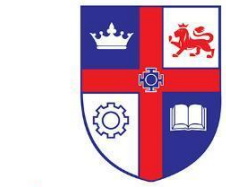
LONDON METROPOLITAN UNIVERSITY



islington college
(इस्लिङ्टन कलेज)

CC5067NI-Smart Data Discovery

60% Individual Coursework

2023-24 Autumn

Student Name: Aavash Sawd

London Met ID: 22085485

College ID: np01cp4s230017

Assignment Due Date: Monday, May 13, 2024

Assignment Submission Date: Sunday, May 12, 2024

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 Figures

Table of Tables

1.Data Understanding

Data understanding is the process of exploring and investing data to get information and knowledge about its content, quantity, and structure using various tools and libraries. To explore data, find patterns and trends, and understand its properties, such libraries as Pandas , NumPy and Matplotlib are employed. Data loading and cleaning, detection of missing values, visualization of data through graphs and figures, and calculation of basic statistical measures all constitute data understanding-related operations. (Maheshwari, 2021). Data understanding is important to ensure the quality of data and identifying relevant features. For the data understanding, dataset is required. Dataset is a collection of data that is often structured in table form. The information presented on the table helps with understanding the information.

The chosen dataset represents a detailed combination of salaries for different roles within the tech industry. Such attributes include the year when the data was collected, the level of experience of an employee, his or her status, job title, salary in a local currency and in USD, country, the percentage of remote work, location for the employer, and the number of employees in a company. The significance of such a dataset is associated with the fact that it allows understanding of the given field’s salaries on different levels.

On the one hand, it can be advantageous for detailed data analysis and understanding the market situations in technology and IT-specific areas. It provides companies with a clear understanding of what their specialists have to be paid based on international standards without referencing only to the national level. Thus, it will be possible to address some challenges related to salary inequality. Many employees working for firms located in different countries often have different payments in the relative currency, but their positions and experience are similar. Data reflects this, meaning that firms can compare the same roles.

Below is a table containing descriptions and data types for each column within the dataset.

SN	Column Name	Description	Data Type
1	work_year	This is important in the sense that salary changes can be analysed based on distinct periods or years, when the salary information was documented.	Integer
2	experience_level	Some level of expertise or experience of the employee, such as Senior Engineer or Midlevel Engineer, among others. This helps in understanding salary variations in relation to different levels of experience.	String
3	employment_type	This data provides information about the salary difference between Permanent roles and the Contract/Temporary roles.	String
4	job_title	This would help to understand which designations get higher pays e.g. Principal Data Scientist, ML Engineer, etc.	String
5	salary	The salary figure stated in the original currency without any conversions. This serves as the fundamental salary data point.	Integer
6	salary_currency	The currency used for salary payments, such as Euros (EUR), US Dollars (USD), and so on. This detail is crucial for grasping the context of the salary amount.	String
7	salary_in_usd	The salary amount converted into US Dollars for standardization and facilitation of comparison across various countries and currencies.	Integer
8	remote_ratio	The variable indicates the percentage of work done, taking values on a closed interval from 0 to 100. The range of this variable has a positive relationship with the dependent variable, so higher	Integer

		remote ratios may have an impact on the levels of salary	
9	employee_residence	The location/country where the employee is located is used, with the abbreviation for Spain being ES, the one for the United States being US, etc. This variable is used for identifying the differences in the levels of salary considering the location of the employee.	String
10	company_size	The size of the company. Regarding the size of companies, the following categories exist: Large, Small, Medium, etc. The variability of these data may have an impact on the levels of salary, and the use of this variable can help with the analysis of compensation structures.	String
11	company_location	Country of the company. The same as for the variable country, the slight abbreviation of the country of the company is used. The designation is used in the analysis to perform assortment based on the location of the company.	String

Table 1:Dataset Table

2.Data Preparation

The process of transforming raw data into a form that is more suitable for modelling is known as Data Preparation (Jaiswal, n.d.).

This process involves several steps, including:

-Data cleaning: handling missing values, removing duplicates, and correcting errors

-Data transformation: converting data types, aggregating data, and creating new features

-Data reduction: selecting the most relevant features, removing correlated features, and handling missing values

-Data visualization: exploring the distribution of data, identifying patterns, and detecting outliers

2.1 Write a python program to load data into pandas Data Frame.

A Data frame is a data structure that, like a spreadsheet that arranges data into a 2-dimensional table of rows and column. Because of its flexibility and intuitive way of storing and working with data, data frames are one of the most used data structures in modern data analytics (Nelamali, 2024).

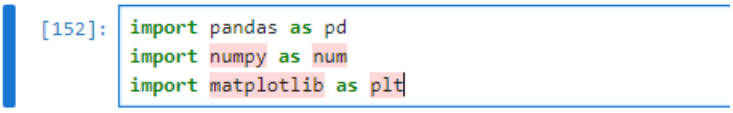


Figure 1:Importing Required Packages

The required packages are being imported in above figure.

Figure 2:Displaying Data

Displaying the data available.

	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:Displaying Dataframe

This code uses the Pandas module in the Python language to import as a CSV (Comma-Separated Values) file by the name as Salary. csv', and it will store its contents in a DataFrame called' data'. The pd. read_csv() function is used for reading the csv file, locating 'Salary'. csv is the value that will be passed as an argument that will initialize the file path. When the data is loaded into the DataFrame, we call the display() function to visually display the contents of the 'data' DataFrame in a tabular format, which is very useful for checking the structure and contents of the loaded data. At the end an empty print() is inserted to fabricate a blank line after displayed DataFrame, which can be useful for visual separation standing information from forthcoming code or output.

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

	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 4:Before Removing Unnecessary Column i.e. Salary and salary currency

The above figure shows the no of columns without removing any.

```
*[156]: # Read the data from CSV
df = pd.read_csv('Salary.csv')

# Remove unnecessary columns i.e ('salary', 'salary_currency')
df.drop(['salary', 'salary_currency'], axis=1, inplace=True)

#print the updated data frame
df.head()
```

Figure 5:Using function to remove column

The following code performs data loading and data preprocessing tasks with the aid of the Pandas library in Python.

1. Reading the data:

- The code uses the pd. read_csv() function to read a CSV file named 'Salary'. takes csv' and creates a DataFrame named 'df from the file content.

- The 'Salary. 'csv' file is considered to be in the same working directory where Python script was found.

- Pandas creates the DataFrame columns by automatically inferring the column names from the first row of the CSV file.

2. Removing unnecessary columns:

- The script takes the 'df' data frame's 'salary' and 'salary_currency' columns out through the drop() function.

- The columns to be removed as a list ['salary', 'salary_currency'], are by the columns parameter of drop().

- The parameter axis=1 tells that the specified columns should be dropped (while axis=0 is used for dropping rows).

- inplace=True parameter stands for non-destructive transformation to the 'df' DataFrame and modification directly without creation of new DataFrame.

3. Previewing the updated DataFrame:

- The code employs the head() function which let's the user sees the first couple number of rows (in a default, five rows) of the updated 'df' DataFrame.

- This option provides a fast way to check the DataFrame after deleting the extra columns.

[156]:

	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

Figure 6:Dataframe after removing column

The above figure shows the dataframe with no of columns after removing unnecessary columns.

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

```
[187]: print(df[df.isna().any(axis=1)])
       #Remove rows with NaN values
       df = df.dropna()

       #Print the updated DataFrame
       df

Empty DataFrame
Columns: [work_year, experience_level, employment_type, job_title, salary, salary_currency, salary_in_usd, employee_residence, remote_ratio, company_location, company_size]
Index: []
```

Figure 7:Checking Nan values

The above code performs two tasks: allowing us to see the rows that are missing values and remove these rows from the DataFrame. Let's go through each part in detail:Let's go through each part in detail:

1.Identifying rows with missing values:Identifying rows with missing values:

- The code df[df. isna(). This (any(axis=1)] is called to demonstrate and show the rows in the DataFrame 'df' that have at least one missing value (NaN).

- df. isna() makes a boolean DataFrame mask of the equal size by rows and colums to 'df', which is where each cell is true if adjacent value in 'df' is missing (NaN) and False otherwise.

- DataFrame, which is looking for rows physically having a cell with all the values being True.

-Consequently, the list of booleans is used to address the initial DataFrame 'df', so that only the rows in which the condition is satisfied are selected.

-At the end, selected rows are printed via print() and expose the rows that display missing values.

2.Removing rows with missing values:Removing rows with missing values:

-The formula "df = df. "dropna() removes all lines of the DataFrame 'df' and corresponding to the rows, which contain at least one null value (NaN).

-Implying the dropna() function to the DataFrame 'df' which will delete any row which bears at least one missing value.

-In the normal scenario, the dropna() function drops rows with any missing values. If you are interested in the package to remove rows only when all the values are absent, you type dropna(how='all').

Now, the updated DataFrame - all rows associated with missing values are removed - is assigned to the variable 'df', therefore replacing the old DataFrame.

3. Displaying the updated DataFrame:

-The code df on a new line just shows the Updated DataFrame 'df' after removing the rows with missing values.

-It will result in a visual inspection of the DataFrame to check if it was completed when the rows with missing values were removed.

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

[158]:
```
# Check for duplicate rows
duplicate_rows = df[df.duplicated()]

if duplicate_rows.empty:
    print("No duplicate rows found.")
else:
    print("Duplicate rows found:")
    display(duplicate_rows)
```

Duplicate rows found:

	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

1171 rows × 9 columns

Figure 8:Checking Duplicate values in data frame

The code above looks for duplicates in the DataFrame 'df' and if there are any found, they will come up.

1. Checking for duplicate rows:

- The code df[df. [df["duplicated"]()] gives the identification and selection of duplicate rows in the DataFrame 'df' through the use of this method.

- The duplicated() function is applied to the DataFrame 'df' and the result is a boolean Series that indicates whether each row is a duplicate of a previous row.

- Behind the scenes, pandas' duplicated() function is set to look for duplicates in all columns by default. If you need to specifically check for duplicates based on some column, then you can use the columns as arguments by writing duplicated(subset=['column1', 'column2']).

- The produced boolean Series is used to index the original DataFrame 'df', and the chosen rows are the ones that are duplicates.

- A variable, duplicate_rows, is defined and holds the records that are duplicates.

2. Checking if duplicate rows exist:

- Code inside the "if" statement checks whether the empty_rows DataFrame is empty or not.

- The empty attribute of a DataFrame returns True if the DataFrame is empty and False otherwise.

- If duplicate_rows. and ID == False, it implies that sets of rows have no duplicates in the DataFrame “df”. Here, this is what the generated code displays: "No duplicate rows found. "

- If duplicate_rows. empty is False, it indicates that the records were found to be the same in the DataFrame 'df'. Here, the code will subtract the subtotal from the sales tax percentage and return the result.

3. Displaying duplicate rows:

- If duplicate rows are found, the code prints the message "Duplicate rows found:" as a seeming way to see that duplicates were found in the DataFrame.

- The display() function is then used to display the duplicate_rows DataFrame, which is the one that has the duplicate rows.

- The display() function has a more visual effect than a simple print()one, especially with DataFrames on .

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

```
[160]: # Iterate over each column in the DataFrame
for column in df.columns:
    # Get the unique values for the current column
    unique_values = df[column].unique()

    # Print the column name and its unique values
    print(f"Unique values in '{column}' column:")
    print(unique_values)
    print()
```

Figure 9:Finding Unique values from all columns

The above figure shows there over loops each column in the DataFrame 'df' and relays the unique values present in each column. Let's break it down step by step:Let's break it down step by step:

1. Iterating over columns:

- The code uses a for loop to get through each column of the DataFrame 'df'.

- The df. columns attributes has the capability to return a column of an index object including the column names of a Dataframe.

- In every iteration of these famous for loops, we assign to the variable column the name of the current one.

2. Getting unique values for each column:2. Getting unique values for each column:

- The loop body accesses the unique values for the current column using df[column]. unique().

- A unique() function is applied using data in Series df[column] which is data in the current column.

- The unique() function returns a NumPy array, holding unique values of every duplicated column, eliminating duplicates.

- The unique values are placed in the unique_values variable.

3. Printing the column name and its unique values:3. Printing the column name and its unique values:

- First, the code is printing the column name and then its unique values using the print (. . .) statements.

- First line (of the code) prints column name by f-string. The price of gasping the f". . . " term is the ability to insert the variable values directly in the string using curly braces {}.

- The second print() statement yields the unique_values array, which is a list of the unique values for the current column.

- Next, printing the unique values and then a print() statement without anything inside is being used to insert a new line acting as a separator in the output.

```
Unique values in 'work_year' column:
[2023 2022 2020 2021]

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

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

Unique values in 'job_title' column:
['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'
 'BI Data Analyst' 'Head of Data Science' 'Insight Analyst'
 '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']
```

Figure 10: Unique values in Column 'work_year','experience_level','employment_type','job_title'

```
 'Principal Data Engineer' 'Staff Data Scientist' 'Finance Data Analyst']

Unique values in 'salary_in_usd' column:
[ 85847  30000  25500 ...  28369 412000  94665]

Unique values in 'employee_residence' column:
['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 'remote_ratio' column:
[100   0  50]

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

Figure 11:Unique values in Column'slaary_in_usd', 'employee_residence','company_size'','remote_ratio','company_location'

2.6 Rename the experience level columns as below.

SE – Senior Level/Expert

MI – Medium Level/Intermediate

EN – Entry Level

EX – Executive Level

	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 12:Data Frame before Renaming

```
[122]:
# Read the data from CSV
df = pd.read_csv('Salary.csv')

# Define mapping for renaming experience levels
experience_level_mapping = {
    'SE': 'Senior Level/Expert',
    'MI': 'Medium Level/Intermediate',
    'EN': 'Entry Level',
    'EX': 'Executive Level'
}

# Rename experience level values
df['experience_level'] = df['experience_level'].replace(experience_level_mapping)

# Display the updated DataFrame
display(df)
```

Figure 13:Renaming

The code is reading the CSV file called 'Salary'. after the output of the command just_read: 'csv', the values in the 'experience_level' column are renamed using an individual mapping and then the updated DataFrame is displayed. Let's go through each part in detail:Let's go through each part in detail:

1. Reading the data from CSV:

- The code creates file called `Salary. csv` and uses pd. read_csv() function to read it. I will be using the 'csv' and store its contents in a DataFrame called 'df'.

- The 'Salary. '/data/input. csv' file is assumed to be in the directory which is same as the python script.

- Pandas ability to take head(1) of csv file and use it to create the DataFrame Columns works very well to save the developed time.

2. Defining the mapping for renaming experience levels:

- The code initiates a dictionary named experience_level_mapping that links the shortened experience level codes to their respective full descriptions.

- The dictionary is represented in key-value pairs format, where the keys are the shortened codes ('SE', 'MI', 'EN', 'EX') and their corresponding descriptions ('Senior Level/Expert', 'Medium Level/Intermediate', 'Entry Level', 'Executive Level') are their values.

- The way the mapping is used as replacement for the abbreviated codes via full descriptions in 'experience_level' column of the DataFrame.

3. Renaming experience level values:

- The code is using the replace() function to replace the values in the 'experience_level' column of the DataFrame 'df' according to the determined mapping.

- The df['experience_level'] syntax serves its purpose; it gives the DataFrame's 'experience_level' column.

- replace() function calls will be done on the `experience_level` column and mapping dictionary will be passed as argument.

- Each value in the 'experience_level' column is checked for the corresponding key in the experience_level_mapping dictionary by the replace() function. The substitute codes are accessed through any match and put back into the DataFrame in place of the previous code.

- As a result of the updated values, the 'experience_level' table in the DataFrame, uses the assigned updated values.

4. Displaying the updated DataFrame:

- The code displays the updated DataFrame 'df' after renaming the experience level values by using the display() function.

- It further produces the result in an up to the mark manner as compared to the standard print() function, especially for the purpose of use in a Jupyter Notebook or the same kind of environment to which the output happens to be connected.

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

3755 rows × 11 columns

Figure 14:Dataframe after renaming

3. Data Analysis

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

```
# Read the data from CS
with open('Salary.csv', 'r') as file:
    data = file.readlines()

# Extract the header row
header = data[0].strip().split(',')

# Find the index of the chosen variable
chosen_variable = 'salary_in_usd'
variable_index = header.index(chosen_variable)

# Initialize variables for summary statistics
total = 0
values = []

# Iterate over the data rows
for row in data[1:]:
    values.append(float(row.strip().split(',')[variable_index]))
```

Figure 15:loading csv file

This code opens a CSV file named 'Salary. csv', extracts values of a selected variable (which is indicated by chosen_variable) from every row and stores them asfloat in a list which is named values. By doing so, it will first open the file, read the contents of it, extract the header row, search for the index of the chosen variable then iterate over the data rows so that the values of the specified variable from the data rows will be appended to the values list.

```
# Calculate summary statistics
n = len(values)
total = sum(values)
mean = total / n

squared_diffs = [(value - mean) ** 2 for value in values]
variance = sum(squared_diffs) / (n - 1)
std_dev = variance ** 0.5
```

Figure 16:Calculating summary

The code calculates statistical summaries of a set of numeric values which are stored in the values list. It calculates three values of it: n to understand number of values, total to observe the sum of all values and mean (average) by dividing the total by n. Squared differences are calculated between each value and mean using list comprehension and are stored in the squared_diffs. The variance is obtained by summing the squares of the differences and dividing by (n - 1). The last step in the process is the computation of standard deviation which involves taking the square root of the variance. These statistics give the mean and dispersion measures for the values in the list expressed as variance and standard deviation.

```
# Calculate skewness
numerator = sum([(value - mean) ** 3 for value in values])
denominator = n * std_dev ** 3
skewness = numerator / denominator
```

Figure 17;Calculating Skewness

The code calculates the skewness of the values into the values list. Skew is a measure for the distribution of values and their asymmetry. The numerator is equal to the sum of the cubed differences between each value and the mean, which is calculated using a list comprehension. The denominator is the number ` (` n``, std_dev, and the standard deviation `(` std_dev^+`)),` where `(` the value of `std_dev^+` is `(` the cube of the standard deviation). Skewness is the ratio of numerator to denominator and is then computed. A positive skewness is the longer right tail of the distribution, while a negative skewness is the longer left tail.

```
# Calculate kurtosis
numerator = sum([(value - mean) ** 4 for value in values])
denominator = n * std_dev ** 4
kurtosis = (numerator / denominator) - 3
```

Figure 18:Calculating kurtosis

The code uses the method to get the kurtosis of the values list values, the dissonant is the product of the two factors: the count of the values (`n`) and the fourth power of the standard deviation (`std_dev`). Kurtosis is then computed by the first part which is the

numerator and the second part which is the denominator and subtracting 3 from the result. A kurtosis that is higher than 1 is called positive and has a similar shape as the distribution of the figure above. On the other hand, if the result would be lower than 1, this is considered a negative kurtosis whose shape is more like that of the normal distribution.

```
# Print the summary statistics
print(f"Summary Statistics for '{chosen_variable}':")
print(f"Sum: {total}")
print(f"Mean: {mean}")
print(f"Standard Deviation: {std_dev}")
print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurtosis}")
```

Figure 19:Printing the values

The code displays the summary statistics for the specified variable (`chosen_variable`) using the formatted string literals (f-strings). There displays the name of the chosen variable along with a table of statistics that have been calculated. The total represents the sum of the values, mean is being computed as average, std_dev is the standard deviation, skewness is responsible for the asymmetricity of a distribution, and kurtosis explains peakedness or flatness. The data are labeled with descriptive names to make it easy for the reader to grasp the features of the variable.

```
Summary Statistics for 'salary_in_usd':
Sum: 516576814.0
Mean: 137570.38988015978
Standard Deviation: 63055.625278224084
Skewness: 0.5359726925230319
Kurtosis: 0.8292585346115984
```

Figure 20:Calculated statistics

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

```
import csv
import pandas as pd

# File path for the CSV
file_path = 'Salary.csv'

# Read the data from the CSV file
data = []
with open(file_path, 'r') as file:
    csv_reader = csv.reader(file)
    headers = next(csv_reader)
    for row in csv_reader:
        data.append(row)
```

Figure 21:Giving file path

This code is an extractor of data from a CSV file called 'Salary. 'generate a csv file using the built-in "csv" module and the pandas library. Initially, it defines the file path to load the CSV data. Next, it reads the file in read mode by using with and the open() function. It creates a csv. reader might oppose such a move and that might not work. After the next() function was written to skip the header row and put the headers in the headers variable. In the last paragraph, the author links each row of the CSV file to the data list using a for loop. This is the end result for the data list that took all the rows from the CSV file while discarding the original header row.

```
# Identify numeric columns
numeric_columns = []
for i in range(len(headers)):
    if all(row[i].replace('.', '', 1).isdigit() for row in data if row[i]):
        numeric_columns.append(i)
```

Figure 22:Identifying numeric columns

This line of code locates the data containing numbers. It goes through each column index using the for loop and the range() function. For each column, it uses a list comprehension and the isdigit() function to scan all the values of that column for being numeric. The := method, is used in the handling of decimal points by removing the first occurrence of a dot (‘. 'Please ask. ' before you ask if it is a digit. If all the values in a column other than zero are numeric, then that column index number is appended to the

numeric_columns list. Following the loop, list_columns has the indices of the columns that consists of numerical values only.

```
# Calculate the correlation matrix for numeric columns only
num_columns = len(numeric_columns)
correlation_matrix = [[None] * num_columns for _ in range(num_columns)]
for i in range(num_columns):
    for j in range(i, num_columns):
        if i == j:
            correlation_matrix[i][j] = 1.0
        else:
            x = []
            y = []
            for row in data:
                if row[numeric_columns[i]].replace('.', '', 1).isdigit() and row[numeric_columns[j]].replace('.', '', 1).isdigit():
                    x.append(float(row[numeric_columns[i]]))
                    y.append(float(row[numeric_columns[j]]))
            if len(x) > 0 and len(y) > 0:
                n = len(x)
                mean_x = sum(x) / n
                mean_y = sum(y) / n
                std_dev_x = (sum((value - mean_x) ** 2 for value in x) / (n - 1)) ** 0.5
                std_dev_y = (sum((value - mean_y) ** 2 for value in y) / (n - 1)) ** 0.5
                covariance = sum((x[k] - mean_x) * (y[k] - mean_y) for k in range(n)) / (n - 1)
                correlation = covariance / (std_dev_x * std_dev_y)
                correlation_matrix[i][j] = correlation
                correlation_matrix[j][i] = correlation
```

Figure 23:Calculating correlation matrix

Thus it is the code which shows the correlation matrix for the columns which are found numeric using the previous step. It creates datalist named correlation_matrix of size equal to number of numeric columns. After that the code is realized in nested loops that will iterate over the every pair of numeric columns. This matrix allows the algorithm to check whether the column indices are the same in both matrices (which is the case for diagonal elements) and to set the correlation to 1. 0. It gets values of non-diagonal elements from the data list, however, it ignores rows where there are items, which are not numeric. It is responsible for computing the mean, standard deviation, and covariance for each pair of columns using the given values. Next, the function draws on the covariance computed in order to calculate the correlation coefficient divided by the product of the standard deviations. This result is then saved into the correlation_matrix. The code fills both the upper and lower triangular parts because symmetric, but you can also see that the code is a triangular matrix.

```
# Create a DataFrame from the correlation matrix for numeric columns only
numeric_headers = [headers[i] for i in numeric_columns]
df_correlation = pd.DataFrame(correlation_matrix, columns=numeric_headers, index=numeric_headers)

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

Figure 24:Printing the answer

Here, I have created a pandas dataframe called df_correlation which holds the correlation matrix calculated in the previous step. It calls header_extractor function using list comprehension and the numeric_columns list and takes out the corresponding headers for the numeric columns. The pd. DataFrame() function is used to construct the DataFrame, with the correlation_matrix as the data, the numeric_headers as the column names, and the numeric_headers as the row index. Lastly, it print a message which is says that below output is a correlation matrix and display the df_correlation DataFrame which is a tabular format containing correlation coefficients between each pair of numeric columns.

```
Correlation Matrix:
```

]:

	work_year	salary	salary_in_usd	remote_ratio
work_year	1.000000	-0.094724	0.228290	-0.236430
salary	-0.094724	1.000000	-0.023676	0.028731
salary_in_usd	0.228290	-0.023676	1.000000	-0.064171
remote_ratio	-0.236430	0.028731	-0.064171	1.000000

Figure 25:Output

4. Data Exploration

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

```
[90]: import pandas as pd
      import matplotlib.pyplot as plt

      # Read the data from CSV
      df = pd.read_csv('Salary.csv')

      # Get the top 15 jobs
      top_jobs = df['job_title'].value_counts().head(15)

      # Define color palette (you can choose any other color palette)
      colors = plt.cm.tab10(range(len(top_jobs)))

      # Plotting the bar graph
      plt.figure(figsize=(10, 6))
      bars = top_jobs.plot(kind='bar', color=colors)
      plt.title('Top 15 Jobs')
      plt.xlabel('Job Title')
      plt.ylabel('Frequency')
      plt.xticks(rotation=45, ha='right')
      plt.tight_layout()

      # Add accurate values to the bars
      for bar in bars.patches:
          plt.text(bar.get_x() + bar.get_width() / 2,  # x-coordinate
                   bar.get_height() + 0.05,            # y-coordinate (adding a small offset)
                   f'{bar.get_height():.0f}',         # formatted text (without decimal)
                   ha='center', va='bottom',           # alignment
                   fontsize=8)                         # fontsize for the text

      plt.show()
```

Figure 26:Top 15 job calculation in graphThe enclosed script is written in Python and utilizes the libraries of Pandas and Matplotlib to graphically visualize the chart of top 15 job titles from the named CSV file 'Salary. csv'.

The provided code is written in Python and utilizes the Pandas and Matplotlib libraries to generate a bar chart visualization of the top 15 most frequent job titles from a CSV file named "Salary.csv".

The code first imports the necessary libraries: Pandas for data manipulation and Matplotlib for data visualization. Then, it reads the data from the "Salary.csv" file using the `pd.read_csv` function and stores it in a Pandas DataFrame named `df`.

Next, it retrieves the top 15 most frequent job titles from the 'job_title' column of the DataFrame using the `value_counts()` method and stores them in the `top_jobs` variable.

The code then defines a color palette using the `plt.cm.tab10` colormap from Matplotlib to assign different colors to each bar in the chart.

The main part of the code sets up the bar chart visualization. It creates a figure with a size of 10 by 6 inches and plots the `top_jobs` data using the `plot` method from

Pandas, specifying the type of plot as a bar chart (`kind='bar'`). The chart is given a title "Top 15 Jobs", and labels for the x-axis ("Job Title") and y-axis ("Frequency") are set.

The `plt.xticks` function is used to rotate the x-axis labels by 45 degrees and align them to the right for better readability. The `plt.tight_layout()` function is called to ensure that the labels and title fit properly within the figure.

Finally, a loop iterates over each bar in the chart and adds the corresponding frequency value as text above the bar. The text is positioned using the `get_x()` and `get_height()` methods of each bar patch, with a small offset to prevent overlapping with the bar itself. The text is formatted using the `f-string` notation to display only the integer part of the frequency value.

After all the customizations, the `plt.show()` function is called to display the final bar chart visualization in a separate window.

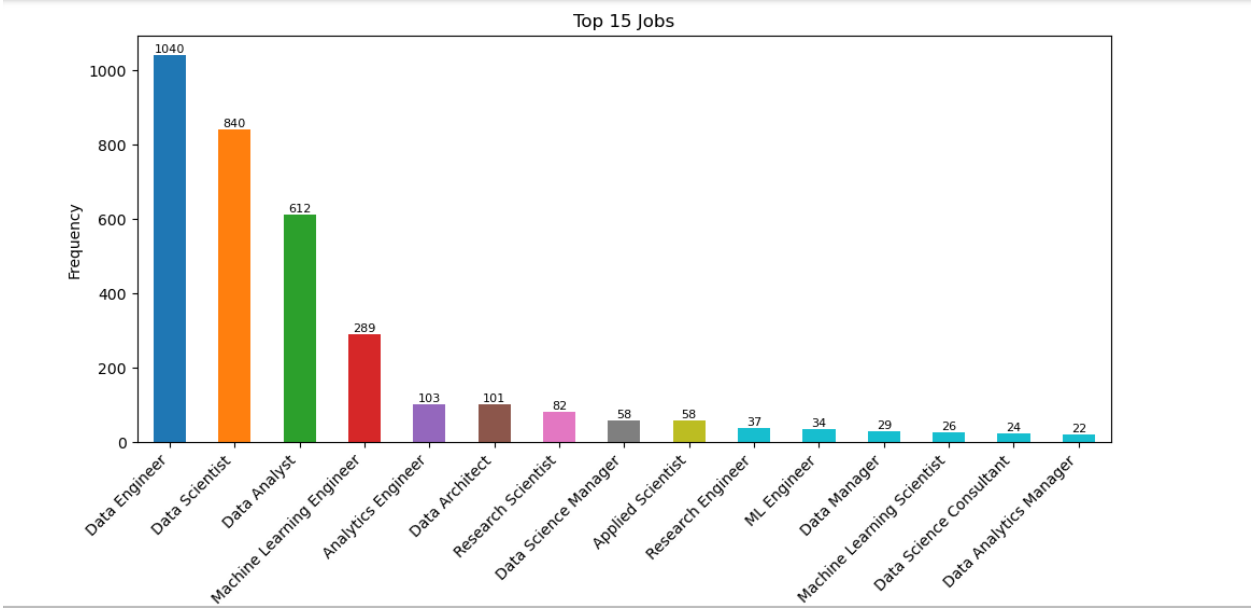


Figure 27:Top 15 jobs in graph

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

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the data from CSV
df = pd.read_csv('Salary.csv')

# Calculate the average salary for each job title
average_salary_by_job = df.groupby('job_title')['salary_in_usd'].mean().sort_values(ascending=False)

# Get the job with the highest average salary
highest_salary_job = average_salary_by_job.idxmax()
highest_salary = average_salary_by_job.max()

# Define color palette
colors = plt.cm.tab10(range(len(average_salary_by_job)))

# Plotting the horizontal bar graph
plt.figure(figsize=(12, 20))  # Adjusted figure size for better layout
bars = average_salary_by_job.plot(kind='barh', color=colors)  # Using barh for horizontal bar plot
plt.title('Average Salary by Job Title', fontsize=16)  # Increased title font size
plt.xlabel('Average Salary (USD)', fontsize=12)  # Adjusted x-axis label font size
plt.ylabel('Job Title', fontsize=12)  # Adjusted y-axis label font size
plt.axvline(x=highest_salary, color='red', linestyle='--', label=f'Highest Salary: {highest_salary_job} (${highest_salary:.2f})')
plt.legend(fontsize=10, loc='upper right')  # Adjusted legend font size and position
plt.subplots_adjust(left=0.25)  # Adjusted spacing between bars and y-axis labels

# Add accurate values to the bars
for bar in bars.patches:
    plt.text(bar.get_width(), bar.get_y() + bar.get_height() / 2,
             f'{bar.get_width():,.0f}', ha='left', va='center', fontsize=10)

plt.tight_layout()  # Adjust layout to prevent overlapping
plt.show()
```

Figure 28:Calculating job with highest salery

The provided code is written in Python and utilizes the Pandas and Matplotlib libraries to generate a horizontal bar chart visualization of the average salary for each job title from a CSV file named "Salary.csv".

The code begins by importing the necessary libraries and reading the data from the "Salary.csv" file into a Pandas DataFrame named `df`. It then calculates the average salary for each job title by grouping the data by the 'job_title' column and taking the mean of the 'salary_in_usd' column. The resulting series is sorted in descending order and stored in the `average_salary_by_job` variable.

The code identifies the job title with the highest average salary by using the `idxmax()` method on the `average_salary_by_job` series and stores the job title and the corresponding salary value in the `highest_salary_job` and `highest_salary` variables, respectively.

Next, the code defines a color palette using the `plt.cm.tab10` colormap from Matplotlib to assign different colors to each bar in the chart.

The main part of the code sets up the horizontal bar chart visualization. It creates a figure with a size of 12 by 20 inches and plots the `average_salary_by_job` data using the `plot` method from Pandas, specifying the type of plot as a horizontal bar chart (`kind='barh'`). The chart is given a title "Average Salary by Job Title", and labels for the x-axis ("Average Salary (USD)") and y-axis ("Job Title") are set with adjusted font sizes.

The `plt.axvline` function is used to draw a vertical dashed red line at the position of the highest average salary value. A legend is added to identify this line, displaying the job title and the corresponding highest salary value.

The `plt.subplots_adjust` function is called to adjust the spacing between the bars and the y-axis labels, ensuring better readability.

A loop iterates over each bar in the chart and adds the corresponding average salary value as text next to the bar. The text is positioned using the `get_width()` and `get_y()` methods of each bar patch, with the text aligned to the left and vertically centered. The salary values are formatted using the `,` thousands separator for better readability.

Finally, the `plt.tight_layout()` function is called to ensure that the labels, title, and legend fit properly within the figure, and the `plt.show()` function is called to display the final horizontal bar chart visualization in a separate window.

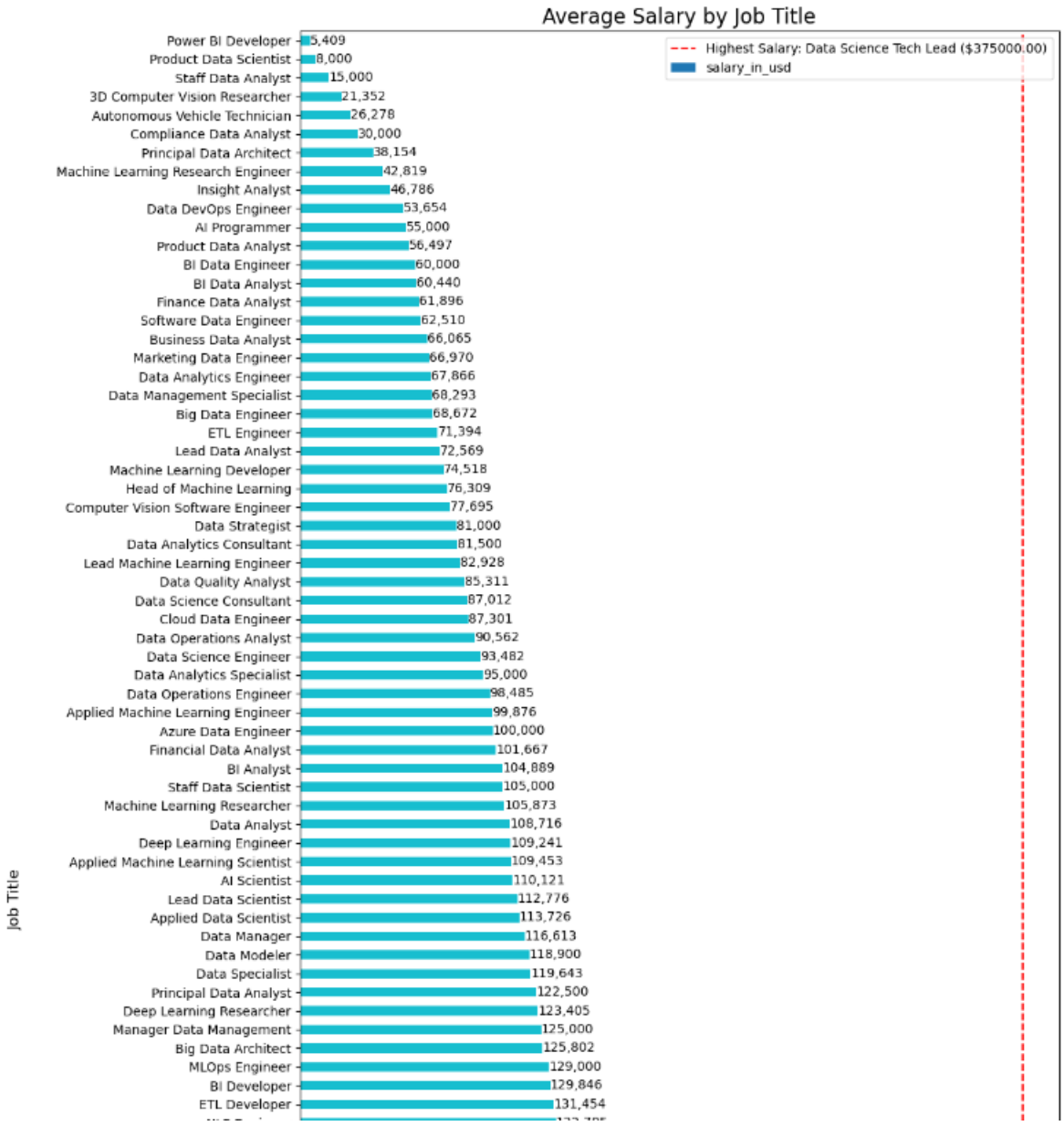


Figure 29:Graph of jobs and their respective slaries

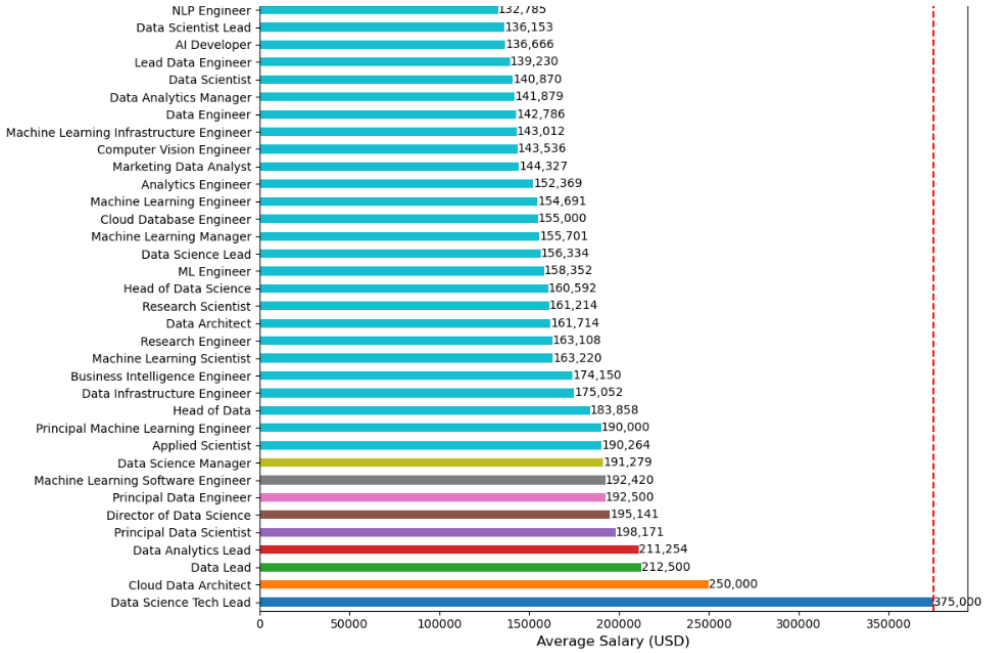


Figure 30:Job with maximum salary

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

```
import matplotlib.pyplot as plt
import csv

# Read the CSV file into a list of rows
file_path = "Salary.csv"
with open(file_path, "r") as file:
    csv_reader = csv.reader(file)
    rows = list(csv_reader)

# Get the index of the "experience_level" and "salary_in_usd" columns
experience_level_index = rows[0].index("experience_level")
salary_index = rows[0].index("salary_in_usd")

# Create a dictionary to store experience levels and their corresponding salaries
experience_salaries = {}

# Iterate over each row (excluding the header row)
for row in rows[1:]:
    experience_level = row[experience_level_index]
    salary = float(row[salary_index])
    if experience_level in experience_salaries:
        experience_salaries[experience_level].append(salary)
    else:
        experience_salaries[experience_level] = [salary]

# Calculate the mean salary for each experience level
mean_salaries = {level: sum(salaries) / len(salaries) for level, salaries in experience_salaries.items()}

# Set custom color palette
color = '#1f77b4'

# Create a bar graph of the mean salaries for each experience level
plt.figure(figsize=(10, 8))
bars = plt.bar(mean_salaries.keys(), mean_salaries.values(), color=color, edgecolor='black', linewidth=1.2, alpha=0.8)
plt.xlabel('Experience Level', fontsize=14, labelpad=10)
plt.ylabel('Mean Salary (USD)', fontsize=14, labelpad=10)
plt.title('Mean Salaries by Experience Level', fontsize=18, pad=20)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()

# Add labels to the bars
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, f"${height:.2f}", ha='center', va='bottom', fontsize=12)

# Add a legend
experience_levels = list(mean_salaries.keys())
legend_labels = [f"{level} (${salary:.2f})" for level, salary in mean_salaries.items()]
plt.legend(bars, legend_labels, title='Experience Levels', title_fontsize=12, fontsize=10, loc='upper right')

plt.show()
```

Figure 31:Calculation for salary based on experience level

The provided code is written in Python and utilizes the Matplotlib library for data visualization and the built-in csv module for reading data from a CSV file. The code generates a bar chart that displays the mean salaries for different experience levels present in the dataset.

The code starts by reading the data from the "Salary.csv" file using the `csv.reader` function and storing the rows in a list. It then finds the indices of the "experience_level" and "salary_in_usd" columns in the header row.

Next, the code creates a dictionary called `experience_salaries` to store the salaries for each experience level. It iterates over each row (excluding the header row) and updates the `experience_salaries` dictionary with the corresponding experience level and salary.

After collecting the salaries for each experience level, the code calculates the mean salary for each experience level and stores the results in the `mean_salaries` dictionary.

The code then sets up the visualization by creating a figure with a specific size and a custom color palette for the bars. It generates a bar chart using the `plt.bar` function, where the x-axis represents the experience levels, and the y-axis represents the mean salaries. The chart is labeled with appropriate titles and axis labels, and the x-axis tick labels are rotated for better readability.

The code then adds labels to the individual bars, displaying the mean salary value on top of each bar. It also creates a legend displaying the experience levels and their corresponding mean salaries.

Finally, the code adds grid lines to the y-axis for better visualization of the salary values and calls the `plt.show()` function to display the final bar chart.

Overall, this code demonstrates how to read data from a CSV file, process the data to calculate mean values grouped by a specific column, and visualize the results using a bar chart with appropriate formatting and labeling.

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

```
data = pd.read_csv(file_path)

# Set custom color palette
colors_histogram = ['#1f77b4', '#aec7e8']  # Two shades for the histogram
color_boxplot = '#1f77b4'  # Single color for boxplot

# Create a histogram for the 'salary_in_usd' column
plt.figure(figsize=(10, 8))
n, bins, patches = plt.hist(data["salary_in_usd"], bins=20, edgecolor='black', linewidth=1.2, alpha=0.8)
for i, patch in enumerate(patches):
    patch.set_facecolor(colors_histogram[i % len(colors_histogram)])
    value = int(n[i])
    plt.text(patch.get_x() + patch.get_width() / 2, patch.get_height(), str(value), ha='center', va='bottom', fontsize=10)
plt.title("Histogram of Salaries (in USD)", fontsize=18)
plt.xlabel("Salary (in USD)", fontsize=14)
plt.ylabel("Count", fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()

# Create a box plot for the 'salary_in_usd' column grouped by 'company_size'
plt.figure(figsize=(10, 8))
company_sizes = data["company_size"].unique()
box_data = [data[data["company_size"] == size]["salary_in_usd"] for size in company_sizes]
bplot = plt.boxplot(box_data, patch_artist=True)
for patch in bplot['boxes']:
    patch.set_facecolor(color_boxplot)
plt.title("Box Plot of Salaries (in USD) by Company Size", fontsize=18)
plt.xlabel("Company Size", fontsize=14)
plt.ylabel("Salary (in USD)", fontsize=14)
plt.xticks(range(1, len(company_sizes) + 1), company_sizes, rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```

Figure 32:creating histogram and box plot of any chosen different variables calculation

The provided code is written in Python and utilizes the Pandas and Matplotlib libraries to generate two visualizations: a histogram and a box plot, both related to the salary data from a CSV file.

The code starts by reading the data from the CSV file using `pd.read_csv` and storing it in the `data` DataFrame.

For the histogram visualization, the code sets up a custom color palette with two shades of blue (`colors_histogram`). It then creates a figure with a specified size and plots a histogram of the 'salary_in_usd' column using `plt.hist`. The histogram is divided into 20 bins, and the bars are styled with black edges, a line width of 1.2, and an alpha value of 0.8 for transparency.

The code then loops through each patch (bar) in the histogram and alternates between the two shades of blue for the bar colors. It also adds the count value (number of data points in each bin) as text on top of each bar.

The histogram is labeled with a title ("Histogram of Salaries (in USD)"), x-axis label ("Salary (in USD)"), and y-axis label ("Count"). The tick labels are formatted with larger font sizes for better readability, and a grid is added to the y-axis for improved visualization.

For the box plot visualization, the code sets up a single shade of blue (`color_boxplot`) for the box colors. It creates another figure with a specified size and generates a box plot of the 'salary_in_usd' column grouped by the 'company_size' column using `plt.boxplot`.

The code first gets the unique company sizes from the data, and then it creates a list of salary data for each company size. The box plot is then plotted using this list of data, and the box colors are set to the specified shade of blue.

The box plot is labeled with a title ("Box Plot of Salaries (in USD) by Company Size"), x-axis label ("Company Size"), and y-axis label ("Salary (in USD)"). The x-axis tick labels are the unique company sizes, rotated by 45 degrees for better readability. The tick labels are formatted with larger font sizes, and a grid is added to the y-axis for improved visualization.

Finally, both visualizations are displayed using `plt.show()`.

Overall, this code demonstrates how to create a histogram and a box plot using the Matplotlib library, with customizations for colors, labels, grid lines, and tick label formatting to enhance the visualizations.

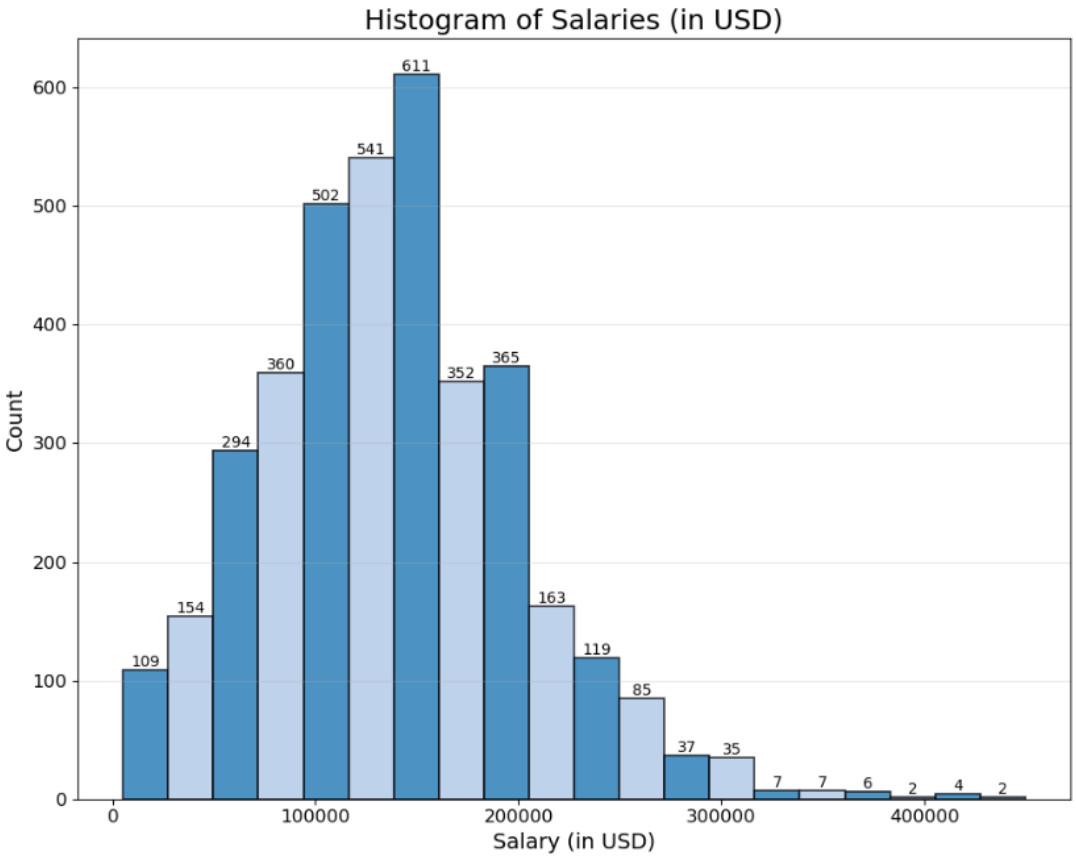


Figure 33:Histogram of salaries

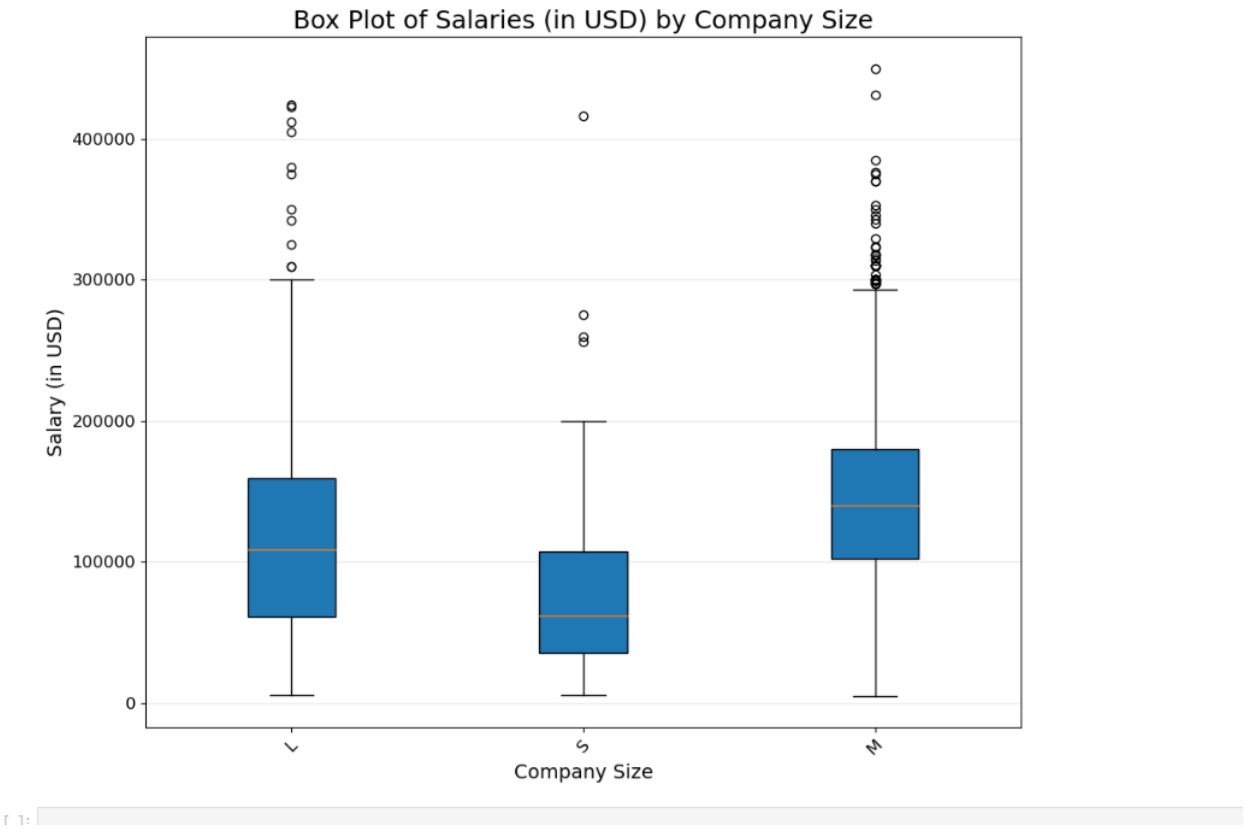


Figure 34:BoxPLot of salaries

5 References

Jaiswal, S. (n.d.). *datacamp*. Retrieved from www.datacamp.com: https://www.datacamp.com/tutorial/data-preparation-with-pandas

Maheshwari, H. (2021, may 21). *towardsdatascience*. Retrieved from towardsdatascience.com: https://towardsdatascience.com/understanding-and-interpreting-data-in-python-db67b11865a0

Nelamali, N. (2024, march 24). *parkbyexamples*. Retrieved from parkbyexamples.com: https://sparkbyexamples.com/pandas/pandas-what-is-dataframe-explained/