[Stage 11:> (0 + 1) / 1]



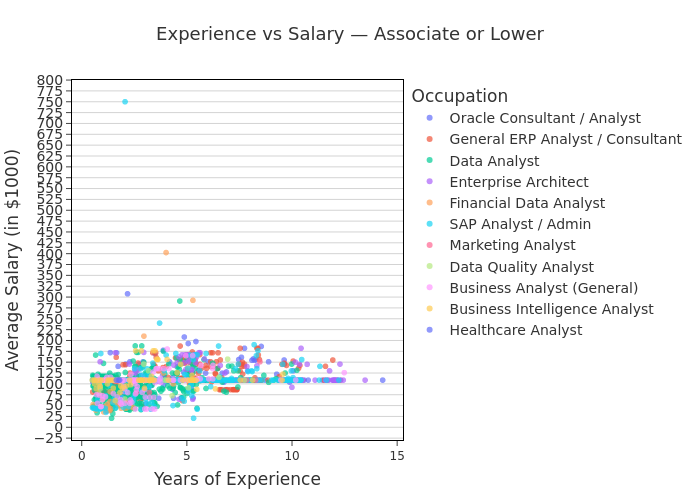
* **Explanation:** The bubble chart shows that job posting volumes vary significantly across LOT occupation types, with roles like Data/Data Mining Analyst and Business Intelligence Analyst dominating in demand. Median salaries, however, differ more clearly after filtering out previously imputed values,revealing that roles in system architecture and engineering tend to command higher median pay.

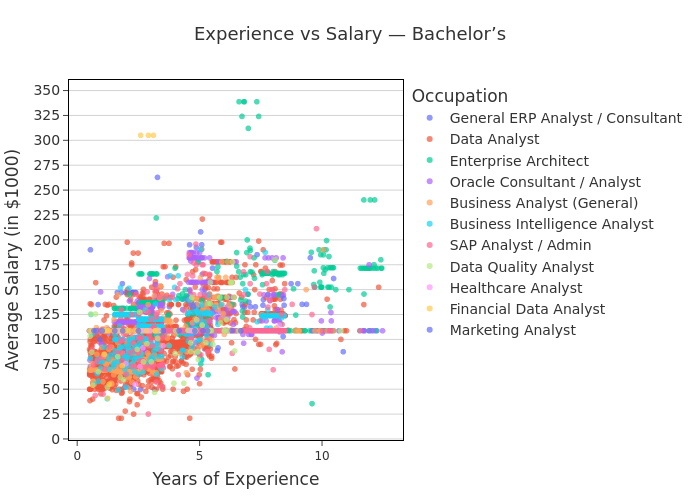
# 5. Salary by Education Level

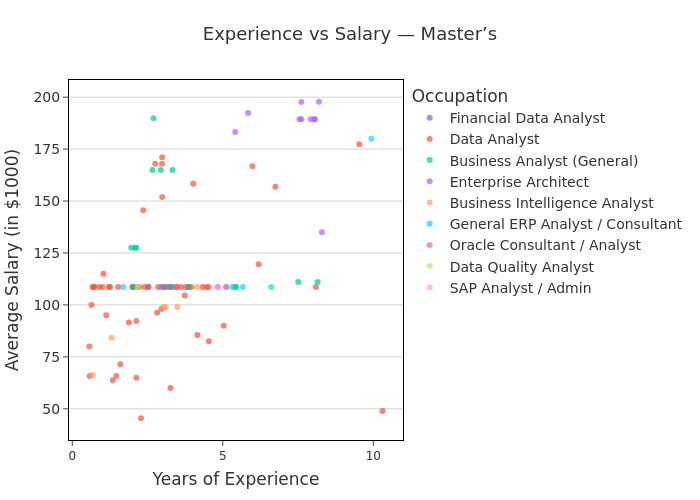
* Create four groups:
  + **Associate’s or lower** (GED, Associate, No Education Listed)
  + **Bachelor’s** (Bachelor’s degree)
  + **Master’s** (Master’s degree)
  + **PhD** (PhD, Doctorate, professional degree)
* Plot scatter plots for each group using, MAX\_YEARS\_EXPERIENCE (with jitter), Average\_Salary, LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME
* **Short explanation** of key insights for each graph:
  + **Associate or Lower**: Most salaries cluster between $50K–$150K. A few outliers exceed $300K–$700K, but they are extremely rare and likely exceptional cases. Increasing years of experience does not significantly improve salary for this group, indicating a potential ceiling for career growth without further education.
  + **Bachelor’s**: There is a noticeable upward trend—salaries increase with experience more consistently than the Associate group. Most salaries fall within $70K–$150K, with the median visibly higher than the Associate group. Broader range of occupations are represented, potentially offering better career mobility.
  + **Master’s**: Few roles pay below $100K, indicating a stronger starting point compared to the lower education groups. Despite the smaller sample size, salaries for experienced professionals (5+ years) often exceed $150K. More linear progression, suggesting Master’s degree offers good return on investment for salary advancement.
  + **PhD**: Limited sample, but high value. Even with low years of experience, this group commands higher salaries—likely due to specialized roles or niche expertise. This suggests deep specialization can offset lack of experience when entering the job market.

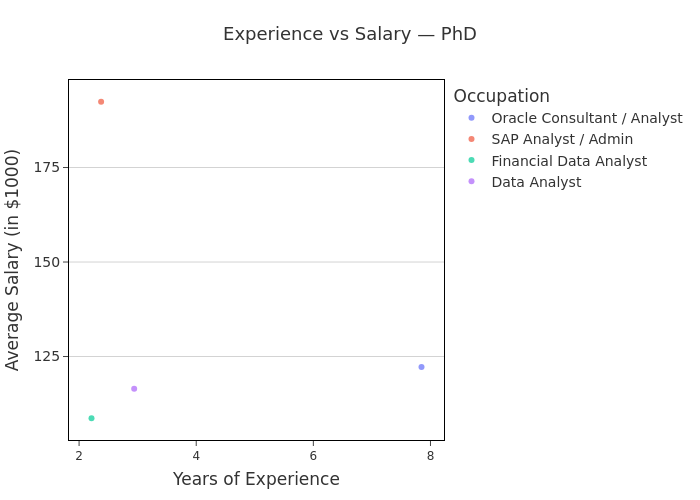
from pyspark.sql.functions import col, when, rand  
  
# Defining education level groupings  
associate\_or\_lower = ["GED", "Associate", "No Education Listed", "High school"]  
bachelor = ["Bachelor"]  
master = ["Master"]  
phd = ["Ph.D", "Doctorate", "professional degree"]  
  
# Adding EDU\_GROUP column  
df = df.withColumn(  
 "EDU\_GROUP",  
 when(col("EDUCATION\_LEVELS\_NAME").rlike("|".join([f"(?i){deg}" for deg in associate\_or\_lower])), "Associate or Lower")  
 .when(col("EDUCATION\_LEVELS\_NAME").rlike("|".join([f"(?i){deg}" for deg in bachelor])), "Bachelor’s")  
 .when(col("EDUCATION\_LEVELS\_NAME").rlike("|".join([f"(?i){deg}" for deg in master])), "Master’s")  
 .when(col("EDUCATION\_LEVELS\_NAME").rlike("|".join([f"(?i){deg}" for deg in phd])), "PhD")  
 .otherwise("Other")  
)  
  
# 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"))  
  
# Adding jitter to avoid overlapping dots in scatter plot  
df = df.withColumn("Jittered\_Experience", col("MAX\_YEARS\_EXPERIENCE") + (rand() - 0.5))  
  
# 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)  
)  
  
# Keeping only four major groups  
df\_filtered = df.filter(  
 col("EDU\_GROUP").isin("Associate or Lower", "Bachelor’s", "Master’s", "PhD")  
)  
  
# Converting to Pandas for plotting  
df\_pd = df\_filtered.select(  
 "EDU\_GROUP",  
 "Jittered\_Experience",  
 "Average\_Salary",  
 "LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME"  
).toPandas()  
  
# Creating scatter plots for each group  
  
edu\_groups = ["Associate or Lower", "Bachelor’s", "Master’s", "PhD"]  
  
for group in edu\_groups:  
 subset = df\_pd[df\_pd["EDU\_GROUP"] == group].copy()  
   
 subset["Salary\_K"] = subset["Average\_Salary"] / 1000  
  
 fig = px.scatter(  
 subset,  
 x="Jittered\_Experience",  
 y="Salary\_K",  
 color="LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME",  
 title=f"Experience vs Salary — {group}",  
 labels={  
 "Jittered\_Experience": "Years of Experience",  
 "Salary\_K": "Average Salary (in $1000)",  
 "LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME": "Occupation"  
 },  
 opacity=0.7,  
 width=1000,  
 height=600  
 )  
  
 fig.update\_layout(  
 font=dict(family="Arial", size=14, color="#333333"),  
 title=dict(x=0.5, xanchor="center", font=dict(size=18)),  
 plot\_bgcolor="white",  
 paper\_bgcolor="white",  
 xaxis=dict(  
 tickangle=0,  
 tickfont=dict(size=12),  
 showline=True,  
 linecolor='black',  
 title\_standoff=10,  
 showgrid=False,  
 zeroline=False,  
 ticks='outside',  
 mirror=True  
 ),  
 yaxis=dict(  
 title="Average Salary (in $1000)",  
 tick0=0,  
 dtick=25,  
 showline=True,  
 linecolor='black',  
 title\_standoff=10,  
 showgrid=True,  
 gridcolor='lightgray',  
 zeroline=False,  
 ticks='outside',  
 mirror=True  
 ),  
 margin=dict(l=60, r=40, t=80, b=60),  
 legend\_title\_text="Occupation",  
 hovermode="closest"  
 )  
  
 fig.show()  
 safe\_group = group.replace("’", "").replace(" ", "\_")   
 fig.write\_image(f"output/Q4\_experience\_salary\_{safe\_group}.svg", width=1500, height=600, scale=1)

[Stage 14:> (0 + 1) / 1]









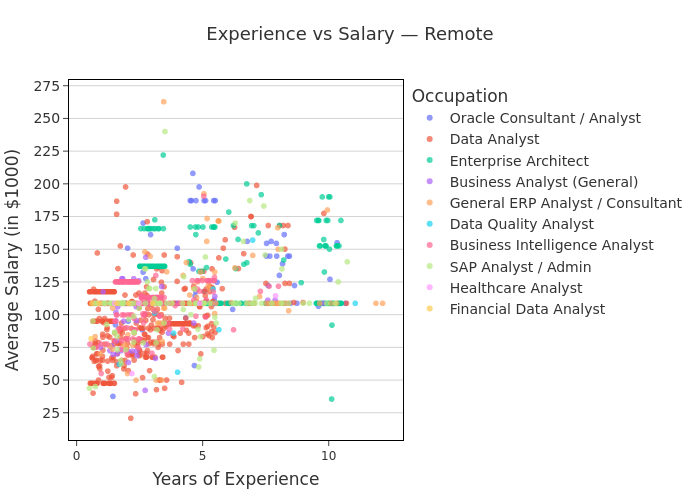
# 6. Salary by Remote Work Type

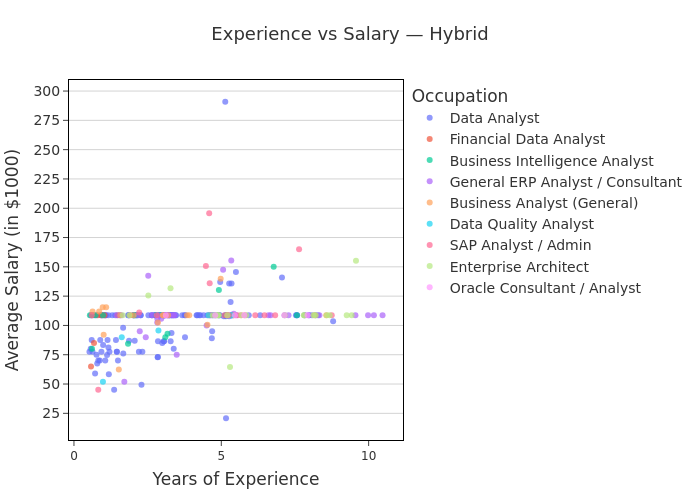
* Split into three groups based on REMOTE\_TYPE\_NAME:
  + Remote
  + Hybrid
  + Onsite (includes [None] and blank)
* Plot scatter plots for each group using, MAX\_YEARS\_EXPERIENCE (with jitter), Average\_Salary, LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME
* Also, create salary histograms for all three groups.
* **After each graph, briefly describe any patterns or comparisons.**

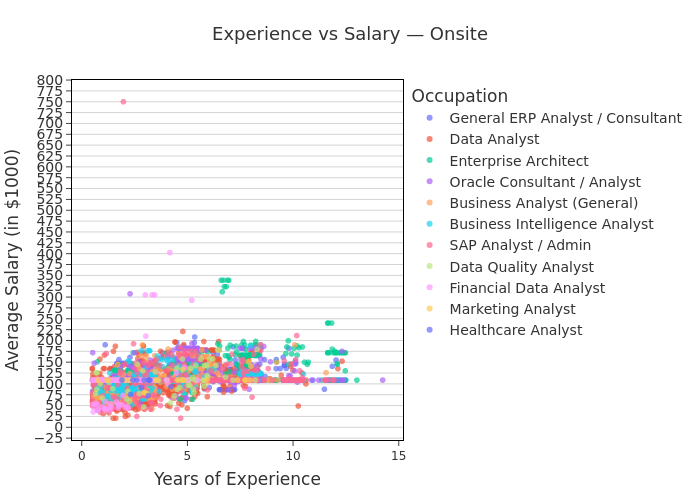
## 6.1 Scatter Plots by Remote Group

from pyspark.sql.functions import col, when, rand  
  
# Creating a new column REMOTE\_GROUP  
df = df.withColumn(  
 "REMOTE\_GROUP",  
 when(col("REMOTE\_TYPE\_NAME") == "Remote", "Remote")  
 .when(col("REMOTE\_TYPE\_NAME") == "Hybrid Remote", "Hybrid")  
 .otherwise("Onsite")  
)  
  
# Adding jitter to avoid overlapping dots  
df = df.withColumn("Jittered\_Experience", col("MAX\_YEARS\_EXPERIENCE") + (rand() - 0.5))  
  
# Filtering for clean values  
df\_filtered = df.filter(  
 col("MAX\_YEARS\_EXPERIENCE").isNotNull() &  
 col("Average\_Salary").isNotNull() &  
 (col("MAX\_YEARS\_EXPERIENCE") > 0) &  
 (col("Average\_Salary") > 0)  
)  
  
# Converting to Pandas for Plotly  
df\_pd = df\_filtered.select(  
 "REMOTE\_GROUP",  
 "Jittered\_Experience",  
 "Average\_Salary",  
 "LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME"  
).toPandas()  
  
# Creating scatter plots for each group  
remote\_groups = ["Remote", "Hybrid", "Onsite"]  
  
for group in remote\_groups:  
 subset = df\_pd[df\_pd["REMOTE\_GROUP"] == group].copy()  
  
 if subset.shape[0] < 10:  
 print(f"Skipping plot for {group} (not enough data)")  
 continue  
  
 subset["Salary\_K"] = subset["Average\_Salary"] / 1000  
  
 fig = px.scatter(  
 subset,  
 x="Jittered\_Experience",  
 y="Salary\_K",  
 color="LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME",  
 title=f"Experience vs Salary — {group}",  
 labels={  
 "Jittered\_Experience": "Years of Experience",  
 "Salary\_K": "Average Salary (in $1000)",  
 "LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME": "Occupation"  
 },  
 opacity=0.7,  
 width=1000,  
 height=600  
 )  
  
 fig.update\_layout(  
 font=dict(family="Arial", size=14, color="#333333"),  
 title=dict(x=0.5, xanchor="center", font=dict(size=18)),  
 plot\_bgcolor="white",  
 paper\_bgcolor="white",  
 xaxis=dict(  
 tickangle=0,  
 tickfont=dict(size=12),  
 showline=True,  
 linecolor='black',  
 title\_standoff=10,  
 showgrid=False,  
 zeroline=False,  
 ticks='outside',  
 mirror=True  
 ),  
 yaxis=dict(  
 title="Average Salary (in $1000)",  
 tick0=0,  
 dtick=25,  
 showline=True,  
 linecolor='black',  
 title\_standoff=10,  
 showgrid=True,  
 gridcolor='lightgray',  
 zeroline=False,  
 ticks='outside',  
 mirror=True  
 ),  
 margin=dict(l=60, r=40, t=80, b=60),  
 legend\_title\_text="Occupation",  
 hovermode="closest"  
 )  
  
 fig.show()  
 safe\_name = group.replace(" ", "\_").lower()  
 fig.write\_image(f"output/Q5\_remote\_salary\_{safe\_name}.svg", width=1200, height=600)

[Stage 15:> (0 + 1) / 1]



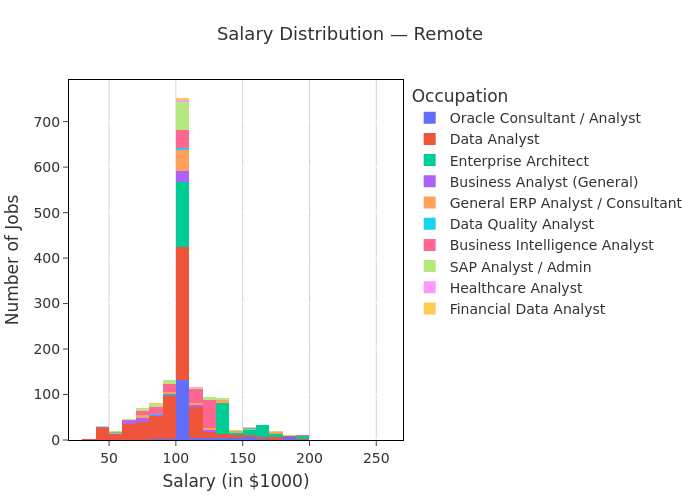


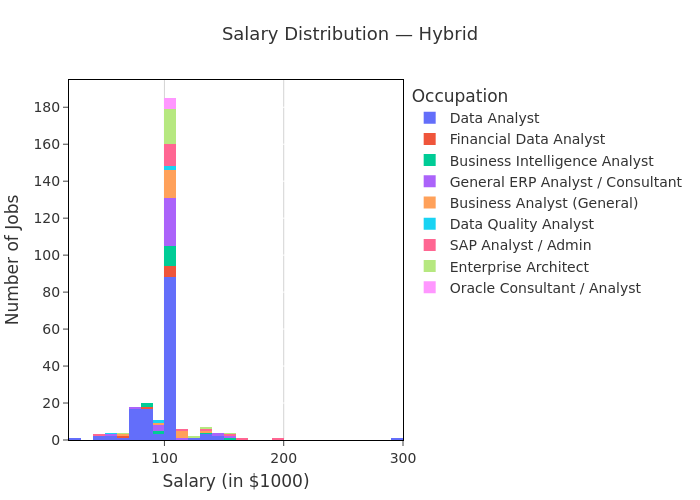


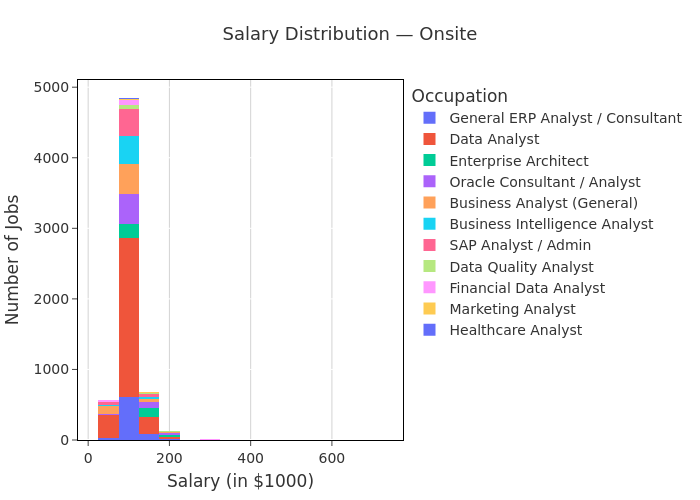
* **Explanation:**
  + **Onsite** roles show the widest salary range, with outliers >$700K, especially among senior roles like Enterprise Architects.
  + **Remote** roles have a more moderate salary distribution, mostly under $250K.
  + **Hybrid** roles show the narrowest salary range, mostly $50K–$200K.
  + **All three show a positive trend**: more experience generally leads to higher salary.
  + **Enterprise Architects** tends to earn the highest salaries, especially in onsite roles.

## 6.2 Histograms of Salary by Remote Group

# Converting salary to $1000 units  
df\_pd["Salary\_K"] = df\_pd["Average\_Salary"] / 1000  
  
# Filtering out rows with missing salary or REMOTE\_GROUP  
filtered\_df = df\_pd.dropna(subset=["Average\_Salary", "REMOTE\_GROUP"])  
  
# Ploting histogram for each remote group  
for remote\_type in ["Remote", "Hybrid", "Onsite"]:  
 subset = filtered\_df[filtered\_df["REMOTE\_GROUP"] == remote\_type]  
  
 fig = px.histogram(  
 subset,  
 x="Salary\_K",  
 color="LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME",  
 nbins=30,  
 title=f"Salary Distribution — {remote\_type}",  
 labels={  
 "Salary\_K": "Average Salary (in $1000)",  
 "LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME": "Occupation"  
 },  
 barmode="stack",  
 width=1000,  
 height=500  
 )  
  
 fig.update\_layout(  
 font=dict(family="Arial", size=14, color="#333333"),  
 title=dict(x=0.5, xanchor="center", font=dict(size=18)),  
 xaxis=dict(  
 title="Salary (in $1000)",  
 tickformat=",d",  
 showline=True,  
 linecolor='black',  
 title\_standoff=10,  
 showgrid=True,  
 gridcolor='lightgray',  
 zeroline=False,  
 ticks='outside',  
 mirror=True  
 ),  
 yaxis=dict(  
 title="Number of Jobs",  
 showline=True,  
 linecolor='black',  
 title\_standoff=10,  
 ticks='outside',  
 mirror=True  
 ),  
 plot\_bgcolor="white",  
 paper\_bgcolor="white",  
 legend\_title="Occupation",  
 margin=dict(l=60, r=40, t=80, b=60)  
 )  
  
 fig.show()  
 fig.write\_image(f"output/Q5\_salary\_histogram\_{remote\_type.lower()}.svg", width=1500, height=600, scale=1)







* **Explanation:**
  + **Remote** roles offer consistency in pay but fewer outliers or variation.
    - **Onsite** roles are much more numerous and include extreme high-salary cases.
    - **Hybrid** roles are the least common but demonstrate a more balanced mix.
    - **Data Analyst** is the most common role across all remote types, underscoring its cross-location demand.