assignment03-Cindy Guzman

September 24, 2025

1 Assignment 03

Cindy Guzman (Boston University)
November 21, 2024

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.

[33]:
```
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)
```

[34]:
```
# 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)

[35]:
```
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()
```

[35]:
```
                                      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

[17]:
```python
# 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 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: Imputing missing salaries, but not experience
df = df.fillna({
  "SALARY_FROM": median_from,
  "SALARY_TO": median_to,
  "SALARY": median_salary
  })

# Step 5: Computing average salary
```

```
df = df.withColumn("Average_Salary", (col("SALARY_FROM") + col("SALARY_TO")) /␣
 ↪2)

# Step 6: 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 7: 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
```

[17]:

```
            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
```

[18]:

```
# 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()
```

[18]:
```
      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

[19]:
```python
import re
pdf["EMPLOYMENT_TYPE_NAME"] = pdf["EMPLOYMENT_TYPE_NAME"].apply(lambda x: re.
↪sub(r"[^\x00-\x7F]+", "", x))
pdf.head()
```

[19]:
```
      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
```

[20]:
```python
# 6. Compute median salary for sorting

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

[20]:
```
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.

[21]:
```
# 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="EMPLOYMENT_TYPE_NAME",
    y="SALARY",
    orientation="h",
    title="Salary Distribution by Employment Type",
    color_discrete_sequence=["#084B21"], # Single neutral color
    boxmode='group',
    points="all",  # Show all outliers
)
fig.update_layout(title_x=0.5)

# 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="Employment Type", 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="Salary ($1000)", 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=0.7,
    plot_bgcolor='white',
    paper_bgcolor='white',
    showlegend=False,
    height=500,
    width=850,
)

 # Show figure
fig.show()
fig.write_html("./output/boxplot_salary_by_employment_type.html")
fig.write_image("./output/boxplot_salary_by_employment_type.png", ␣
↪height=500,width=850, scale=1)
```

[22]:
```
#| 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=["#084B21"])
fig.update_layout(font_family="Calibri", title_font_size=30,␣
↪font_color="black", title_x=0.5, height=900, width=1110)
fig.show()
```

8 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.

[23]:
```
#| eval: false
#| echo: false
# Spark SQL - Median salary and job count per ONET_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
""")

# 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 Occupation Type (Bubble Chart)",
    labels= {"LOT_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="Plasma"
)

# Layout improvements
fig.update_layout(
```

```
    font_family="Calibri",
    font_size=14,
    title_font_size=24,
    xaxis_title="LOT 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()
fig.write_image("./output/bubble_chart_salary_by_onet.png", ␣
↪height=600,width=1000, scale=1)
```

9 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.

[ ]:

```

```

10 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.**

[ ]:
```

```