Dataset and Data Cleaning Overview

Note

For the specific Python codes to perform the data cleaning steps outlined here, please refer to this file in the repository: data_cleaning_steps.txt

Dataset Description

The dataset used in this project (occupation_dataset.csv) is entirely fictional and was created for instructional purposes to demonstrate data cleaning and transformation workflows in Python. The dataset simulates information about employees in an organizational setting.

Variables in the Original Dataset

The dataset contains the following columns:

- **Employee ID**: A unique identifier for each employee
- **Age**: Age of the employee in years
- **Gender**: Gender of the employee
- **Department**: The department in which the employee works (e.g., Marketing, Finance)
- **Job Satisfaction**: A self-reported score of job satisfaction (scale ranging from 1 to 10, where higher scores indicate greater satisfaction)
- **Stress Level**: A self-reported score of stress level (scale ranging from 1 to 10, where higher scores indicate greater stress)
- **Annual Salary**: The employee's annual salary, in thousands of dollars
- **Overtime Hours/Month**: Number of overtime hours worked per month
- **Manager Rating**: A manager's performance rating for the employee (scaling ranging from 1 to 5, where higher scores indicate more favorable ratings)
- **Promotion in Last 5 Years**: Whether the employee was promoted in the last 5 years (Yes/No)
- **Date Hired**: The date (year, followed by numerical indicator of month) that the employee was hired (e.g., 2007/04)
- **Parent Education Level**: The highest education level attained by the employee's parents.
- **Favorite Activity**: The employee's favorite activity
- **Job Description**: A text description of the employee’s job role, including skills necessary for the job

Workflow Overview

We will use Python to clean the dataset and prepare it for analysis. Here are the data cleaning steps we will be performing; each addresses common challenges in data cleansing.

1. *Drop irrelevant columns*
2. *Standardizing values in a column by handling lowercase values and abbreviations*

3. *Filter employees by department, meaning we will choose only employees from certain departments to be included in subsequent analyses*
4. *Drop invalid cases*
5. *Handling missing values*
6. *Spitting data in one column into separate columns*
7. *Removing unwanted strings in a column with numeric data*
8. *Count the number of cases/rows containing specific keywords or substrings in a column containing textual information (language translation)*

1. Drop irrelevant columns
'Parent Education Level' and 'Favorite Activity' are two columns that obviously contain irrelevant (and unneeded) information, so we would want to drop them from any further analyses. We will also drop the column "Unnamed: 0", which represents the far-left unnamed column in the CSV file (starting with values like 1). This column is unnecessary because it duplicates the indexing already provided by the DataFrame. After running this code, the first row of the dataset is re-indexed, starting at 0.

2. Standardizing values in a column by handling lowercase values and abbreviations
The values in the "Gender" column should read either as "Male" or "Female," which most already do. However, some values were entered as abbreviations (M, F, m, f), or in a case-insensitive format (male, female). To ensure consistency and accuracy, we will clean these inconsistent entries by converting them to the proper format.
What they Python code does: The code standardizes the values in the "Gender" column of the dataset. It first strips any whitespace, converts all text to lowercase, and maps the abbreviations "f" and "m" to "Female" and "Male," respectively. If a value doesn't match any of the specified mappings, the original value is retained. In the second step, it replaces uppercase "F" and "M" with "Female" and "Male" to ensure consistency across the column.

3. Filter employees by department
We will filter the dataset to include only employees from the "Marketing," "Finance," "Sales," and "Operations" departments (from the "Department" column), excluding employees from other departments (e.g., IT). This step ensures that our analysis focuses on relevant groups and excludes data that falls outside the scope of our objectives.
What the Python code does: It filters the dataset to focus on employees in the specific departments we want. It first calculates the number of rows that belong to unrelated departments by subtracting the count of rows in the target departments from the total number of rows. Then, it uses **drop()** to remove rows where the "Department" column does not match any of the target values, using a condition with multiple checks **(!=)** combined with the **&** operator. The dataset is thus reduced to only include the specified departments for further analysis.

4. Drop invalid cases
The column “Promotion in Last 5 Years” is supposed to have only two values: Yes or No. However, there are several cases in which the value is “-1” (which likely indicates a missing value). The plan here is to remove the rows containing these invalid cases from the dataset. This step eliminates erroneous or placeholder entries, ensuring the dataset contains only accurate and meaningful information for analysis.

5. Handling missing values
We will address missing values in the "Stress Level" column. Missing values will usually either appear as a “NaN” value, or just a blank cell. There are several methods to handle this situation properly, including imputation by mean, median, or mode (the first two when the data are normally distributed, the last when the data are skewed). Another method is using forward fill imputation, where you replace a missing value with the previous value. Finally, rows with missing values may be deleted. Properly handling missing values is crucial in avoiding potential problems in subsequent analyses.

6. Spitting data in one column into separate columns
The values in the “Date Hired” column are listed in the format of year and month an employee was hired, in numeric form (e.g., 2007/04 indicates April of 2007). Here, we want to split the data in this column into two separate columns: “Year Hired” and “Month Hired.” The values in the “Year Hired” will be the year the employee was hired, and the values in new “Month Hired” column will represent the month of the year the employee was hired. This transformation provides more granular information, making it easier to analyze patterns or trends based on the year or month of hiring.
What the Python code does: The **str.split()** function separates the values in the “Date Hired” column based on the "**/**" delimiter, with expand=True ensuring the results are placed in separate columns. The new columns are then converted to integers using the **astype(int)** function for numerical operations. After this, the original “Date Hired” column is removed using **drop(columns=[‘Date Hired’])**.

7. Removing unwanted strings in a column with numeric data
We need to clean the "Annual Salary" column by removing unwanted strings,"K(est.)," and retaining only the numeric values. This step ensures the column is formatted correctly for numerical analysis.
What the Python code does: the code removes the substring K(est.) using the **str.replace()** function, with the **regex=True** argument enabling the use of regular expressions for pattern matching. After removing the substring, the values in the column are converted to floats using the **astype(float)** function, allowing for numerical operations on the cleaned salary data.

8. Count the number of cases/rows containing specific keywords or substrings in a column containing textual information (language translation)
Let’s suppose we are interested in identifying the skills necessary for the job role of “Operations,” which is one of the departments listed in the “Department” column. We will analyze the “Job Description” column to count the number of cases that include the following

specific keywords or substrings: “data analysis,” “SQL,” “Python,” “Power BI,” “social media,” “Excel,” “R,” and “CRM” in the job description specifically for the role of “Operations.” This step is especially useful in identifying trends and skill requirements within the data.
What the Python code does: It starts by defining the list of skills to search for and creates a regex pattern using the **|** operator with **join()**, enabling the pattern to match any skill in the list. The dataset is then filtered to include only rows where the department is "Operations." For each skill, the **str.contains()** function checks if the skill appears in the Job Description column, with **case=False** ensuring case-insensitivity and **na=False** handling missing values. The counts of each skill are stored in a dictionary, which is converted into a DataFrame for easy viewing. Finally, the results, specific to the Operations department, are displayed.