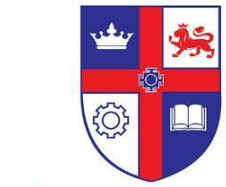
LONDON METROPOLITAN UNIVERSITY



Islington college
(इस्लिङ्टन कलेज)

CC5067NI-Smart Data Discovery

60% Individual Coursework

2023-24 Spring

Student Name: Dikshya Sharma

London Met ID: 22067520

College ID: NP01CP4a220029

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.

Acknowledgement

I would like to take this opportunity to express my sincere gratitude and appreciation to everyone who helped in completion of this coursework. First and foremost, I am so grateful for my module leader Mr. Dipeshwor Silwal, my tutor Mr. Alish Kc, and all my respected teachers for their continuous guidance and all helpful advice and assistance they gave while working on the coursework. Their expertise, feedback and constant encouragement were immensely beneficial. In order for me to complete this coursework on time, my lecturers and tutors were of the greatest possible help. Whether it was in person or online through google hangouts, they were always willing to assist. Furthermore, I am thankful to the university and all the members of Islington College for assigning this coursework. I can undoubtedly say that the abilities and knowledge that I learned from this coursework will be beneficial to me in the days to come. Finally, I extend my gratitude to all my friends for helping me overcome the obstacles with their skills. I greatly appreciate it. Thank you all so much.

Abstract

This coursework aims to provide the fundamental knowledge and ability to use Python for data analysis, with an emphasis on a thorough examination of data science salary. The primary objective of this coursework is to delve into the numerous factors influencing salary levels within the field and to meticulously prepare the dataset for in-depth analysis and mining. Tools like pandas and matplotlib are employed to analyse data and create visual representations which further include understanding the dataset, cleaning up messy data, identifying trends through statistical analysis, and presenting findings clearly and organized.

Table of Contents

Table of Figures

Table of Tables

1. INTRODUCTION

In the ever-growing field of data science, understanding the wide range of factors influencing professionals' journeys is not only useful but also crucial for both individuals and organizations seeking success in this ever-changing industry. This coursework goes into the ***analysis of data science salaries***, with the goal of providing insights into the factors that influence salary levels. Through Python programming and data analysis techniques, a journey to explore, prepare, and analyze a dataset containing information on various factors affecting data science salaries was embarked.

Data science is all about implementing modern techniques and tools to analyze massive amounts of data which assists businesses in identifying patterns, extracting usable information, and making informed decisions. Data science predicts the future using complex algorithms that are essential for businesses seeking to increase efficiency, promote innovation, and adapt to market changes. It identifies hidden patterns, predicts outcomes, and personalizes client interactions, resulting in enhanced profitability and growth (Biswal, 2024).

This document outlines a significant journey—a 60% weighted assignment—focused on analyzing data science salary while demonstrating technical skills and critical thinking. The dataset contains critical insights into salary criteria such as experience and job titles, and the goal is to uncover trends and factors that influence data scientist salaries. The coursework is divided into five distinct parts: Data Understanding, Data Preparation, Data Analysis, Data Exploration, and Document Organization. Throughout, Python serves as a guiding tool, documenting progress with screenshots and insightful comments.

1.1. Aims and Objectives

The primary aim of this course is to develop proficiency in utilizing Python for data analysis and perform a thorough analysis of data science salaries to better understand the numerous elements that influence salary levels and to prepare the data for further mining and analysis.

Below are the objectives of this coursework:

- To gain proficiency in Python libraries such as ***pandas and matplotlib*** for data analysis.
- To analyze and document the given dataset's characteristics to better understand its structure and details.
- To develop Python programs utilizing pandas to load, clean, and prepare data, including handling missing values and removing duplicates to maintain data accuracy and integrity.
- To identify and remove unnecessary columns from the dataset, ensuring data accuracy and relevance.
- To perform statistical analysis, including summary statistics and correlation calculations, to understand data trends and relationships.
- To visualize data exploration findings through bar graphs and other graphical representations.
- To organize the technical report effectively, including a structured report structure and adequate documentation of Python programs with screenshots, results, and user guides.

1.2. Tools Used

The coursework was carried out using a variety of tools and packages. The main programming language was Python, which provided resilience and flexibility for data processing. To further improve the analytical capabilities, a variety of libraries, including **matplotlib** and **pandas**, were used for **data manipulation**. For executing and documenting the code, **Jupyter Notebook or Jupyter Lab**, which are part of the Anaconda distribution. All the insights that were discovered during the investigation were combined and presented in a report that was prepared using **Microsoft Word.**

- **Libraries Used**

- **Pandas**

Pandas is a well-known Python open-source toolkit that is widely used for data science, machine learning, and data analysis tasks. One of Pandas's most well-known data manipulation skills is its ability to interact with many other Python data science modules in an easy-to-use manner (S, 2020). Pandas seamlessly integrates with numerous other Python modules within the data science realm.

- **Matplotlib**

Matplotlib is a cross-platform toolkit for Python and its numeric extension NumPy that facilitates data visualization and graphical plotting (bar charts, scatter plots, histograms, etc.). As such, it provides a strong open-source substitute for MATLAB. Plots can also be included in GUI programs by developers using the matplotlib APIs (Application Programming Interfaces) (M, 2021).

Matplotlib enables users to create a wide range of visualizations to represent data effectively. This includes line plots, bar charts, scatter plots, histograms, heatmaps, and more. These visualizations aid in understanding patterns, trends, and relationships within the data

2. Data Understanding

In the realm of data science, getting a project correctly begins with understanding the problem and getting to know your data thoroughly. Understanding data involves delving into its basic components to get insight into its structure and content. The goals of data understanding include numerous fundamental components. First, it requires understanding the dataset's basic qualities, which include recognizing the numerous variables and their properties. Furthermore, data understanding entails summarizing the dataset by identifying key properties such as data amount and total number of variables included (3 Pillar Global, 2016).

Furthermore, it requires a thorough investigation of potential dataset issues such as missing values, errors, and outliers, all of which might have an impact on the analysis's trustworthiness. Finally, visualization tools help to validate key data properties and identify any anomalies in summary statistics. Finally, visualization tools help to validate key data properties and identify any anomalies in summary statistics. Through these aims, understanding data serves as the foundation for effective analysis and interpretation, enabling informed decision-making based on accurate insights gained from data (3 Pillar Global, 2016).

Furthermore, the dataset is generated from internal corporate records, surveys, and external databases, providing an in-depth analysis of the factors that influence salary in the data science field. It includes variables such as experience level, employment type, and job title, among others, which help to provide a comprehensive understanding of salary factors. This dataset, formatted in CSV format with 11 columns and 3756 rows spanning 2020 to 2023, provides sufficient information for study and is easily accessible. As a result, it is an excellent resource for learning about the various elements that influence employee wages.

The datasets are further explained below:

2.1. Dataset Columns

The CSV file contains 11 columns and 3756 rows of data. These columns include information such as work year, experience level, employment type, job title, salary, salary currency, salary in USD, employee residence, remote ratio, company location, and company size. Each column provides valuable insights into different aspects of employee salaries, ranging from their level of experience and job title to the location and size of their employing company. This comprehensive dataset allows for thorough analysis of factors influencing salary levels within the data science industry, providing a rich source of information for understanding salary dynamics and trends over time.

Below is the further explanation of columns of datasets provided.

a. Work Year:

The column “Work Year” is critical for understanding data science salaries since it represents both the year of collecting data and an individual's career starts in the industry. It is displayed as an integer to represent personal experience. It demonstrates how incomes often improve with more tenure, reflecting professional advancement and experience. Furthermore, "Work Year" is the first column in the CSV file, indicating when employees joined the organization, with years spanning from 2020 to 2023.

b. Experience Level:

The "Experience Level" column categorizes employees according to their level of expertise, which influences their salary. It consists of four levels: Entry Level (EL) for beginners, Mid-level/Intermediate (MI) for professionals, Executive Level (EX) for executives, and Senior level/Expert (SE) for highly experienced professionals. This column, which is commonly recorded as a string object, shows the relationship between experience level and salary in the dataset.

c. Employment Type:

The column “Employment Type” categorizes employees into Full Time (FT), Part Time (PT), and Contractual (CT) types, providing insights into employment arrangements that influence salaries and overall income levels. FT employees work full-time, PT employees work part-time (potentially in multiple companies), and CT employees are contracted for specific tasks.

d. Job Title:

The column “Job Title” provides a comprehensive overview of the various positions or roles held by employees within the data science industry. It encompasses a diverse range of job titles, including but not limited to Data Analyst, Data Scientist, Data Engineer, Machine Learning Engineer, and more. These titles reflect the specialized expertise and responsibilities of individuals within the field. By examining this column, one can gain valuable insights into how salaries are distributed among different job titles, as each role typically commands varying levels of compensation based on its skill requirements and market demand.

e. Salary:

This column records the annual salary received by employees based on their job title, experience level, and tenure within the data science industry. The salaries are stored as integers in the native currency of their respective nations. By analysing this data, one can gain valuable insights into salary distribution across different job titles within the field. Additionally, it facilitates the examination of salary trends, variations based on job title and experience level, and other financial aspects pertinent to the data science Industry.

f. Salary Currency:

The column “Salary Currency” of the CSV file specifies the currency used for distributing salaries across various nations within the data science industry. It contains string data representing different currencies such as USD, GBP, EUR, and others. This information offers a valuable overview of currency diversity, facilitating the analysis and comparison of salaries on an international scale. Understanding currency fluctuations is essential for maintaining data consistency globally, enabling more accurate assessments of salary trends and comparisons across different regions.

g. Salary in USD:

The column "Salary in USD," contains the salaries of employees converted into US Dollars (USD) to ensure standardization and facilitate easy comparison among different nations. By converting salaries into a common currency, this column maintains consistency, enabling straightforward analysis and comparison across various regions. This standardization not only simplifies salary assessment but also enhances the clarity and accuracy of international salary comparisons within the data science industry.

h. Employee Residence:

The column "Employee Residence," specifies the residential location or address of each employee, denoting the country where they reside. This column contains string data representing countries such as US, CA, NIG, and others. By examining this information, one can gain a comprehensive overview of the global distribution of the employee workforce within the data science industry. Moreover, it facilitates the analysis of various factors such as job demand, economic conditions, regional salary discrepancies, employment trends, and workforce demographics worldwide.

i. Remote Ratio:

The ninth column in the CSV file, "Remote Ratio," indicates the proportion of employees engaged in remote work arrangements. This column employs numerical values (0-100%) to denote the extent of remote work, ranging from fully remote to on-site positions. A value of 0 signifies physical presence at the workplace, 50 indicates a hybrid arrangement where employees split time between office and remote work, and 100 represents full remote work capability. By analysing this data, one can glean insights into remote work trends and its correlation with salary levels, shedding light on how work arrangements influence compensation within the data science industry.

j. Company Location:

The column in the CSV file, "Company Location," denotes the geographical location of the company where employees are employed. This column contains string data representing countries such as GB, US, CA, NG, and others. By examining this information, one can gain valuable insights into the geographical distribution of companies within the data science industry. This data enables analysis on various factors such as job opportunities, salary levels, and industry branches across different regions, thereby facilitating a deeper understanding of the global landscape of the data science industry.

k. Company Size:

The "Company Size," categorizes companies into small, medium, and large based on factors such as the number of employees and company revenue. This classification is denoted by string data representing "Large," "Medium," and "Small" sizes. Analysing this column provides insights into the distribution of company sizes within the dataset and can offer valuable information on potential factors influencing salary structures. Understanding the size of companies can shed light on various aspects such as organizational dynamics, growth potential, and workforce demographics within the data science industry.

2.2. Dataset Column Description Table

This table consists of several columns, each of which contains useful information about data science industry employees. Each column is assigned a unique summary and data type to ensure reliable data storage.

S. No	COLUMN NAME	DESCRIPTION	DATA TYPE
1.	work_year	This column lists the year an employee started their professional career with the company.	Integer(int64)
2.	experience_level	This column lists each employee's level of experience for their unique job title.	Object (String)
3.	employment_type	This column lists each individual's job status, including whether they work full-time or on a contract basis.	Object (String)
4.	job_title	This column indicates each employee's precise role or position in the data science industry.	Object (String)
5.	salary	This column lists each employee's salary in the currency of their respective nations.	Integer(int64)
6.	salary_currency	This column lists the currency used to specify the salary.	Object (String)

7.	salary_in_usd	This column lists salaries in USD to preserve uniformity and make comparisons easier.	Integer(int64)
8.	employee_residence	This column lists each employee's address or residing location.	Object (String)
9.	remote_ratio	This column lists the percentage of each employee that works remotely. .	Integer(int64)
10.	company_location	This column lists the geographical location of the company where employees are employed.	Object (String)
11.	company_size	This column lists the company size into three categories: small, medium, and giant.	Object (String)

Table 1: Dataset Column Description Table

2.3. Outline of Data Types

Data is usually classified into two types: quantitative and qualitative. Quantitative data has measurable values, but qualitative data contains descriptive language and categorizations that are not immediately measurable. Qualitative data is collected through various feedback channels such as comments, reviews, surveys, and consultations. Quantitative data is essential for statistical analysis and trend predictions. These data kinds work together to support informed business decisions and strategic planning (Rudderstack, 2024).

The datasets offered contain a wide range of data kinds, each with a distinct purpose in understanding data science salary. For example, columns such as *"salary," "salary_in_usd,"* and *"remote_ratio"* have numerical data types, allowing for quantitative examination of salary amounts. These columns allow for statistical computations and comparisons of salaries across several categories. Columns like *"experience_level," "employment_type," "job_title," "salary_currency," "employee_residence," "company_location," and "company_size"* contain character data (strings) that convey qualitative information. They organize data into groups, with "*experience_level," "employment_type," and "job_title"* clearly identifying various groups. Furthermore, the *"work_year"* column contains dates, providing an arranged context.

2.4. Data Quality

Data quality is defined as the level to which data meets a company's requirements for accuracy, validity, completeness, and consistency. It is an important part of data management to ensure that the information utilized for analysis, reporting, and decision-making is reliable and trustworthy (Suer, 2023).

The dataset provided has high data quality, with no missing or null values in any of its columns. This completeness ensures that the dataset is ready for future analysis without the need for data modification, hence increasing its dependability and correctness. The lack of missing values reduces the possibility of errors and biased results, increasing the dataset's dependability for research purposes. But there are cases of duplicate job titles with similar functions and kinds, resulting in data inconsistency. Addressing this issue involves removing such duplicates while importing the data into the Data Frame. Furthermore, deleting duplicate values helps to improve analysis results by minimizing memory usage and increasing the efficiency of statistical calculations.

In summary, maintaining data consistency simplifies preprocessing processes, allowing for more effective data exploration and analysis. Finally, the dataset maintains data integrity, allowing for efficient processing, analysis, and exploration, resulting in useful insights for informed decision-making.

2.5. Data Insights

Data insights are significant objects of information obtained through comprehensive data analysis. These insights serve as the foundation for making intelligent choices by revealing patterns and complexities in a dataset (James, 2024).

The dataset provides significant insights on the factors that influence data science salary. Among these factors, *"work_year"* records salary trends over time, but *"experience_level"* describes how individual talents influence salary, with more experienced professionals typically receiving higher incomes. The variable "*job_title"* indicates various salaries based on job responsibilities and required skills. In particular, blunders in this column, such as the overlapping between "*ML Engineer" and "Machine Learning Engineer,"* will be corrected to maintain data clarity and consistency.

Moreover, the *"employment_type"* factor plays a role in shaping salary structures, as full-time employees generally receive higher wages compared to their part-time or contractual counterparts. Additionally, variations in "*salary_currency"* based on *"residence_area"* require conversion to US dollars (USD) to ensure standardized data for consistent evaluation and analysis. Furthermore, differences between "*company_location" and "employee_residence"* highlight the impact of geographical factors on salaries, while the "*remote_ratio*" indicates company policies on remote work, which can also affect salary considerations.

Likewise, *"company_size"* mirrors salary variations among small, medium, and large companies, typically with larger firms offering higher pay. These insights collectively shed light on diverse factors influencing salary levels, informing data preparation, analysis, visualization, mathematical operations, and decision-making processes.

3. Data Preparation

Data preparation is the crucial stage in filtering raw data for analysis and processing which plays a critical role in enhancing data quality by rectifying errors, inconsistencies, and missing values. It validates and verifies data to ensure correctness and completeness, thereby mitigating the risk of inaccurate analysis (Khan, 21).

The "**Data Science Salary Analysis**" project's first phase deals mainly with careful data preparation. This stage consists of a number of crucial steps meant to refine the dataset and make sure it is well-organized, accurate, and clear. By carefully formatting the data, the way for seamless analysis is paved, setting the stage for extracting profound insights that lie within the dataset.

Below are the different activities performed in this data preparation:

- **Importing required libraries for data analysis and visualization.**

The code snippet provided serves as the foundation for data analysis and visualization in Python, essential for your **"Data Science Salary Analysis"**. Two crucial libraries **pandas** and **matplotlib** was imported, assigning them aliases as **“pd”** and **“plt”** respectively. Pandas simplifies data manipulation and analysis with DataFrame and Series structures, while matplotlib facilitates easy visualization creation. By incorporating these libraries into the Python environment, a robust foundation for handling and exploring the dataset was established.

Importing required libraries for data analysis and visualization

In [1]:
```
# Importing the pandas library as pd for data manipulation and analysis
import pandas as pd

# Importing the matplotlib.pyplot module as plt for creating visualizations and plots
import matplotlib.pyplot as plt
```

Figure 1: Importing required libraries for data analysis and visualization.

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

To load data into a panda DataFrame, the **read_csv()** function was utilized provided by the pandas library in Python. This function was specifically designed to read data from CSV files and store it into a DataFrame.

The below code reads the CSV file named **'CourseworkDataset.csv'** and stores its contents into a DataFrame variable named **'dataSets'**. The **pd.read_csv()** function is a part of the Pandas library and is specifically designed to read data from CSV files into a DataFrame, a tabular data structure commonly used for data analysis in Python. Once the data is read and stored in the 'dataSets' variable, it can be manipulated, analysed, or visualized using various Pandas methods and functions.

Program to Load the dataset from the provided CSV file for further analysis

```
In [2]: # Reading the CSV file 'CourseworkDataset.csv' and storing the data in a DataFrame variable named 'data'
        dataSets=pd.read_csv('CourseworkDataset.csv')
```

Figure 2: Program to load the dataset from the provided CSV file for further analysis.

This code snippet displays a sample of the loaded dataset from the **'DataScienceSalaries'** CSV file. First, it prints a descriptive message to indicate that it's displaying a sample of the dataset.Then, it uses the **sample()** function provided by pandas to select a random sample of ten rows from the DataFrame **dataSets**. This function randomly selects rows from the DataFrame and returns them as a new DataFrame. The argument **10** specifies the number of rows to be included in the sample.

Finally, the selected sample DataFrame is printed to the console, allowing the user to inspect a subset of the dataset and verify its contents.

Displaying loaded datasets from CSV file

In [3]:
```
# Displaying a sample of the loaded dataset to verify its contents
print("\nSample of the DataScienceSalaries csv file:")
print("\n================================================================================\n")

print(dataSets.sample(10))  # Displaying a random sample of ten rows from the DataFrame
```

```
Sample of the DataScienceSalaries csv file:

================================================================================

      work_year experience_level employment_type           job_title  salary  \
694        2023               EN              FT       Data Engineer   73900
2674       2022               SE              FT   Applied Scientist  191475
152        2023               SE              FT  Analytics Engineer  143200
1795       2023               SE              FT       Data Engineer  180000
12         2023               SE              FT        Data Analyst  100000
2443       2022               SE              FT      Data Scientist  104000
3598       2021               EN              FT      Data Scientist   31000
2233       2022               SE              FT      Data Architect  175000
2552       2022               SE              FT        Data Analyst   95000
705        2023               MI              FT       Data Engineer   70000

     salary_currency  salary_in_usd employee_residence  remote_ratio  \
694              USD          73900                 US             0
2674             USD         191475                 US           100
152              USD         143200                 US             0
1795             USD         180000                 US             0
12               USD         100000                 US           100
2443             USD         104000                 US           100
3598             EUR          36643                 FR            50
2233             USD         175000                 US           100
2552             USD          95000                 US             0
705              GBP          85066                 GB           100

     company_location company_size
694                US            M
2674               US            M
152                US            M
1795               US            M
12                 US            M
2443               US            M
3598               FR            L
2233               US            M
2552               US            M
705                GB            M
```

Figure 3: Displaying loaded datasets from CSV file.

After completing the extraction tasks, the dataset is now loaded into a Pandas DataFrame, allowing users to effortlessly perform various data operations. This is achieved by employing the **pd.DataFrame()** function, where the variable name storing the extracted dataset is passed as an argument. Therefore, the program for loading data into a Pandas DataFrame appears as follows:

Write a python program to load data into pandas DataFrame

```
In [4]: #Loading the data into a pandas DataFrame
        df = pd.DataFrame(dataSets)
```

Figure 4: Program to load data into pandas dataframe.

The below code segment prints the original DataFrame for visualization purposes. It begins by printing a descriptive header indicating that the following output displays the original DataFrame. Then, it prints the DataFrame itself using the **print()** function, followed by a line break for clarity. The DataFrame being printed is referred to as **df**, and it contains the data that has been loaded and processed earlier in the program. This step allows users to visually inspect the contents of the DataFrame and verify its structure and data before proeeding with any further analysis or manipulation tasks.

```
In [5]: # Printing the original DataFrame for visualization
        print("\n==============================Original DataFrame ==============================\n")
        print(df)
        print("\n================================================================================\n")
```

```
==============================Original DataFrame ==============================

      work_year experience_level employment_type                 job_title  \
0          2023               SE              FT  Principal Data Scientist
1          2023               MI              CT               ML Engineer
2          2023               MI              CT               ML Engineer
3          2023               SE              FT            Data Scientist
4          2023               SE              FT            Data Scientist
...         ...              ...             ...                       ...
3750       2020               SE              FT            Data Scientist
3751       2021               MI              FT  Principal Data Scientist
3752       2020               EN              FT            Data Scientist
3753       2020               EN              CT     Business Data Analyst
3754       2021               SE              FT      Data Science Manager

       salary salary_currency  salary_in_usd employee_residence  remote_ratio  \
0       80000             EUR          85847                ES           100
1       30000             USD          30000                US           100
2       25500             USD          25500                US           100
3      175000             USD         175000                CA           100
4      120000             USD         120000                CA           100
...       ...             ...            ...               ...           ...
3750   412000             USD         412000                US           100
3751   151000             USD         151000                US           100
3752   105000             USD         105000                US           100
3753   100000             USD         100000                US           100
3754  7000000             INR          94665                IN            50

     company_location company_size
0                  ES            L
1                  US            S
2                  US            S
3                  CA            M
4                  CA            M
...               ...          ...
3750               US            L
3751               US            L
3752               US            S
3753               US            L
3754               IN            L

[3755 rows x 11 columns]

================================================================================
```

Figure 5: Printing the original dataframe for visualization.

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

This program demonstrates the process of eliminating irrelevant columns, such as **'salary'** and **'salary_currency'**, from the dataset to enhance its relevance and clarity for analysis. By removing these unnecessary columns, the DataFrame becomes more streamlined and suitable for further operations.

To achieve this, the Pandas function **drop()** is utilized. This function allows the removal of specified columns along the column **axis (axis=1).** By setting the parameter **inplace=True**, the changes are applied directly to the original DataFrame. Subsequently, the modified DataFrame is printed again to showcase the elimination of the irrelevant columns. In principle, this program ensures that only relevant columns remain in the DataFrame, facilitating clearer data analysis and visualization.

Below is the refined version of the program:

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

In [6]:
```
# Removing the 'salary' column from the DataFrame 'df'
df.drop('salary', axis=1, inplace=True)

# Removing the 'salary_currency' column from the DataFrame 'df'
df.drop('salary_currency', axis=1, inplace=True)
```

Figure 6: Removing the unnecessary column(salary and salary currency).

This code segment displays the DataFrame after removing unnecessary columns which allows users to visually inspect the DataFrame post-removal of unnecessary columns, ensuring that only relevant columns are retained for further analysis and visualization.

```
In [7]: # Displaying the DataFrame after removing unnecessary columns
        print("\n------------------------------ Modified DataFrame ------------------------------\n")
        print(df)
        print("\n--------------------------------------------------------------------------------\n")
```

```
------------------------------ Modified DataFrame ------------------------------

      work_year experience_level employment_type                job_title  \
0          2023               SE              FT  Principal Data Scientist
1          2023               MI              CT               ML Engineer
2          2023               MI              CT               ML Engineer
3          2023               SE              FT            Data Scientist
4          2023               SE              FT            Data Scientist
...         ...              ...             ...                       ...
3750       2020               SE              FT            Data Scientist
3751       2021               MI              FT  Principal Data Scientist
3752       2020               EN              FT            Data Scientist
3753       2020               EN              CT     Business Data Analyst
3754       2021               SE              FT      Data Science Manager

      salary_in_usd employee_residence  remote_ratio company_location  \
0             85847                ES           100               ES
1             30000                US           100               US
2             25500                US           100               US
3            175000                CA           100               CA
4            120000                CA           100               CA
...             ...               ...           ...              ...
3750         412000                US           100               US
3751         151000                US           100               US
3752         105000                US           100               US
3753         100000                US           100               US
3754          94665                IN            50               IN

     company_size
0               L
1               S
2               S
3               M
4               M
...           ...
3750            L
3751            L
3752            S
3753            L
3754            L

[3755 rows x 9 columns]

--------------------------------------------------------------------------------
```

Figure 7: Displaying the Dataframe after removing unnecessary columns.

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

This code segment provides a comprehensive view of all columns within the DataFrame to enhance understanding. By displaying all the column labels, users gain a clear understanding of the structure and composition of the DataFrame, which is crucial for data analysis and manipulation.

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

```
In [8]: # Providing a comprehensive view of all columns within the DataFrame for better understanding
        print("Columns of the DataFrame:")
        print("-"* 80)
        print(df.columns)

Columns of the DataFrame:
--------------------------------------------------------------------------------
Index(['work_year', 'experience_level', 'employment_type', 'job_title',
       'salary_in_usd', 'employee_residence', 'remote_ratio',
       'company_location', 'company_size'],
      dtype='object')
```

Figure 8: Program to remove the NaN missing values from updated dataframe.

This program illustrates the process of removing any **NaN** missing values from the updated DataFrame to ensure accurate analysis outcomes. Addressing missing values is crucial as they can significantly impact analysis results. Initially, the program displays all columns of the DataFrame using the **.columns** function, providing users with an overview of available columns. Then, it identifies and checks for the presence of null values in each column using the **isnull()** function, displaying the results in Boolean form. Following this, the program calculates and displays the count of NaN missing values using the **.sum()** function. Since the provided dataset does not contain any missing values, it reports a count of zero for all columns.

Additionally, the program tests for the presence of missing values row-wise using the **isnull().any(axis=1)** function. Again, as there are no missing values in the dataset, it confirms the absence of null missing values for each row. Throughout these steps, the program reassures users by displaying outcomes that highlight the absence of null missing values, ensuring the DataFrame is ready for further analysis.

In [9]:
```
# Checking for missing values in each column of the DataFrame
missing_values=df.isnull().any()

# Displaying the presence of NaN missing values within the DataFrame, if any
print("\nPresence of NaN Missing Values within the DataFrame:\n", "-" *60)
missing_values
```

```
Presence of NaN Missing Values within the DataFrame:
 ------------------------------------------------------------
```

Out[9]:
```
work_year             False
experience_level      False
employment_type       False
job_title             False
salary_in_usd         False
employee_residence    False
remote_ratio          False
company_location      False
company_size          False
dtype: bool
```

Figure 9: Checking for missing values in each column of the Dataframe.

This code calculates the column-wise sum of NaN (missing) values within the DataFrame and stores the result in the variable **missing_value_count**. The expression **df.isnull()** generates a DataFrame of the same shape as **df**, where each cell contains a Boolean value indicating whether it is a missing value (True) or not (False). Then, **.sum()** is applied to this DataFrame, resulting in a Series object that contains the sum of missing values for each column.

Finally, the code prints a header indicating the purpose of the output ("Count of NaN Missing Values within the DataFrame:"), followed by a line of dashes for visual separation, and then prints the **missing_value_count** Series, which displays the count of missing values for each column.

```
In [10]: # Calculating the column-wise sum of NaN (missing) values
         missing_value_count = df.isnull().sum()

         # Displaying the count of NaN Missing Values within the DataFrame, if any
         print("\nCount of NaN Missing Values within the DataFrame:\n", "-" * 50)
         missing_value_count

         Count of NaN Missing Values within the DataFrame:
          --------------------------------------------------
Out[10]: work_year             0
         experience_level      0
         employment_type       0
         job_title             0
         salary_in_usd         0
         employee_residence    0
         remote_ratio          0
         company_location      0
         company_size          0
         dtype: int64
```

Figure 10: Calculating the column-wise sum of NaN (missing) values.

This code checks for rows in the DataFrame that contain NaN (missing) values and stores them in the variable **rows_with_missing_values**. If there are no missing values (i.e., the DataFrame is empty), it prints a message indicating that there are no NaN missing values present. If missing values are found, it prints a header indicating the presence of rows with NaN missing values, followed by displaying those rows in the DataFrame. This process ensures that users are informed about the presence of any missing values within the DataFrame, facilitating further data handling or cleaning procedures if necessary.

```
In [11]: # Checking for rows with NaN (missing) values and displaying them if present
         rows_with_missing_values = df[df.isnull().any(axis=1)]

         # If there are no missing values, display a message
         if rows_with_missing_values.empty:
             print("\nNo NaN Missing Values present within the DataFrame.")
         else:
             print("\nRows with NaN Missing Values within the DataFrame:")
             print(rows_with_missing_values)

         No NaN Missing Values present within the DataFrame.
```

Figure 11: Checking for rows with NaN(missing) values and displaying them if present.

This code removes NaN (missing) values from the DataFrame using the **dropna()** function.When **inplace=True** is set, it modifies the DataFrame directly, without needing to assign the result to a new variable.The variable **missing_values_removed** captures the return value of the **dropna()** function, which is either None (if no missing values were removed) or a DataFrame with missing values removed.The code then checks if any missing values were removed. If **missing_values_removed** is None, it prints a message indicating that no NaN missing values were found in the DataFrame and thus no rows were removed.

Otherwise, it prints a message confirming the removal of missing values, along with the value stored in **missing_values_removed**, which would be the modified DataFrame with missing values removed.This process ensures that users are informed about the outcome of the missing value removal operation.

No NaN missing values present within the DataFrame.

In [12]:
```
# Removing NaN missing values from the DataFrame
missing_values_removed = df.dropna(inplace=True)

# If there are no missing values removed, print a message
if missing_values_removed is None:
    print("\nNo NaN missing values found in the DataFrame. No rows were removed.")
else:
    print("\nMissing Values Removed:", missing_values_removed)
```

```
No NaN missing values found in the DataFrame. No rows were removed.
```

Figure 12: Removing NaN missing values from the dataframe.

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

This code checks for the presence of duplicate values in the DataFrame using the duplicated() function. The **duplicated()** function returns a Boolean Series where each element indicates whether the corresponding row is a duplicate of a previous row (True) or not (False). The variable **duplicate_values** store this Boolean Series. The **duplicate_values.any()** function call checks if there are any True values in the Series, indicating the presence of duplicate values. Finally, it prints the result of this check, indicating whether duplicate values exist in the DataFrame (True) or not (False). This process provides users with information about the presence of duplicate values in the DataFrame, which may require further handling or cleaning.

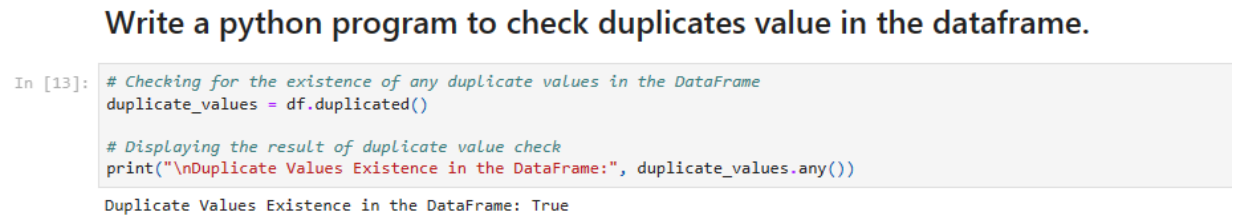


Figure 13: Program to check duplicates values in the dataframe.

This code calculates the number of duplicate rows in the DataFrame by summing up the True values in the **duplicate_values** Boolean Series, which was obtained from the **duplicated()** function earlier. The variable **duplicated_rows_count** stores this count.

Finally, the code prints the number of duplicated rows in the DataFrame, providing users with information about the extent of duplication present in the dataset. This step helps users understand the scale of duplication within the DataFrame, which can be crucial for data quality assessment and cleaning processes.

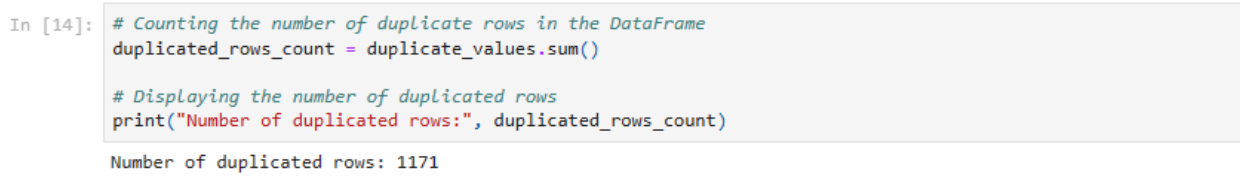


Figure 14: Counting the number of duplicate rows in Dataframe.

This code selects rows in the DataFrame df that are duplicates based on all columns and creates a new DataFrame named **duplicates_entries**. The variable **duplicate_values** contain a Boolean Series indicating whether each row is a duplicate or not. Using this Boolean Series as a mask, **df[duplicate_values]** selects only those rows from the DataFrame df where the corresponding value in **duplicate_values** is True, effectively isolating the duplicate entries. The resulting DataFrame duplicates_entries contains only the rows that are duplicates based on all columns.

Finally, the code prints the **DataFrame duplicates entries**, displaying the duplicate values for further examination or processing. This step helps users identify and analyze the duplicated entries in the dataset, which is essential for data quality assurance and cleaning procedures.

```
In [15]: # Selecting rows in DataFrame 'df' that are duplicates based on all columns and creating a new DataFrame 'Duplicates'
duplicates_entries = df[duplicate_values]

# Displaying the DataFrame consisting of duplicate values only
print("\nDataFrame with Duplicated Values:")
print("-"* 80)
print(duplicates_entries)
```

```
DataFrame with Duplicated Values:
--------------------------------------------------------------------------------
      work_year experience_level employment_type           job_title  \
115        2023               SE              FT      Data Scientist
123        2023               SE              FT  Analytics Engineer
153        2023               MI              FT       Data Engineer
154        2023               MI              FT       Data Engineer
160        2023               SE              FT       Data Engineer
...         ...              ...             ...                 ...
3439       2022               MI              FT      Data Scientist
3440       2022               SE              FT       Data Engineer
3441       2022               SE              FT       Data Engineer
3586       2021               MI              FT       Data Engineer
3709       2021               MI              FT      Data Scientist

      salary_in_usd employee_residence  remote_ratio company_location  \
115          150000                 US             0                US
123          289800                 US             0                US
153          100000                 US           100                US
154           70000                 US           100                US
160          115000                 US             0                US
...             ...                ...           ...               ...
3439          78000                 US           100                US
3440         135000                 US           100                US
3441         115000                 US           100                US
3586         200000                 US           100                US
3709          90734                 DE            50                DE

     company_size
115             M
123             M
153             M
154             M
160             M
...           ...
3439            M
3440            M
3441            M
3586            L
3709            L

[1171 rows x 9 columns]
```

Figure 15: Selecting rows in df that are duplicates based on all columns and creating a new dataframe 'Duplicates'.

This code removes duplicate rows from the DataFrame **df** using the **drop_duplicates()** function. When **inplace=True** is set, it modifies the DataFrame directly, without needing to assign the result to a new variable. After removing duplicates, the code then prints the updated DataFrame **df**, displaying it for further analysis or processing.

The **drop_duplicates()** function removes duplicate rows from the DataFrame based on all columns by default. This process ensures that the DataFrame contains only unique rows, eliminating any redundancy in the dataset. It's worth noting that the code includes the **drop_duplicates()** function call twice, which might be a mistake. Typically, it's only necessary to call this function once to remove duplicates from the DataFrame.

In [16]:
```
# Dropping duplicate rows from the DataFrame 'df'
df.drop_duplicates(inplace=True)

# Displaying the updated DataFrame after removing duplicates
print("\nThe DataFrame after removing duplicate values:")
df.drop_duplicates(inplace=True)
df
```

```
The DataFrame after removing duplicate values:
```

Out[16]:

	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

2584 rows × 9 columns

Figure 16: Dropping duplicates rows from the Dataframe 'df'.

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

This code prints the list of column names present in the DataFrame df. It first prints a header "Column Names:" to indicate the nature of the following output. Then, it prints the column names themselves using df.columns, which returns an Index object containing the column labels of the DataFrame.

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

```
In [17]: # Printing the list of column names in the DataFrame
         print("Column Names:")
         print(df.columns)

Column Names:
Index(['work_year', 'experience_level', 'employment_type', 'job_title',
       'salary_in_usd', 'employee_residence', 'remote_ratio',
       'company_location', 'company_size'],
      dtype='object')
```

Figure 17: Program to see the unique values from all the columns in the dataframe.

This code iterates over each column in the DataFrame df and prints the unique values present in each column. It utilizes a for loop to iterate through each column name in the DataFrame's columns. Within the loop, it retrieves the unique values of the current column using **df[column].unique(),** storing them in the variable **unique_values**.

Then, it prints a message for each column, indicating the column name along with its unique values.

```
In [18]: # Iterate over each column in the DataFrame and print its unique values
         for column in df.columns:
             unique_values = df[column].unique()
             print(f"Unique values in column '{column}': {unique_values}")
```

```
Unique values in column 'work_year': [2023 2022 2020 2021]
Unique values in column 'experience_level': ['SE' 'MI' 'EN' 'EX']
Unique values in column 'employment_type': ['FT' 'CT' 'FL' 'PT']
Unique values in column 'job_title': ['Principal Data Scientist' 'ML Engineer' 'Data Scientist'
 'Applied Scientist' 'Data Analyst' 'Data Modeler' 'Research Engineer'
 'Analytics Engineer' 'Business Intelligence Engineer'
 'Machine Learning Engineer' 'Data Strategist' 'Data Engineer'
 'Computer Vision Engineer' 'Data Quality Analyst'
 'Compliance Data Analyst' 'Data Architect'
 'Applied Machine Learning Engineer' 'AI Developer' 'Research Scientist'
 'Data Analytics Manager' 'Business Data Analyst' 'Applied Data Scientist'
 'Staff Data Analyst' 'ETL Engineer' 'Data DevOps Engineer' 'Head of Data'
 'Data Science Manager' 'Data Manager' 'Machine Learning Researcher'
 'Big Data Engineer' 'Data Specialist' 'Lead Data Analyst'
 'BI Data Engineer' 'Director of Data Science'
 'Machine Learning Scientist' 'MLOps Engineer' 'AI Scientist'
 'Autonomous Vehicle Technician' 'Applied Machine Learning Scientist'
 'Lead Data Scientist' 'Cloud Database Engineer' 'Financial Data Analyst'
 'Data Infrastructure Engineer' 'Software Data Engineer' 'AI Programmer'
 'Data Operations Engineer' 'BI Developer' 'Data Science Lead'
 'Deep Learning Researcher' 'BI Analyst' 'Data Science Consultant'
 'Data Analytics Specialist' 'Machine Learning Infrastructure Engineer'
 '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']
Unique values in column 'salary_in_usd': [ 85847  30000  25500 ...  28369 412000  94665]
Unique values in column 'employee_residence': ['ES' 'US' 'CA' 'DE' 'GB' 'NG' 'IN' 'HK' 'PT' 'NL' 'CH' 'CF' 'FR' 'AU'
 'FI' 'UA' 'IE' 'IL' 'GH' 'AT' 'CO' 'SG' 'SE' 'SI' 'MX' 'UZ' 'BR' 'TH'
 'HR' 'PL' 'KW' 'VN' 'CY' 'AR' 'AM' 'BA' 'KE' 'GR' 'MK' 'LV' 'RO' 'PK'
 'IT' 'MA' 'LT' 'BE' 'AS' 'IR' 'HU' 'SK' 'CN' 'CZ' 'CR' 'TR' 'CL' 'PR'
 'DK' 'BO' 'PH' 'DO' 'EG' 'ID' 'AE' 'MY' 'JP' 'EE' 'HN' 'TN' 'RU' 'DZ'
 'IQ' 'BG' 'JE' 'RS' 'NZ' 'MD' 'LU' 'MT']
Unique values in column 'remote_ratio': [100   0  50]
Unique values in column 'company_location': ['ES' 'US' 'CA' 'DE' 'GB' 'NG' 'IN' 'HK' 'NL' 'CH' 'CF' 'FR' 'FI' 'UA'
 'IE' 'IL' 'GH' 'CO' 'SG' 'AU' 'SE' 'SI' 'MX' 'BR' 'PT' 'RU' 'TH' 'HR'
 'VN' 'EE' 'AM' 'BA' 'KE' 'GR' 'MK' 'LV' 'RO' 'PK' 'IT' 'MA' 'PL' 'AL'
 'AR' 'LT' 'AS' 'CR' 'IR' 'BS' 'HU' 'AT' 'SK' 'CZ' 'TR' 'PR' 'DK' 'BO'
 'PH' 'BE' 'ID' 'EG' 'AE' 'LU' 'MY' 'HN' 'JP' 'DZ' 'IQ' 'CN' 'NZ' 'CL'
 'MD' 'MT']
Unique values in column 'company_size': ['L' 'S' 'M']
```

Figure 18: Iterating over each columns in the Dataframe and print its unique values.

3.6. Rename the experience level columns as below. SE SeniorLevel/Expert MI – Medium Level/Intermediate EN – Entry Level EX – Executive Level

This code defines a dictionary named **title_rename_mapping** to specify the renaming of **'SE' (Software Engineer) to 'Senior Level/Expert'**. Then, it replaces the job titles in the DataFrame df using this renaming dictionary. The **replace()** function is applied with **inplace=True** to modify the DataFrame directly.

Rename the experience level columns as below.

SE – Senior Level/Expert MI – Medium Level/Intermediate EN – Entry Level EX – Executive Level

Define a dictionary to rename 'SE' to 'Senior Level/Expert'

In [19]:
```
# Defining a dictionary to rename 'SE' (Software Engineer) to 'Senior Level/Expert'
title_rename_mapping = {'SE': "Senior Level/Expert"}

# Replacing job titles in the DataFrame 'df' using the renaming dictionary
df.replace(title_rename_mapping, inplace=True)
```

In [20]:
```
df
```

Out[20]:

	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	MI	CT	ML Engineer	30000	US	100	US	S
2	2023	MI	CT	ML Engineer	25500	US	100	US	S
3	2023	Senior Level/Expert	FT	Data Scientist	175000	CA	100	CA	M
4	2023	Senior Level/Expert	FT	Data Scientist	120000	CA	100	CA	M
...	...	...	...	...	...	...	...	...	...
3750	2020	Senior Level/Expert	FT	Data Scientist	412000	US	100	US	L
3751	2021	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	Senior Level/Expert	FT	Data Science Manager	94665	IN	50	IN	L

2584 rows × 9 columns

Figure 19: Defining a dictionary to rename 'SE' to 'Senior Level/Expert'.

This code snippet defines a dictionary named **title_rename_mapping** to facilitate the renaming of **'MI'** (denoting a job title) to **'Medium Level/Intermediate'**.

Next, it applies this renaming dictionary to replace the job titles in the DataFrame df using the **replace()** function, with the parameter **inplace=True** indicating that the changes should be applied directly to the original DataFrame.

Define a dictionary to rename 'MI' to 'Medium Level/Intermediate'

In [21]:
```
# Defining a dictionary to rename  'MI' to 'Medium Level/Intermediate'
title_rename_mapping =  {'MI': "Medium Level/Intermediate"}

# Replacing job titles in the DataFrame 'df' using the renaming dictionary
df.replace(title_rename_mapping, inplace=True)
```

In [22]:
```
df
```

Out[22]:

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

2584 rows × 9 columns

Figure 20: Defining a dictionary to rename 'MI' to 'Medium Level/Intermediate'.

This code defines a dictionary named **title_update_mapping** to facilitate the renaming of **'EN'** (denoting a job title as Entry Level) to **'Entry Level'**.

Then, it applies this renaming dictionary to replace the job titles in the DataFrame df using the **replace()** function, with the parameter **inplace=True** indicating that the changes should be applied directly to the original DataFrame.

Define a dictionary to rename 'EN' to 'Entry Level'

```
In [23]: # Defining a dictionary to rename 'EN' (Entry Level) to 'Entry Level'
         title_update_mapping = {'EN': "Entry Level"}

         # Replacing job titles in the DataFrame 'df' using the renaming dictionary
         df.replace(title_update_mapping, inplace=True)

In [24]: df
```

Out[24]:

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

2584 rows × 9 columns

Figure 21: Defining a dictionary to rename 'EN' to 'Entry Level'

This code defines a dictionary named **title_update_mapping** to facilitate the renaming of **'EX'** (denoting a job title as Executive Level) to **'Executive Level'**.

Then, it applies this renaming dictionary to replace the job titles in the DataFrame df using the **replace()** function, with the parameter **inplace=True** indicating that the changes should be applied directly to the original DataFrame.

Define a dictionary to rename 'EX' to 'Executive Level'

In [25]:
```
# Defining a dictionary to rename 'EX' to 'Executive Level'
title_update_mapping = {'EX': "Executive Level"}

# Replacing job titles in the DataFrame 'df' using the renaming dictionary
df.replace(title_update_mapping, inplace=True)
```

In [26]:
```
df
```

Out[26]:

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

2584 rows × 9 columns

Figure 22: Defining a dictionary to rename 'EX' to 'Executive Level'.

4. Data Analysis

Data analysis is a structured process encompassing the systematic gathering, cleansing, transforming, describing, modeling, and interpretation of data which holds significant importance in both scientific inquiry and business realms, particularly as the demand for data-driven decision-making has surged in recent times. Through the application of diverse data analysis methodologies, significant insights can be obtained from datasets, which can then be utilized to inform operational choices or direct future research initiatives (Eldridge, 2024).

The course "Data Science Salary Analysis" highlights the importance of data analysis in obtaining insightful information from the dataset. It involves analyzing each column one by one in order to determine how it affects the salary levels. Through various statistical methods and mathematical operations such as summarizing statistics (sum, mean, standard deviation, skewness, and kurtosis), identifying patterns and trends, and calculating correlations among variables, the analysis aims to uncover significant relationships. Additionally, it enables comparisons across different categories such as salary levels over time, geographic locations, and company types, empowering informed decision-making within organizations.

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

This code segment analyzes salary statistics from the DataFrame df. First, it calculates descriptive statistics for the **'salary_in_usd'** column using the **describe()** method and stores the results in the variable **salary_description**. Then, it computes additional statistics such as **total salary (sum), average salary (mean), standard deviation of salary, skewness, and kurtosis**.

Finally, it prints the calculated statistics, providing a comprehensive overview of the salary distribution within the dataset. This analysis offers valuable insights into various aspects of salary data, including central tendency, variability, skewness, and shape of the distribution, aiding in understanding and interpreting salary trends within the dataset.

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

In [27]:
```
# Analyzing salary statistics
salary_description = df['salary_in_usd'].describe()  # Calculate descriptive statistics

# Calculate additional statistics
total_salary = df['salary_in_usd'].sum()  # Calculate sum
average_salary = df['salary_in_usd'].mean()  # Calculate mean
std_dev_salary = df['salary_in_usd'].std()  # Calculate standard deviation
salary_skew = salary_description.skew()  # Calculate skewness
salary_kurt = salary_description.kurtosis()  # Calculate kurtosis

# Display the statistics
print("Salary Statistics:")
print("-" * 25)
print("Description:")
print(salary_description)
print()
print("Total Salary:", total_salary)
print("Average Salary:", average_salary)
print("Standard Deviation of Salary:", std_dev_salary)
print("Salary Skewness:", salary_skew)
print("Salary Kurtosis:", salary_kurt)
```

```
Salary Statistics:
-------------------------
Description:
count      2584.000000
mean     133409.280186
std       67136.837329
min        5132.000000
25%       84975.000000
50%      130000.000000
75%      175000.000000
max      450000.000000
Name: salary_in_usd, dtype: float64

Total Salary: 344729580
Average Salary: 133409.28018575851
Standard Deviation of Salary: 67136.83732925021
Salary Skewness: 1.8456151844884958
Salary Kurtosis: 4.151486799923131
```

Figure 23: Program to show summary ststistics of sum, mean, standard deviation, skewness, and kurtosis of any choosen variable.

In addition, similar activities were performed to calculate the statistics summary with sum, mean, standard deviation, skewness, and kurtosis for the variable ***"remote_ratio_summary"*** Here, all the function as used for **"salary_in_usd"** are utilized to perform the corresponding computation. Thus, it provide a better understanding of remote work trends along with its scenario in the dataset. Also, it calculates the total portion of remote work across all the dataset observations along with the average portion with measuring its asymmetry distribution and peakedness or flatness of the distribution.

Thus, the program calculating summary statistics of sum, mean, standard deviation, skewness, and kurtosis of ***"remote_ratio"*** column is shown below:

Analyzing 'remote_ratio' for various statistical measures

In [28]:
```
# Analyzing 'remote_ratio' for various statistical measures

# Summary statistics
remote_ratio_summary = df['remote_ratio'].describe()

# Calculating sum
remote_ratio_sum = df['remote_ratio'].sum()

# Calculating mean
remote_ratio_mean = df['remote_ratio'].mean()

# Calculating standard deviation
remote_ratio_std = df['remote_ratio'].std()

# Calculating skewness
remote_ratio_skewness = df['remote_ratio'].skew()

# Calculating kurtosis
remote_ratio_kurtosis = df['remote_ratio'].kurtosis()

# Printing the results
print("\nSummary Statistics for 'remote_ratio':\n", remote_ratio_summary)
print("-" * 55)
print("\nSum of 'remote_ratio':", remote_ratio_sum)
print("\nMean of 'remote_ratio':", remote_ratio_mean)
print("\nStandard Deviation of 'remote_ratio':", remote_ratio_std)
print("\nSkewness of 'remote_ratio':", remote_ratio_skewness)
print("\nKurtosis of 'remote_ratio':", remote_ratio_kurtosis)
```

```
Summary Statistics for 'remote_ratio':
 count    2584.000000
mean       50.483746
std        48.163707
min         0.000000
25%         0.000000
50%        50.000000
75%       100.000000
max       100.000000
Name: remote_ratio, dtype: float64
-------------------------------------------------------

Sum of 'remote_ratio': 130450

Mean of 'remote_ratio': 50.48374613003096

Standard Deviation of 'remote_ratio': 48.16370698235042

Skewness of 'remote_ratio': -0.01931889440725452

Kurtosis of 'remote_ratio': -1.9229957652422922
```

Figure 24: Analyzing remote ratio for varuous statistical measures.

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

This program demonstrates the essential process of calculating and displaying correlations between variables within a dataset. This step is crucial as it shows potential associations and cause-and-effect linkages between different numerical variables, aiding in pattern recognition and dependency identification. By analyzing correlations, we can discern whether changes in one variable correspond to changes in another, thereby gauging the impact level of these relationships. Moreover, correlation analysis helps pinpoint data quality issues, including errors or outliers.

To compute and display correlations, the program initially selects numerical columns from the DataFrame using the **select_dtypes()** function with the parameter **include='number'**, ensuring that only columns with numerical data types, such as integers or floats, are considered. This filtration ensures the suitability of variables for correlation analysis, as correlation coefficients are typically computed between numerical values. Here, the columns *'work_year', 'salary_in_usd', and 'remote_ratio'* were filtered.

Following this, the program calculates the correlation matrix using the **corr() function**, which computes pairwise correlations between all selected columns. The correlation matrix is a square matrix where each position represents the correlation coefficient between two variables. Finally, after calculating the correlations, the program displays them to provide valuable insights into the relationships between numerical values in the dataset.

In interpreting the correlation coefficients, values ranging from -1 to 1 are considered. A coefficient close to 1 indicates a strong positive correlation, -1 denotes a strong negative correlation, and a coefficient close to 0 signifies no correlation between variables. Positive correlation suggests that an increase in one variable is accompanied by an increase in another, while negative correlation implies that an increase in one variable leads to a decrease in another.

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

In [29]:
```
# Selecting numerical columns
numerical_df = df.select_dtypes(include='number')

# Calculate correlation matrix
correlation_matrix = numerical_df.corr()

# Display the correlation matrix
print("Correlation Matrix of Numerical Variables:")
print("-" * 55)
print(correlation_matrix)
```

```
Correlation Matrix of Numerical Variables:
-------------------------------------------------------
              work_year  salary_in_usd  remote_ratio
work_year      1.000000       0.236958     -0.219160
salary_in_usd  0.236958       1.000000     -0.084502
remote_ratio  -0.219160      -0.084502      1.000000
```

Figure 25: Program to calculate and show correlation of all variables.

5. Data Exploration

Analyzing raw data to identify trends, qualities, and correlations between elements is known as **data exploration**. Analysts gain insights before data is recorded in databases or data warehouses by using it to help identify the dataset's structure, anomalies, and data distribution. As part of the process, features and their interactions are evaluated, data quality is guaranteed, and data is examined both statistically and graphically. Data exploration is critical because it facilitates the selection of useful data, expedites analysis, and lets users see patterns in data that spreadsheets would not show (Salatino, 2024).

5.1. Renaming similar job_titles

On examining the DataFrame, it was observed that multiple positions displayed identical job titles. These comparable job titles were renamed to guarantee consistency and readability in the data. To do this, a dictionary entitled **"job_title_mapping"** was established, in which the titles of all the previous jobs were mapped to the titles of the new jobs.

To preserve consistency among similar job titles representing the same profession, the title “AI Programmer" was renamed to "AI Developer". In this way, about fourteen job names that were similar were renamed in order to standardize the depiction of job titles and improve consistency and clarity for additional data investigation.

The **"replace()"** function was used to implement the renaming, replacing instances of similar job titles with the renamed versions that were specified in the " **job_title_mapping** " dictionary. It was ensured that the changes were applied to the original DataFrame by setting the "inplace" argument to "True". More precise analysis and interpretation of data related to jobs were made possible by this systematic method. The code snippet that shows how to rename and replace elements in a DataFrame is given below, along with an explanation of the process in a detailed remark.

Organizing job titles by renaming similar roles to ensure consistency and clarity of data.

In [30]:
```
# Creating a dictionary to rename old job titles to new job titles
job_title_mapping = {
    'AI Programmer': "AI Developer",  # Renaming 'AI Programmer' to 'AI Developer'
    'AI Scientist': "Artificial Intelligence Scientist",  # Renaming 'AI Scientist' to 'Artificial Intelligence Scientist'
    'Analytics Engineer': "Data Analytics Engineer",  # Renaming 'Analytics Engineer' to 'Data Analytics Engineer'
    'Applied Data Scientist': "Data Scientist",  # Renaming 'Applied Data Scientist' to 'Data Scientist'
    'Applied Machine Learning Engineer': "Machine Learning Engineer",  # Renaming 'Applied Machine Learning Engineer' to 'Machine Learning Engineer'
    'Applied Machine Learning Scientist': "Machine Learning Scientist",  # Renaming 'Applied Machine Learning Scientist' to 'Machine Learning Scientist'
    'Applied Scientist': "Data Scientist",  # Renaming 'Applied Scientist' to 'Data Scientist'
    'BI Analyst': "BI Data Analyst",  # Renaming 'BI Analyst' to 'BI Data Analyst'
    'BI Data Engineer': "Business Intelligence Engineer",  # Renaming 'BI Data Engineer' to 'Business Intelligence Engineer'
    'Data Science Lead': "Data Science Tech lead",  # Renaming 'Data Science Lead' to 'Data Science Tech lead'
    'Data Scientist Lead': "Lead Data Scientist",  # Renaming 'Data Scientist Lead' to 'Lead Data Scientist'
    'Financial Data Analyst': "Finance Data Analyst",  # Renaming 'Financial Data Analyst' to 'Finance Data Analyst'
    'ML Engineer': "Machine Learning Engineer",  # Renaming 'ML Engineer' to 'Machine Learning Engineer'
    'MLOps Engineer': "Machine Learning Operations Engineer",  # Renaming 'MLOps Engineer' to 'Machine Learning Operations Engineer'
}
# Replacing old job titles with new job titles in the DataFrame 'df'
df.replace(job_title_mapping, inplace=True)
```

Figure 26: Organizing job titles by renaming similar roles to ensure consistency and clarity of data.

After ensuring consistency among similar job names, this code presents the DataFrame. It displays the updated DataFrame with consistent job titles, which facilitates the analysis and interpretation of employment-related data.

Displaying the dataframe after organizing job titles to ensure consistency and clarity.

In [31]:
```
# Displaying the DataFrame after organizing similar job titles to maintain consistency.
print("\nThe DataFrame after standardizing similar job titles is:")
print("-" * 65)
print(df)
```

```
The DataFrame after standardizing similar job titles is:
-----------------------------------------------------------------
      work_year           experience_level employment_type  \
0          2023       Senior Level/Expert              FT
1          2023  Medium Level/Intermediate             CT
2          2023  Medium Level/Intermediate             CT
3          2023       Senior Level/Expert              FT
4          2023       Senior Level/Expert              FT
...         ...                       ...             ...
3750       2020       Senior Level/Expert              FT
3751       2021  Medium Level/Intermediate             FT
3752       2020               Entry Level              FT
3753       2020               Entry Level              CT
3754       2021       Senior Level/Expert              FT

                     job_title  salary_in_usd employee_residence  \
0      Principal Data Scientist          85847                ES
1     Machine Learning Engineer          30000                US
2     Machine Learning Engineer          25500                US
3                Data Scientist         175000                CA
4                Data Scientist         120000                CA
...                         ...            ...               ...
3750             Data Scientist         412000                US
3751   Principal Data Scientist         151000                US
3752             Data Scientist         105000                US
3753      Business Data Analyst         100000                US
3754       Data Science Manager          94665                IN

      remote_ratio company_location company_size
0              100               ES            L
1              100               US            S
2              100               US            S
3              100               CA            M
4              100               CA            M
...            ...              ...          ...
3750           100               US            L
3751           100               US            L
3752           100               US            S
3753           100               US            L
3754            50               IN            L

[2584 rows x 9 columns]
```

Figure 27: Displaying the dataframe after organizing job titles to ensure consistency and clarity.

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

The program that is provided shows how to identify the top 15 jobs from the list of jobs that are available in the DataFrame and properly display the results in a bar graph. Therefore, it helps in understanding how jobs are distributed within the industry. Additionally, the visual representation facilitates understanding and comparison of employment trends over time. It also supports decisions about strategy concerning industry trends and recruitment practices, which aim to improve the company's standing and elevate the least desirable positions.

In this section the **".value_counts()"** method from the updated DataFrame was used to present the unique job list after renaming the similar **job_title** together with their count before determining the top fifteen jobs. This helps to analyze the list of jobs along with their occurrence in the DataFrame. It assists in determining the top job and by default displays the list in descending order. Analyzing the frequency of employment distribution is therefore helpful. Below is a sample of code that finds every unique job title together with its count:

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

```
In [32]: #Counting the occurrences of each job title
         job_counts = df['job_title'].value_counts()
         job_counts

Out[32]: job_title
         Data Engineer                598
         Data Scientist               579
         Data Analyst                 396
         Machine Learning Engineer    242
         Data Analytics Engineer       97
                                     ...
         Azure Data Engineer             1
         Data DevOps Engineer            1
         Staff Data Analyst              1
         Compliance Data Analyst         1
         Staff Data Scientist            1
         Name: count, Length: 82, dtype: int64
```

Figure 28: Program to find out top 15 jobs

The software uses the **".value_counts()"** function on the **'job_title'** column to extract the top 15 jobs after first retrieving all job titles and their numbers. The top 15 job titles and their corresponding counts are retrieved by adding **".head(15)"** to the **".value_counts()"** function; by default, the results are shown in descending order. By streamlining the process of identifying the most common job titles in the dataset, additional analysis and decision-making are made easier.

```
In [33]: # Getting the top 15 job titles
top_jobs = df['job_title'].value_counts().head(15)

# Display the top 15 job titles
print("\nTop 15 Most Common Job Titles:\n")
print("-" * 35)
print(top_jobs)
```

```
Top 15 Most Common Job Titles:

-----------------------------------
job_title
Data Engineer                 598
Data Scientist                579
Data Analyst                  396
Machine Learning Engineer     242
Data Analytics Engineer        97
Research Scientist             65
Data Architect                 64
Data Science Manager           52
Machine Learning Scientist     38
Research Engineer              33
BI Data Analyst                24
Data Manager                   23
Data Science Consultant        23
Computer Vision Engineer       18
Data Analytics Manager         18
Name: count, dtype: int64
```

Figure 29: Getting the top 15 job titles.

This code segment generates a bar chart to visually represent the frequency distribution of the top 15 job titles in the DataFrame. Initially, the figure size is specified to be 10x6 units using **plt.figure(figsize=(10, 6))**, ensuring adequate space for the plot. The **plot()** function is then utilized to create a bar chart of the top_jobs data, with the plot type set to 'bar'. The bars are customized to have a brown color, partial transparency (alpha=0.5), and black edges for clearer delineation. The chart is titled 'Sales Distribution of Top 15 Jobs' using **plt.title()**, with the x-axis labeled as 'Job Title' (**plt.xlabel()**) and the

y-axis as 'Sales' (**plt.ylabel()**). To enhance readability, the x-axis labels are rotated 90 degrees clockwise for better alignment using **plt.xticks(rotation=90, ha='right')**. The layout is adjusted to prevent overlap of labels and title with **plt.tight_layout()**. Finally, the completed chart is displayed using **plt.show()**.

Bar graph to visualize the sales count for the top 15 job titles

In [34]:
```
# Creating a bar chart to visualize the frequency distribution of the top 15 job titles

# Setting the figure size
plt.figure(figsize=(10, 6))

# Plotting a bar chart of the top 15 job titles
top_jobs.plot(kind='bar', color="brown", alpha = 0.5, edgecolor='black')

# Setting the title of the chart
plt.title('Sales Distribution of Top 15 Jobs')

# Setting the label for the x-axis
plt.xlabel('Job Title')

# Setting the label for the y-axis
plt.ylabel('Sales')

# Rotating x-axis labels for better readability
plt.xticks(rotation=90, ha='right')

# Adjusting layout to prevent overlap of labels
plt.tight_layout()

# Displaying the chart
plt.show()
```

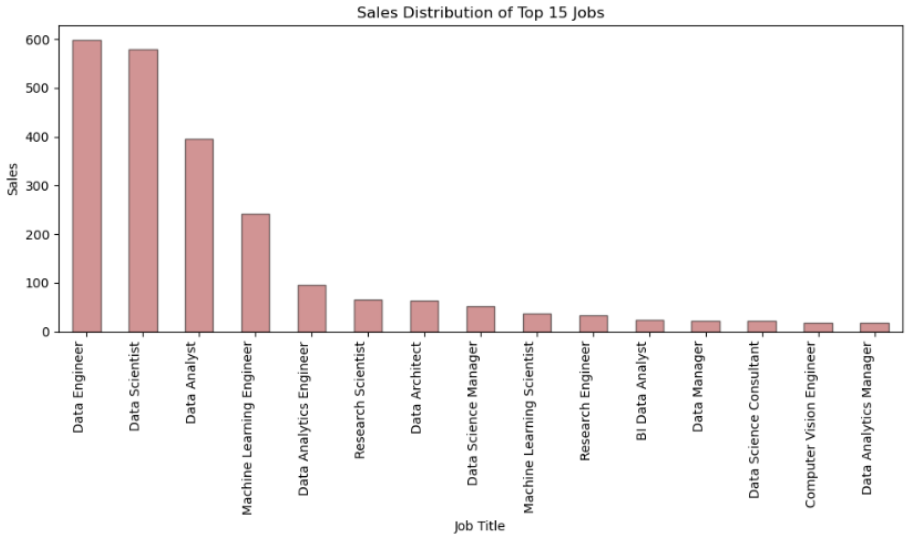


Figure 30: Bar graph to visualize the sales count for the top 15 job titles.

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

This code segment aims to identify the job title with the highest average salary from the DataFrame and display the result. Initially, the dataset is grouped by the 'job_title' column using the **groupby()** function, and the mean (average) salary for each unique job title is calculated using the **mean()** function, storing the results in the variable **mean_salary_by_job**. Then, the **idxmax()** function is applied to **mean_salary_by_job** to find the job title corresponding to the highest mean salary, which is stored in the variable **highest_salary_job_title**. Next, the average salary of the highest-paying job is retrieved using the **loc[ ]** function applied to **mean_salary_by_job** with the **highest_salary_job_title**, and the result is stored in the variable **highest_salary_job_amount**. Finally, the job title with the highest average salary along with its corresponding mean salary is printed to the console using a formatted string, providing the desired output.

Below is the code snippet to find the job with the highest salary including the amount :

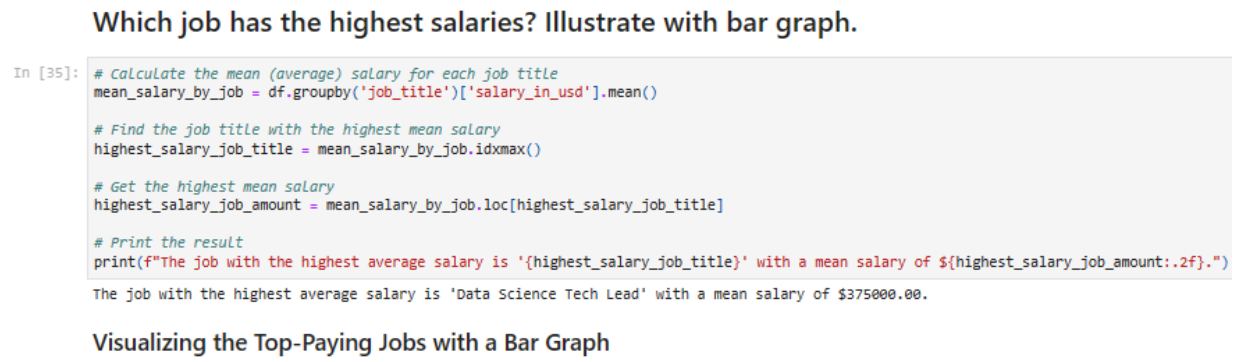


Figure 31: Job with highest average salary.

This code segment calculates the mean salary for each job title by grouping the DataFrame 'df' based on the 'job_title' column and then computing the average salary using the **mean()** function. The resulting mean salaries are stored in the **mean_salary_by_job** variable. Next, the job title with the highest mean salary is identified using the **idxmax()** function and stored in the variable **highest_salary_job_title**.

To visualize the average salary by job title, the size of the figure is set to 20x10 using the **plt.figure(figsize=(20, 10))** function. Colors for the bar graph are defined based on whether each job title matches the highest paying job title, with the highest paying job highlighted in purple.The bar graph is created using the **plot()** function with **kind='bar'**, and the defined colors are applied to each bar. Additional customization includes setting the alpha value to 0.5 for transparency and setting the edge color of the bars to black for better visibility.

Titles are added to the chart using **plt.title()**, and the x-axis and y-axis are labeled with 'Job Title' and 'Mean Salary in USD', respectively, using **plt.xlabel()** and **plt.ylabel()**. To improve readability, the x-axis labels are rotated 90 degrees to the right using **plt.xticks(rotation=90, ha='right')**.Finally, the layout is adjusted to prevent overlapping of labels and titles using **plt.tight_layout()**, and the plot is displayed using **plt.show()**. Overall, this code segment effectively visualizes the average salary distribution across different job titles, highlighting the job title with the highest mean salary for easy interpretation.

Below is the code snippet to find the job with the highest salary in a bar graph:

Which job has the highest salaries?(Bar Graph)

In [36]:
```
# Calculating the mean salary for each job title
mean_salary_by_job = df.groupby('job_title')['salary_in_usd'].mean()

# Identifying the job title with the highest mean salary
highest_salary_job_title = mean_salary_by_job.idxmax()

# Setting the size of the figure
plt.figure(figsize=(20, 10))

# Defining colors for the bar graph, highlighting the highest paying job in purple
colors = ['green' if job == highest_salary_job_title else 'brown' for job in mean_salary_by_job.index]

# Creating the bar graph with appropriate colors
mean_salary_by_job.plot(kind='bar', color=colors, alpha=0.5, edgecolor='black')

# Adding a title to the chart
plt.title('Average Salary by Job Title')

# Labeling the x-axis
plt.xlabel('Job Title')

# Labeling the y-axis
plt.ylabel('Mean Salary in USD')

# Rotating the x-axis labels for better readability
plt.xticks(rotation=90, ha='right')

# Adjusting the layout to prevent overlapping of labels and titles
plt.tight_layout()

# Displaying the plot
plt.show()
```

Figure 32: Bar graph to show highest salaries.

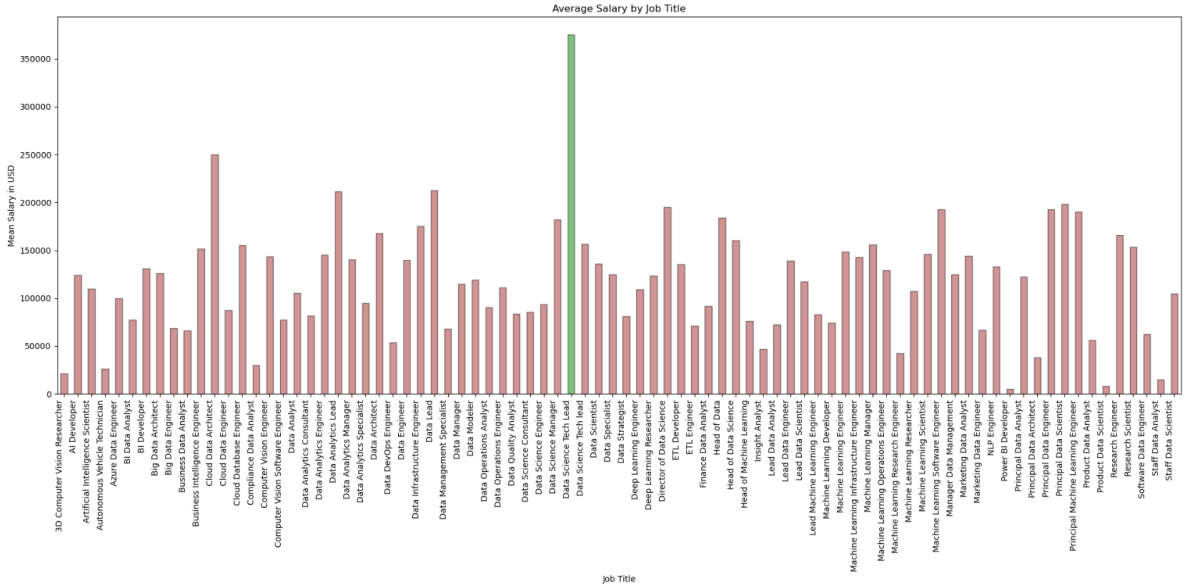


Figure 33: Bar graph of average salary by job title.

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

This code segment calculates the mean salary for each unique experience level present in the DataFrame 'df' by grouping the data based on the **'experience_level'** column using the **groupby()** function. The **mean()** function is then applied to compute the average salary corresponding to each experience level.

The resulting mean salaries for different experience levels are stored in the variable **ExprienceLevel**. Finally, the output is displayed followed by the mean salary for each experience level. This provides insights into how salaries vary across different levels of experience within the dataset.

Figure 34: Finding out salaries based on experience level.

This code segment calculates the average salary for each distinct group of experience levels present in the DataFrame 'df' by grouping the data based on the **'experience_level'** column using the **groupby()** function. The mean() function is then applied to compute the mean salary corresponding to each experience level group.

After calculating the average salary for each group, the data is sorted in ascending order based on the average salary using the **sort_values()** function. This ensures that the output displays the experience levels in increasing order of average salary. Finally, the output is displayed, followed by the average salary for each experience level group. This provides insights into how salaries vary across different levels of experience within the dataset, with the groups arranged from the lowest to the highest average salary.

```
In [38]: # Calculating the mean salary for each different experience level group
average_salary_by_experience = df.groupby('experience_level')['salary_in_usd'].mean()

# Sorting the data by average salary for each experience level
average_salary_by_experience = average_salary_by_experience.sort_values()

# Displaying the average salary based on experience level
print("\nAverage Salary by Experience Level:\n")
print("-" * 45)
print(average_salary_by_experience)
```

```
Average Salary by Experience Level:

---------------------------------------------
experience_level
Entry Level                  72648.685185
Medium Level/Intermediate   101828.783133
Senior Level/Expert         153897.435650
Executive Level             191078.208333
Name: salary_in_usd, dtype: float64
```

Figure 35: Average Salary by experience level.

This code snippet creates a bar graph to visualize salaries based on different experience levels. Firstly, it sets the size of the figure to (20, 15) inches using **plt.figure(figsize=(20, 15))** to ensure a suitable visual representation. Then, it generates a bar chart using the **plot()** function with **kind='bar'**, specifying the color of the bars as brown with an alpha value of **0.5** for **transparency**, and setting the edge color of the bars to black for better visibility.

Following that, it sets the title of the chart to **'Salaries by Experience Level'** with a font size of **16 using plt.title().** The **x-axis** label is set to **'Experience Level'** with a font size of 16 using **plt.xlabel()**, and the **y-axis** label is set to **'Salaries (USD)'** with a font size of 16 using **plt.ylabel().**To enhance readability, the x-axis labels are rotated 360 degrees (essentially keeping them horizontal) and set to a font size of 15 using **plt.xticks(rotation=360, fontsize=15),** while the y-axis font size is set to 15 using **plt.yticks(fontsize=15).**

Finally, the layout is adjusted to prevent label overlap using **plt.tight_layout(),** and the chart is displayed using **plt.show().** This visualization provides a clear depiction of how salaries vary across different experience levels within the dataset.

```
In [39]: # Creating a bar graph to visualize salaries based on experience level

         plt.figure(figsize=(20, 15))  # Set the figure size

         # Create a bar chart
         average_salary_by_experience.plot(kind='bar', color='brown', alpha = 0.5, edgecolor='black')

         plt.title('Salaries by Experience Level', fontsize=16)  # Set the title of the chart
         plt.xlabel('Experience Level', fontsize=16)  # Set the label for the x-axis
         plt.ylabel('Salaries (USD)', fontsize=16)  # Set the label for the y-axis

         plt.xticks(rotation=360, fontsize=15)  # Rotate x-axis labels for better readability
         plt.yticks(fontsize=15)  # Set font size for y-axis ticks

         plt.tight_layout()  # Adjust layout to prevent overlap of labels
         plt.show()  # Display the chart
```

Figure 36: Bar graph to visualize salaries based on experience level.

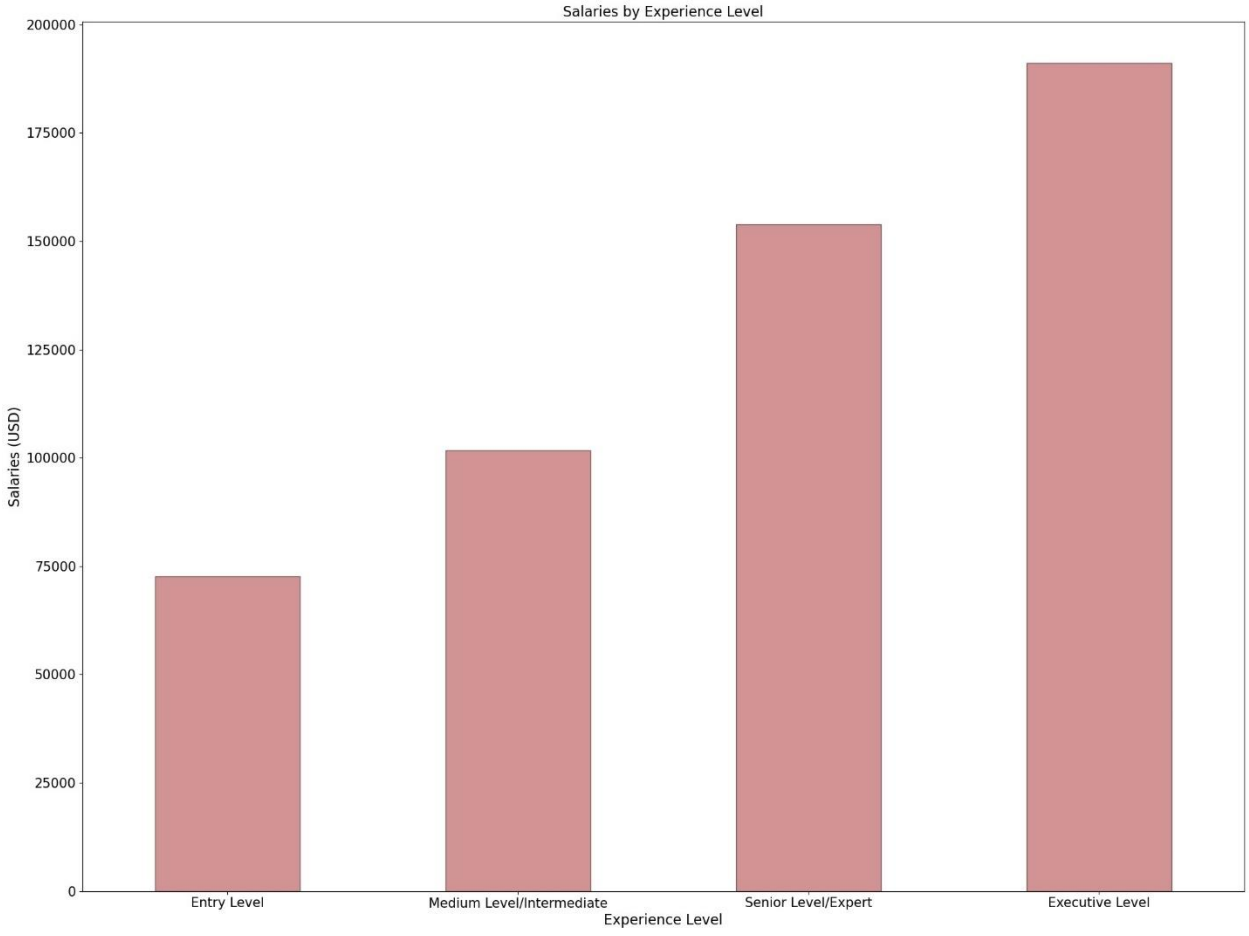


Figure 37: Bar graph of experience level.

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

This code snippet creates a histogram to visualize the distribution of salaries present in the DataFrame. Firstly, it sets the size of the figure to **(20, 14)** inches using **plt.figure(figsize=(20, 14))** to ensure a suitable visual representation. Then, it generates the histogram using the **hist()** function, specifying the salary data column **(df['salary_in_usd'])** to plot and the number of **bins (25)** to divide the salary range into intervals.

The histogram bars are colored brown with black edges for better visibility, and they are made slightly transparent **(alpha=0.5)** to aid in distinguishing overlapping bars. Gridlines are added to the plot using **plt.grid(True, linestyle='--', alpha=0.5)**. Next, the title of the histogram is set to **'Histogram of Salary Distribution'** using **plt.title(),** and the x-axis label is set to **'Salary in USD'** using **plt.xlabel().** The y-axis is labeled with **'Frequency'**, representing the number of occurrences of salaries within each bin.

To enhance readability, the **x-axis** labels are rotated **90 degrees** to ensure they don't overlap, and they are aligned to the right using **plt.xticks(rotation=90, ha='right')**. Additionally, the layout is adjusted to prevent overlapping of labels and titles using **plt.tight_layout()**. Finally, the histogram is displayed using **plt.show().** Overall, this visualization provides insights into the distribution of salaries within the dataset, allowing for a clear understanding of the salary trends and variability present in the industry.

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

Histogram of choosen variables -- salary_in_usd

In [40]:
```
# Visualizing the distribution of salaries using a histogram

# Setting the size of the figure
plt.figure(figsize=(20, 14))

# Creating the histogram with 20 bins, skyblue color, black edges, and 70% transparency
plt.hist(df['salary_in_usd'], bins=25, color='brown', edgecolor='black', alpha=0.5, linewidth=1)

# Adding gridlines to the plot
plt.grid(True, linestyle='--', alpha=0.5)

# Setting the title of the histogram
plt.title('Histogram of Salary Distribution')

# Labeling the x-axis with salary values in USD
plt.xlabel('Salary in USD')

# Labeling the y-axis with frequency of salaries
plt.ylabel('Frequency')

# Rotating the x-axis labels for better readability and alignment
plt.xticks(rotation=90, ha='right')

# Adding text annotation for mean salary
mean_salary = df['salary_in_usd'].mean()

# Adjusting the layout to prevent overlapping of labels and titles
plt.tight_layout()

# Displaying the histogram
plt.show()
```

Figure 38: Program to show histogram plot of salary_in_usd.

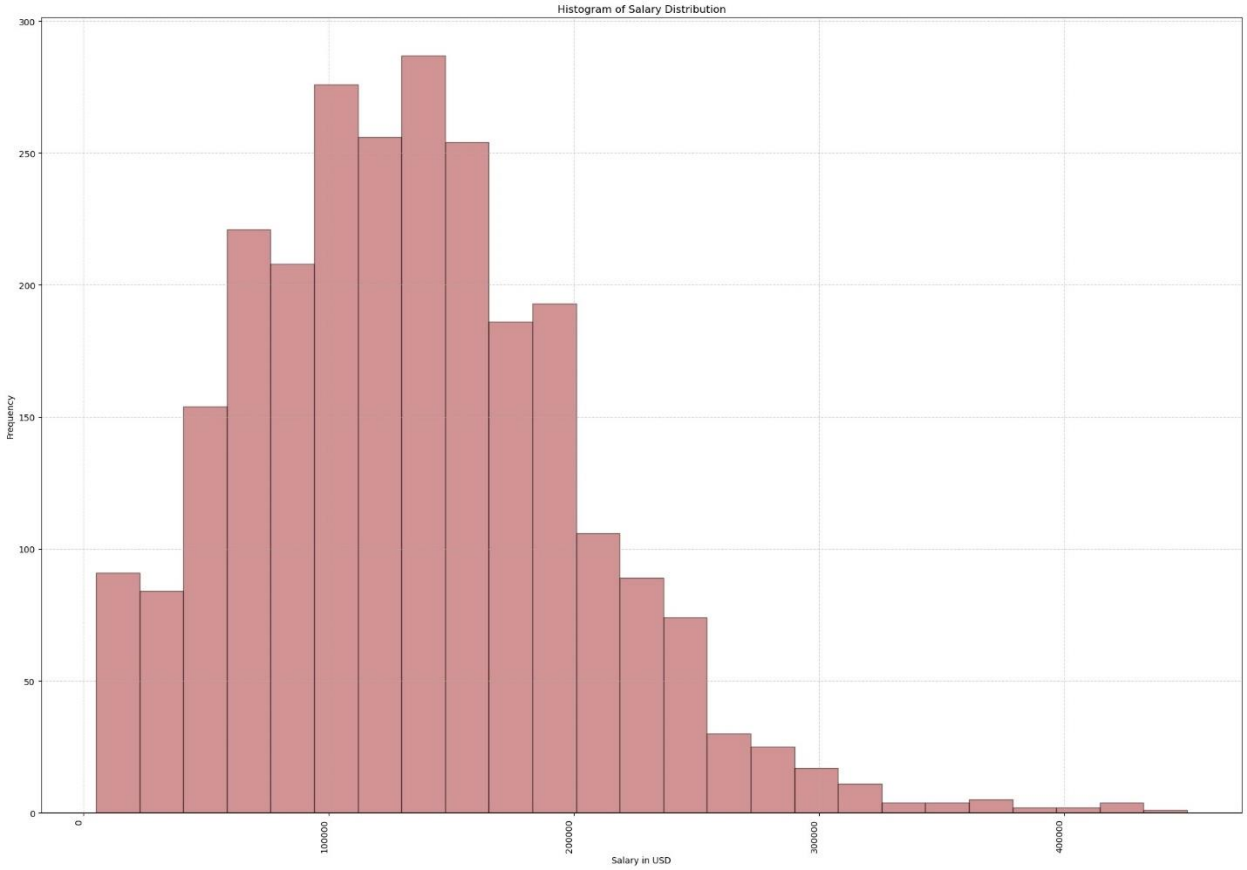


Figure 39: Histogram of Salary_in_usd.

This code snippet creates a box plot to illustrate the distribution of salaries present in the DataFrame. Firstly, it sets the size of the figure to **(20, 14)** inches using **plt.figure(figsize=(20, 14))** to ensure a suitable visual representation. Then, the box plot for the salary distribution is plotted using the **boxplot()** function, specifying the salary data column **(df['salary_in_usd']).**The **patch_artist=True** argument is used to fill the box plot with color, and the **boxprops** parameter is used to customize the appearance of the box, setting its **facecolor to brown**, **edge color to black**, and **transparency** to **0.5**.

Additionally, the **medianprops** parameter is used to set the color of the **median** line to black for **better visibility**. The title of the box plot is set to **'Box Plot of Salary Distribution'** using **plt.title()**, and the x-axis label is set to **'Salary in USD'** using **plt.xlabel().** Similarly, the **y-axis** label is set to **'Salary (USD)' using plt.ylabel().**To ensure proper layout, the function **plt.tight_layout()** is used to prevent overlapping of labels and titles. Finally, the box plot is displayed using **plt.show().**

Overall, this visualization provides insights into the distribution of salaries within the dataset, highlighting key statistics such as median, quartiles, and outliers, allowing for a clear understanding of the salary distribution and variability present in the data.

Box plot of chosen variables--salary_in_usd

In [41]:
```
# Illustrating the distribution of salaries using a box plot

# Setting the size of the figure
plt.figure(figsize=(20, 14))

# Plotting the box plot for salary distribution
plt.boxplot(df['salary_in_usd'], patch_artist=True, boxprops=dict(facecolor='brown',  alpha= 0.5, color='black'), medianprops=dict(color='black'))

# Setting the title of the chart
plt.title('Box Plot of Salary Distribution', fontsize=18)

# Setting the label for the x-axis
plt.xlabel('Salary in USD', fontsize=18)

# Setting the label for the y-axis
plt.ylabel('Salary (USD)', fontsize=18)

# Adjusting the layout to prevent overlapping of labels and titles
plt.tight_layout()

# Displaying the plot
plt.show()
```

Figure 40: Program to box plot of salary_in_usd.

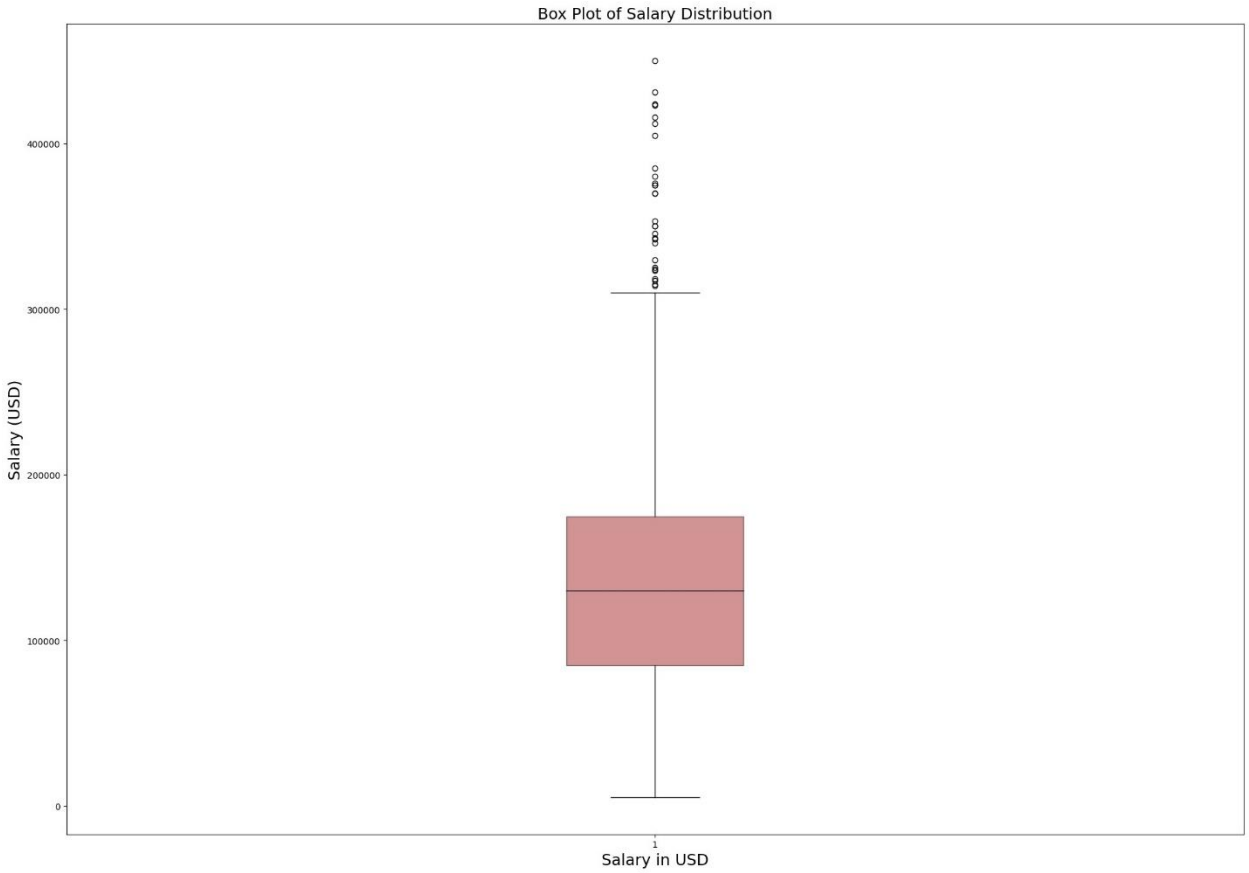


Figure 41: Box Plot of salery_in_usd.

Below is the combined visual representation showcasing both a histogram and a box plot for the salary data. The histogram provides a detailed view of the salary distribution, while the box plot offers a summary of key statistics. This combined visualization offers comprehensive insights into the distribution and characteristics of the salary data, aiding in analysis and decision-making processes.

Illustrating Histogram and Box plot together of chossen variable -- salary_in_usd

In [42]:
```
#Plotting the box plot and histogram

#Set the figure size
plt.figure(figsize=(25, 10))

#Subplot 1: Histogram

#Create a subplot for Histogram
plt.subplot(1, 2, 1)


 #Creating a histogram for the 'salary_in_usd' column
df['salary_in_usd'].hist(bins=20, linewidth=0.5, color="brown", alpha=0.5, edgecolor = 'black')

# Setting the title of the chart
plt.title('Histogram of Salary', fontsize=18)

# Setting the label for the x-axis
plt.xlabel('salary_in_usd', fontsize=18)

# Setting the label for the y-axis
plt.ylabel('Frequency', fontsize=18)

#Subplot 2: Box Plot

#Create a subplot for boxplot
plt.subplot(1, 2, 2)

# Plotting the box plot with customized colors and styles
plt.boxplot(df['salary_in_usd'], patch_artist=True, boxprops=dict(facecolor='brown', alpha=0.5, color='black'), medianprops=dict(color='black'))


# #Creating a box plot for the 'salary_in_usd' column
# df.boxplot(column='salary_in_usd')

# Setting the title of the chart
plt.title('Box Plot of Salary', fontsize=18)

# Setting the label for the x-axis
plt.xlabel('Salary in USD', fontsize=18)

# Setting the label for the y-axis
plt.ylabel('salary(USD)', fontsize=18)

# Adjusting the layout to prevent overlapping of labels and titles
plt.tight_layout()

# Displaying the plot
plt.show()
```

Figure 42:Combined Histogram and Box Plot of salary_in_usd.

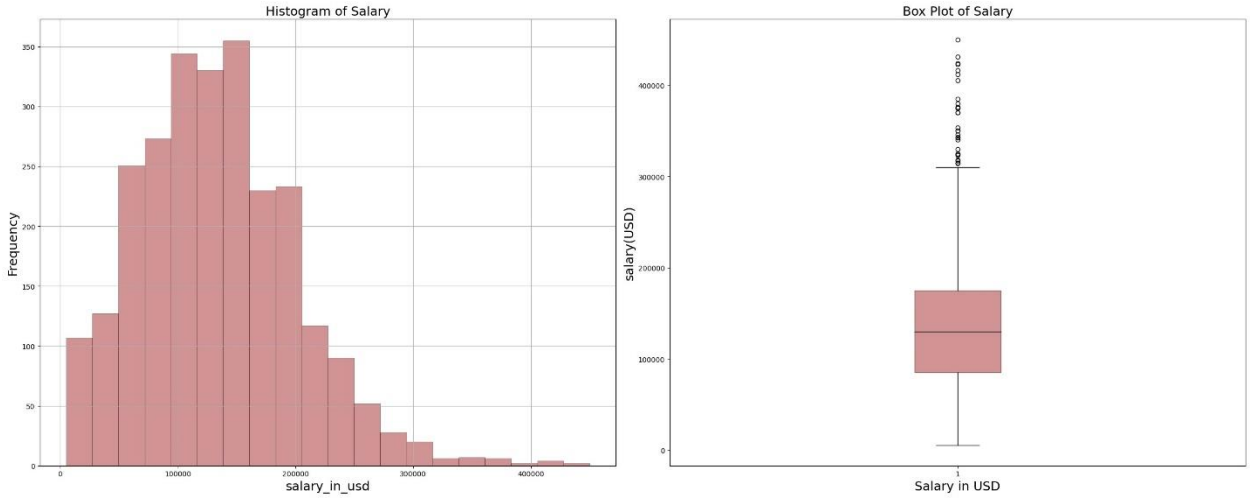


Figure 43: Combined histogram and box plot of salary_in_usd.

6. Conclusion

The data analysis coursework in the field of data science salary has been an insightful experience, with the goal of thoroughly understanding, organizing, evaluating, and analyzing datasets to extract important insights into the dynamics of pay in the sector. Using Python packages like matplotlib and pandas, the coursework carefully examined every stage of the data analytics lifecycle. Initially the preparation and understanding of the data were prioritized. This included dealing with missing values and duplicates, classifying data types across columns, and identifying and correcting inconsistencies.

The careful preparation of the dataset ensured its consistency and integrity, providing a strong basis for any further study. Quantitative calculations and qualitative data analysis methods were used as the data analysis stage to find patterns, distributions, and correlations in the dataset. To identify relationships between various variables and provide insight into the factors influencing pay levels, measures of central tendency, dispersion, and correlation were computed. By using various graphs, such as bar graphs, boxplots, and histograms, to visually convey findings, the data exploration phase further enhanced the study. This made it easier to understand how salaries are distributed among various job titles and experience levels by emphasizing trends and abnormalities in the data.

To sum up, this course not only gives students the fundamental knowledge and abilities to analyze data, but it also offers priceless information that can be used to make well-informed decisions on pay scales, hiring practices, and personnel management in the data science sector. This project establishes a strong foundation for improving industry practices and promoting additional breakthroughs in the field by adhering to industry standards and making good use of Python modules.

7. References

3 Pillar Global, 2016. *Understanding the Data in Data Science | 3Pillar Global.* [Online] Available at: https://www.3pillarglobal.com/insights/blog-posts/understanding-data-data-science/
[Accessed 12 May 2024].

Biswal, A., 2024. *What is Data Science? A Simple Explanation and More.* [Online] Available at: https://www.simplilearn.com/tutorials/data-science-tutorial/what-is-data-science
[Accessed 12 May 2024].

Eldridge, S., 2024. *Data analysis | Definition, Research, & Methodology | Britannica.* [Online]
Available at: https://www.britannica.com/science/data-analysis
[Accessed 10 May 2024].

James, V., 2024. *Data Insights: What They Are, Ways to Gather Them & More.* [Online] Available at: https://www.zuar.com/blog/what-are-data-insights/
[Accessed 11 May 2024].

Khan, F., 21. *What Is Data Preparation? + 9 Steps For Effective Data Prep.* [Online] Available at: https://www.astera.com/type/blog/data-preparation/
[Accessed 1 May 2024].

M, R., 2021. *What Is Matplotlib In Python? How to use it for plotting? - ActiveState.* [Online]
Available at: https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/
[Accessed 10 May 2024].

Rudderstack, 2024. *Quantitative vs. Qualitative Data.* [Online] Available at: https://www.rudderstack.com/learn/data-analytics/quantitative-vs-qualitative-data/
[Accessed 12 May 2024].

Salatino, F., 2024. *Data Exploration: aDefinition, Importance & How It Differs from Data Analytics (CData Software).* [Online]
Available at: https://www.cdata.com/blog/what-is-data-exploration
[Accessed 7 May 2024].

S, S., 2020. *What Is Pandas in Python? Everything You Need to Know - ActiveState.* [Online]
Available at: https://www.activestate.com/resources/quick-reads/what-is-pandas-in-python-everything-you-need-to-know/
[Accessed 10 May 2024].

Suer, M., 2023. *What Is Data Quality and Why Is It Important?* | *Alation.* [Online]
Available at: https://www.alation.com/blog/what-is-data-quality-why-is-it-important/
[Accessed 12 May 2024].