LONDON METROPOLITAN UNIVERSITY

Islington college
(इस्लिङ्टन कलेज)

CC5067NI-Smart Data Discovery

60% Individual Coursework

2023-24 Autumn

Student Name: Aryan Jaiswal

London Met ID: 22085502

College ID: NP01CP4S23016

Assignment Due Date: Monday, May 13, 2024

Assignment Submission Date: Sunday, May 12, 2024

Word Count: 2116

I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Table of Contents

Table of Tables

Table Of Figures

1. Data Understanding

This dataset contains information about individuals' employment details, including their work year, experience level (categorized as Senior, Mid, or Entry), employment type (full-time or contract), job title, salary in local currency, salary currency, salary converted to USD, employee residence country, remote work ratio, company location, and company size. These attributes collectively provide insights into employment trends, salary structures, and workforce characteristics, enabling analysis of factors influencing job roles, compensation levels, and geographical variations in employment conditions. The dataset facilitates comparisons across different regions and company sizes, offering a comprehensive view of employment dynamics and market conditions.

S.N	Column Name	Description	Data Type
1.	Work_Year	This column indicates the specific year when employment data was recorded and It helps track changes over time in job titles, salaries, and employment patterns.	Integer
2.	Experience_Level	The experience level column categorizes individuals based on their experience, such as "SE" (Senior), "MI" (Mid), or "EN" (Entry). Higher experience levels often correspond to more senior job titles and higher salaries.	String(object)
3.	Employment_Type	The employment_type column specifies whether individuals are employed full-time or under contract. This can influence job stability and benefits, which in turn affect salary levels.	String(object)

4.	Job_Title	The Job_Title column describes the role or position held by each individual. Job titles directly reflect the nature of work, required skills, and salary range within the dataset.	String(object)
5.	Salary	The Salary column denotes the actual salary earned by each individual in their respective currency.	Integer
6.	Salary_Currency	Salary_Currency specifies the currency used to report salaries. It's important for currency conversion (if needed) to compare salaries across different regions or countries.	String(object)
7.	Salary_in_usd	Salary_in_usd column is the salary amount which is converted in the US Dollar for a proper analysis and comparison.	Integer
8.	Employee_Residence	The Employee_Residence column indicates the country where each employees resides. Its also relevant on comparing the salary of different employees of different location on the basis of cost of living and market conditions.	String(object)
9.	Remote_Ratio	Remote_Ratio represents the remote work offered for each employee.	Integer
10.	Company_Location	Company_Location specifies the geographical location of the employing company. Company location impacts salary levels due to regional economic conditions and market demand of	String(object)

		different location.	
11.	Company_Size	Company_Size categorizes the size of each company, reflecting its organizational scale and resources. Company size can affect job roles, responsibilities, and salary scales.	String(object)

Table 1 Dataset Table

2. Data Preparation.

Write a python program to load data into pandas DataFrame.

```
[127]: #Importing python libraries
       import pandas as pd
       import numpy as np

[129]: # Reading the csv file in a dataframe
       data = pd.read_csv("Coursework.csv")
       data.head()
```

[129]:

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	SE	FT	Principal Data Scientist	80000	EUR	85847	ES	100	ES	L
1	2023	MI	CT	ML Engineer	30000	USD	30000	US	100	US	S
2	2023	MI	CT	ML Engineer	25500	USD	25500	US	100	US	S
3	2023	SE	FT	Data Scientist	175000	USD	175000	CA	100	CA	M
4	2023	SE	FT	Data Scientist	120000	USD	120000	CA	100	CA	M

Figure 1 Loading Data into Pandas Data Frame

The code above begins by importing the required libraries, pandas and numpy. Then, it loads the CSV file named "Coursework.csv" into a DataFrame object using the **pd.read_csv()** function. This function reads the CSV file and converts its contents into a tabular format, storing it in the DataFrame named **data**. Finally, the **data.head()** function display the first few rows of the DataFrame.

Write a python program to remove unnecessary columns i.e., salary and salary currency.

```
[93]: #removing unnecessary column
rem = ['salary', 'salary_currency']
data = data.drop(columns=rem)
data
```

[93]:

	work_year	experience_level	employment_type	job_title	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	SE	FT	Principal Data Scientist	85847	ES	100	ES	L
1	2023	MI	CT	ML Engineer	30000	US	100	US	S
2	2023	MI	CT	ML Engineer	25500	US	100	US	S
3	2023	SE	FT	Data Scientist	175000	CA	100	CA	M
4	2023	SE	FT	Data Scientist	120000	CA	100	CA	M
...	...	...	...	...	...	...	...	...	...
3750	2020	SE	FT	Data Scientist	412000	US	100	US	L
3751	2021	MI	FT	Principal Data Scientist	151000	US	100	US	L
3752	2020	EN	FT	Data Scientist	105000	US	100	US	S
3753	2020	EN	CT	Business Data Analyst	100000	US	100	US	L
3754	2021	SE	FT	Data Science Manager	94665	IN	50	IN	L

3755 rows × 9 columns

Figure 2 Removing the columns

This Python program removes unnecessary columns, 'salary' and 'salary currency', from the Data Frame named 'data'. First, a list named 'rem' is declared, containing the names of the columns to be removed, which are 'salary' and 'salary currency'. Then, the **data. Drop(columns=rem)** method call removes these specified columns from the 'data' DataFrame. This method takes the list of column names to be removed as input and returns a new DataFrame without those columns.

Finally, the modified DataFrame, which no longer contains the specified columns, is assigned back to the variable 'data', effectively updating it with the removed columns.

Write a python program to remove the NaN missing values from updated dataframe.

```
[135]: #Removing NAN missing values
data = data.dropna()
data
```

[135]:

	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	SE	FT	Principal Data Scientist	80000	EUR	85847	ES	100	ES	L
1	2023	MI	CT	ML Engineer	30000	USD	30000	US	100	US	S
2	2023	MI	CT	ML Engineer	25500	USD	25500	US	100	US	S
3	2023	SE	FT	Data Scientist	175000	USD	175000	CA	100	CA	M
4	2023	SE	FT	Data Scientist	120000	USD	120000	CA	100	CA	M
...	...	...	...	...	...	...	...	...	...	...	...
3750	2020	SE	FT	Data Scientist	412000	USD	412000	US	100	US	L
3751	2021	MI	FT	Principal Data Scientist	151000	USD	151000	US	100	US	L
3752	2020	EN	FT	Data Scientist	105000	USD	105000	US	100	US	S
3753	2020	EN	CT	Business Data Analyst	100000	USD	100000	US	100	US	L
3754	2021	SE	FT	Data Science Manager	7000000	INR	94665	IN	50	IN	L

3755 rows × 11 columns

Figure 3 Removing NaN missing values

The code **dropna()** function is used here to eliminate NaN missing values from the DataFrame. When applied to the DataFrame named 'data', the function removes rows with any missing data. As a result, the DataFrame is updated to exclude these rows. However, in this case the original DataFrame contains no missing values so the output remains same without any changes.

Write a python program to check duplicates value in the dataframe.

```
[3]: #Checking duplicate values
data.duplicated().sum()

[3]: 1171
```

Figure 4 Checking Duplicate values in dataframe

The **duplicated ()** function is used in this code to identify duplicate values within the DataFrame. By calling. sum **()** on the result of **data. Duplicated ()**, the program calculates the total number of duplicate values present in the DataFrame. This count represents the number of duplicate rows within the DataFrame.

Write a python program to see the unique values from all the columns in the dataframe.

```
[101]: # Printing unique values from all the column
for column in data:
    # Checking if the data type of the column is 'object'
    if data[column].dtype == 'object':
        unique_values = data[column].unique()
         # Printing the unique values along with the column name
        print(f"Unique values in column '{column}':{unique_values}\n")
    print()
```

```
Unique values in column 'experience_level':['SE' 'MI' 'EN' 'EX']


Unique values in column 'employment_type':['FT' 'CT' 'FL' 'PT']


Unique values in column 'job_title':['Principal Data Scientist' 'ML Engineer' 'Data Scientist'
 'Applied Scientist' 'Data Analyst' 'Data Modeler' 'Research Engineer'
 'Analytics Engineer' 'Business Intelligence Engineer'
 'Machine Learning Engineer' 'Data Strategist' 'Data Engineer'
 'Computer Vision Engineer' 'Data Quality Analyst'
 'Compliance Data Analyst' 'Data Architect'
 'Applied Machine Learning Engineer' 'AI Developer' 'Research Scientist'
 'Data Analytics Manager' 'Business Data Analyst' 'Applied Data Scientist'
 'Staff Data Analyst' 'ETL Engineer' 'Data DevOps Engineer' 'Head of Data'
 'Data Science Manager' 'Data Manager' 'Machine Learning Researcher'
 'Big Data Engineer' 'Data Specialist' 'Lead Data Analyst'
 'BI Data Engineer' 'Director of Data Science'
 'Machine Learning Scientist' 'MLOps Engineer' 'AI Scientist'
 'Autonomous Vehicle Technician' 'Applied Machine Learning Scientist'
 'Lead Data Scientist' 'Cloud Database Engineer' 'Financial Data Analyst'
 'Data Infrastructure Engineer' 'Software Data Engineer' 'AI Programmer'
 'Data Operations Engineer' 'BI Developer' 'Data Science Lead'
 'Deep Learning Researcher' 'BI Analyst' 'Data Science Consultant'
 'Data Analytics Specialist' 'Machine Learning Infrastructure Engineer'
 'Deep Learning Engineer' 'Machine Learning Software Engineer'
 'Big Data Architect' 'Product Data Analyst'
 'Computer Vision Software Engineer' 'Azure Data Engineer'
 'Marketing Data Engineer' 'Data Analytics Lead' 'Data Lead'
 'Data Science Engineer' 'Machine Learning Research Engineer'
 'NLP Engineer' 'Manager Data Management' 'Machine Learning Developer'
 '3D Computer Vision Researcher' 'Principal Machine Learning Engineer'
 'Data Analytics Engineer' 'Data Analytics Consultant'
 'Data Management Specialist' 'Data Science Tech Lead'
 'Data Scientist Lead' 'Cloud Data Engineer' 'Data Operations Analyst'
 'Marketing Data Analyst' 'Power BI Developer' 'Product Data Scientist'
 'Principal Data Architect' 'Machine Learning Manager'
 'Lead Machine Learning Engineer' 'ETL Developer' 'Cloud Data Architect'
 'Lead Data Engineer' 'Head of Machine Learning' 'Principal Data Analyst'
 'Principal Data Engineer' 'Staff Data Scientist' 'Finance Data Analyst']


Unique values in column 'employee_residence':['ES' 'US' 'CA' 'DE' 'GB' 'NG' 'IN' 'HK' 'PT' 'NL' 'CH' 'CF' 'FR' 'AU'
 'FI' 'UA' 'IE' 'IL' 'GH' 'AT' 'CO' 'SG' 'SE' 'SI' 'MX' 'UZ' 'BR' 'TH'
 'HR' 'PL' 'KW' 'VN' 'CY' 'AR' 'AM' 'BA' 'KE' 'GR' 'MK' 'LV' 'RO' 'PK'
 'IT' 'MA' 'LT' 'BE' 'AS' 'IR' 'HU' 'SK' 'CN' 'CZ' 'CR' 'TR' 'CL' 'PR'
 'DK' 'BO' 'PH' 'DO' 'EG' 'ID' 'AE' 'MY' 'JP' 'EE' 'HN' 'TN' 'RU' 'DZ'
 'IQ' 'BG' 'JE' 'RS' 'NZ' 'MD' 'LU' 'MT']


Unique values in column 'company_location':['ES' 'US' 'CA' 'DE' 'GB' 'NG' 'IN' 'HK' 'NL' 'CH' 'CF' 'FR' 'FI' 'UA'
 'IE' 'IL' 'GH' 'CO' 'SG' 'AU' 'SE' 'SI' 'MX' 'BR' 'PT' 'RU' 'TH' 'HR'
 'VN' 'EE' 'AM' 'BA' 'KE' 'GR' 'MK' 'LV' 'RO' 'PK' 'IT' 'MA' 'PL' 'AL'
 'AR' 'LT' 'AS' 'CR' 'IR' 'BS' 'HU' 'AT' 'SK' 'CZ' 'TR' 'PR' 'DK' 'BO'
 'PH' 'BE' 'ID' 'EG' 'AE' 'LU' 'MY' 'HN' 'JP' 'DZ' 'IQ' 'CN' 'NZ' 'CL'
 'MD' 'MT']


Unique values in column 'company_size':['L' 'S' 'M']
```

Figure 5 Viewing unique values from all the columns

The code iterates through each column in the DataFrame. For columns containing string or categorical data, it extracts the unique values using the **unique ()** function and prints them along with the column name.

The output displayed the unique values found in each column. The code prints the unique values for each column separately. This makes it easy to check the different categories present in the DataFrame.

Rename the experience level columns as below.

SE – Senior Level/Expert

MI – Medium Level/Intermediate

EN – Entry Level

EX – Executive Level

```
[21]: # Renaming experience level columns
data["experience_level"].replace({"SE": "Senior Level/Expert",
                                  "MI": "Medium Level/Intermediate",
                                  "EN": "Entry Level",
                                  "EX": "Executive Level"}, inplace=True)
data
```

[21]:

	work_year	experience_level	employment_type	job_title	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
0	2023	Senior Level/Expert	FT	Principal Data Scientist	85847	ES	100	ES	L
1	2023	Medium Level/Intermediate	CT	ML Engineer	30000	US	100	US	S
2	2023	Medium Level/Intermediate	CT	ML Engineer	25500	US	100	US	S
3	2023	Senior Level/Expert	FT	Data Scientist	175000	CA	100	CA	M
4	2023	Senior Level/Expert	FT	Data Scientist	120000	CA	100	CA	M
...	...	...	...	...	...	...	...	...	...
3750	2020	Senior Level/Expert	FT	Data Scientist	412000	US	100	US	L
3751	2021	Medium Level/Intermediate	FT	Principal Data Scientist	151000	US	100	US	L
3752	2020	Entry Level	FT	Data Scientist	105000	US	100	US	S
3753	2020	Entry Level	CT	Business Data Analyst	100000	US	100	US	L
3754	2021	Senior Level/Expert	FT	Data Science Manager	94665	IN	50	IN	L

3755 rows × 9 columns

Figure 6 Renaming the experience level columns

The code replaces the short forms in the "experience level" column with their full versions. It uses the **replace ()** function to do this. The code replaced SE as Senior Level/Expert, MI as Medium Level/Intermediate, EN as Entry Level, EX as Executive Level. After running the code, the DataFrame 'data' displayed the updated "experience level" column with the full name.

3. Data Analysis

Write a Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of any chosen variable.

```
[54]: # Choosing the variable for summary statistics
selected_column = 'salary_in_usd'

# Retrieve the column of the chosen variable
selected_data = data[selected_column]

# Calculating summary statistics
n = len(selected_data)
total = 0
```

Figure 7 Showing Summary Statistics

The code above begins by selecting the variable 'salary_in_usd' for calculating summary statistics. It then retrieves the column of the chosen variable from the DataFrame 'data' and stores it in the variable 'selected_data'. Following this, the code initializes variables 'n' and 'total' to capture the length of 'selected_data' and the total sum of the values, respectively.

```
[10]: # Calculating sum
for value in selected_data:
    total += value
print(f"Sum: {total}")

Sum: 516576814
```

Figure 8 Calculating sum values

The code iterates through each value in the 'selected_data' column, accumulating their sum into the variable 'total'. After iterating through all the values, it prints out the total sum, which is 516,576,814. This value represents the combined sum of all the values in the 'selected_data' column.

```
[11]: # Calculating mean
      mean = total / n
      print(f"Mean: {mean}")

      Mean: 137570.38988015978
```

Figure 9 Calculating mean

This code calculates the mean (average) of the values in the 'selected_data' column. It divides the total sum of values by the total number of data points to calculate the mean. The resulting mean value is approximately 137,570.39, as indicated by the output.

```
[11]: # Calculating sum again for the variance calculation
      total_sum = 0
      for value in selected_data:
          total_sum += (value - mean) ** 2
      variance = total_sum / n

      # Calculating standard deviation
      std_deviation = variance ** 0.5
      print(f"Standard Deviation: {std_deviation}")

      Standard Deviation: 63047.22849740541
```

Figure 10 Calculating Variance and Standard Deviation

This code calculates the variance and standard deviation of the values in the 'selected_data' column.

It first initializes a variable 'total_sum to zero. Then, it iterates through each value in the column, calculating the squared difference between each value and the mean. These squared differences are summed up in the total_sum' variable.

Next, the code divides total_sum' by the total number of data points ('n') to calculate the variance.

Finally, it computes the standard deviation by taking the square root of the variance. The resulting standard deviation value is approximately 63,047.23, as displayed in the output.

```
[12]: #Calculating Skewness
skewness_sum = 0

for value in selected_data:
    skewness_sum += (value - mean) ** 3
skewness = (skewness_sum / n) / (std_deviation ** 3)
print(f"Skewness: {skewness}")
```

```
Skewness: 0.5361868674235561
```

Figure 11 Calculating Skewness

This part of the code calculates the skewness of the values in the 'selected_data' column.

First, a variable 'skewness_sum' is set to zero. Then, the code goes through each value in the column, finding the difference between each value and the mean, cubing it, and adding up these cubed differences in 'skewness_sum'.

Next, skewness is calculated by dividing 'skewness_sum' by the total number of data points ('n') and by the cube of the standard deviation ('std_deviation').

The resulting skewness value is approximately 0.536, indicating the degree of asymmetry in the distribution of values in the column.

```
[13]: #Calculating Kurtosis
kurtosis_sum = 0
for value in selected_data:
    kurtosis_sum += (value - mean) ** 4
kurtosis = (kurtosis_sum / n) / (variance ** 2) - 3
print(f"Kurtosis: {kurtosis}")
```

```
Kurtosis: 0.8312989014514449
```

Figure 12 Calculating Kurtosis

In this code, the kurtosis of the values in the 'selected_data' column is calculated. First, a variable 'kurtosis_sum' is set to zero. Then, the code iterates through each value in the column, finding the difference between each value and the mean, raising it to the fourth power, and accumulating these fourth power differences in 'kurtosis_sum'.

Next, kurtosis is calculated by dividing 'kurtosis_sum' by the total number of data points ('n') and by the square of the variance ('variance'), then subtracting 3 from the result.

The resulting kurtosis value is approximately 0.831, indicating the degree of peakedness or flatness in the distribution of values in the column.

Write a Python program to calculate and show correlation of all variables.

```
[15]: #Correlation of all the variables
numerical_cols = data.select_dtypes(include=['int','float']).columns
correlation = data[numerical_cols]
correlation.corr()
```

[15]:

	work_year	salary_in_usd	remote_ratio
work_year	1.00000	0.228290	-0.236430
salary_in_usd	0.22829	1.000000	-0.064171
remote_ratio	-0.23643	-0.064171	1.000000

Figure 13 Displaying Correlation of all variables

This code analyses the correlation between numerical variables in the dataset. This code ensures that only columns with numerical data are included in the correlation analysis.

Next, the code calculates the correlation matrix using the 'corr()' method on the selected numerical columns. The output of the code is the correlation matrix, which shows the correlation coefficients between all pairs of numerical variables in the dataset.

4. Data Exploration

Write a python program to find out top 15 jobs. Make a bar graph of salesaswell.

```
import matplotlib.pyplot as plt

# Counting the job title and storing the result in 'data'
data = data['job_title'].value_counts()
# Calculating the total sum of job counts
data.sum()
# Extract top 15 jobs
top_15_jobs = data.head(15)

# Plotting bar graph
plt.figure(figsize=(13, 8))
top_15_jobs.plot(kind='bar',color='orange', legend=False, figsize=(13, 8)) #Plotting the bar graph
#Setting the label for x and y axis
plt.title('Top 15 Jobs')
plt.xlabel('Job Title')
plt.ylabel('Sales')
plt.xticks(rotation=70) # Rotating the x-axis labels for better readability
plt.tight_layout()
plt.show() #Displaying the plot
```

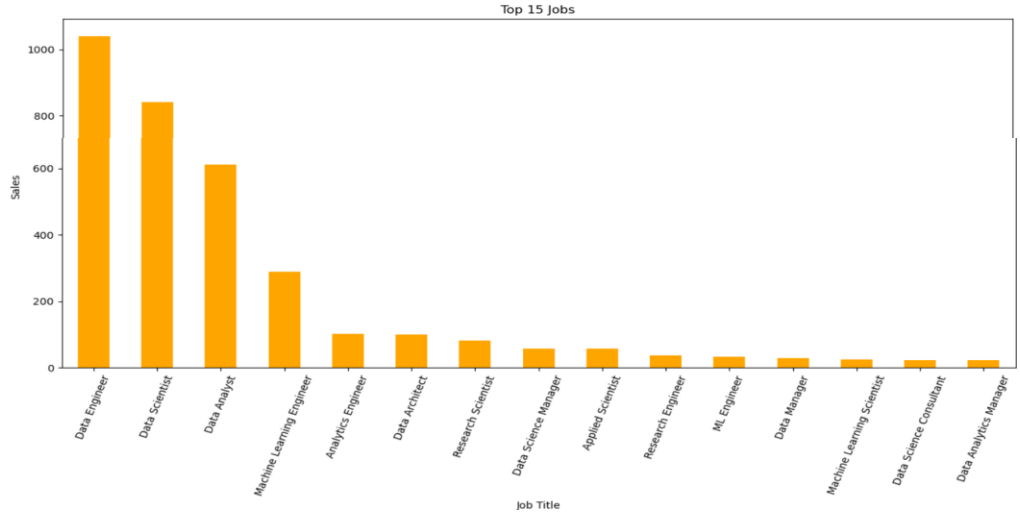

Figure 14 Bar Graph of top 15 jobs

This code generates a bar graph displaying the top 15 jobs based on their frequency counts in the 'job_title' column of the DataFrame 'data'. Initially, it calculates the frequency count of each unique job title using the 'value_counts()' method, resulting in a Series where job titles are the index and their respective counts are the values. Then, it selects the top 15 most frequent jobs by calling the 'head(15)' method on the frequency count Series. Subsequently, a bar graph is created using Matplotlib's 'plot' function with the 'kind' parameter set to 'bar'. The figure size is adjusted to (13, 8) for better visualization. The bars are colored with 'skyblue' for aesthetic appeal, and the legend is disabled. Additionally, the graph's title, x-axis label ('Job Title'), y-axis label ('Sales'), and x-axis tick labels are customized for clarity. Finally, the graph is displayed using 'plt.show()'.

Which job has the highest salaries? Illustrate with bar graph.

```
[27]: # Finding the job title with the highest average salary
highest_job_salary = data.groupby('job_title')['salary_in_usd'].mean().idxmax()
# Print the job title with the highest average salary
print(f"Job with the highest salary: {highest_job_salary}")

# Illustrating the job with the highest average salary with a bar graph
salary_by_job = data.groupby('job_title')['salary_in_usd'].mean()
salary_by_job.plot(kind='bar', figsize=(40,20))# Plotting a bar graph of average salary by job title
plt.title('Average Salary by Job Title')
#Setting the label for x and y axis
plt.xlabel('Job Title')
plt.ylabel('Average Salary-_in_usd')
plt.show() #Displaying the plot
```

```
Job with the highest salary: Data Science Tech Lead
```

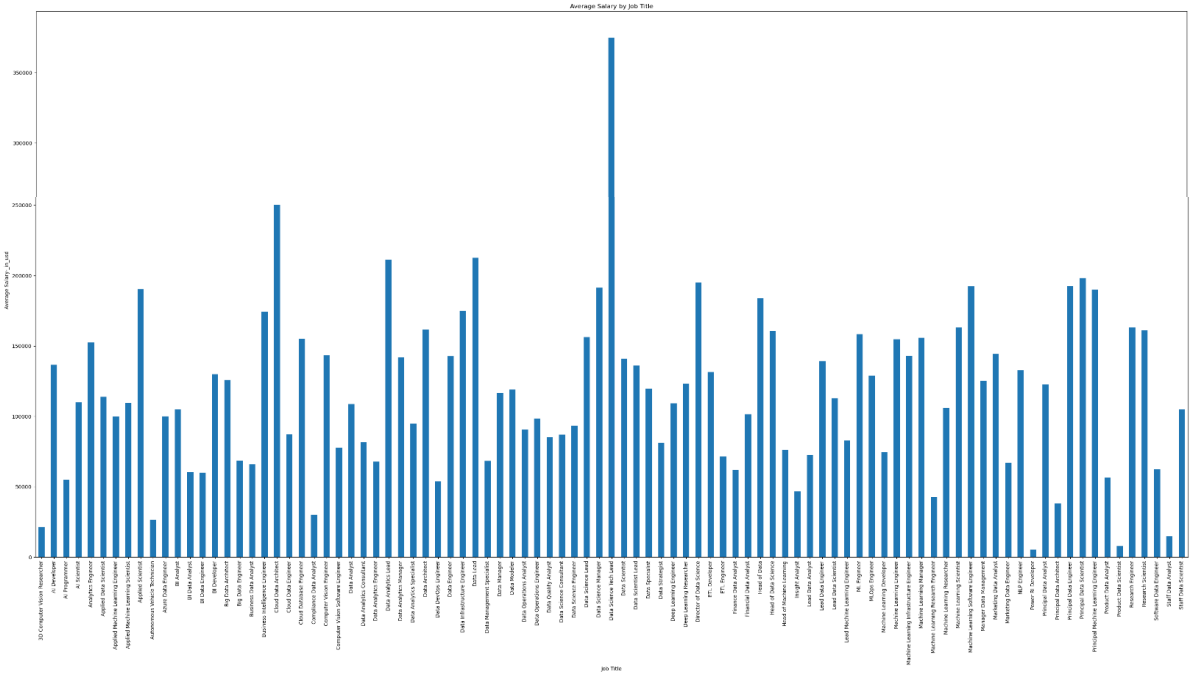

Figure 15 Bar graph of highest salary job

Data Science Tech Lead Job has the highest salaries. Here in this code mean salary was calculated for each job title by grouping the data by 'job_title' and calculating the mean of the 'salary_in_usd' column. Then, it identifies the job title with the highest average salary using the 'idxmax()' method, which returns the maximum value. A bar graph is created using Matplotlib's 'plot' function with the 'kind' parameter set to 'bar'. The figure size is adjusted to (40, 20) for better visualization. Finally, the graph is displayed using 'plt.show()'. This visualization allows for a clear comparison of average salaries across different job titles.

Write a python program to find out salaries based on experience level. Illustrate it through bar graph.

```
[61]: # Calculating salaries based on experience level
salary_by_exp = data.groupby('experience_level')['salary_in_usd'].mean()
# Plotting a bar graph to illustrate salary by experience level
salary_by_exp.plot(kind='bar', figsize=(10, 6))
plt.title('Average Salary by Experience Level')
#Setting the label for x and y axis
plt.xlabel('Experience Level')
plt.ylabel('Average Salary (USD)')
plt.show() #Displaying the plot
```

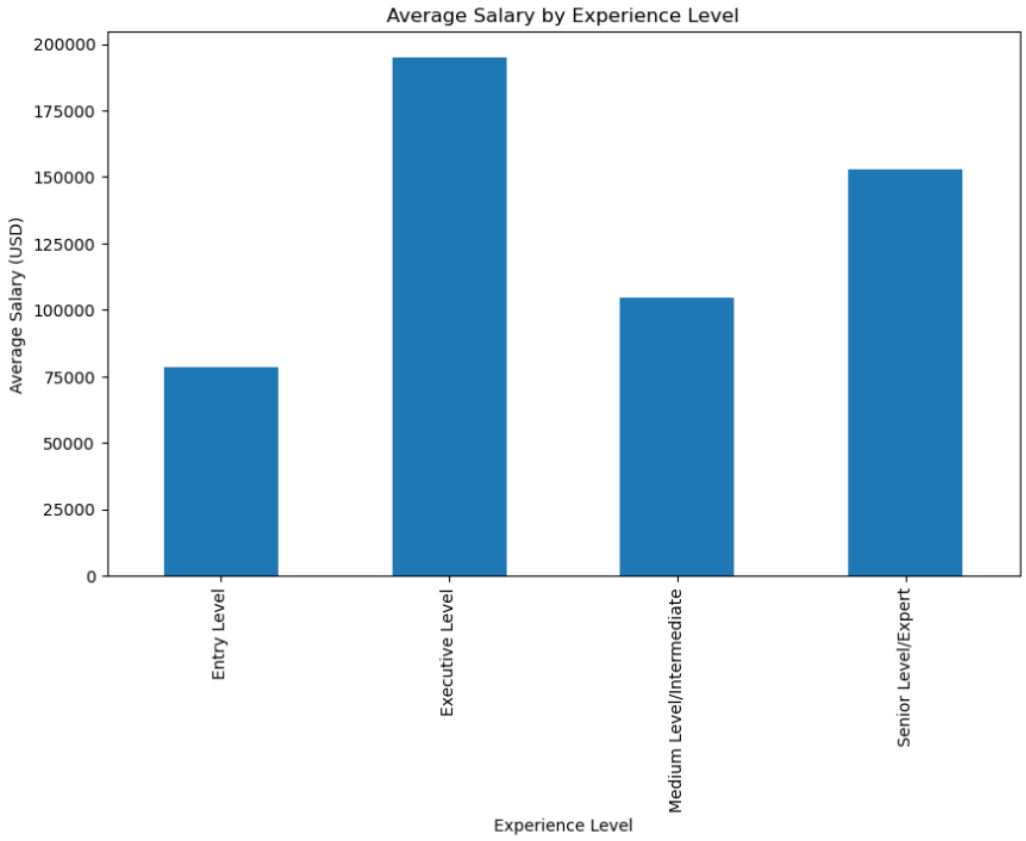

Figure 16 Bar graph of salaries based on experience level

Here in this code average salary was calculated based on experience levels and calculates the mean of the 'salary_in_usd' column for each group. Then, it created a bar graph using Matplotlib's 'plot' function with the 'kind' parameter set to 'bar'. The figure size is adjusted to (10, 6) for better visualization. The graph's title is set to 'Average Salary by Experience Level', and the x-axis label and y-axis label is set. Finally, the graph is displayed using 'plt.show()'.

Write a Python program to show histogram and box plot of any chosen different variables. Use proper labels in the graph.

```
[149]: # Histogram and box plot of a chosen variable
       # Selected variable for analysis
       Selected_variable = 'salary_in_usd'
       # Plot histogram
       plt.figure(figsize=(12, 6))
       plt.subplot(1, 2, 1)
       plt.hist(data[Selected_variable], bins=20, color='skyblue', edgecolor='black')
       plt.title('Histogram of Salary')
       # Setting the label for x axis and y axis
       plt.xlabel('Salary (USD)')
       plt.ylabel('Frequency')
```

```
[149]: Text(0, 0.5, 'Frequency')
```

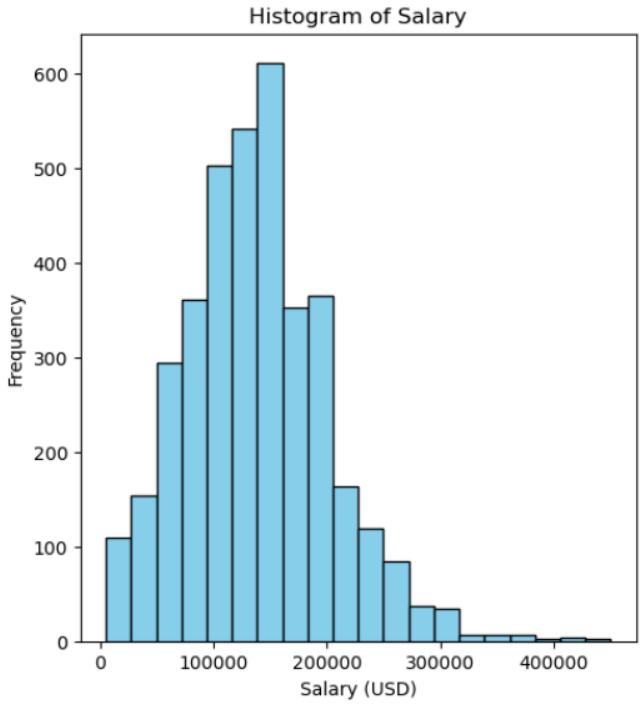

Figure 17 Histogram of Salary

This code generated a histogram and a box plot for the selected variable 'salary_in_usd' in the DataFrame 'data'. Firstly, a Matplotlib figure with a size of (12, 6) is created. Subsequently, a subplot grid with one row and two columns is defined, and the first subplot is designated for the histogram. The 'hist' function is then utilized to plot the histogram of the

'salary_in_usd' variable with 20 bins. The histogram bars are colored sky blue with black edges for clarity. Finally, the histogram is displayed.

```
•[33]: # Creating a subplot for the box plot
plt.subplot(1, 2, 2)
plt.boxplot(data[Selected_variable])
plt.title('Boxplot of Salary')
# Set label for the y-axis
plt.ylabel('Salary (USD)')
# Hide x-axis ticks
plt.xticks([1], [''])
plt.tight_layout() # Adjust layout to prevent overlap
plt.show() #Displaying the plot
```

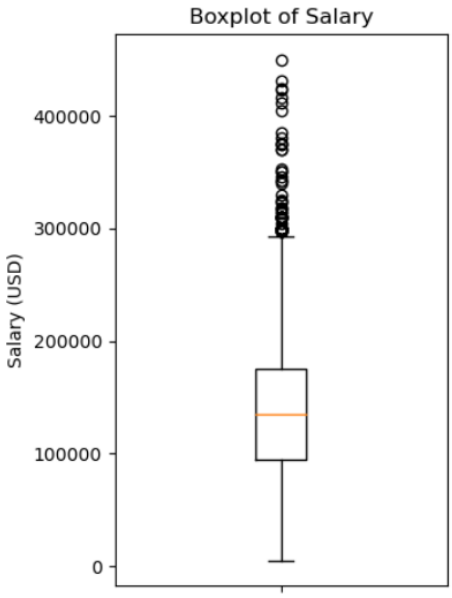

Figure 18 Boxplot of Salary

This code creates a box plot for the selected variable 'salary_in_usd' in the DataFrame 'data'. It starts by defining a subplot grid with one row and two columns, and then designates the second subplot for the box plot. The 'boxplot' function is used to generate the box plot of the 'salary_in_usd' variable. The x-axis tick labels are customized to show an empty string, as indicated by an empty list passed to 'xticks'. This removes the x-axis tick labels for better clarity. Finally, 'plt.tight_layout()' is called to adjust the layout of the subplots for improved appearance, and the plot is displayed using 'plt.show()'.