* In any organization, managing and analyzing employee data is crucial for efficient human resource management, strategic decision-making, and operational effectiveness. Employee data encompasses a wide range of personal, demographic, and professional attributes that provide valuable insights into workforce composition, employment history, and trends. The dataset presented here includes extensive details about employees, capturing critical information such as identification details, personal demographics, employment records, and geographic locations. This introduction provides an in-depth overview of the various attributes included in the dataset and their significance in workforce analysis.
* Below is a breakdown of each column:

|  |
| --- |
| **Emp ID:** Unique identifier for each employee. |
| Example: 857211, 514341, etc. |
| Type: Numeric (Integer). |

|  |
| --- |
| **Name Prefix:** Title or prefix of the employee's name (e.g., Mr., Ms., Prof.). |
| Example: Ms., Mr., Prof. |
| Type: Categorical (String). |

|  |
| --- |
| **First Name:** The first name of the employee. |
| Example: Hermila, Antonio, Sebastian. |
| Type: String. |

|  |
| --- |
| **Middle Initial**: refers to the first letter of an employee's middle name (if they have one). For example, if someone’s full name is **John Michael Smith**, their middle initial would be **M** (the first letter of the middle name "Michael"). |
| Example: J, Q, S. |
| Type: String (Single character). |

|  |
| --- |
| **Last Name:** The last name of the employee. |
| Example: Suhr, Joy, Moores. |
| Type: String. |

|  |
| --- |
| **Gender:** Gender of the employee. |
| Example: F(Female), M (Male). |
| Type: Categorical (String). |

|  |
| --- |
| **E Mail:** Email address of the employee. |
| Example: hermila.suhr@gmail.com, antonio.joy@yahoo.com. |
| Type: String. |

|  |
| --- |
| **Father's Name:** Full name of the employee's father. |
| Example: Todd Suhr, Clark Joy. |
| Type: String. |

|  |
| --- |
| **Mother's Name:** Full name of the employee's mother. |
| Example: Cathrine Suhr, Clarisa Joy. |
| Type: String. |

|  |
| --- |
| **Mother's Maiden Name:** Maiden name of the employee's mother (last name before marriage). |
| Example: Hinojosa, Gagliardi. |
| Type: String. |

|  |
| --- |
| **Date of Birth:** Employee's date of birth in `MM/DD/YYYY` format. |
| Example: 9/4/1992, 12/24/1989. |
| Type: Date. |

|  |
| --- |
| **Time of Birth:** Time of birth of the employee in HH:MM:SS AM/PM format. |
| Example: 4:29:56 AM, 8:01:44 AM. |
| Type: Time. |

|  |
| --- |
| **Age in Yrs:** Age of the employee in years (calculated as of a specific date).If an employee was born on **September 4, 1992**, and today's date is **February 20, 2025**, the employee would be **24.91 years old** (approximately 24 years and 11 months old). |
| Example: 24.91, 27.61. |
| Type: Numeric (Float). |

|  |
| --- |
| **Weight in Kgs:** Weight of the employee in kilograms. |
| Example: 57, 55. |
| Type: Numeric (Integer). |

|  |
| --- |
| **Date of Joining:** Date when the employee joined the company in MM/DD/YYYY format. |
| Example: 9/9/2014, 8/2/2011. |
| Type: Date. |

|  |
| --- |
| **Quarter of Joining:** Quarter of the year when the employee joined (Q1, Q2, Q3, Q4). |
| Example: Q3, Q2. |
| Type: Categorical (String). |

|  |
| --- |
| **Half of Joining:** Half of the year when the employee joined (H1 for Jan-Jun, H2 for Jul-Dec). |
| Example: H2, H1. |
| Type: Categorical (String). |

|  |
| --- |
| **Year of Joining:** Year when the employee joined the company. |
| Example: 2014, 2011. |
| Type: Numeric (Integer). |

|  |
| --- |
| **Month of Joining:** Month number when the employee joined (1-12). |
| Example: 9, 8. |
| Type: Numeric (Integer). |

|  |
| --- |
| **Month Name of Joining:** Full name of the month when the employee joined. |
| Example: September, August. |
| Type: Categorical (String). |

|  |
| --- |
| **Short Month:** Abbreviated name of the month when the employee joined. |
| Example: Sep, Aug. |
| Type: Categorical (String). |

|  |
| --- |
| **Day of Joining:** Day of the month when the employee joined. |
| Example: 9, 2. |
| Type: Numeric (Integer). |

|  |
| --- |
| **DOW of Joining:** Day of the week when the employee joined (full name). |
| Example: Tuesday, Sunday. |
| Type: Categorical (String). |

|  |
| --- |
| **Short DOW:** Abbreviated day of the week when the employee joined. |
| Example: Tue, Sun. |
| Type: Categorical (String). |

|  |
| --- |
| **Age in Company (Years):** Number of years the employee has been with the company (as of a specific date). |
| Example: 2.88, 5.99. |
| Type: Numeric (Float). |

|  |
| --- |
| **Salary:** Current salary of the employee. |
| Example: 168991, 53504. |
| Type: Numeric (Integer). |

|  |
| --- |
| **Last % Hike:** Percentage of the last salary hike received by the employee. |
| Example: 12%, 30%. |
| Type: String (Percentage). |

|  |
| --- |
| **SSN**: Social Security Number of the employee (hyphen-separated). |
| Example: 275-17-8844, 646-23-6213. |
| Type: String. |

|  |
| --- |
| **Phone No:** Phone number of the employee (hyphen-separated). |
| Example: 479-539-4593, 229-234-6154. |
| Type: String. |

|  |
| --- |
| **Place Name:** Name of the place where the employee resides. |
| Example: Peach Orchard, Rocky Ford. |
| Type: String. |

|  |
| --- |
| **County:** County where the employee resides. |
| Example: Clay, Screven. |
| Type: String. |

|  |
| --- |
| **City:** City where the employee resides. |
| Example: Peach Orchard, Rocky Ford. |
| Type: String. |

|  |
| --- |
| **State:** state where the employee resides (abbreviated). |
| Example: AR, GA. |
| Type: Categorical (String). |

|  |
| --- |
| **Zip:** ZIP code of the employee's residence. |
| Example: 72453, 30455. |
| Type: Numeric (Integer). |

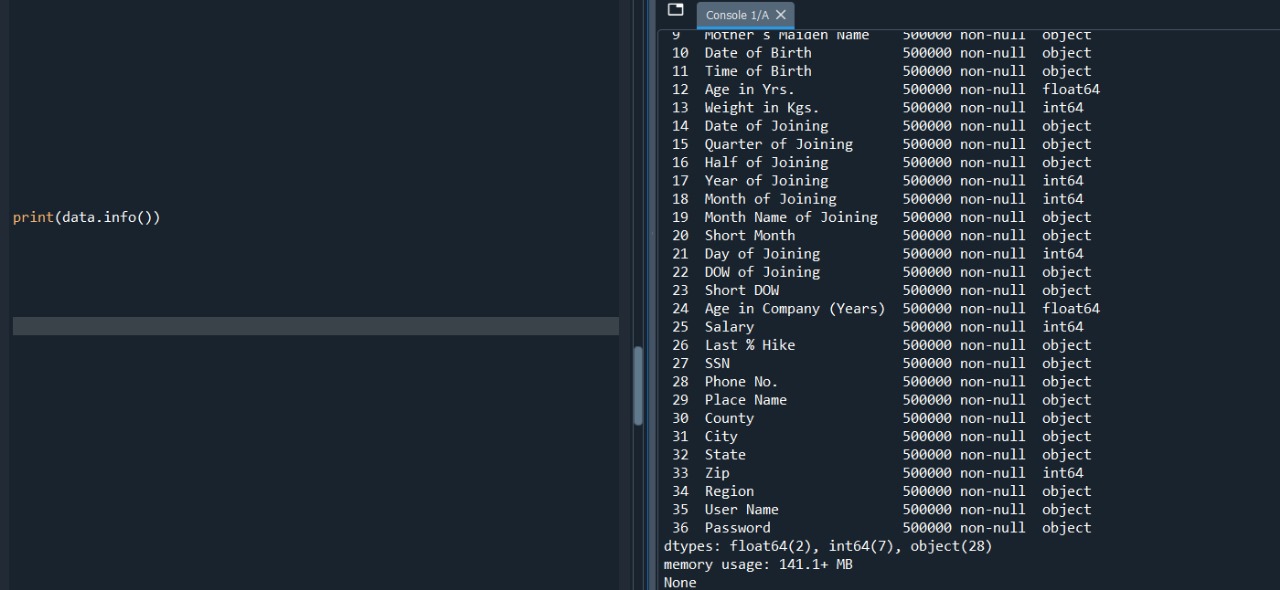
|  |
| --- |
| **Region:** Geographic region of the employee's residence (e.g., South, Northeast). |
| Example: South, Northeast. |
| Type: Categorical (String). |

|  |
| --- |
| **User Name:** Username of the employee (likely for company systems). |
| Example: hjsuhr, aqjoy. |
| Type: String. |

|  |
| --- |
| **Password:** Password of the employee (likely for company systems). |
| Example: oZ%{<6wN!A, 7\_[%FE;saZ:B. |
| Type: String |

* **Data cleaning and preprocessing**

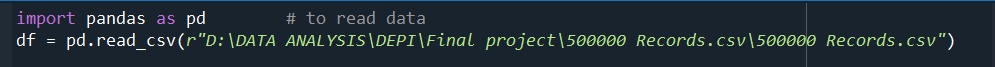
In this project, we will be working with a dataset using **Python** and the **Pandas** library for data analysis and manipulation. The following sections will outline the essential steps in loading, exploring, and preparing the dataset. Each step includes Python code and an explanation of its functionality.

This point will cover:

* Importing necessary libraries
* Loading and inspecting the dataset
* Handling missing values and duplicate entries
* Extracting key information from the dataset

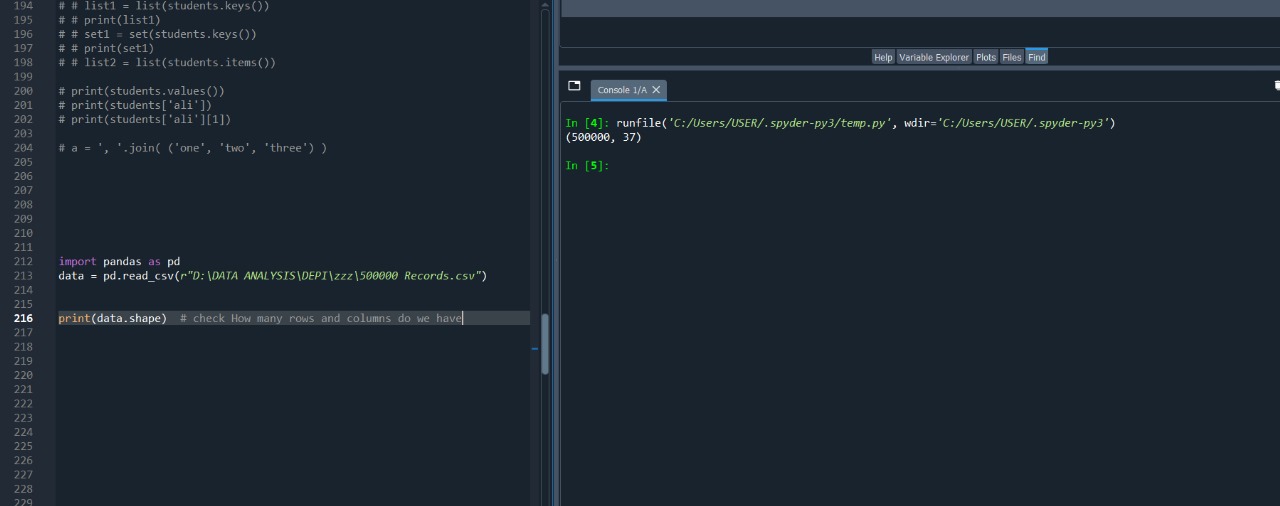
By following these steps, we aim to ensure that the data is clean, structured, and ready for further analysis.

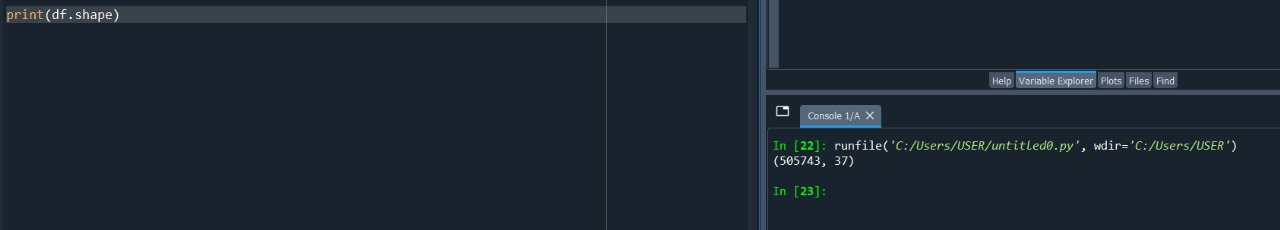
1. **Importing and loading**

This line imports the pandas library and gives it an alias (pd). also loads the dataset from a ‘CSV’ (Comma Separated Values) file located at the specified location.

1. **Checking the shape of the DataFrame**

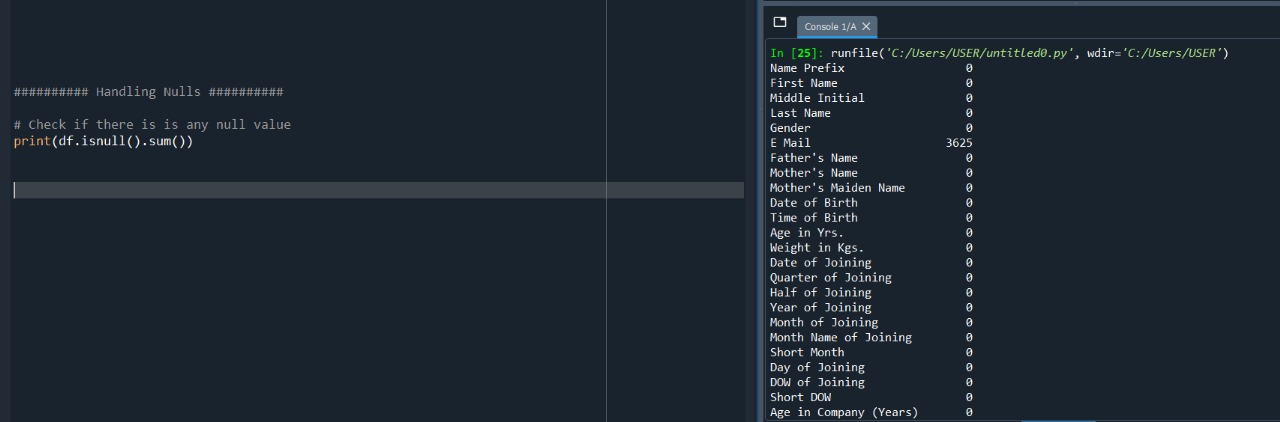
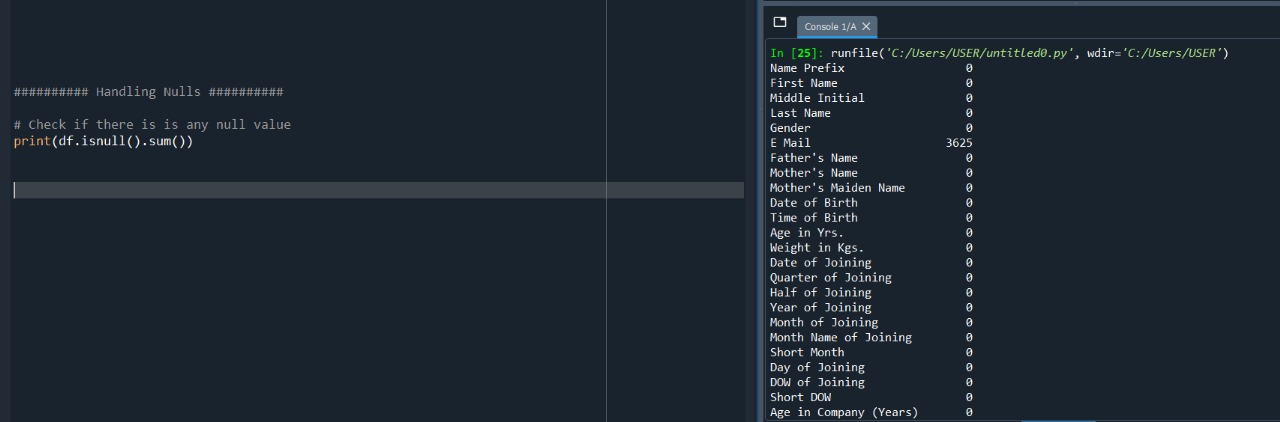
The data.shape command returns the ‘dimensions’ of the dataset, specifically the number of rows and columns. This is useful for quickly understanding the size of the dataset.





1. **Counting the number of missing values in each column**

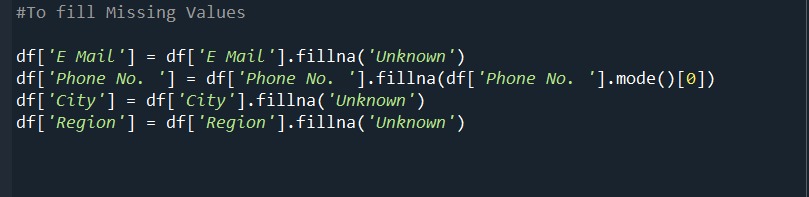
The data.isnull() method creates a boolean DataFrame where each value is True if the value in that position is missing (null), and False if the value is present. The .sum() function adds up the number of True values for each column, effectively counting how many ‘missing’ values exist in each column. This helps identify columns with missing data that may need to be cleaned or imputed.



****

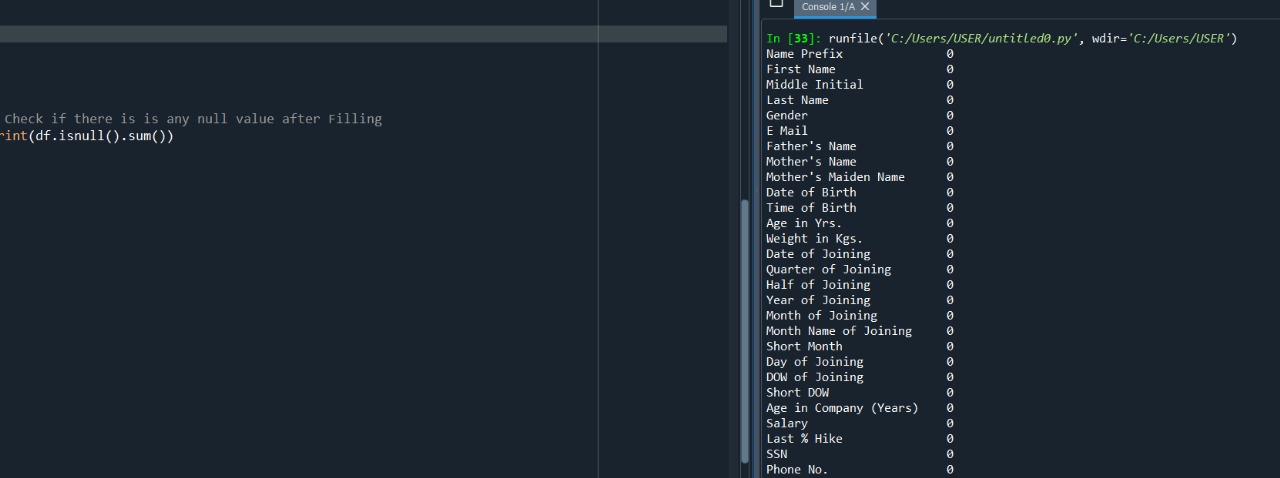
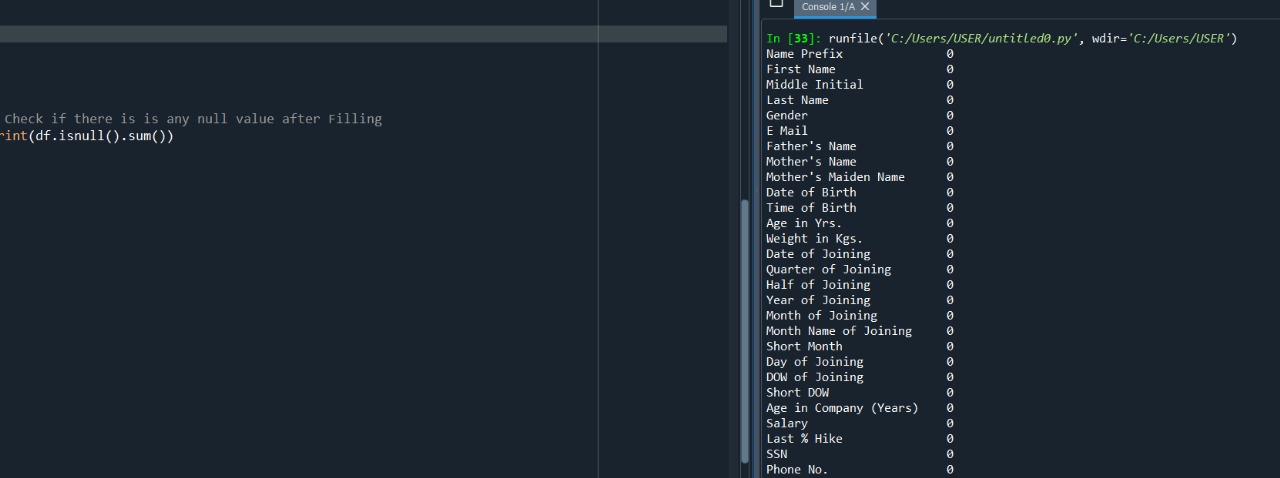
1. **To fill the missing values**

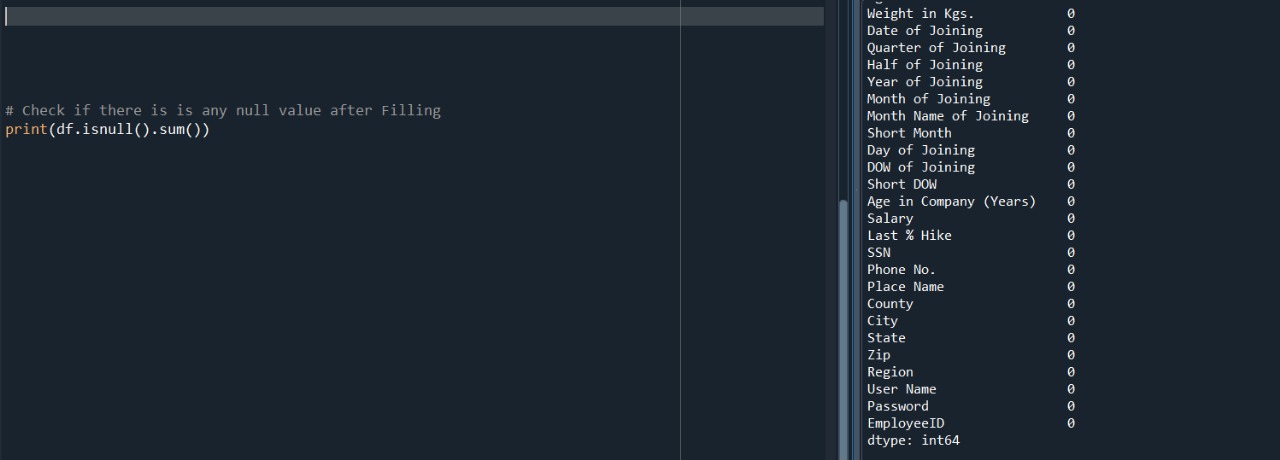
The .fillna() method is used to replace missing values in the dataset. Missing email, city, and region values are replaced with 'Unknown' to maintain data consistency. For the phone number column, missing values are filled with the most frequent (mode) phone number in the dataset to ensure data integrity. This step helps handle missing data without losing valuable records.



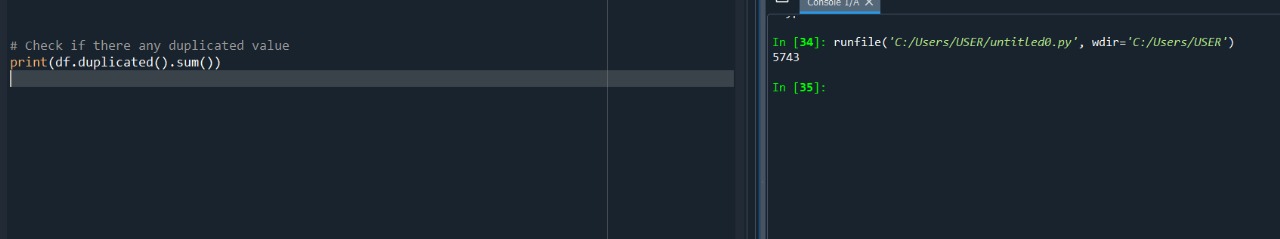
1. **Checking for Null Values After Filling**

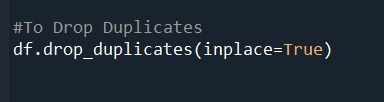
The df.isnull().sum() method is used to check for any remaining missing values in the dataset after filling them. It returns the count of null values for each column. If all values are 0, it confirms that missing data has been successfully handled. This step ensures data completeness before further analysis.

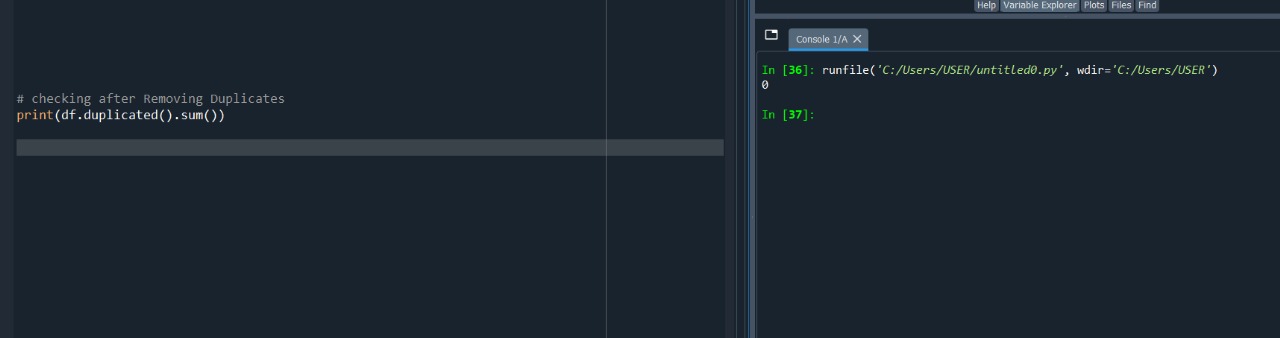
****

****

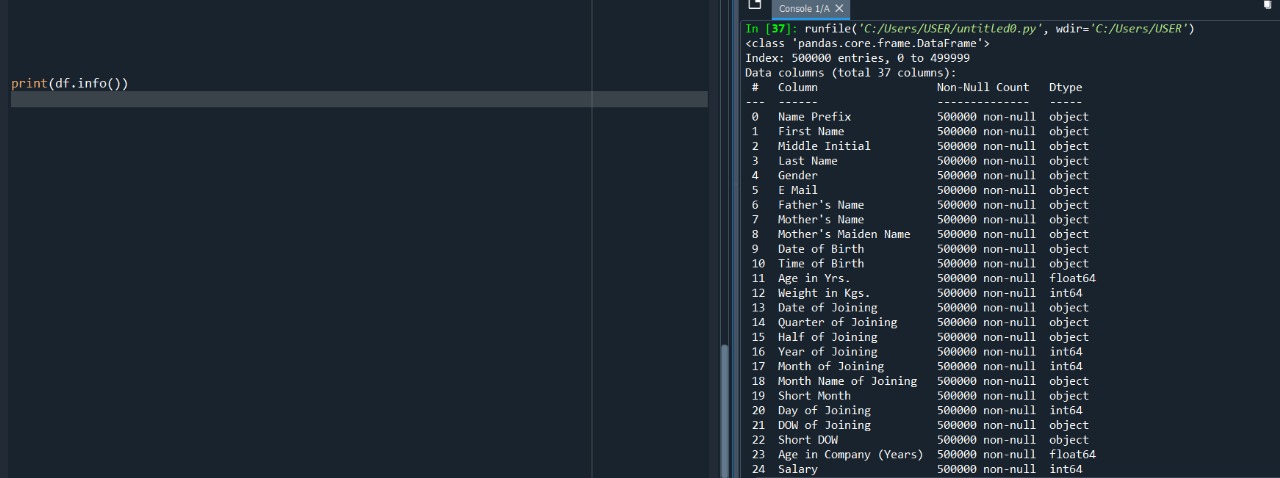
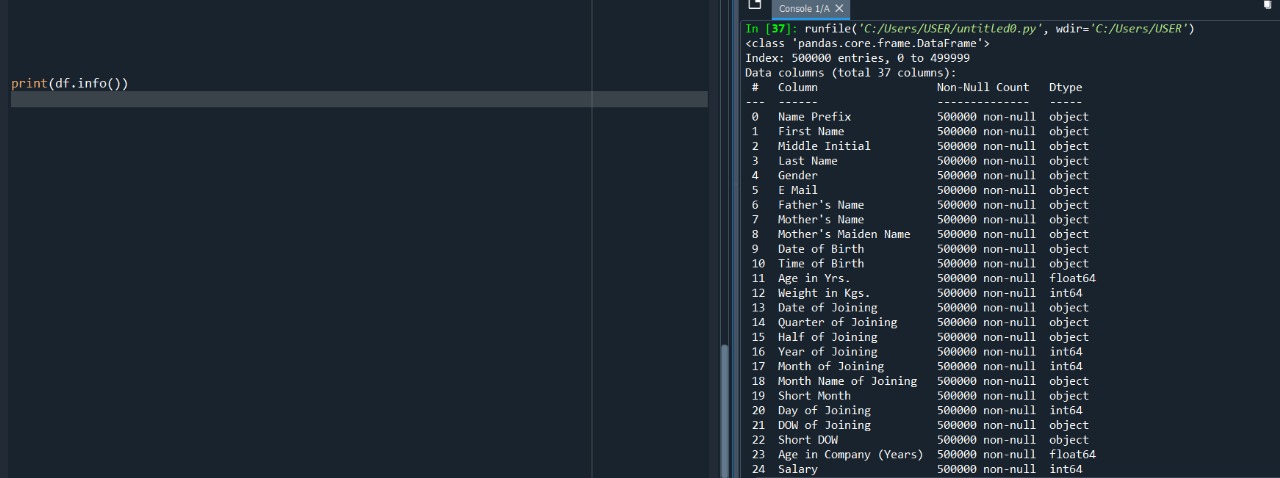
1. **Checking for Duplicate Values**

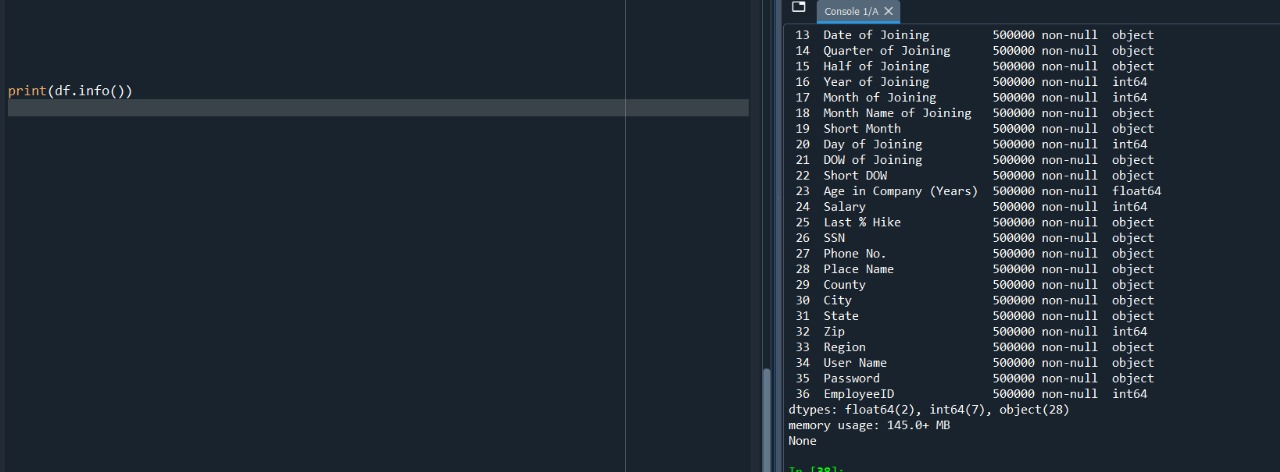
****The df.duplicated().sum() method checks for duplicate rows in the dataset. It returns the total count of duplicate rows, helping to identify redundant entries that may affect data integrity. In this case, the output shows 5743 duplicate rows, indicating that further action, such as removal or handling of duplicates, may be required.

1. **Dropping Duplicate Rows**  
   The df.drop\_duplicates(inplace=True) method removes duplicate rows from the dataset. The inplace=True parameter ensures that the changes are applied directly to the DataFrame without needing to reassign it. This step is crucial for maintaining data integrity and preventing redundant entries from skewing analysis.
2. **Verifying Duplicate Removal in the Dataset**

After removing duplicate rows, the df.duplicated().sum() method is used to check if any duplicates remain in the DataFrame. This function returns the count of duplicate rows, ensuring data consistency. The output 0 in the console confirms that all duplicate entries have been successfully removed, and the dataset is now free from redundancies. This step is crucial in data preprocessing to maintain accuracy and reliability in further analysis.

1. **Getting basic information about the dataset**

The data.info() method provides a ‘summary’ of the DataFrame. It includes: - The number of entries (rows). - The column names. - The number of non-null (non-missing) entries in each column. - The data type of each column (e.g., integer, float, string).

****

1. **Statistical Overview of the Dataset**

To gain a better understanding of the dataset, the df.describe() function was utilized. This function provides a comprehensive statistical summary of all numerical columns, offering key insights into the dataset’s distribution and characteristics.

The output includes:

**Count**: The total number of non-null values for each variable, ensuring data completeness.

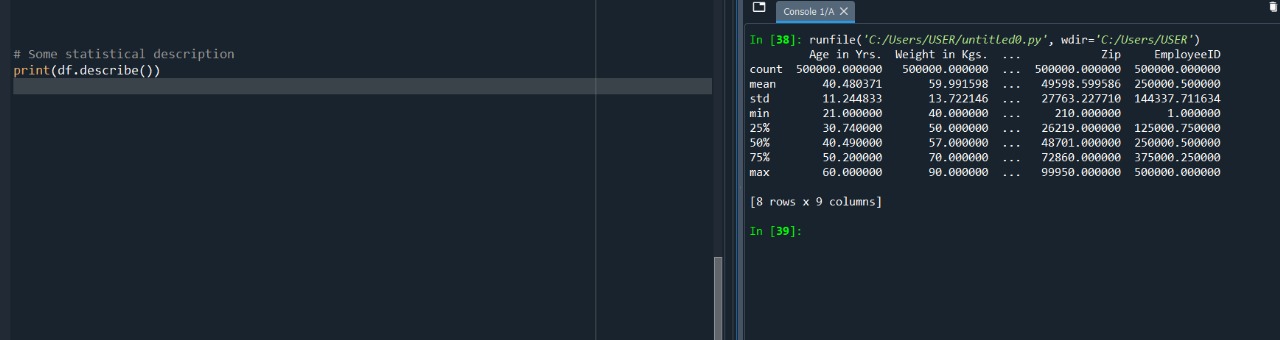
**Mean**: The average value of each numerical attribute, giving an estimate of central tendency.

**Standard Deviation (std):** A measure of data dispersion, indicating how spread out the values are.

**Minimum (min) and Maximum (max):** The smallest and largest values recorded in each column.

**Quartiles (25%, 50%, 75%):** These percentiles provide insights into the distribution and median values of the dataset.

This statistical summary is crucial in the data preprocessing stage, allowing for the detection of potential anomalies, missing values, and overall data patterns. Understanding these metrics helps ensure the dataset’s accuracy and reliability before proceeding with further analysis or model development**.**



* **Build Data Module:**

Previously, we examined the **original structure** of the dataset, which contained redundant and unstructured data. To improve **data integrity, eliminate redundancy, and optimize performance**, we performed **normalization using SQL, and then we applied a star schema for fast querying and reporting.**

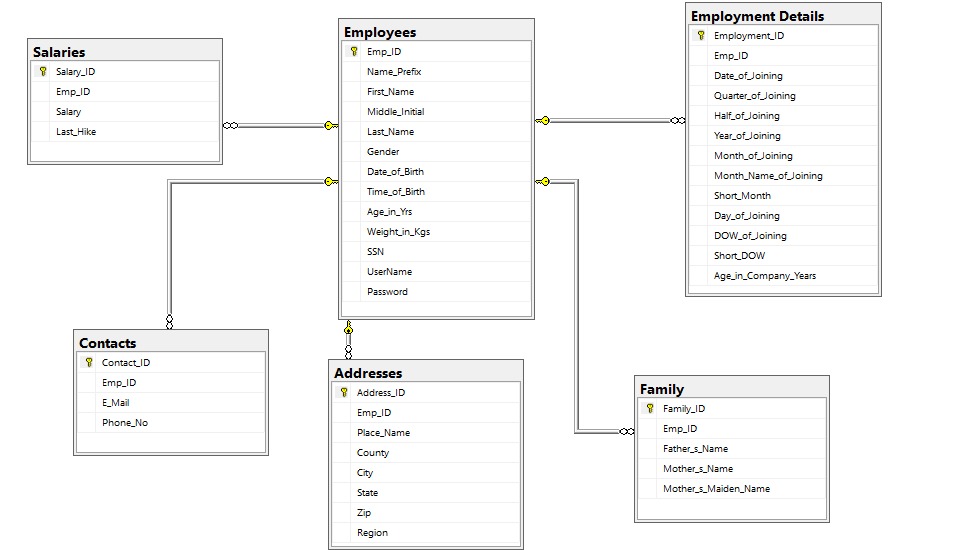
**Key improvements after normalization:**

* **Eliminated data redundancy** by breaking large tables into smaller, related tables.
* **Established clear relationships** between entities using **primary keys (PK) and foreign keys (FK)**.
* **Improved data consistency** by ensuring that each attribute is stored in the most appropriate table.
* **Enhanced database efficiency** by making queries more structured and reducing storage needs.

**Key improvements after applying star schema design:**

* **Simple & Easy to Understand –** The design is intuitive and easy to query**.**
* **Faster Query Performance** – Uses **denormalized** tables, reducing complex joins.
* **Efficient for OLAP & Reporting –** Ideal for Business Intelligence (BI) tools like Power BI & Tableau.
* **Better Aggregation –** Designed for quick summarization of data**.**

Below is the **normalized database model** after applying **SQL normalization techniques and then Star schema design:**



This new structure makes the dataset more **organized, scalable, and easier to manage** for further data analysis.

* **Analysis Questions Phase:**

Before we start analyzing data and asking business questions**,** we applied the exploratory data analysis (**EDA**) step, which helps to understand the dataset’s structure, detect patterns, and identify any issues. The data has been prepared for this phase.

We conducted an initial study of the Human Resources (HR) business to define the strategic questions and Key Performance Indicators (KPI’s), as well as to determine the objectives we will focus on in the coming period.

The study helped us understand the overall context of HR data, enabling us to prioritize and direct the analysis in alignment with business goals**.**