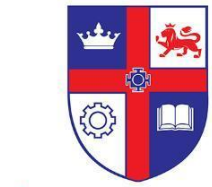


CC5067NI-Smart Data Discovery

60% Individual Coursework

2023-24 Autumn

Student Name: Jenish Katuwal

London Met ID: 22068751

College ID: NP01CP4A220214

Assignment Due Date: Monday, May 13, 2024

Assignment Submission Date: Monday, May 13, 2024

Word Count: 4760

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.

Acknowledgement

I am very thankful to Mr. Dipeshor Silwal, the teacher leading the Smart Data Discovery (CC5067NI) course at Islington College. His knowledge and love for the subject have really helped me learn. This coursework assignment he gave us is a great way for me to apply what I've learned about understanding data, getting it ready, exploring it, and doing some initial analysis. Mr. Silwal's guidance has made me see how important and complex good data science practices are.

I also want to thank Mr. Alish Kc, who has been a wonderful tutor to us. His support and mentoring have been so helpful during this course. Mr. Alish Kc’s way of explaining difficult ideas in a simple way has really helped me understand the topics better. He is patient and open, which creates a nice environment for asking questions until I get it. The feedback from Mr. Kc has improved my analysis skills and how I solve problems by making me think in a more thoughtful and realistic way.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The dataset comprising information about several variables, including experience, work level, job title, and more, that may have an impact on data scientists' pay is analysed in this research. Finding any possible patterns or trends in the data as well as improving comprehension of the factors influencing data scientists' pay are the main goals of this investigation.

Data about data science employment, such as job titles, experience levels, work levels, and other pertinent variables, are included in the dataset utilized for this analysis. By carrying out tasks including data interpretation, data preparation, preliminary data analysis, and data exploration, the analysis will concentrate on getting the data ready for more data mining and analysis.

To get insight into the structure and substance of the data, the data understanding phase will entail analysing the data resources and their attributes. The data will be loaded into a panda DataFrame, extraneous columns will be removed, missing values will be handled, duplicates will be checked, and columns will be renamed for clarity as part of the data preparation process.

The first step in the data analysis process will involve producing summary statistics for a subset of the variables, including skewness, kurtosis, mean, total, and standard deviation, and figuring out how the variables are correlated.

The process of data exploration will entail determining the top 15 job titles, using bar graphs to visualize the incomes associated with various job titles and experience levels, and using histograms and box plots to analyse the distribution of particular variables.

The report will be formatted to include code portions, screenshots, and a concise user manual for the Python applications that were produced, along with the findings from each analysis phase.

2. Data Understanding

Data understanding is the process of gaining familiarity with the dataset and comprehending its characteristics, structure, and content. In the context of this coursework, data understanding involves:

1. Identifying the data sources and their formats.

2. Determining the number of instances (rows) and attributes (columns) in the dataset.

3. Exploring the data types (numeric, categorical, text, etc.) and potential data quality issues (missing values, inconsistencies, outliers, etc.).

4. Understanding the meaning and relevance of each attribute/column in the dataset.

5. Gaining insights into the relationships and patterns within the data.

6. Assessing the completeness and suitability of the dataset for the given analysis task.

The data understanding step is crucial as it lays the foundation for subsequent data preparation, analysis, and modeling tasks. It helps in identifying potential challenges, determining appropriate data cleaning and transformation strategies, and ensuring that the dataset aligns with the analysis objectives.

2.1 Dataset Overview

The provided dataset contains information about various factors that influence the salaries of data science professionals. It consists of 3756 rows and 11 columns, covering details such as work year, experience level, employment type, job title, salary amount, currency, employee residence, remote work ratio, company location, and company size. The dataset seems to be comprehensive, capturing a wide range of relevant attributes that could potentially impact data science salaries.

The data encompasses employees from multiple countries, including the United States, Canada, Spain, Germany, Great Britain, Nigeria, and India. It covers different experience levels, from entry-level to executive-level positions, and includes various job titles related to data science, such as Data Scientist, Machine Learning Engineer, Data Analyst, and others. The dataset also provides information about employment types (full-time or contract), salary currencies, and the size of the companies (small, medium, or large). Additionally, it includes the remote work ratio, which could be an essential factor in today's work environment.

2.2 Summary of current provided dataset

SL No.	Column Name	Description	Data Types
1.	Work_year	This column represents the year for which the salary data is recorded.	Integer.
2.	Experience_level	This column categorizes employees based on their level of experience, such as 'SE' (Senior Level/Expert), 'MI' (Medium Level/Intermediate), 'EN' (Entry Level), and 'EX' (Executive Level).	String
3.	Employment_type	This column indicates whether the employee is employed on a 'FT' (Full-Time) or 'CT' (Contract) basis. It is likely stored as a string or categorical data type.	String

4.	Job_title	This column contains the job titles of employees, such as 'Data Scientist', 'Machine Learning Engineer', 'Data Analyst',	String
5.	Salary	This column represents the salary amount earned by the employee.	integer
6.	Salary_currency	This column indicates the currency in which the salary is paid, such as 'USD', 'EUR', 'INR', and many more.	String
7.	Employee_redidence	This column represents the country where the employee resides, such as 'US', 'CA', 'ES', 'DE', 'GB', 'NG', and 'IN'.	String
8.	Remote_ratio	This column indicates the percentage of remote work for the employee, ranging from 0 (non-remote) to 100 (fully remote)	float
9.	Company_location	This column specifies the country where the company is located.	String
10.	Company_size	This column provides the size of the company, categorized as 'S' (Small), 'M' (Medium), or 'L' (Large).	String

Table 1 Description of dataset.

3. Importing all the necessary libraries to complete the task.

```
[2]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     print("Import successfull !")

     Import successfull !
```

Figure 1 Importing required libraries.

Figure 2 Numpy (Numpy, 2024).

Numpy:

NumPy (Numerical Python) is a powerful open-source Python library designed for efficient numerical computing with arrays and matrices. It provides a multidimensional array object and a collection of routines for array manipulation, mathematical operations, linear algebra, random number generation, and more. NumPy forms the core of many scientific and data analysis libraries in the Python ecosystem, such as Pandas, SciPy, and Matplotlib. With its efficient array operations and high-performance vectorized computations, NumPy enables developers and researchers to write concise and optimized code for handling large datasets and complex mathematical operations, making it an essential tool for scientific and data-intensive applications (Numpy, 2024).

Pandas:

Figure 3 pandas (logowik, 2024).

Pandas is an open-source Python library that provides high-performance, easy-to-use data structures and data analysis tools. It is particularly well-suited for working with structured (tabular, multidimensional, potentially heterogeneous) and time series data. Pandas offers two main data structures: Series (one-dimensional labeled array) and DataFrame (two-dimensional labeled data structure with columns of potentially different data types). With Pandas, users can efficiently load, manipulate, analyze, and visualize data from various sources, such as CSV files, SQL databases, and Excel spreadsheets. It provides powerful data manipulation capabilities, including indexing, merging, reshaping, and data cleaning operations, as well as advanced data analysis tools like grouping, windowing, and time series analysis. Pandas is widely used in data science, finance, academic research, and many other domains that involve working with structured data (GeeksforGeeks, 2024).

Matplotlib:

Figure 4 matplotlib (brandfetch, 2024).

Matplotlib is a powerful data visualization library in Python that produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. It can be used in Python scripts, the Python and IPython shells, web application servers, and various graphical user interface toolkits. Matplotlib is designed to be flexible and user-friendly, allowing users to create a wide range of static, animated, and interactive visualizations. It supports a variety of plot types, including line plots, scatter plots, bar charts, histograms, pie charts, and more. Additionally, Matplotlib provides low-level drawing capabilities, enabling users to create custom visualizations and incorporate them into graphical user interfaces. With its extensive documentation, active community, and seamless integration with other Python libraries like NumPy and

Pandas, Matplotlib has become a standard tool for data visualization and exploration in the scientific and data analysis communities (GeeksforGeeks, 2024).

4. Data Preparation

Data preparation is the process of transforming raw data into a clean and structured format that is suitable for analysis and modeling. It involves a series of steps to ensure that the data is accurate, consistent, and ready for further processing.

The main goals of data preparation are:

Data cleaning: Identifying and correcting or removing inaccurate, incomplete, or irrelevant data. This includes handling missing values, removing duplicates, and identifying and dealing with outliers or anomalies.

Data integration: Combining data from multiple sources into a coherent and consistent dataset. This may involve resolving any conflicts or inconsistencies between different data sources.

Data transformation: Converting data into an appropriate format or structure for analysis. This can include tasks like normalization, scaling, encoding categorical variables, and creating new features or attributes from existing ones.

Data reduction: Reducing the size and complexity of the dataset by selecting relevant features or instances, or by using techniques like dimensionality reduction or data compression.

4.1 Write a python program to load data into pandas DataFrame.

```
•[4]: file_path ="D:\\Coursework\\DataScienceSalaries.csv"  #Define the file path where the 'DataScienceSalaries.csv' file is located
#Read the 'DataScienceSalaries.csv' file using the pd.read_csv() function and store the resulting DataFrame in the variable 'df'
df = pd.read_csv(file_path)
df
```

[4]:

	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 5 Loading data into pandas data frame.

Code Implementation:

The code reads a CSV file named "DataScienceSalaries.csv" located at the specified file path using the pandas library in Python. The file path indicates that the file is stored in the "D:\Course df", which allows for easy manipulation and analysis of the data.

Code Outcome:

The code displays the contents of the DataFrame "df", presenting the data from the CSV file in a structured format for examination or further processing. Overall, this code effectively loads the dataset into memory, enabling subsequent data exploration and analysis tasks.

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

```
[5]: ## Remove columns using 'drop' method.
     df = df.drop(['salary', 'salary_currency'], axis = 1)
     #Verify remaining columns and printing.
     df
```

[5]:

	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 6 Removing column salary and salary currency.

Code Implementation:

df = df.drop(['salary', 'salary_currency'], axis=1): This line removes the 'salary' and 'salary_currency' columns from the DataFrame df. The drop() method is used to remove rows or columns from a DataFrame. In this case, we provide a list of column names (['salary', 'salary_currency']) to be dropped. The axis=1 parameter specifies that we want to drop columns .

Code Outcome:

When we execute this code, it will remove the 'salary' and 'salary_currency' columns from the original DataFrame df. The resulting DataFrame df will be printed, showing the remaining columns and their data.

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

```
[44]: df = df.dropna() # Remove rows with missing values (NaN)
      df.head() # Print the first 5 rows of the updated DataFrame
```

[44]:

	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	Machine Learning Engineer	30000	US	100	US	S
2	2023	Medium Level/Intermediate	CT	Machine Learning 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

Figure 7 Removing NaN missing values.

Code Implementation:

df = df.dropna(): This line removes all rows containing missing values (NaN) from the DataFrame df. The dropna() method is used to remove rows or columns with missing values. By default, it removes rows containing any NaN values.

df.head(): This line prints the first 5 rows of the updated DataFrame df after removing rows with missing values.

Code Outcome:

When we execute this code, it will remove all rows containing missing values (NaN) from the DataFrame df. The resulting DataFrame df will only contain rows with complete data with no missing values.

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

```
[8]: #Replace 'ML Engineer' with 'Machine Learning Engineer' in the 'job_title' column
     df['job_title'] = df['job_title'].replace('ML Engineer', 'Machine Learning Engineer')
     duplicate_rows = df[df.duplicated()] #Identify duplicate rows in the DataFrame
     print(f"Number of duplicate rows are :{duplicate_rows.shape[0]}")  #Print the number of duplicate rows

     duplicate_rows  #Print the duplicate rows
```

```
Number of duplicate rows are :1179
```

[8]:

	work_year	experience_level	employment_type	job_title	salary_in_usd	employee_residence	remote_ratio	company_location	company_size
115	2023	SE	FT	Data Scientist	150000	US	0	US	M
123	2023	SE	FT	Analytics Engineer	289800	US	0	US	M
153	2023	MI	FT	Data Engineer	100000	US	100	US	M
154	2023	MI	FT	Data Engineer	70000	US	100	US	M
160	2023	SE	FT	Data Engineer	115000	US	0	US	M
...	...	...	...	...	...	...	...	...	...
3439	2022	MI	FT	Data Scientist	78000	US	100	US	M
3440	2022	SE	FT	Data Engineer	135000	US	100	US	M
3441	2022	SE	FT	Data Engineer	115000	US	100	US	M
3586	2021	MI	FT	Data Engineer	200000	US	100	US	L
3709	2021	MI	FT	Data Scientist	90734	DE	50	DE	L

1179 rows × 9 columns

Figure 8 Checking duplicates values in data frame.

Code Implementation:

df['job_title'] = df['job_title'].replace('ML Engineer', 'Machine Learning Engineer'): This line replaces the string 'ML Engineer' with 'Machine Learning Engineer' in the 'job_title' column of the DataFrame df. The replace() method is used to substitute one value with another value in a DataFrame .

duplicate_rows = df[df.duplicated()]: This line identifies the duplicate rows in the DataFrame df. The duplicated() method returns a boolean Series indicating whether each row is a duplicate or not. By passing this boolean Series to the DataFrame df, we filter out the unique rows and store the duplicate rows in the variable duplicate_rows.

print(f"Number of duplicate rows are :{duplicate_rows.shape[0]}"): This line prints the number of duplicate rows in the DataFrame. The shape attribute of a DataFrame returns a tuple containing the number of rows and columns. By accessing duplicate_rows.shape[0], we get the number of rows in the duplicate_rows DataFrame, which corresponds to the number of duplicate rows in the original DataFrame df.

duplicate_rows: This line prints the duplicate_rows DataFrame, which contains only the duplicate rows from the original DataFrame df.

Code Outcome:

Replace the string 'ML Engineer' with 'Machine Learning Engineer' in the 'job_title' column of the DataFrame df.

Identify the duplicate rows in the DataFrame df and store them in the duplicate_rows DataFrame.

Print the number of duplicate rows found in the DataFrame.

Print the duplicate_rows DataFrame, which will display the duplicate rows from the original DataFrame df.

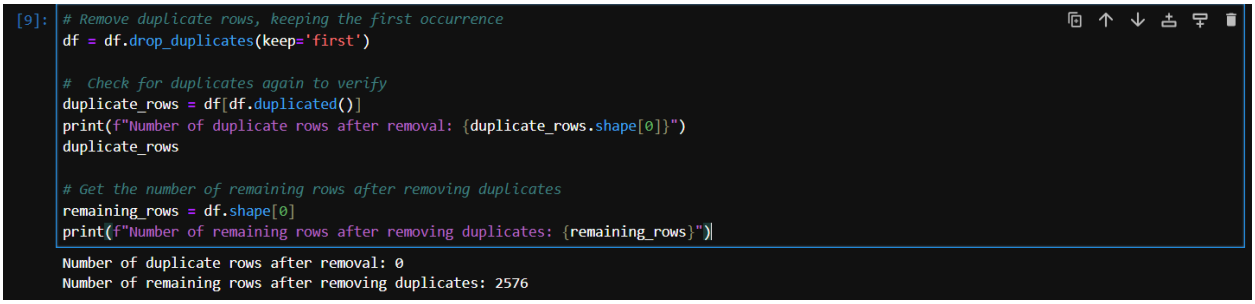


Figure 9 Confirming removed duplicate values.

Code Implementation:

df = df.drop_duplicates(keep='first'): This line removes all duplicate rows from the DataFrame df, while keeping the first occurrence of each duplicate row. The drop_duplicates() method is used to remove duplicate rows from a DataFrame. The keep='first' parameter specifies that the first occurrence of each duplicate row should be kept, while subsequent duplicates are dropped.

duplicate_rows = df[df.duplicated()]: This line checks for any remaining duplicate rows in the DataFrame df after removing duplicates. The duplicated() method returns a

boolean Series indicating whether each row is a duplicate or not. By passing this boolean Series to the DataFrame df, we filter out the unique rows and store any remaining duplicate rows in the variable duplicate_rows.

print(f"Number of duplicate rows after removal: {duplicate_rows.shape[0]}"): This line prints the number of duplicate rows remaining in the DataFrame after removing duplicates. The shape attribute of a DataFrame returns a tuple containing the number of rows and columns. By accessing duplicate_rows.shape[0], we get the number of rows in the duplicate_rows DataFrame, which corresponds to the number of duplicate rows remaining in the DataFrame df.

duplicate_rows: This line prints the duplicate_rows DataFrame, which contains any remaining duplicate rows from the DataFrame df after removing duplicates.

remaining_rows = df.shape[0]: This line gets the number of remaining rows in the DataFrame df after removing duplicates. The shape attribute of a DataFrame returns a tuple containing the number of rows and columns. By accessing df.shape[0], we get the number of rows in the DataFrame df.

print(f"Number of remaining rows after removing duplicates: {remaining_rows}"): This line prints the number of remaining rows in the DataFrame df after removing duplicates.

Code Outcome:

Remove all duplicate rows from the DataFrame df, keeping the first occurrence of each duplicate row.

Check for any remaining duplicate rows in the DataFrame df after removing duplicates.

Print the number of remaining duplicate rows (if any) found in the DataFrame.

Print the duplicate_rows DataFrame, which will display any remaining duplicate rows from the DataFrame df.

Print the number of remaining rows in the DataFrame df after removing duplicates.

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

```python
# Loop through each column in the DataFrame
for column in df.columns:
    unique_values = df[column].unique()  #Get the unique values for the current column
    print(column)
    for value in unique_values: #Print each unique value in the column
        print(f"- {value}")
```

Figure 10 Program to see the unique values from all the columns.

Code Implementation:

for column in df.columns: This line starts a loop that iterates over each column in the DataFrame df.

unique_values = df[column].unique(): Inside the loop, this line gets the unique values for the current column by using the unique() method on the Series df[column]. The unique() method returns an array of unique values in the Series or column.

print(column) This line prints the name of the current column.

for value in unique_values: This line starts an inner loop that iterates over each unique value in the unique_values array.

print(f"- {value}") Inside the inner loop, this line prints each unique value in the current column.

Code Outcome:

When we execute this code, it will loop through each column in the DataFrame df and print the unique values present in each column. The output will look something like this:

```
work_year
- 2023
- 2022
- 2020
- 2021
experience_level
- SE
- MI
- EN
- EX
employment_type
- FT
- CT
- FL
- PT
job_title
- Principal Data Scientist
- Machine Learning Engineer
- Data Scientist
- Applied Scientist
- Data Analyst
- Data Modeler
- Research Engineer
- Analytics Engineer
- Business Intelligence Engineer
- Data Strategist
- Data Engineer
- Computer Vision Engineer
- Data Quality Analyst
- Compliance Data Analyst
- Data Architect
- Applied Machine Learning Engineer
- AI Developer
```

Figure 11 Output of the unique values.

4.6 Rename the experience level columns as below

SE – Senior Level/Expert.

MI – Medium Level/Intermediate

EN – Entry Level

EX – Executive Level

```
[13]: # Define a dictionary to map the experience level codes to their full names
position_rename = {
    'SE': 'Senior Level/Expert',
    'MI': 'Medium Level/Intermediate',
    'EN': 'Entry Level',
    'EX': 'Executive Level'
}
# Replace the experience level codes with their full names in the 'experience_level' column
df['experience_level'] = df['experience_level'].replace(position_rename)

df #Print the updated DataFrame
```

[13]:

	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	Machine Learning Engineer	30000	US	100	US	S
2	2023	Medium Level/Intermediate	CT	Machine Learning 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

2576 rows × 9 columns

Figure 12 Renaming the columns name as per required.

Code Implementation:

position_rename = {...} This line defines a Python dictionary position_rename that maps the experience level codes ('SE', 'MI', 'EN', 'EX') to their corresponding full names ('Senior Level/Expert', 'Medium Level/Intermediate', 'Entry Level', 'Executive Level').

df['experience_level'] = df['experience_level'].replace(position_rename): This line replaces the values in the 'experience_level' column of the DataFrame df with their corresponding full names using the replace() method. The position_rename dictionary is used as a mapping for the replacement. The replace() method replaces the values in the 'experience_level' column with the values from the position_rename dictionary.

df This line prints the updated DataFrame df after replacing the experience level codes with their full names.

Code Outcome:

After executing this code, the 'experience_level' column in the DataFrame df will have the experience level codes replaced with their corresponding full names. The output will look like the original DataFrame, but with the 'experience_level' column showing the full names instead of the short form.

5. Data Analysis

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

```
# Select the 'salary_in_usd' column for analysis
Salary_data = 'salary_in_usd'

# Calculate the summary statistics
sum = df[Salary_data ].sum()
mean = df[Salary_data ].mean()
standarDeviation = df[Salary_data ].std()
skewness = df[Salary_data ].skew()
kurtosis = df[Salary_data ].kurt()

# Print the summary statistics
print("summary stats are describe below:")
print("Sum:", sum)
print("mean:", mean)
print("standarDeviation:", standarDeviation)
print("Skewness:", skewness)
print("Kurtosis:", kurtosis)
```

```
summary stats are describe below:
Sum: 343206451
mean: 133232.31793478262
standarDeviation: 67110.61549757926
Skewness: 0.624093592127214
Kurtosis: 0.837974393681983
```

Figure 13 Statistical data of salary_in_usd.

Code Implementation:

Salary_data = 'salary_in_usd': This line assigns the string 'salary_in_usd' to the variable Salary_data, which represents the column name for salary data in US dollars.

sum = df[Salary_data].sum(): This line calculates the sum of all values in the 'salary_in_usd' column of the DataFrame df using the sum() method.

mean = df[Salary_data].mean(): This line calculates the mean (average) of all values in the 'salary_in_usd' column using the mean() method.

standarDeviation = df[Salary_data].std(): This line calculates the standard deviation of all values in the 'salary_in_usd' column using the std() method.

skewness = df[Salary_data].skew(): This line calculates the skewness (measure of asymmetry) of the distribution of values in the 'salary_in_usd' column using the skew() method.

kurtosis = df[Salary_data].kurt(): This line calculates the kurtosis (measure of tailedness) of the distribution of values in the 'salary_in_usd' column using the kurt() method.

print("summary stats are describe below:"): This line prints a header indicating that the summary statistics will be displayed.

print("Sum: ", sum): This line prints the sum of all values in the 'salary_in_usd' column.

print("mean: ", mean): This line prints the mean (average) of all values in the 'salary_in_usd' column.

print("standarDeviation: ", standarDeviation): This line prints the standard deviation of all values in the 'salary_in_usd' column.

print("Skewness: ", skewness): This line prints the skewness of the distribution of values in the 'salary_in_usd' column.

print("Kurtosis: ", kurtosis): This line prints the kurtosis of the distribution of values in the 'salary_in_usd' column.

Code Outcome:

When we execute this code, it will calculate and print the summary statistics (sum, mean, standard deviation, skewness, and kurtosis) for the 'salary_in_usd' column in the DataFrame df.

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

```
[30]: # Select only numerical columns for correlation
      numerical_df = df.select_dtypes(include=['number'])

      # Calculate the correlation matrix
      correlation_matrix = numerical_df.corr()

      # Print the correlation matrix
      print("Correlation Matrix:")
      print(correlation_matrix)

      Correlation Matrix:
                    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 14 Calculating correlation variables.

Code Implementation:

numerical_df = df.select_dtypes(include=['number']): This line creates a new DataFrame numerical_df that contains only the numerical columns from the original DataFrame df. The select_dtypes() method is used to select columns based on their data types. By setting include=['number'], it selects all columns with numeric data types.

correlation_matrix = numerical_df.corr(): This line calculates the correlation matrix for the numerical_df DataFrame using the corr() method. The correlation matrix shows the pairwise correlation coefficients between all numerical columns in the DataFrame.

print("Correlation Matrix:"): This line prints a header indicating that the correlation matrix will be displayed.

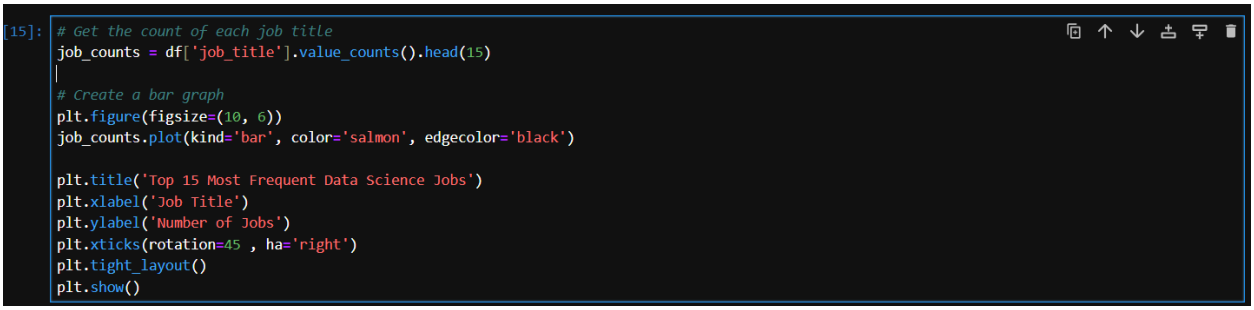
print(correlation_matrix): This line prints the correlation matrix as a table, showing the correlation coefficients between each pair of numerical columns.

Code Outcome:

When we execute this code, it will first create a new DataFrame numerical_df that contains only the numerical columns from the original DataFrame df. Then, it will calculate the correlation matrix for these numerical columns using the corr() method.

6. Data Exploration

6.1 Write a python program to find out top 15 jobs. Make a bar graph of sales as well.



```
[15]: # Get the count of each job title
      job_counts = df['job_title'].value_counts().head(15)

      # Create a bar graph
      plt.figure(figsize=(10, 6))
      job_counts.plot(kind='bar', color='salmon', edgecolor='black')

      plt.title('Top 15 Most Frequent Data Science Jobs')
      plt.xlabel('Job Title')
      plt.ylabel('Number of Jobs')
      plt.xticks(rotation=45 , ha='right')
      plt.tight_layout()
      plt.show()
```

Figure 15 Top 15 jobs.

Code Implementation:

job_counts = df['job_title'].value_counts().head(15): This line calculates the count of each unique job title in the 'job_title' column of the DataFrame df. The value_counts() method counts the occurrences of each unique value in the column. The head(15) method selects the top 15 most frequent job titles.

plt.figure(figsize=(10, 6)): This line creates a new figure with a specified size (10 inches wide by 6 inches tall).

job_counts.plot(kind='bar', color='salmon', edgecolor='black'): This line creates a bar plot using the job_counts data. The kind='bar' parameter specifies that a bar chart should be created. The color='salmon' parameter sets the color of the bars to salmon, and edgecolor='black' sets the edge color of the bars to black.

plt.title('Top 15 Most Frequent Data Science Jobs'): This line sets the title of the plot.

plt.xlabel('Job Title'): This line sets the label for the x-axis.

plt.ylabel('Number of Jobs'): This line sets the label for the y-axis.

plt.xticks(rotation=45, ha='right'): This line rotates the x-axis tick labels by 45 degrees and aligns them to the right for better readability.

plt.tight_layout(): This line adjusts the spacing between the plot elements to ensure that nothing is cut off or overlapping.

plt.show(): This line displays the plot in a separate window.

Code Outcome:

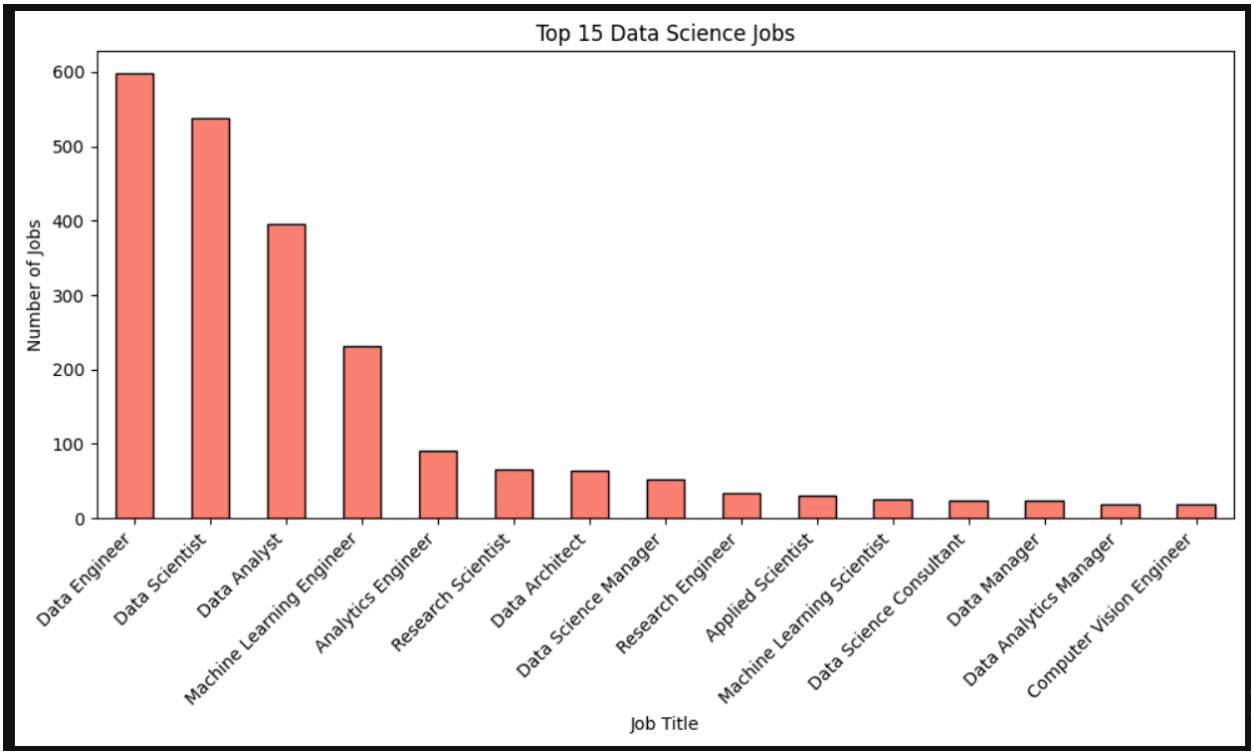


Figure 16 Bar graphs of Top 15 jobs.

When we execute this code, it will create a bar chart showing the top 15 most frequent job titles in the dataset. The x-axis will display the job titles, and the y-axis will show the count or frequency of each job title.

The bars in the chart will be coloured salmon with black edges. The title of the chart will be "Top 15 Most Frequent Data Science Jobs", and the x-axis and y-axis labels will be "Job Title" and "Number of Jobs", respectively.

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

```
# Calculate the average salary for each job title and sort in descending order
salary_by_job = df.groupby('job_title')['salary_in_usd'].mean().sort_values(ascending=False)

# Select the top 10 highest paying jobs
top_10_jobs = salary_by_job.head(10)

# Create a bar graph
plt.figure(figsize=(10, 6))
top_10_jobs.plot(kind='bar', color='lightgreen', edgecolor='black')
plt.title('Top 10 Highest Paying Data Science Jobs')
plt.xlabel('Job Title')
plt.ylabel('Average Salary (USD)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

Figure 17 Highest salary data.

Code Implementation:

salary_by_job=df.groupby('job_title')['salary_in_usd'].mean().sort_values(ascending=False): This line calculates the average salary for each job title in the 'job_title' column of the DataFrame df. The groupby('job_title') groups the data by job title, and ['salary_in_usd'].mean() calculates the mean (average) of the 'salary_in_usd' column for each group. The sort_values(ascending=False) sorts the resulting Series in descending order

top_10_jobs = salary_by_job.head(10): This line selects the top 10 highest paying job titles from the salary_by_job Series.

plt.figure(figsize=(10, 6)): This line creates a new figure with a specified size (10 inches wide by 6 inches tall).

top_10_jobs.plot(kind='bar', color='lightgreen', edgecolor='black'): This line creates a bar plot using the top_10_jobs data. The kind='bar' parameter specifies that a bar chart should be created. The color='lightgreen' parameter sets the color of the bars to light green, and edgecolor='black' sets the edge color of the bars to black.

plt.title('Top 10 Highest Paying Data Science Jobs'): This line sets the title of the plot.

plt.xlabel('Job Title'): This line sets the label for the x-axis.

plt.ylabel('Average Salary (USD)'): This line sets the label for the y-axis.

plt.xticks(rotation=45, ha='right'): This line rotates the x-axis tick labels by 45 degrees and aligns them to the right for better readability.

plt.tight_layout(): This line adjusts the spacing between the plot elements to ensure that nothing is cut off or overlapping.

plt.show(): This line displays the plot in a separate window.

Code Outcome:

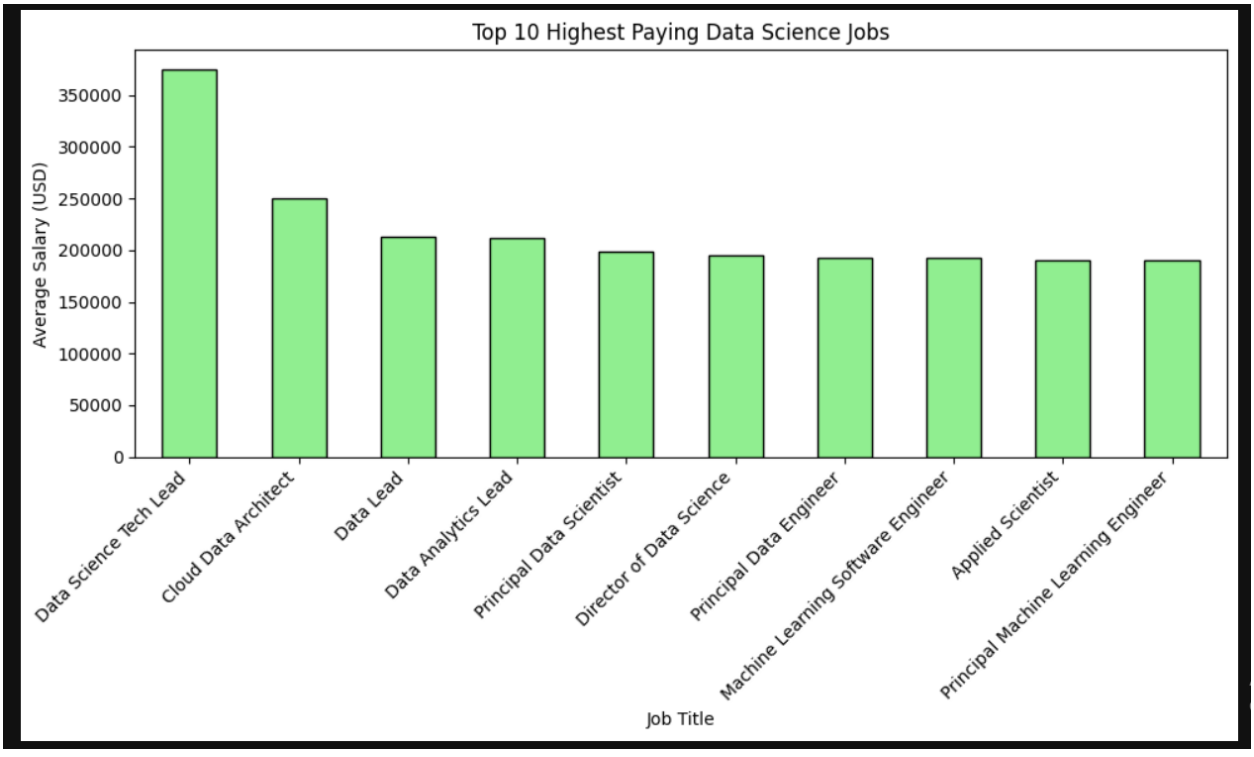


Figure 18 Bar graph of highest salary data.

When we execute this code, it will create a bar chart showing the top 10 highest paying job titles in the dataset based on the average salary. The x-axis will display the job titles, and the y-axis will show the average salary in US dollars for each job title.

The bars in the chart will be colored light green with black edges. The title of the chart will be "Top 10 Highest Paying Data Science Jobs", and the x-axis and y-axis labels will be "Job Title" and "Average Salary (USD)", respectively.

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

```
# Group salaries by experience level and calculate the mean
salary_by_experience = df.groupby('experience_level')['salary_in_usd'].mean().sort_values(ascending=False)

# Create a bar graph
plt.figure(figsize=(8, 6))
salary_by_experience.plot(kind='bar', color='skyblue', edgecolor='black')

plt.title('Average Salary by Experience Level')
plt.xlabel('Experience Level')
plt.grid(True, linestyle='dashed', alpha=0.7)
plt.ylabel('Average Salary (USD)')
plt.xticks(rotation=30, ha='right')  # Rotate x-axis labels for readability
plt.tight_layout()
plt.show()
```

Figure 19 Data of the salaries based on experience level.

Code Implementation:

salary_by_experience = df.groupby('experience_level')['salary_in_usd'].mean().sort_values(ascending=False): This line groups the data in the DataFrame df by the 'experience_level' column, calculates the mean (average) of the 'salary_in_usd' column for each experience level group, and sorts the resulting Series in descending order.

plt.figure(figsize=(8, 6)): This line creates a new figure with a specified size (8 inches wide by 6 inches tall).

salary_by_experience.plot(kind='bar', color='skyblue', edgecolor='black'): This line creates a bar plot using the salary_by_experience data. The kind='bar' parameter specifies that a bar chart should be created. The color='skyblue' parameter sets the color of the bars to sky blue, and edgecolor='black' sets the edge color of the bars to black.

plt.title('Average Salary by Experience Level'): This line sets the title of the plot.

plt.xlabel('Experience Level'): This line sets the label for the x-axis.

plt.grid(True, linestyle='dashed', alpha=0.7): This line adds a dashed grid to the plot with an opacity of 0.7 (70%) for better readability.

plt.ylabel('Average Salary (USD)'): This line sets the label for the y-axis.

plt.xticks(rotation=30, ha='right'): This line rotates the x-axis tick labels by 30 degrees and aligns them to the right for better readability.

plt.tight_layout(): This line adjusts the spacing between the plot elements to ensure that nothing is cut off or overlapping.

plt.show(): This line displays the plot in a separate window.

Code Outcome:

Figure 20 Bar graph of average salary by experience level.

When we execute this code, it will create a bar chart showing the average salary for each experience level. The x-axis will display the experience levels, and the y-axis will show the average salary in US dollars for each experience level.

The bars in the chart will be colored sky blue with black edges. The title of the chart will be "Average Salary by Experience Level", and the x-axis and y-axis labels will be "Experience Level" and "Average Salary (USD)", respectively.

The x-axis tick labels (experience levels) will be rotated by 30 degrees and aligned to the right for better readability.

Additionally, a dashed grid will be added to the plot with an opacity of 70% to improve readability and make it easier to compare the average salaries across different experience levels.

The resulting plot will be displayed in a separate window, allowing you to analyze the relationship between experience level and average salary in the dataset.

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

```
# Choose your variables
variable_one = 'salary_in_usd'
variable_two= 'work_year'

# Create subplots for histogram and box plot
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Histogram for variable1
axes[0].hist(df[variable1], bins=20, edgecolor='black')
axes[0].set_title(f'Histogram of {variable_one}')
axes[0].set_xlabel(variable_one)
axes[0].set_ylabel('Frequency')

# Box plot for variable2
axes[1].boxplot(df[variable2], vert=False, patch_artist=True, showfliers=False,
                boxprops=dict(facecolor='lightblue', color='black'),
                whiskerprops=dict(color='black'),
                capprops=dict(color='black'),
                medianprops=dict(color='red'))
axes[1].set_title(f'Box Plot of {variable_two}')
axes[1].set_xlabel(variable_two)

# Adjust spacing and show the plots
plt.tight_layout()
plt.show()
```

Figure 21 plotting histogram and box plot on two different variables.

Code Implementation:

variable_one = 'salary_in_usd' and **variable_two = 'work_year':** .

fig, axes = plt.subplots(1, 2, figsize=(12, 5)): This line creates a figure with two subplots arranged horizontally. The figsize parameter sets the size of the figure to 12 inches wide by 5 inches tall.

axes[0].hist(df[variable_one], bins=20, edgecolor='black'): This line creates a histogram for variable_one on the first subplot. The bins=20 parameter specifies the number of bins or bars in the histogram. The edgecolor='black' parameter sets the edge color of the bars to black.

axes[0].set_title(f'Histogram of {variable_one}'), axes[0].set_xlabel(variable_one), and axes[0].set_ylabel('Frequency'): These lines set the title, x-axis label, and y-axis label for the histogram subplot.

axes[1].boxplot(df[variable_two], vert=False, patch_artist=True, showfliers=False, boxprops=dict(facecolor='lightblue', color='black'), whiskerprops=dict(color='black'), capprops=dict(color='black'), medianprops=dict(color='red')): This line creates a horizontal box plot for variable_two on the second subplot. The vert=False parameter makes the box plot horizontal. The patch_artist=True parameter allows customization of the box plot elements. The showfliers=False parameter hides the outlier points. The boxprops, whiskerprops, capprops, and medianprops parameters customize the appearance of the box plot elements, such as the box color, whisker color, cap color, and median line color.

axes[1].set_title(f'Box Plot of {variable_two}') and axes[1].set_xlabel(variable_two): These lines set the title and x-axis label for the box plot subplot.

plt.tight_layout(): This line adjusts the spacing between the subplots and the figure elements to ensure that nothing is cut off or overlapping.

plt.show(): This line displays the figure with the two subplots in a separate window.

Code Outcome:

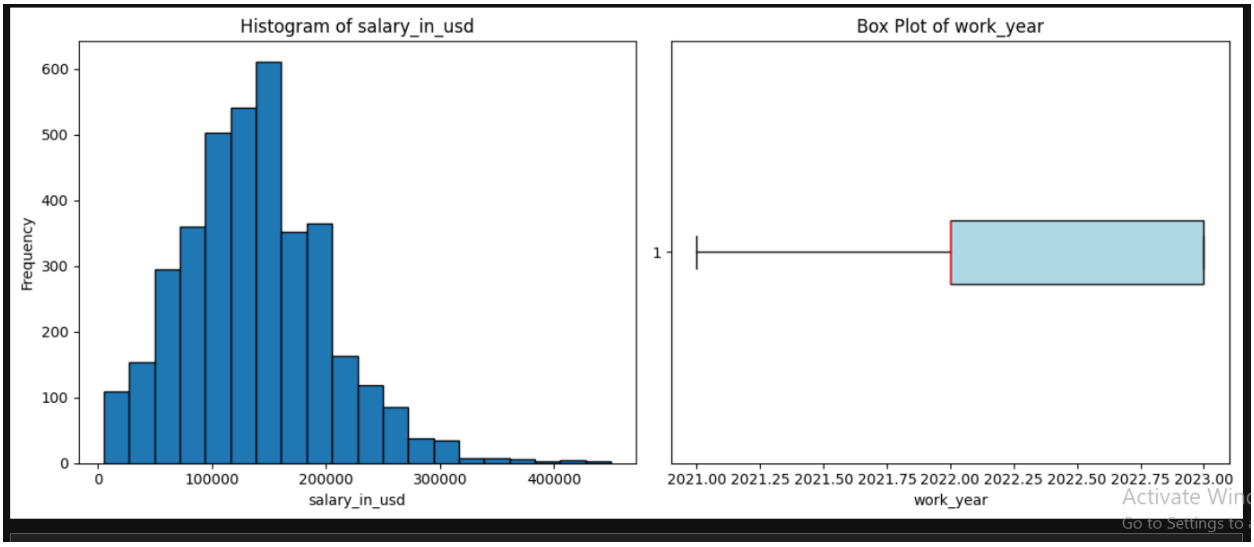


Figure 22 Showing histogram and box plot of own chosen variable.

When we execute this code, it will create a figure with two subplots: a histogram for variable_one (salary_in_usd) and a horizontal box plot for variable_two (work_year).

The histogram subplot will display the distribution of salary_in_usd data, with the x-axis representing the salary range and the y-axis representing the frequency or count of data points within each bin or salary range. The histogram will have 20 bins, and the bars will have black edges.

The box plot subplot will display the distribution of work_year data in a horizontal format. The box plot will show the median value (red line), the interquartile range (IQR) as a blue box, and the whiskers extending from the box to the minimum and maximum values (excluding outliers). The box plot will not show outlier points, as specified by the showfliers=False parameter.

The resulting figure will be displayed in a separate window, allowing you to analyze the distributions of salary_in_usd and work_year variables simultaneously.

7. Conclusion

In conclusion, the dataset has provided insightful information and useful conclusions that can guide strategic efforts and decision-making processes. We made sure that the data we obtained was reliable and authentic by carefully managing the preprocessing, loading, and analysis stages. By identifying the highest-ranking job positions, analyzing pay distributions, and looking at experience-level-based salary differences, we have given stakeholders a thorough grasp of the dataset's main patterns. Furthermore, correlation analysis has illuminated plausible associations among variables, providing other pathways for examination and research. Overall, this coursework emphasizes the value of making decisions based on data and the effectiveness of using data analytics tools to draw meaningful conclusions from large, complicated datasets, which in turn leads to well-informed decisions and fosters organizational success.

8. Reference

brandfetch, 2024. *brandfetch.* [Online]
Available at: https://brandfetch.com/matplotlib.org
[Accessed 12 May 2024].

GeeksforGeeks, 2024. *GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/introduction-to-pandas-in-python/
[Accessed 12 May 2024].

GeeksforGeeks, 2024. *GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/python-introduction-matplotlib/
[Accessed 12 May 2024].

logowik, 2024. *Logowik.* [Online]
Available at: https://logowik.com/pandas-logo-vector-svg-png-free-download-22087.html
[Accessed 12 May 2024].

Numpy, 2024. *Numpy.* [Online]
Available at: https://numpy.org/doc/stable/user/whatisnumpy.html
[Accessed 12 May 2024].