assignment03-Cindy Guzman

September 25, 2025

1 Assignment 03

Cindy Guzman (Boston University)
September 21, 2025

2 1. Load the Dataset

The instruction below provides you with general keywords for columns used in the lightcast file. See the data schema generated after the load dataset code above to use proper column name. For each visualization, **customize colors, fonts, and styles** to avoid a **2.5-point deduction**. Also, **provide a two-sentence explanation** describing key insights drawn from the graph.

1. **Load the Raw Dataset**: - -Use Pyspark to the 'lightcast_data.csv' file into DataFrame: -You can reuse the previous code. -Copying code from your friend constitutes plagiarism. DO NOT DO THIS.

3 Data Exploration and Visualization

Dataset imported successfully, Plotly will be utilized to explore and visualize the data.

[4]:
```python
import pandas as pd
import plotly.express as px
import plotly.io as pio
pio.renderers.default = "png+jpg+svg"
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"

# 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 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).
25/09/25 19:20:14 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
```

[5]:
```
# Histogram of SALARY distribution
from pyspark.sql.functions import floor

binned_df = df.filter(col("SALARY").isNotNull() & (col("SALARY") > 0)) \
              .withColumn("SALARY_BIN", floor(col("SALARY") / 5000) * 5000) \
              .groupBy("SALARY_BIN").count() \
              .orderBy("SALARY_BIN") \
              .toPandas()

fig = px.bar(binned_df, x="SALARY_BIN", y="count", title="Salary Distribution␣
↪by Bin")
fig.update_layout(xaxis_title="Salary Bin", yaxis_title="Frequency", bargap=0.1)
```

4 2. Step 1: Create Companies Table (Primary Key: company_id)

[6]:
```
companies_df = df.select(
  col("company"),
  col("company_name"),
  col("company_raw"),
  col("company_is_staffing")
).distinct().withColumn("company_id", monotonically_increasing_id())
# companies_df.show(5)
companies = companies_df.toPandas()
```

```
companies.drop(columns=["company"], inplace=True)
companies.rename(columns={"company_is_staffing": "is_staffing"}, inplace=True)
companies.to_csv("./output/companies.csv", index=False)
companies.head()
```

[6]:

```
                                company_name  \
0                                      Crowe
1                    The Devereux Foundation
2                             Elder Research
3                                   NTT DATA
4  Frederick National Laboratory For Cancer Research

                                 company_raw is_staffing  company_id
0                                      Crowe       False           0
1                    The Devereux Foundation       False           1
2                             Elder Research       False           2
3                               NTT DATA Inc       False           3
4  Frederick National Laboratory for Cancer Research       False           4
```

5 3. Data Preparation

[7]:

```
# Step 1: Casting salary and experience columns
from pyspark.sql.functions import when
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 medians 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: Fill missing values with medians
df = df.fillna({
```

```
  "SALARY_FROM": median_from,
  "SALARY_TO": median_to,
  "SALARY": median_salary
  })

#Step 4: Compute Average_Salary using filled values
df = df.withColumn(
    "Average_Salary",
    when(
        col("SALARY_FROM").isNull() & col("SALARY_TO").isNull(),
        col("SALARY")
    ).otherwise((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))
pdf.head() # Preview the first few rows
```

```
Medians: 87295.0 130042.0 115024.0
```

```
Data cleaning complete. Rows retained: 72498
```

[7]:

```
             EDUCATION_LEVELS_NAME REMOTE_TYPE_NAME  MAX_YEARS_EXPERIENCE  \
0    [\n  "Bachelor's degree"\n]           [None]                   2.0
1  [\n  "No Education Listed"\n]           Remote                   3.0
2    [\n  "Bachelor's degree"\n]           [None]                   NaN
3  [\n  "No Education Listed"\n]           [None]                   NaN
4  [\n  "No Education Listed"\n]           [None]                   NaN

   Average_Salary    SALARY LOT_V6_SPECIALIZED_OCCUPATION_NAME
0        108668.5  115024.0   General ERP Analyst / Consultant
```

```
1         108668.5  115024.0        Oracle Consultant / Analyst
2         108668.5  115024.0                       Data Analyst
3         108668.5  115024.0                       Data Analyst
4          92500.0   92500.0        Oracle Consultant / Analyst
```

[8]:

```
# Your code for 1st question here
import pandas as pd
# Filter out missing or zero salary values
pdf = df.filter((df["SALARY"] > 0) & (df["EMPLOYMENT_TYPE_NAME"].isNotNull())).
 ↪select("EMPLOYMENT_TYPE_NAME", "SALARY").toPandas()
pdf.head()
```

[8]:

	EMPLOYMENT_TYPE_NAME	SALARY
0	Full-time (> 32 hours)	115024.0
1	Full-time (> 32 hours)	115024.0
2	Full-time (> 32 hours)	115024.0
3	Full-time (> 32 hours)	115024.0
4	Part-time / full-time	92500.0

6 5. Clean employment type names for better readability

[9]:

```
import re
pdf["EMPLOYMENT_TYPE_NAME"] = pdf["EMPLOYMENT_TYPE_NAME"].apply(lambda x: re.
 ↪sub(r"[^\x00-\x7F]+", "", x))
pdf.head()
```

[9]:

	EMPLOYMENT_TYPE_NAME	SALARY
0	Full-time (> 32 hours)	115024.0
1	Full-time (> 32 hours)	115024.0
2	Full-time (> 32 hours)	115024.0
3	Full-time (> 32 hours)	115024.0
4	Part-time / full-time	92500.0

[10]:

```
# 6. Compute median salary for sorting

median_salaries = pdf.groupby("EMPLOYMENT_TYPE_NAME")["SALARY"].median()
median_salaries.head()
```

[10]:

```
EMPLOYMENT_TYPE_NAME
Full-time (> 32 hours)     115024.0
Part-time ( 32 hours)      115024.0
Part-time / full-time      115024.0
Name: SALARY, dtype: float32
```

7 9. 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.
  - Calculate **salary percentiles** (25th, 50th, 75th) for each group.
- **Visualize results**
  - Create a **box plot** where:
  - **X-axis** = `NAICS2_NAME`
  - **Y-axis** = `SALARY_FROM`, or `SALARY_TO`, or `SALARY`
  - Group by `EMPLOYMENT_TYPE_NAME`.
- Customize colors, fonts, and styles.
- **Explanation:** Write two sentences about what the graph reveals.

[11]:
```
# Sort employment types based on median salary in descending order
sorted_employment_types = median_salaries.sort_values(ascending=False).index

# Apply sorted categories
pdf["EMPLOYMENT_TYPE_NAME"] = pd.Categorical(pdf["EMPLOYMENT_TYPE_NAME"], 
 ↪categories=sorted_employment_types, ordered=True)



# Box Plot with horizontal orientation grid lines
fig = px.box(
    pdf,
    x="SALARY",
    y="EMPLOYMENT_TYPE_NAME",
    orientation="h",
    title="Salary Distribution by Employment Type",
    color="EMPLOYMENT_TYPE_NAME",
    color_discrete_map={
    "Full-time (  32 hours)": "#1f77b4",
    "Part-time (32 hours)": "#ff7f0e",
    "Part-time / Full-time": "#2ca02c"
}, # Single neutral color
    boxmode='group',
    points="outliers",  # Show all outliers
)
fig.update_layout(title_x=0.5)

# Improve outlier visibility
fig.update_traces(
    marker=dict(
        size=7,           # Larger point size
```

```
        opacity=0.7,      # Slight transparency
        line=dict(width=0.5, color='black')  # Thin border for contrast
    ),
    jitter=0.4,           # Spread overlapping points
    boxpoints='outliers'  # Ensure outliers are shown
)

# Layout improvements, font styles, and axis labels
fig.update_layout(
    title=dict(
       text="Salary Distribution by Employment Type",
       font=dict(size=30, family="Calibri", color="black")
    ),
    xaxis=dict(
        title=dict(text="Salary (USD $1000)", font=dict(size=14,␣
 ↪family="Calibri", color="black")), # Bigger label for x-axis
        tickangle=0,
        tickfont=dict(size=12, family="Calibri", color="black"),
        showline=True,
        linewidth=2,
        linecolor='black',
        mirror=True,
        showgrid=False,
        categoryorder='array',
        categoryarray=sorted_employment_types.tolist()
    ),
    yaxis=dict(
        title=dict(text="Employment Type", font=dict(size=14, family="Calibri",␣
 ↪color="black")), # Bigger label for y-axis
        tickvals=[0, 50000, 100000, 150000, 200000, 250000, 300000, 350000,␣
 ↪400000, 450000, 500000],
        ticktext=["0", "50", "100", "150", "200", "250", "300", "350", "400",␣
 ↪"450", "500"],
        tickfont=dict(size=12, family="Calibri", color="black", weight="bold"),
        showline=True,
        linewidth=2,
        linecolor='black',
        mirror=True,
        showgrid=True,
        gridcolor='lightgrey',
        gridwidth=0.5,
    ),
    font=dict(family="Calibri", size=12, color="black"),
    boxgap=0.5,
    plot_bgcolor='white',
    paper_bgcolor='white',
    showlegend=False,
```

```
    height=600,
    width=900,
)

 # Show figure
fig.show()
fig.write_html("./output/boxplot_salary_by_employment_type.html")
```

8 Salary Distribution by Employment Type Insight

The box plot reveals that full-time roles offer a higher median salaries compared to part-time or mixed employment types. Salary variability is also greater in full-time positions, which suggests a wider range of compensation across industries.

[12]:
```
#| eval: false
#| echo: true
#| fig-align :center
pdf = df.select("NAICS2_NAME", "SALARY").toPandas()
fig = px.box(pdf, x="NAICS2_NAME", y="SALARY", title="Salary Distribution by␣
↪Industry", color_discrete_sequence=["#1f77b4"])
fig.update_layout(font_family="Calibri", title_font_size=30,␣
↪font_color="black", title_x=0.5, height=900, width=1110)
fig.show()
```

9 Salary Distribution by Industry Type and NAICS2_NAME

The box plot reveals significant variation in salary distributions across a variety of disciplines. Fields such as Engineering, Computer Science, and Legal Studies show higher median salaries and broader ranges, this indicates a strong earning potential as well as variability. In contrast, disciplines like Theology, Library Science, and the Arts tend to have lower salary medians with narrower spreads, this suggest more consistent but modest compensation.

10 10. 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.

- **Explanation:** Write two sentences about what the graph reveals.

[13]:
```
#| eval: false
#| echo: false
# Spark SQL - Median salary and job count per ONET_NAME
df.createOrReplaceTempView("Job_Postings")
salary_analysis = spark.sql("""
                            SELECT
    LOT_SPECIALIZED_OCCUPATION_NAME AS Occupation_Name,
    PERCENTILE(SALARY, 0.5) AS Median_Salary,
    COUNT(*) AS Job_Postings
    FROM Job_Postings
    GROUP BY LOT_SPECIALIZED_OCCUPATION_NAME
    ORDER BY Job_Postings DESC
    LIMIT 10
""")

# Convert to Pandas DataFrame for visualization
salary_pdf = salary_analysis.toPandas()
salary_pdf.head()

# Bubble chart using plotly
import plotly.express as px

fig = px.scatter(
    salary_pdf,
    x="Occupation_Name",
    y="Median_Salary",
    size="Job_Postings",
    title="Salary Analysis by LOT Specialized Occupation Type (Bubble Chart)",
    labels= {"LOT_SPECIALIZED_OCCUPATION_NAME": "LOT Occupation",
             "Median_Salary": "Median Salary ($1000)",
             "Job_Postings": "Number of Job Postings"},
             hover_name="Occupation_Name",
             size_max=60,
             width=1000,
             height=600,
             color="Job_Postings",
             color_continuous_scale="Spectral"
)

# Layout improvements
fig.update_layout(
    font_family="Calibri",
    font_size=14,
    title_font_size=24,
    title_x=0.5,
```

```
    xaxis_title="LOT Specialized Occupation",
    yaxis_title="Median Salary ($1000)",
    plot_bgcolor='white',
    xaxis=dict(
        tickangle=45,
        showline=True,
        linecolor='black',
    ),
    yaxis=dict(
        showline=True,
        linecolor='black'
    )
)

# Show figure
fig.show()
```

```
25/09/25 19:21:18 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'.
```

11 Salary Analysis by LOT Specialized Occupation Type

The bubble chart reveals that even though the median salaries across specialized data-related occupations are relatively consistent, job demand varies significantly. Roles such as Data Analyst and Business Analyst have larger bubbles, which indicates higher posting volumes. Looking at niche positions such as Enterprise Architect or SAP Analyst, they offer similar pay but the opportunities are fewer .

12 11. Salary by Education Level

- Create two groups:
  - **Bachelor's or lower** (Bachelor's, GED, Associate, No Education Listed)
  - **Master's or PhD** (Master's degree, Ph.D. or professional degree)
- Plot scatter plots for each group using, `MAX_YEARS_EXPERIENCE` (with jitter), `Average_Salary`, `LOT_V6_SPECIALIZED_OCCUPATION_NAME`
- Then, plot histograms overlaid with KDE curves for each group.
- This would generate two scatter plots and two histograms.
- **After each graph, add a short explanation** of key insights.

[24]:
```
from pyspark.sql.functions import col, regexp_replace, split, explode

# Step 1: Clean up brackets and newlines
df_clean = df.withColumn(
    "EDUCATION_LEVELS",
    regexp_replace(col("EDUCATION_LEVELS"), "[\\[\\]\\n]", "")
)
```

```
# Step 2: Split into array on commas
df_clean = df_clean.withColumn(
    "EDUCATION_LEVELS",
    split(col("EDUCATION_LEVELS"), ",\\s*")
)

# Step 3: Explode into rows
df_exploded = df_clean.withColumn("EDU_LEVEL", explode(col("EDUCATION_LEVELS")))

# Step 4: Trim spaces just in case
from pyspark.sql.functions import trim
df_exploded = df_exploded.withColumn("EDU_LEVEL", trim(col("EDU_LEVEL")))

# Preview result
## df_exploded.select("EDUCATION_LEVELS", "EDU_LEVEL").show(truncate=False)
```

[25]:

```
from pyspark.sql.functions import col, explode, trim, lower, when,␣
 ↪regexp_replace, split
from pyspark.sql.types import ArrayType, StringType

# EDUCATION_LEVELS

df_clean = df.withColumn(
    "EDUCATION_LEVELS",
    regexp_replace(col("EDUCATION_LEVELS"), "[\\[\\]\\n]", "")
)

# Split into array on commas
df_clean = df_clean.withColumn(
    "EDUCATION_LEVELS",
    split(col("EDUCATION_LEVELS"), ",\\s*")
)

# Explode into rows
df_exploded = df_clean.withColumn("EDU_CODE", explode(col("EDUCATION_LEVELS")))

# Trim spaces
df_exploded = df_exploded.withColumn("EDU_CODE", trim(col("EDU_CODE")))


# Map numeric codes -> labels (adjust if your mapping is different)

df_exploded = df_exploded.withColumn(
    "EDU_LEVEL",
    when(col("EDU_CODE") == "0", "No Education Listed")
    .when(col("EDU_CODE") == "1", "GED")
```

```python
    .when(col("EDU_CODE") == "2", "High School")
    .when(col("EDU_CODE") == "3", "Associate")
    .when(col("EDU_CODE") == "4", "Bachelor's Degree")
    .when(col("EDU_CODE") == "5", "Master's Degree")
    .when(col("EDU_CODE") == "6", "PhD / Professional Degree")
    .when(col("EDU_CODE") == "99", "Other / Unknown")
    .otherwise("Other")
)

# Normalize and group

df_exploded = df_exploded.withColumn("EDU_LEVEL", trim(lower(col("EDU_LEVEL"))))

associate_or_lower = [x.lower() for x in [
    "ged", "no education listed", "high school", "high school or ged",␣
↪"associate"
]]
bachelor = ["bachelor's degree", "bachelor"]
masters = ["master's degree", "masters"]
phd = ["phd / professional degree", "phd", "doctorate", "professional degree"]

df_exploded = df_exploded.withColumn(
    "EDU_GROUP",
    when(col("EDU_LEVEL").isin(associate_or_lower), "Associate")
    .when(col("EDU_LEVEL").isin(bachelor), "Bachelor's")
    .when(col("EDU_LEVEL").isin(masters), "Master's")
    .when(col("EDU_LEVEL").isin(phd), "PhD")
    .otherwise("Other")
)


# Clean numeric columns

df_exploded = df_exploded.withColumn(
    "MAX_YEARS_EXPERIENCE", col("MAX_YEARS_EXPERIENCE").cast("float")
)
df_exploded = df_exploded.withColumn(
    "Average_Salary", col("Average_Salary").cast("float")
)


# Filter valid rows

df_filtered = df_exploded.filter(
    col("MAX_YEARS_EXPERIENCE").isNotNull() &
    (col("MAX_YEARS_EXPERIENCE") > 0) &
    col("Average_Salary").isNotNull() &
```

```
    (col("Average_Salary") > 0) &
    col("EDU_GROUP").isin(["Associate", "Bachelor's", "Master's", "PhD"])
)

# Convert to Pandas for visualization

df_pd = df_filtered.toPandas()

# Check distribution of EDU_GROUP
##print(df_pd.head())
##print(df_pd["EDU_GROUP"].value_counts())
```

[26]:
```
#Jitter and Trim
df_pd["MAX_YEARS_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)

df_pd = df_pd[df_pd["Average_Salary_JITTER"] <= 399000]
df_pd.head()
```

[26]:
```
                                         ID LAST_UPDATED_DATE  \
0  1f57d95acf4dc67ed2819eb12f049f6a5c11782c          9/6/2024
1  229620073766234e814e8add21db7dfaef69b3bd         10/9/2024
2  229620073766234e814e8add21db7dfaef69b3bd         10/9/2024
3  138ce2c9453b47a9b33403c364d4fd80996caa4f         8/10/2024
4  138ce2c9453b47a9b33403c364d4fd80996caa4f         8/10/2024

   LAST_UPDATED_TIMESTAMP  DUPLICATES    POSTED   EXPIRED  DURATION  \
0 2024-09-06 20:32:57.352           0  6/2/2024  6/8/2024       6.0
1 2024-10-09 18:07:44.758           0  6/2/2024  8/1/2024       NaN
2 2024-10-09 18:07:44.758           0  6/2/2024  8/1/2024       NaN
3 2024-08-10 19:36:49.244           5  6/2/2024  8/9/2024       NaN
4 2024-08-10 19:36:49.244           5  6/2/2024  8/9/2024       NaN

                                        SOURCE_TYPES  \
0                                  [\n  "Company"\n]
1                                  [\n  "Company"\n]
2                                  [\n  "Company"\n]
3  [\n  "Job Board",\n  "Education",\n  "Recruite…
4  [\n  "Job Board",\n  "Education",\n  "Recruite…

                                             SOURCES  \
0                            [\n  "brassring.com"\n]
```

```
1                                  [\n  "3ds.com"\n]
2                                  [\n  "3ds.com"\n]
3  [\n  "silkroad.com",\n  "hercjobs.org",\n  "di…
4  [\n  "silkroad.com",\n  "hercjobs.org",\n  "di…

                                                 URL  …  \
0  [\n  "https://sjobs.brassring.com/TGnewUI/Sear…  …
1  [\n  "https://www.3ds.com/careers/jobs/sr-mark…  …
2  [\n  "https://www.3ds.com/careers/jobs/sr-mark…  …
3  [\n  "https://main.hercjobs.org/jobs/20166141/…  …
4  [\n  "https://main.hercjobs.org/jobs/20166141/…  …

                               NAICS_2022_5_NAME NAICS_2022_6  \
0       Automotive Parts and Accessories Retailers       441330
1      Computer Systems Design and Related Services       541511
2      Computer Systems Design and Related Services       541511
3  Colleges, Universities, and Professional Schools       611310
4  Colleges, Universities, and Professional Schools       611310

                               NAICS_2022_6_NAME Average_Salary  \
0       Automotive Parts and Accessories Retailers       108668.5
1             Custom Computer Programming Services        92962.0
2             Custom Computer Programming Services        92962.0
3  Colleges, Universities, and Professional Schools       108668.5
4  Colleges, Universities, and Professional Schools       108668.5

  REMOTE_GROUP  EDU_CODE    EDU_LEVEL  EDU_GROUP MAX_YEARS_EXPERIENCE_JITTER  \
0       Onsite         2  high school  Associate                        1.78
1       Onsite         2  high school  Associate                        2.16
2       Onsite         3    associate  Associate                        1.84
3       Remote         1          ged  Associate                        4.95
4       Remote         2  high school  Associate                        4.85

   Average_Salary_JITTER
0              108384.97
1               91977.68
2               92953.22
3              108691.38
4              110094.66

[5 rows x 138 columns]
```

[ ]:

```python
# Standardize and lock the four groups (fix common spelling/quote variants)
all_groups = ["Associate", "Bachelor's", "Master's", "PhD"]
df_pd["EDU_GROUP"] = (
    df_pd["EDU_GROUP"]
    .astype(str)
```

```
    .str.strip()
    .replace({
        "associate": "Associate",
        "associates": "Associate",
        "bachelors": "Bachelor's",
        "bachelor’s": "Bachelor's",
        "masters": "Master's",
        "master’s": "Master's",
        "phd": "PhD",
        "ph.d.": "PhD",
        "ph.d": "PhD",
        "doctorate": "PhD",
        "professional degree": "PhD",
    })
)
# Keep only the 4 buckets (optional-remove this line if you want to keep␣
↪"Other")
df_pd = df_pd[df_pd["EDU_GROUP"].isin(all_groups)].copy()
df_pd["EDU_GROUP"] = pd.Categorical(df_pd["EDU_GROUP"], categories=all_groups,␣
↪ordered=True)

# Coerce numerics (prevents dtype errors during jitter math)
for c in ["MAX_YEARS_EXPERIENCE", "Average_Salary"]:
    df_pd[c] = pd.to_numeric(df_pd[c], errors="coerce")
df_pd = df_pd.dropna(subset=["MAX_YEARS_EXPERIENCE", "Average_Salary"]).copy()

# Jitter and Trim (again, after filtering)
np.random.seed(42)
df_pd["MAX_YEARS_EXPERIENCE_JITTER"] = (
    df_pd["MAX_YEARS_EXPERIENCE"] + np.random.uniform(-0.3, 0.3,␣
↪size=len(df_pd))
)
df_pd["Average_Salary_JITTER"] = (
    df_pd["Average_Salary"] + np.random.uniform(-500, 500, size=len(df_pd))
)

# Hover column (only include if it exists)
hover_cols = [c for c in ["LOT_V6_SPECIALIZED_OCCUPATION_NAME"] if c in df_pd.
↪columns] or None

# Fixed color mapping so each group is always the same color
color_map = {
    "Associate":  "#187145",
    "Bachelor's": "#45A274",
    "Master's":   "#22E529",
    "PhD":        "#86F51E",
}
```

```python
# Plotting of four education groups
fig = px.scatter(
    df_pd,
    x="MAX_YEARS_EXPERIENCE_JITTER",
    y="Average_Salary_JITTER",
    color="EDU_GROUP",
    hover_data=hover_cols,
    title="Experience vs. Average Salary by Education Level",
    opacity=0.7,
    category_orders={"EDU_GROUP": all_groups},  # stable ordering
    color_discrete_map=color_map,               # stable colors
    labels={
        "MAX_YEARS_EXPERIENCE_JITTER": "Years of Experience",
        "Average_Salary_JITTER": "Average Salary (USD)",
        "EDU_GROUP": "Education Level",
    },
)
fig.update_traces(marker=dict(size=7, line=dict(width=1, color="black")))

# If any group has 0 rows, add an invisible trace so it still appears in the␣
 ↪legend
present = set(df_pd["EDU_GROUP"].dropna().unique().tolist())
for g in [grp for grp in all_groups if grp not in present]:
    fig.add_scatter(
        x=[None], y=[None], mode="markers", name=g, showlegend=True,
        marker=dict(color=color_map[g])
    )

# Layout refinements
fig.update_layout(
    font_family="Calibri",
    font_size=14,
    title_font_size=24,
    title_x=0.5,
    xaxis_title="Max Years of Experience",
    yaxis_title="Average Salary (USD)",
    plot_bgcolor="white",
    paper_bgcolor="white",
    legend_title="Education Level",
    hoverlabel=dict(bgcolor="white", font_size=12, font_family="Calibri"),
    xaxis=dict(showline=True, linecolor="black"),
    yaxis=dict(showline=True, linecolor="black"),
)

# Show figure
fig.show()
```

13 Analysis of Salary by Education Level

The scatter plot shows that those individuals who possess higher education levels, especially Master's or PhD degrees, tend to have higher average salaries as their experience increases, with some salaries reaching up to $350K. Contrary to this, individuals with a Bachelor's degree or lower have a modest salary growth with experience, with the majority of salaries clustering below $150K. We also observe that after 6 years of experience, the salary growth for Bachelor's or lower degrees tends to plateau, while those with advanced degrees continue to see significant salary increases.

[ ]:
```
# Local copy and cleanup
_df = df_pd.copy()
for c in ["MAX_YEARS_EXPERIENCE", "Average_Salary"]:
    _df[c] = pd.to_numeric(_df[c], errors="coerce")
_df = _df.dropna(subset=["EDU_GROUP","MAX_YEARS_EXPERIENCE","Average_Salary"])
_df = _df[(_df["MAX_YEARS_EXPERIENCE"] > 0) & (_df["Average_Salary"] > 0)]

# Standardize EDU_GROUP & map to super-groups
_df["EDU_GROUP"] = (
    _df["EDU_GROUP"].astype(str).str.strip().replace({
        "associate":"Associate","associates":"Associate",
        "bachelors":"Bachelor's","bachelor's":"Bachelor's",
        "masters":"Master's","master's":"Master's",
        "phd":"PhD","ph.d.":"PhD","ph.d":"PhD",
        "doctorate":"PhD","professional degree":"PhD",
        "hs":"Associate","high school":"Associate","ged":"Associate",
        "no education listed":"Associate",
    })
)
def to_super(x:str):
    xlow = x.lower()
    if any(k in xlow for k in␣
  ↪["master","mba","phd","doctor","md","jd","llm","dnp","edd","psyd","pharmd","dvm"]):
  ↪
        return "Master's or PhD"
    if any(k in xlow for k in ["bachelor","associate","ged","high school","no␣
  ↪education"]):
        return "Bachelor's or lower"
    return "Bachelor's or lower"
_df["EDU_SUPER_GROUP"] = _df["EDU_GROUP"].map(to_super)

# Subset, jitter, hover
d = _df[_df["EDU_SUPER_GROUP"]=="Bachelor's or lower"].copy()
rng = np.random.default_rng(42)
d["MAX_YEARS_EXPERIENCE_JITTER"] = d["MAX_YEARS_EXPERIENCE"] + rng.uniform(-0.
  ↪3,0.3,len(d))
d["Average_Salary_JITTER"]       = d["Average_Salary"]       + rng.
  ↪uniform(-500,500,len(d))
```

```
hover_cols = [c for c in ["LOT_V6_SPECIALIZED_OCCUPATION_NAME"] if c in d.
 ↪columns] or None

fig = px.scatter(
    d, x="MAX_YEARS_EXPERIENCE_JITTER", y="Average_Salary_JITTER",
    opacity=0.75, color_discrete_sequence=["#45A274"],
    title="Experience vs. Average Salary - Bachelor’s or lower",
    hover_data=hover_cols,
    labels={"MAX_YEARS_EXPERIENCE_JITTER":"Years of Experience (jittered)",
            "Average_Salary_JITTER":"Average Salary (USD, jittered)"}
)
fig.update_traces(marker=dict(size=7,line=dict(width=1,color="black")),␣
 ↪showlegend=False)
fig.
 ↪update_layout(font_family="Calibri",font_size=14,title_font_size=24,title_x=0.
 ↪5,
                  plot_bgcolor="white",paper_bgcolor="white",
                  xaxis=dict(showline=True,linecolor="black"),
                  yaxis=dict(showline=True,linecolor="black"))
fig.show()
```

14 Analysis of Salary for Bachelor’s or Lower Education Level

The scatter plot shows that while salary tends to increase with years of experience for individuals possessing a Bachelor’s degree or lower, the relationship varies and is non-linear. More notably, we see outliers with salaries that are significantly higher than the median for the experience level, this can indicate other contributing factors such as location, industry, or specific skillsets that may influence the salary outside of the education level and experience.

[45]:
```
_df = df_pd.copy()
_df["Average_Salary"] = pd.to_numeric(_df["Average_Salary"], errors="coerce")
_df = _df.dropna(subset=["EDU_GROUP","Average_Salary"])
_df = _df[_df["Average_Salary"] > 0]
_df["EDU_GROUP"] = (
    _df["EDU_GROUP"].astype(str).str.strip().replace({
        "associate":"Associate","associates":"Associate",
        "bachelors":"Bachelor's","bachelor’s":"Bachelor's",
        "masters":"Master's","master’s":"Master's",
        "phd":"PhD","ph.d.":"PhD","ph.d":"PhD",
        "doctorate":"PhD","professional degree":"PhD",
        "hs":"Associate","high school":"Associate","ged":"Associate",
        "no education listed":"Associate",
    })
)
def to_super(x:str):
    xl=x.lower()
```

```
    if any(k in xl for k in
["master","mba","phd","doctor","md","jd","llm","dnp","edd","psyd","pharmd","dvm"]):

        return "Master's or PhD"
    if any(k in xl for k in ["bachelor","associate","ged","high school","no
education"]):
        return "Bachelor's or lower"
    return "Bachelor's or lower"
_df["EDU_SUPER_GROUP"] = _df["EDU_GROUP"].map(to_super)

vals = _df.loc[_df["EDU_SUPER_GROUP"]=="Bachelor's or lower","Average_Salary"].
dropna().astype(float).values
if vals.size==0:
    print("[WARN] No data for Bachelor's or lower.");
else:
    fig = px.histogram(x=vals, nbins=50, histnorm="probability density",
opacity=0.75)
    fig.update_traces(marker_color="#45A274", marker_line_color="black",
marker_line_width=1, showlegend=False)

    x_min, x_max = float(np.min(vals)), float(np.max(vals))
    pad = 0.05 * (x_max - x_min if x_max>x_min else 1.0)
    grid = np.linspace(x_min - pad, x_max + pad, 512)
    counts, edges = np.histogram(vals, bins=50, density=True)
    centers = 0.5*(edges[:-1]+edges[1:])
    bw = 1.06*np.std(vals)*(vals.size**(-1/5)) if vals.size>1 and np.
std(vals)>0 else 0.1*(x_max-x_min if x_max>x_min else 1.0)
    bw = max(bw, 1e-9)
    kbins = max(2, int(0.5*len(centers)*bw/(x_max-x_min+1e-9)))
    kx = np.linspace(-3,3,2*kbins+1); kernel = np.exp(-0.5*(kx**2)); kernel /=
kernel.sum()
    smooth = np.convolve(counts, kernel, mode="same")
    smooth_grid = np.interp(grid, centers, smooth)
    fig.add_trace(go.Scatter(x=grid, y=smooth_grid, mode="lines",
line=dict(color="#45A274", width=3)))

    fig.update_layout(
        title="Salary Distribution - Bachelor's or lower (Histogram + KDE)",
        xaxis_title="Average Salary (USD)", yaxis_title="Density",
        plot_bgcolor="white", paper_bgcolor="white",
        xaxis=dict(showline=True,linecolor="black"),
        yaxis=dict(showline=True,linecolor="black"),
        font_family="Calibri", font_size=14, title_font_size=24, title_x=0.5,
        showlegend=False
    )
    fig.show()
```

15 Analysis of Salary distribution for Bachelor's or Lower Education Level

The histogram reveals that the salary distribution for individuals with a Bachelor's degree or lower is right-skewed, a significant concentration of salaries falls between $70K and $140K. The long tail extension towards the higher salary range shows that while most individuals earn moderate salarries, a few outling individuals earn a substantially higher salary, this indicates variability in this education group.

16 Note:

The data returned no values for Master's or PhD education levels, hence the corresponding plots are not included.

17 12. 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.**

[18]:
```python
from pyspark.sql.functions import when, col, trim

# Categorize and clean remote work types
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")
)

# Filter for relevant columns and valid values
df_filtered = df.filter(
    col("MAX_YEARS_EXPERIENCE").isNotNull() &
    (col("MAX_YEARS_EXPERIENCE") > 0) &
    col("Average_Salary").isNotNull() &
    (col("Average_Salary") > 0) &
    col("REMOTE_GROUP").isin(["Remote", "Hybrid", "Onsite"])
)
```

```python
# Convert to Pandas DataFrame
df_pd = df_filtered.select("MAX_YEARS_EXPERIENCE", "Average_Salary",␣
↪"LOT_V6_SPECIALIZED_OCCUPATION_NAME","REMOTE_GROUP", ).toPandas()
df_pd.head()
```

[18]:

```
   MAX_YEARS_EXPERIENCE  Average_Salary LOT_V6_SPECIALIZED_OCCUPATION_NAME  \
0                   2.0        108668.5   General ERP Analyst / Consultant
1                   3.0        108668.5        Oracle Consultant / Analyst
2                   7.0        108668.5   General ERP Analyst / Consultant
3                   2.0         92962.0                       Data Analyst
4                   5.0        108668.5                       Data Analyst

  REMOTE_GROUP
0       Onsite
1       Remote
2       Onsite
3       Onsite
4       Remote
```

[48]:

```python
# Addition of Jitter and 2 decimal places trim
df_pd["MAX_EXPERIENCE_JITTER"] = df_pd["MAX_YEARS_EXPERIENCE"] + np.random.
↪uniform(-0.20, 0.20, size=len(df_pd))
df_pd["Average_Salary_JITTER"] = df_pd["Average_Salary"] + np.random.
↪uniform(-5000, 5000, size=len(df_pd))
df_pd = df_pd.round(2)
df_pd.head()

# Removal of outlier values higher than 399K
df_pd = df_pd[df_pd["Average_Salary_JITTER"] <= 399000]
```

[58]:

```python
# Scatter plot with trend lines
fig = 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. Average Salary by Remote Work Type<b>",
    opacity=0.7,
    category_orders={"REMOTE_GROUP": ["Remote", "Hybrid", "Onsite"]},  # lock␣
↪order
    color_discrete_sequence=["#2ca02c", "#F7EA76", "#B83198"],  # Remote,␣
↪Hybrid, Onsite
)
```

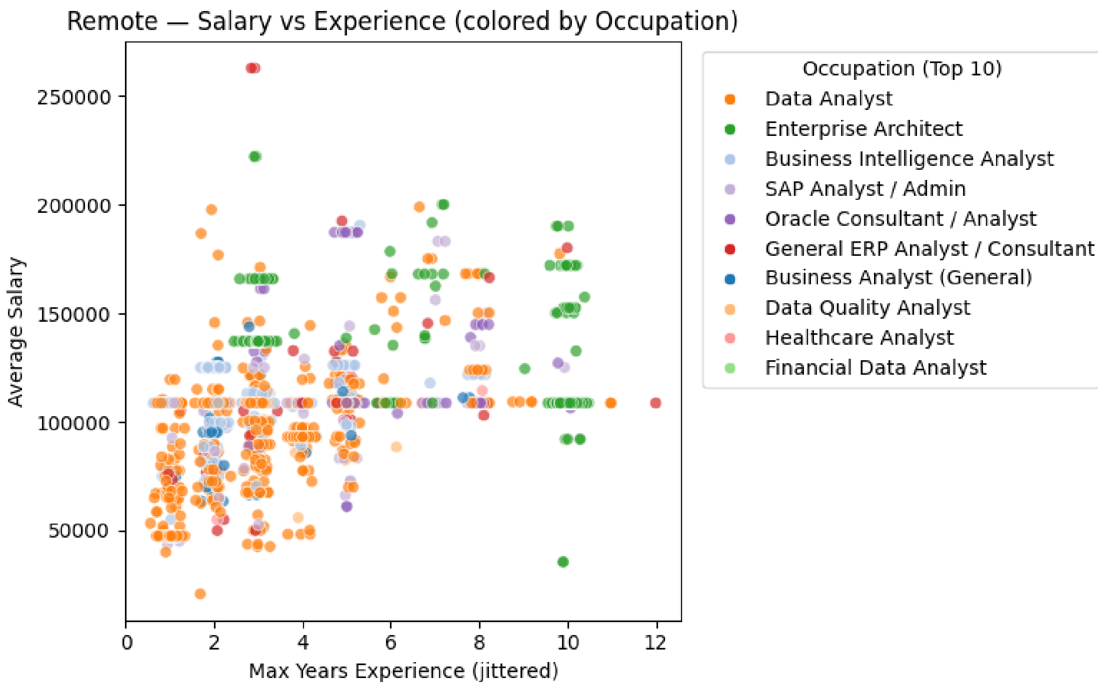
```python
# Layout Improvements

fig.update_traces(marker=dict(size=7, line=dict(width=1, color="black")))
fig.update_layout(
    plot_bgcolor="white",
    paper_bgcolor="white",
    font=dict(family="Calibri", size=14),
    title_font=dict(size=22),
    title_x=0.5,
    xaxis_title="Years of Experience",
    yaxis_title="Average Salary ($1000)",
    legend_title="Remote Work Type",
    hoverlabel=dict(bgcolor="white", font_size=12, font_family="Calibri"),
    margin=dict(l=40, r=40, t=80, b=40),
    xaxis=dict(
        gridcolor="black",
        tickmode="linear",
        tick0=1,
        dtick=1,
        tickangle=0,
    ),
    yaxis=dict(gridcolor="black"),
    legend=dict(
        orientation="h",
        yanchor="bottom",
        y=1.02,
        xanchor="right",
        x=1,
    ),
)

fig.show()
```

18 Salary Analysis by Experience and Remote Work Type

The initial scatter plot shows an overall view of the three different remote work types, Remote, Hybrid, and Onsite. We can see that the remote positions have a wider range of salaries compared to onsite and hybrid roles. This suggests that remote work may offer more flexibility in compensation, possibly due to a broader talent pool and varying cost of living considerations. Onsite roles tend to cluster around lower salary ranges, indicating more standardized pay scales. While hybrid roles fall somewhere in the middle, there is still a wide range of salaries, suggesting that hybrid work arrangements can vary significantly in terms of compensation.

[59]:

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

remote = df_pd[df_pd["REMOTE_GROUP"] == "Remote"].copy()
remote["LOT_V6_SPECIALIZED_OCCUPATION_NAME"] =␣
↪remote["LOT_V6_SPECIALIZED_OCCUPATION_NAME"].fillna("Unknown")

# Jitter on x
rng = np.random.default_rng(42)
jitter = rng.normal(loc=0.0, scale=0.15, size=len(remote))
x = remote["MAX_YEARS_EXPERIENCE"].to_numpy() + jitter
y = remote["Average_Salary"].to_numpy()

# Color by occupation (stable mapping per run)
occ = remote["LOT_V6_SPECIALIZED_OCCUPATION_NAME"].astype("category")
colors = plt.cm.tab20(occ.cat.codes.to_numpy() % 20)

plt.figure(figsize=(8, 5))
plt.scatter(x, y, c=colors, alpha=0.7, edgecolors="white", linewidths=0.5)
plt.xlabel("Max Years Experience (jittered)")
plt.ylabel("Average Salary")
plt.title("Remote - Salary vs Experience (colored by Occupation)")
# Optional compact legend: top N occupations
top_occ = occ.value_counts().index[:10]
handles = [plt.Line2D([0],[0], marker='o', linestyle='',
                      markerfacecolor=plt.cm.tab20(occ.cat.categories.
↪get_indexer([o])[0] % 20),
                      markeredgecolor='white', markeredgewidth=0.5, label=o)␣
↪for o in top_occ]
plt.legend(handles=handles, title="Occupation (Top 10)", bbox_to_anchor=(1.02,␣
↪1), loc="upper left", frameon=True)
plt.tight_layout()
plt.show()
```

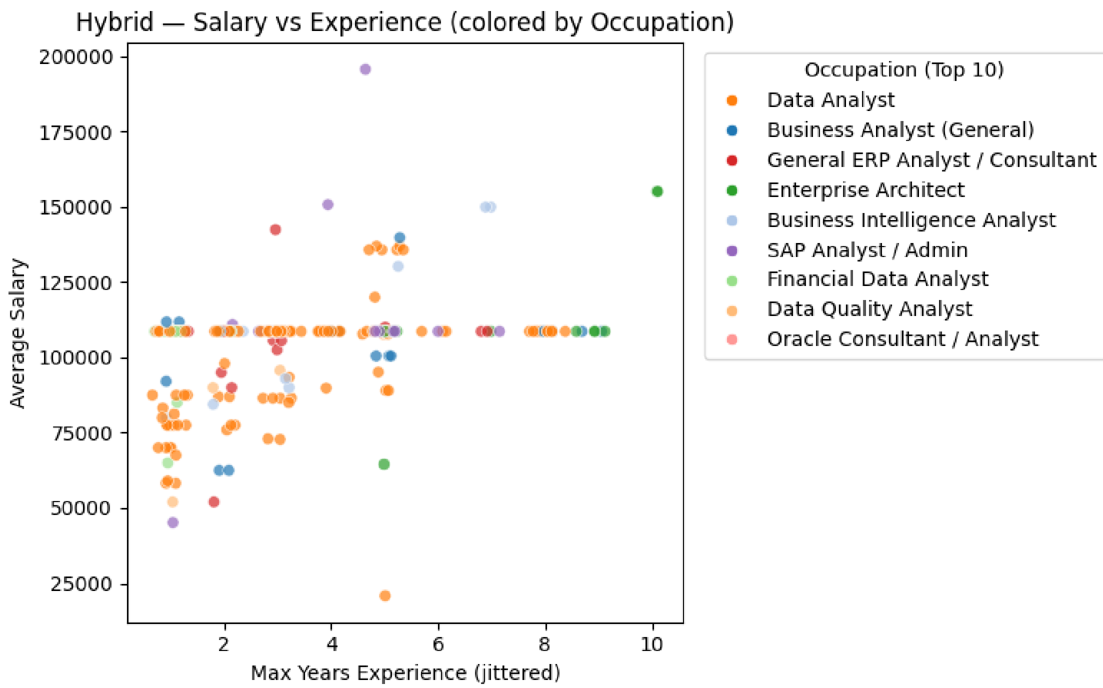


19 Analysis of Salary for Remote Work Type based on Experience

The scatter plot shows that the remote roles for data and analytics have a wider salary range across all levels of experience. We see that roles such a Data Analyst, Data Scientist, and Business Analyst tend to lean towards remote work when compared to other role types.

[51]:
```
import numpy as np
import matplotlib.pyplot as plt

hybrid = df_pd[df_pd["REMOTE_GROUP"] == "Hybrid"].copy()
hybrid["LOT_V6_SPECIALIZED_OCCUPATION_NAME"] =␣
↪hybrid["LOT_V6_SPECIALIZED_OCCUPATION_NAME"].fillna("Unknown")

rng = np.random.default_rng(43)
jitter = rng.normal(loc=0.0, scale=0.15, size=len(hybrid))
x = hybrid["MAX_YEARS_EXPERIENCE"].to_numpy() + jitter
y = hybrid["Average_Salary"].to_numpy()

occ = hybrid["LOT_V6_SPECIALIZED_OCCUPATION_NAME"].astype("category")
colors = plt.cm.tab20(occ.cat.codes.to_numpy() % 20)

plt.figure(figsize=(8, 5))
plt.scatter(x, y, c=colors, alpha=0.7, edgecolors="white", linewidths=0.5)
plt.xlabel("Max Years Experience (jittered)")
```

```
plt.ylabel("Average Salary")
plt.title("Hybrid - Salary vs Experience (colored by Occupation)")
top_occ = occ.value_counts().index[:10]
handles = [plt.Line2D([0],[0], marker='o', linestyle='',
                      markerfacecolor=plt.cm.tab20(occ.cat.categories.
 ↪get_indexer([o])[0] % 20),
                      markeredgecolor='white', markeredgewidth=0.5, label=o)␣
 ↪for o in top_occ]
plt.legend(handles=handles, title="Occupation (Top 10)", bbox_to_anchor=(1.02,␣
 ↪1), loc="upper left", frameon=True)
plt.tight_layout()
plt.show()
```

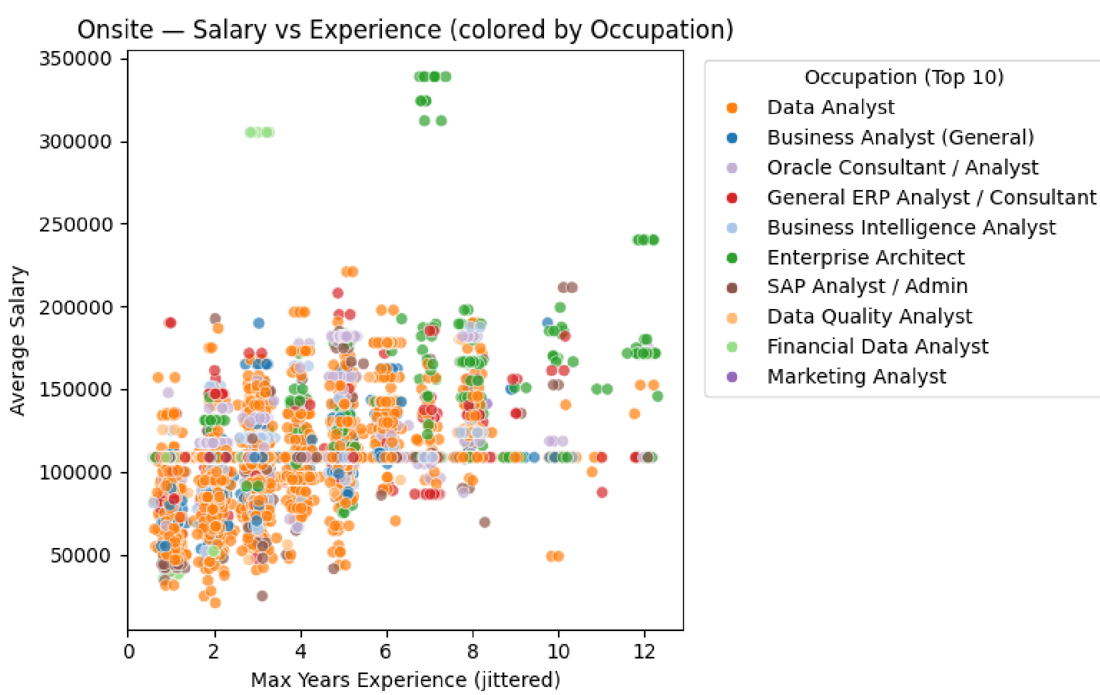


20 Analysis of Salary for Hybrid Work Type based on Experience

The scatter plot indicates that hybrid roles in data and analytics tend to have a moderate salary range while roles such as Enterprise Architect and Data Engineer have consistently higher salaries.

[54]:
```
import numpy as np
import matplotlib.pyplot as plt

onsite = df_pd[df_pd["REMOTE_GROUP"] == "Onsite"].copy()
onsite["LOT_V6_SPECIALIZED_OCCUPATION_NAME"] =␣
 ↪onsite["LOT_V6_SPECIALIZED_OCCUPATION_NAME"].fillna("Unknown")
```

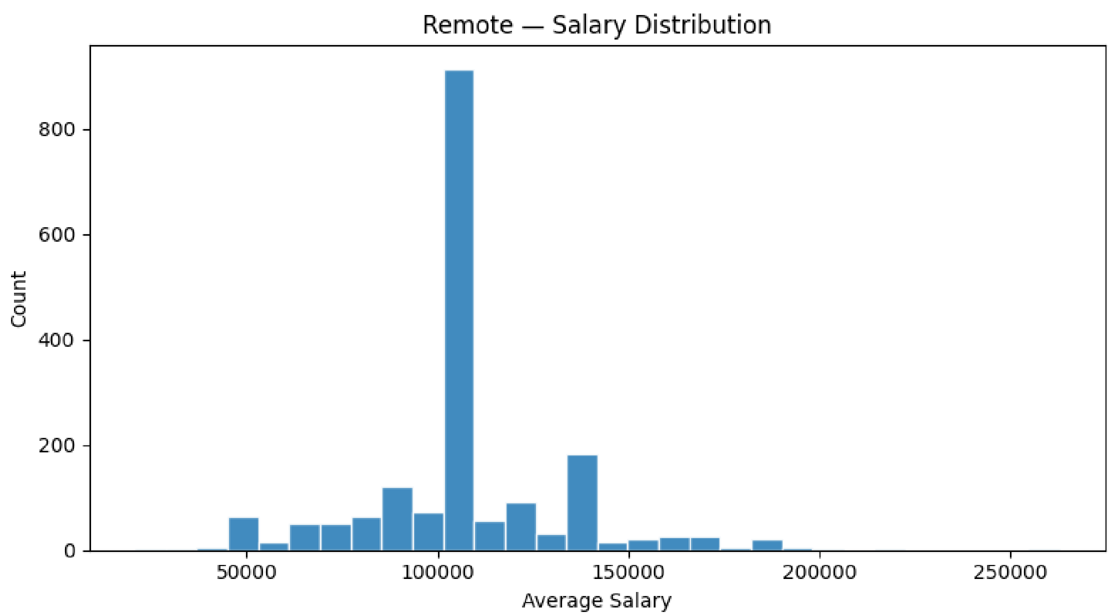
```
rng = np.random.default_rng(44)
jitter = rng.normal(loc=0.0, scale=0.15, size=len(onsite))
x = onsite["MAX_YEARS_EXPERIENCE"].to_numpy() + jitter
y = onsite["Average_Salary"].to_numpy()

occ = onsite["LOT_V6_SPECIALIZED_OCCUPATION_NAME"].astype("category")
colors = plt.cm.tab20(occ.cat.codes.to_numpy() % 20)

plt.figure(figsize=(8, 5))
plt.scatter(x, y, c=colors, alpha=0.7, edgecolors="white", linewidths=0.5)
plt.xlabel("Max Years Experience (jittered)")
plt.ylabel("Average Salary")
plt.title("Onsite - Salary vs Experience (colored by Occupation)")
top_occ = occ.value_counts().index[:10]
handles = [plt.Line2D([0],[0], marker='o', linestyle='',
                      markerfacecolor=plt.cm.tab20(occ.cat.categories.get_indexer([o])[0] % 20),
                      markeredgecolor='white', markeredgewidth=0.5, label=o) for o in top_occ]
plt.legend(handles=handles, title="Occupation (Top 10)", bbox_to_anchor=(1.02, 1), loc="upper left", frameon=True)
plt.tight_layout()
plt.show()
```



21 Analysis of Salary for Onsite Work Type based on Experience

For the onsite work type we observe a wide variety of salaries across different experience levels. The highest salaries are seen for roles such as Enterprise Architect, Oracle Consultants, and Data Engineers, which suggests that these positions command higher pay regardless of the work arrangement. Other roles like Data Analysts and Business Analysts show a broader range of salaries, indicating that job title may have a stronger influence on salary than the onsite work type itself.

[55]:
```
import matplotlib.pyplot as plt

remote = df_pd[df_pd["REMOTE_GROUP"] == "Remote"]
plt.figure(figsize=(8, 4.5))
plt.hist(remote["Average_Salary"], bins=30, alpha=0.85, edgecolor="white")
plt.xlabel("Average Salary")
plt.ylabel("Count")
plt.title("Remote - Salary Distribution")
plt.tight_layout()
plt.show()
```

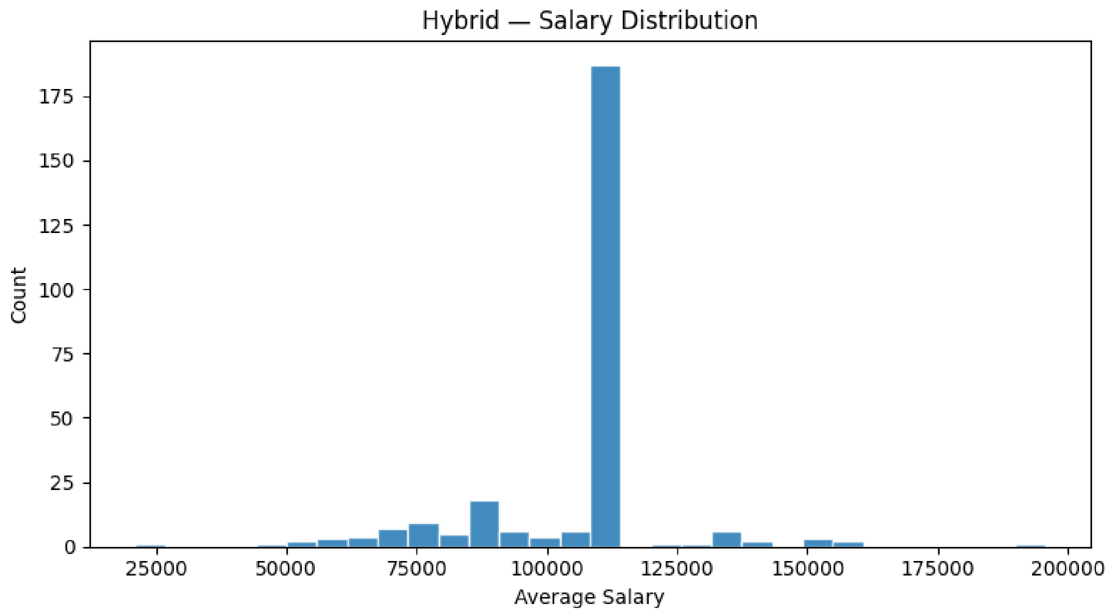


22 Analysis of Salary Distribution by Remote Work Type

The histogram reveals that remote positions have a wider salary distribution, with the highest cluster of roles earning between $100K to $130K.

[56]:
```
import matplotlib.pyplot as plt

hybrid = df_pd[df_pd["REMOTE_GROUP"] == "Hybrid"]
```

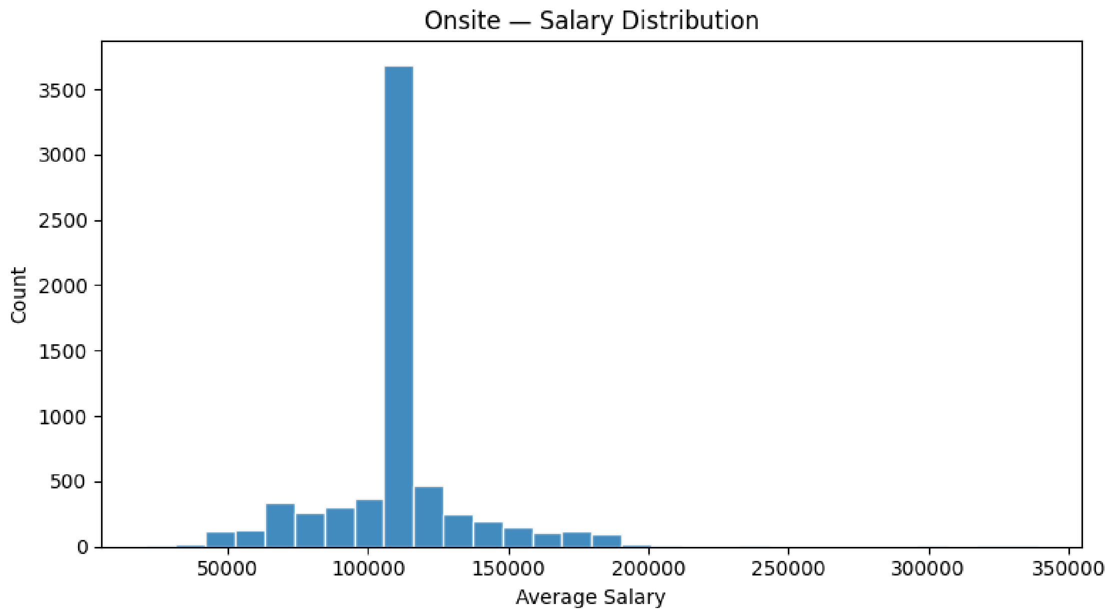
```
plt.figure(figsize=(8, 4.5))
plt.hist(hybrid["Average_Salary"], bins=30, alpha=0.85, edgecolor="white")
plt.xlabel("Average Salary")
plt.ylabel("Count")
plt.title("Hybrid - Salary Distribution")
plt.tight_layout()
plt.show()
```



23 Analysis of Salary Distribution by Hybrid Work Type

The histogram shows that for hybrid roles, salaries most commonly offered are at around $120K. We then see that there are fewer roles below and over this median salary, indicating that there may be more structure around compensation for hybrid roles.

[57]:
```
import matplotlib.pyplot as plt

onsite = df_pd[df_pd["REMOTE_GROUP"] == "Onsite"]
plt.figure(figsize=(8, 4.5))
plt.hist(onsite["Average_Salary"], bins=30, alpha=0.85, edgecolor="white")
plt.xlabel("Average Salary")
plt.ylabel("Count")
plt.title("Onsite - Salary Distribution")
plt.tight_layout()
plt.show()
```



24 Analysis of Salary Distribution by Onsite Work Type

The histogram shows that for onsite roles, salaries most commonly offered are at about $110K. We then observe that as the salary inscreases the number of onsite roles decrease, suggesting that higher paying onsite roles are less prevelant than the latter.