assignment03-Cindy Guzman

# Assignment 03[¶](#Assignment-03)

Cindy Guzman (Boston University)  
September 21, 2025

# 1. Load the Dataset[¶](#X4afd4f0e2a52a813cbb0231b0ea0b67255a67ab)

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.

# Data Exploration and Visualization[¶](#Data-Exploration-and-Visualization)

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

In [4]:

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

In [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)

# 2. Step 1: Create Companies Table (Primary Key: company\_id)[¶](#Xcd25e25cbb28ed5d34fabd7d34d75f65f4c033d)

In [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()

Out[6]:

|  | company\_name | company\_raw | is\_staffing | company\_id |
| --- | --- | --- | --- | --- |
| 0 | Crowe | Crowe | False | 0 |
| 1 | The Devereux Foundation | The Devereux Foundation | False | 1 |
| 2 | Elder Research | Elder Research | False | 2 |
| 3 | NTT DATA | NTT DATA Inc | False | 3 |
| 4 | Frederick National Laboratory For Cancer Research | Frederick National Laboratory for Cancer Research | False | 4 |

# 3. Data Preparation[¶](#X6f6239c6d7afd36ac2a2c3a5bc5c6b4c25257cc)

In [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

Out[7]:

|  | EDUCATION\_LEVELS\_NAME | REMOTE\_TYPE\_NAME | MAX\_YEARS\_EXPERIENCE | Average\_Salary | SALARY | LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | [\n "Bachelor's degree"\n] | [None] | 2.0 | 108668.5 | 115024.0 | General ERP Analyst / Consultant |
| 1 | [\n "No Education Listed"\n] | Remote | 3.0 | 108668.5 | 115024.0 | Oracle Consultant / Analyst |
| 2 | [\n "Bachelor's degree"\n] | [None] | NaN | 108668.5 | 115024.0 | Data Analyst |
| 3 | [\n "No Education Listed"\n] | [None] | NaN | 108668.5 | 115024.0 | Data Analyst |
| 4 | [\n "No Education Listed"\n] | [None] | NaN | 92500.0 | 92500.0 | Oracle Consultant / Analyst |

In [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()

Out[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 |

# 5. Clean employment type names for better readability[¶](#Xcfa3a3c65aee023a0e9e45268aaecc269f7472a)

In [9]:

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

Out[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 |

In [10]:

# 6. Compute median salary for sorting  
  
median\_salaries = pdf.groupby("EMPLOYMENT\_TYPE\_NAME")["SALARY"].median()  
median\_salaries.head()

Out[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

# 9. Salary Distribution by Industry and Employment Type[¶](#X61557126211bd732d728c722f83c09b347ec94e)

* 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.

In [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")

# Salary Distribution by Employment Type Insight[¶](#X07e48bd25e30d1ad1c4fe73861c370d4798594d)

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.

In [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()

# Salary Distribution by Industry Type and NAICS2\_NAME[¶](#X7910b57ed6cf8c3d83c11039df024b862d11898)

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. Salary Analysis by ONET Occupation Type (Bubble Chart)[¶](#Xae78f2aad18183fed88cc4ccbb1305f611483df)

* 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.

In [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'.

# Salary Analysis by LOT Specialized Occupation Type[¶](#X3aa549ad105d4aed2bba073d8d4deae67c2d8cf)

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 .

# 11. Salary by Education Level[¶](#Xdcaf04b551cc71fcc70e4cd5eb6773579c00e90)

* 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.

In [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)

In [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")  
 .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())

In [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()

Out[26]:

|  | ID | LAST\_UPDATED\_DATE | LAST\_UPDATED\_TIMESTAMP | DUPLICATES | POSTED | EXPIRED | DURATION | SOURCE\_TYPES | SOURCES | URL | ... | NAICS\_2022\_5\_NAME | NAICS\_2022\_6 | NAICS\_2022\_6\_NAME | Average\_Salary | REMOTE\_GROUP | EDU\_CODE | EDU\_LEVEL | EDU\_GROUP | MAX\_YEARS\_EXPERIENCE\_JITTER | Average\_Salary\_JITTER |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1f57d95acf4dc67ed2819eb12f049f6a5c11782c | 9/6/2024 | 2024-09-06 20:32:57.352 | 0 | 6/2/2024 | 6/8/2024 | 6.0 | [\n "Company"\n] | [\n "brassring.com"\n] | [\n "https://sjobs.brassring.com/TGnewUI/Sear... | ... | Automotive Parts and Accessories Retailers | 441330 | Automotive Parts and Accessories Retailers | 108668.5 | Onsite | 2 | high school | Associate | 1.78 | 108384.97 |
| 1 | 229620073766234e814e8add21db7dfaef69b3bd | 10/9/2024 | 2024-10-09 18:07:44.758 | 0 | 6/2/2024 | 8/1/2024 | NaN | [\n "Company"\n] | [\n "3ds.com"\n] | [\n "https://www.3ds.com/careers/jobs/sr-mark... | ... | Computer Systems Design and Related Services | 541511 | Custom Computer Programming Services | 92962.0 | Onsite | 2 | high school | Associate | 2.16 | 91977.68 |
| 2 | 229620073766234e814e8add21db7dfaef69b3bd | 10/9/2024 | 2024-10-09 18:07:44.758 | 0 | 6/2/2024 | 8/1/2024 | NaN | [\n "Company"\n] | [\n "3ds.com"\n] | [\n "https://www.3ds.com/careers/jobs/sr-mark... | ... | Computer Systems Design and Related Services | 541511 | Custom Computer Programming Services | 92962.0 | Onsite | 3 | associate | Associate | 1.84 | 92953.22 |
| 3 | 138ce2c9453b47a9b33403c364d4fd80996caa4f | 8/10/2024 | 2024-08-10 19:36:49.244 | 5 | 6/2/2024 | 8/9/2024 | NaN | [\n "Job Board",\n "Education",\n "Recruite... | [\n "silkroad.com",\n "hercjobs.org",\n "di... | [\n "https://main.hercjobs.org/jobs/20166141/... | ... | Colleges, Universities, and Professional Schools | 611310 | Colleges, Universities, and Professional Schools | 108668.5 | Remote | 1 | ged | Associate | 4.95 | 108691.38 |
| 4 | 138ce2c9453b47a9b33403c364d4fd80996caa4f | 8/10/2024 | 2024-08-10 19:36:49.244 | 5 | 6/2/2024 | 8/9/2024 | NaN | [\n "Job Board",\n "Education",\n "Recruite... | [\n "silkroad.com",\n "hercjobs.org",\n "di... | [\n "https://main.hercjobs.org/jobs/20166141/... | ... | Colleges, Universities, and Professional Schools | 611310 | Colleges, Universities, and Professional Schools | 108668.5 | Remote | 2 | high school | Associate | 4.85 | 110094.66 |

5 rows × 138 columns

In [ ]:

# 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",  
}  
  
# 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()

# Analysis of Salary by Education Level[¶](#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.

In [ ]:

# 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()

# Analysis of Salary for Bachelor's or Lower Education Level[¶](#X5240c18783de5712ed9cecfdfb9d7eb167e9328)

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.

In [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()

# Analysis of Salary distribution for Bachelor's or Lower Education Level[¶](#X3114797d2b2b60dfdc257997983a81728804720)

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.

# Note:[¶](#Note:)

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

# 12. Salary by Remote Work Type[¶](#X6ff38626a13cd8ea3d16fcf4e51bfe07efa08d1)

* 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.**

In [18]:

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"])  
)  
  
# 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()

Out[18]:

|  | MAX\_YEARS\_EXPERIENCE | Average\_Salary | LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME | REMOTE\_GROUP |
| --- | --- | --- | --- | --- |
| 0 | 2.0 | 108668.5 | General ERP Analyst / Consultant | Onsite |
| 1 | 3.0 | 108668.5 | Oracle Consultant / Analyst | Remote |
| 2 | 7.0 | 108668.5 | General ERP Analyst / Consultant | Onsite |
| 3 | 2.0 | 92962.0 | Data Analyst | Onsite |
| 4 | 5.0 | 108668.5 | Data Analyst | Remote |

In [48]:

# 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]

In [58]:

# 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  
)  
  
  
# 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()

# Salary Analysis by Experience and Remote Work Type[¶](#Xc902710092223c99396bb1f9d4f100caa69a3fe)

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.

In [59]:

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()

![No description has been provided for this image](data:image/png;base64;base64,)

# Analysis of Salary for Remote Work Type based on Experience[¶](#X48d7a1d820c6b527f39248c3efe88239c1521f8)

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.

In [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()

![No description has been provided for this image](data:image/png;base64;base64,)

# Analysis of Salary for Hybrid Work Type based on Experience[¶](#X5043db3b8037e136c26f3c9d0e9ee10804608af)

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 consistiently higher salaries.

In [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()

![No description has been provided for this image](data:image/png;base64;base64,)

# Analysis of Salary for Onsite Work Type based on Experience[¶](#X98ab7492b161493c900bea9ad9d399e3bc66558)

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.

In [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()

![No description has been provided for this image](data:image/png;base64;base64,)

# Analysis of Salary Distribution by Remote Work Type[¶](#X99ba925a5eb59d2db0fd3b0edf79991f0369ae1)

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

In [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()

![No description has been provided for this image](data:image/png;base64;base64,)

# Analysis of Salary Distribution by Hybrid Work Type[¶](#X640cc90b8ee144f16c90de1af884f7304304264)

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.

In [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()

![No description has been provided for this image](data:image/png;base64;base64,)

# Analysis of Salary Distribution by Onsite Work Type[¶](#X9d9c27b4a95e2b661d5a3df64882d987da9dbb7)

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.