Project 3 Report on:

Missing Data Cleaner

By

Ratnesh Ranjan

Roll – 2312res506

Contents

Project Workflow:

Overview

The “Missing Data Cleaner” project is a Python-based tool designed to address the common data science challenge of handling missing values in datasets. Missing data can significantly impact the reliability of analyses, making it essential to clean datasets effectively. This program detects missing values in numeric columns and fills them using statistical methods (mean, median, or mode), ensuring the dataset is ready for downstream analysis. This workflow provides a detailed explanation of user-friendly data cleaning python program for data input, cleaning, analysis, visualization, and output generation, with robust error handling and flexible input methods.

Project Framework

1. **Objective**: To create a robust tool that detects and imputes missing values in numeric columns of a dataset, making it suitable for data analysis by providing a clean dataset.
2. **Tools**:
    a. **Python**: Core programming language for script execution.
    b. **Pandas**: For efficient data manipulation and handling of CSV files and DataFrames.
    c. **os**: For directory scanning and file management.
    d. **StringIO**: For parsing manual terminal input into a CSV-like format.
3. **Input Methods**:
    a. **CSV File Path**: Users provide a specific file path (e.g., sample.csv or C:/Users/YourName/data.csv).
    b. **Directory Scan**: Automatically selects the first CSV file (alphabetically) in the current directory, excluding cleaned_data.csv.
    c. **Manual Terminal Input**: Users enter column names and data rows interactively.
    d. **Exit Option**: Allows users to terminate the program gracefully.
4. **Output**:
    a. Cleaned dataset saved as cleaned_data.csv.
    b. Terminal display of original dataset, missing value counts, cleaned dataset, and fill values used.

Workflow Steps

1. Data Input Handling

- **Purpose**: To collect a dataset with user-defined columns and data, supporting multiple input methods to accommodate different user preferences and scenarios.
- **Detailed Approach**:
    a. **Option 1: CSV File Path**:
        i. Users input a file path to a CSV file (e.g., sample.csv or a full path like C:/Users/YourName/data.csv).
        ii. The program uses pandas.read_csv() to load the file into a DataFrame.
        iii. A retry loop handles invalid paths (e.g., FileNotFoundError) by prompting the user to try again or type 'back' to return to the main menu.
        iv. This method is ideal for users with a specific dataset file in a known location.

```
Choose how to provide your dataset:
1. Enter a CSV file path (e.g., 'sample.csv' or 'C:/Users/YourName/data.csv')
2. Use a CSV file from the current directory (e.g., 'sample.csv')
3. Enter data manually in the terminal
4. Exit
Enter choice (1, 2, 3, or 4): 1
Enter the path to your CSV file (e.g., 'sample.csv') or type 'back' to return to main menu: D:/DataScience/RawData/MissingData.
csv

Dataset loaded successfully from D:/DataScience/RawData/MissingData.csv
```

    b. **Option 2: Directory Scan**:
        i. The program scans the current directory using os.listdir() to find all files with a .csv extension, excluding cleaned_data.csv to avoid processing the output file.
        ii. Files are sorted alphabetically, and the first CSV file is automatically selected using pandas.read_csv().
        iii. If no CSV files are found, an error message prompts the user to place a CSV file in the directory and try again or choose another input method.
        iv. This method simplifies input for users who have a CSV file in the same directory as the script.

```
Choose how to provide your dataset:
1. Enter a CSV file path (e.g., 'sample.csv' or 'C:/Users/YourName/data.csv')
2. Use a CSV file from the current directory (e.g., 'sample.csv')
3. Enter data manually in the terminal
4. Exit
Enter choice (1, 2, 3, or 4): 2

Dataset loaded successfully from MissingData.csv
```

    c. **Option 3: Manual Terminal Input**:
        i. Users enter data interactively, with the first line specifying column names (e.g., Name,Age,Salary) and subsequent lines providing data rows (e.g., Ravi,28, , Meena,,45000).

ii. Input is collected until the user presses Enter twice (empty line).

iii. The input is parsed into a DataFrame using StringIO and pandas.read_csv().

iv. Clear instructions guide users to enter data in CSV format, with examples provided to ensure correct formatting.

v. This method is suitable for small datasets or users without a prepared CSV file.

```
Choose how to provide your dataset:
1. Enter a CSV file path (e.g., 'sample.csv' or 'C:/Users/YourName/data.csv')
2. Use a CSV file from the current directory (e.g., 'sample.csv')
3. Enter data manually in the terminal
4. Exit
Enter choice (1, 2, 3, or 4): 3


Enter dataset in CSV format (press Enter twice to finish).

First row will be considered as column names,
You can Enter any number of Column e.g., Name,Age,Salary,....

Then Second and further row will be considered as data rows,
Enter data row in same format as column e.g., 'Ravi,28, ...' or 'Meena,,45000...' or 'Kumar,30,50000...'
Enter row no. 1: Name,Age,Salary
Enter row no. 2: Ravi,28,
Enter row no. 3: Meena,,45000
Enter row no. 4: Kumar,30,50000
Enter row no. 5:
Dataset loaded successfully from terminal input.
```

d. **Option 4: Exit**:

i. Users can exit the program at any time by selecting option 4, ensuring a graceful termination with a farewell message.

```
Choose how to provide your dataset:
1. Enter a CSV file path (e.g., 'sample.csv' or 'C:/Users/YourName/data.csv')
2. Use a CSV file from the current directory (e.g., 'sample.csv')
3. Enter data manually in the terminal
4. Exit
Enter choice (1, 2, 3, or 4): 4
Exiting the program. Goodbye!
```

e. The loaded dataset is displayed using df.to_string() to confirm successful input and allow users to verify the data structure (columns and rows).

f. Example:

1.

```
Original dataset:
    Name   Age   Salary
0   Ravi  28.0      NaN
1  Meena   NaN  45000.0
2  Kumar  30.0  50000.0
```

2.

```
Dataset loaded successfully from MissingData.csv

Original dataset:
      Name   Age   Salary  Height  Weight
0    Aryan  21.0  67000.0     NaN    70.0
1    Ayush  24.0  80000.0     5.5    75.0
2  Ratnesh  25.0      NaN     5.9    76.0
3    Rohit   NaN  90000.0     NaN    64.0
4      Ram   NaN      NaN     6.4    77.0
5   Satyam  27.0  99000.0     5.3     NaN
6   Aditya  20.0  65000.0     5.2    80.0
7   Raunak  21.0      NaN     6.0    55.0
```

- **Error Handling**:
    a. Handles FileNotFoundError for invalid file paths or missing files in the directory.
    b. Handles parsing errors for terminal input (e.g., mismatched columns or non-CSV format).
    c. Retries are offered for all errors, allowing users to correct inputs or return to the main menu.
    d. Example:

```
try:
    df = pd.read_csv(file_path)
    print()
    print(f"Dataset loaded successfully from {file_path}")
    return df, None
except FileNotFoundError:
    print("Error: File not found. Please check the file path and try again.")
except Exception as e:
    print(f"Error loading file: {e}")
```

2. Data Cleaning and Preprocessing

- **Purpose**: To ensure the dataset is valid, consistent, and ready for missing value imputation, focusing on numeric columns for statistical processing.
- **Detailed Approach**:
    a. **Missing Value Detection**: Use df.isnull() to identify missing values (NaN) in the dataset. This is critical to determine which columns and rows require cleaning.
    b. **Data Validation**:
        i. For terminal input, validate that at least one row of data is provided after column names.
        ii. Ensure CSV parsing handles varying numbers of columns by relying on pandas' robust CSV reader.
        iii. Non-numeric columns (e.g., Name) are preserved but excluded from imputation to avoid errors with statistical methods.
    c. **Data Type Handling**: Identify numeric columns (float64, int64) using df.select_dtypes() to ensure only appropriate columns are processed for mean, median, or mode calculations.
    d. **Error Handling**: Catch parsing errors during terminal input (e.g., malformed CSV) or file loading (e.g., incorrect file format) and prompt the user to retry or return to the main menu.

- **Example**:

```python
def fill_missing_values(df, method):
    # Filling missing values in numeric columns using specified method
    df_filled = df.copy()
    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
    fill_values = {}

    for column in numeric_columns:
        if method == 'mean':
            fill_value = df[column].mean()
        elif method == 'median':
            fill_value = df[column].median()
        elif method == 'mode':
            fill_value = df[column].mode()[0] if not df[column].mode().empty else None
        else:
            return df_filled, f"Invalid method: {method}. Choose 'mean', 'median', or 'mode'."
        if fill_value is not None:
            df_filled[column] = df_filled[column].fillna(fill_value)
            fill_values[column] = fill_value
    return df_filled, None, fill_values
```

3. Data Analysis

- **Purpose**: To analyze missing values and fill them using user-selected statistical methods, ensuring the dataset is complete for further use.
- **Detailed Approach**:
    a. **Missing Value Summary**: Display the count of missing values per column using df.isnull().sum() in a clear, text-based format.
    b. **Filling Method Selection**:
        i. Prompt the user to choose a method (mean, median, or mode) with a sample input (e.g., mean).
        ii. Validate the input in a retry loop, ensuring only valid methods are accepted.
    c. **Imputation**:
        i. For each numeric column with missing values:
            1. **Mean**: Calculate the average of non-missing values using df[column].mean().
            2. **Median**: Calculate the middle value using df[column].median().
            3. **Mode**: Select the most frequent value using df[column].mode()[0].
        ii. Apply the selected method using df.fillna() and store fill values for reporting.
    d. **Validation**: Display the cleaned dataset and fill values used to confirm successful imputation.
- **Error Handling**: Handle cases where no valid fill value is available (e.g., empty mode) by skipping imputation for that column and continuing with others.

- **Example**:

```
Original dataset:
      Name   Age   Salary  Height  Weight
0    Aryan  21.0  67000.0     NaN    70.0
1    Ayush  24.0  80000.0     5.5    75.0
2  Ratnesh  25.0      NaN     5.9    76.0
3    Rohit   NaN  90000.0     NaN    64.0
4      Ram   NaN      NaN     6.4    77.0
5   Satyam  27.0  99000.0     5.3     NaN
6   Aditya  20.0  65000.0     5.2    80.0
7   Raunak  21.0      NaN     6.0    55.0

Missing values in each column:
Name      0
Age       2
Salary    3
Height    2
Weight    1

Choose a method to fill missing values (enter 'mean', 'median', or 'mode'):
Example: mean
Enter method: mean

Cleaned dataset:
      Name   Age   Salary    Height  Weight
0    Aryan  21.0  67000.0  5.716667    70.0
1    Ayush  24.0  80000.0  5.500000    75.0
2  Ratnesh  25.0  80200.0  5.900000    76.0
3    Rohit  23.0  90000.0  5.716667    64.0
4      Ram  23.0  80200.0  6.400000    77.0
5   Satyam  27.0  99000.0  5.300000    71.0
6   Aditya  20.0  65000.0  5.200000    80.0
7   Raunak  21.0  80200.0  6.000000    55.0

Fill values used:
Age: 23.0
Salary: 80200.0
Height: 5.716666666666666
Weight: 71.0
```

4. Data Visualization

- **Purpose**: To present the original dataset, missing value counts, cleaned dataset, and fill values in a clear, text-based format for user verification.
- **Detailed Approach**:
    a. **Original Dataset**: Display the full dataset using df.to_string() to show all columns and rows, including missing values (NaN).
    b. **Missing Value Counts**: Present a text-based table of missing values per column using df.isnull().sum().to_string().
    c. **Cleaned Dataset**: Display the imputed dataset using df_cleaned.to_string() to allow comparison with the original.
    d. **Fill Values**: List the values used for imputation (e.g., Age: 29.0, Salary: 47500.0) in a concise format.
    e. **Usability**: The text-based output is formatted for readability, with clear section headers and spacing to separate different outputs.

- **Example**:

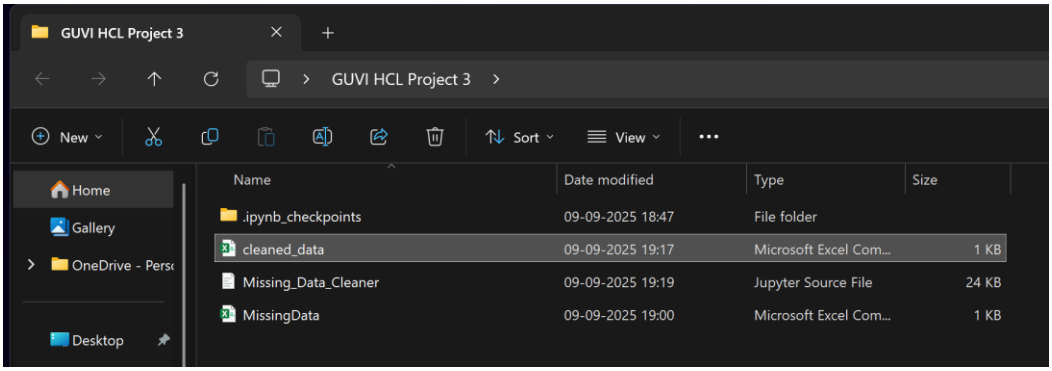
```
Original dataset:
      Name   Age   Salary  Height  Weight
0    Aryan  21.0  67000.0     NaN    70.0
1    Ayush  24.0  80000.0     5.5    75.0
2  Ratnesh  25.0      NaN     5.9    76.0
3    Rohit   NaN  90000.0     NaN    64.0
4      Ram   NaN      NaN     6.4    77.0
5   Satyam  27.0  99000.0     5.3     NaN
6   Aditya  20.0  65000.0     5.2    80.0
7   Raunak  21.0      NaN     6.0    55.0

Missing values in each column:
Name      0
Age       2
Salary    3
Height    2
Weight    1

Choose a method to fill missing values (enter 'mean', 'median', or 'mode'):
Example: mean
Enter method: mean

Cleaned dataset:
      Name   Age   Salary    Height  Weight
0    Aryan  21.0  67000.0  5.716667    70.0
1    Ayush  24.0  80000.0  5.500000    75.0
2  Ratnesh  25.0  80200.0  5.900000    76.0
3    Rohit  23.0  90000.0  5.716667    64.0
4      Ram  23.0  80200.0  6.400000    77.0
5   Satyam  27.0  99000.0  5.300000    71.0
6   Aditya  20.0  65000.0  5.200000    80.0
7   Raunak  21.0  80200.0  6.000000    55.0

Fill values used:
Age: 23.0
Salary: 80200.0
Height: 5.716666666666666
Weight: 71.0
```

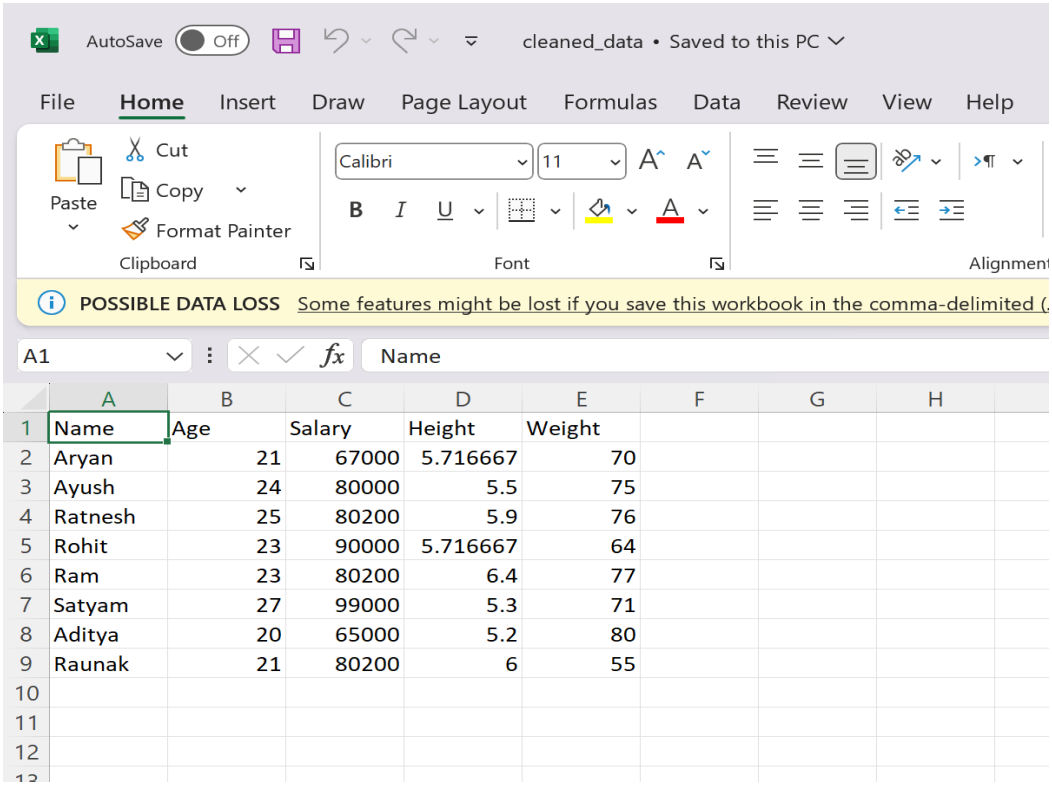
5. Cleaned Dataset File Generation

- **Purpose**: To save the cleaned dataset as a reusable output file for further analysis or sharing.
- **Detailed Approach**:
    a. Automatically save the cleaned dataset to cleaned_data.csv using df.to_csv(index=False) in the current directory.
    b. Display a confirmation message with the file path (e.g., Cleaned dataset saved to cleaned_data.csv).
    c. Ensure non-numeric columns (e.g., Name) are preserved in the output file to maintain the dataset's integrity.
    d. Handle potential file-saving errors (e.g., permission issues) using try-except and display an error message if saving fails.

✓ **Example**:

```
Cleaned dataset saved to cleaned_data.csv
```





	A	B	C	D	E	F	G	H
1	Name	Age	Salary	Height	Weight			
2	Aryan	21	67000	5.716667	70			
3	Ayush	24	80000	5.5	75			
4	Ratnesh	25	80200	5.9	76			
5	Rohit	23	90000	5.716667	64			
6	Ram	23	80200	6.4	77			
7	Satyam	27	99000	5.3	71			
8	Aditya	20	65000	5.2	80			
9	Raunak	21	80200	6	55			
10								
11								
12								

6. Error Handling and Code Structure

- **Purpose**: To ensure the program is robust, user-friendly, and maintainable, with clear code organization and error recovery.
- **Detailed Approach**:
    a. **Error Handling**:
        i. **File Input Errors**: Catch FileNotFoundError for invalid file paths (Option 1) or missing CSV files (Option 2), prompting retries or return to the main menu.
        ii. **Terminal Input Errors**: Handle malformed CSV input (e.g., mismatched columns) with clear error messages and retries.
        iii. **Invalid Choice Errors**: Validate main menu choices (1, 2, 3, 4) and filling methods (mean, median, mode), offering retries for invalid inputs.
        iv. **Exit Option**: Allow graceful exit with Option 4, displaying a farewell message.

b. **Code Structure**:

   i. Organized into modular functions: load_dataset_from_path, load_dataset_from_directory, load_dataset_from_terminal, display_missing_values, fill_missing_values, save_cleaned_dataset.

   ii. Each function has a single responsibility, improving readability and maintainability.

   iii. Descriptive variable names (e.g., df, df_cleaned, fill_values) enhance code clarity.

   iv. Concise comments explain key logic (e.g., # Allow retries for file path input until valid or user returns to main menu).

c. **Retry Mechanism**:

   i. Main menu loop allows retries for invalid choices or failed inputs.

   ii. File path input (Option 1) offers retries or 'back' to main menu.

   iii. Filling method input includes a retry loop for invalid methods.

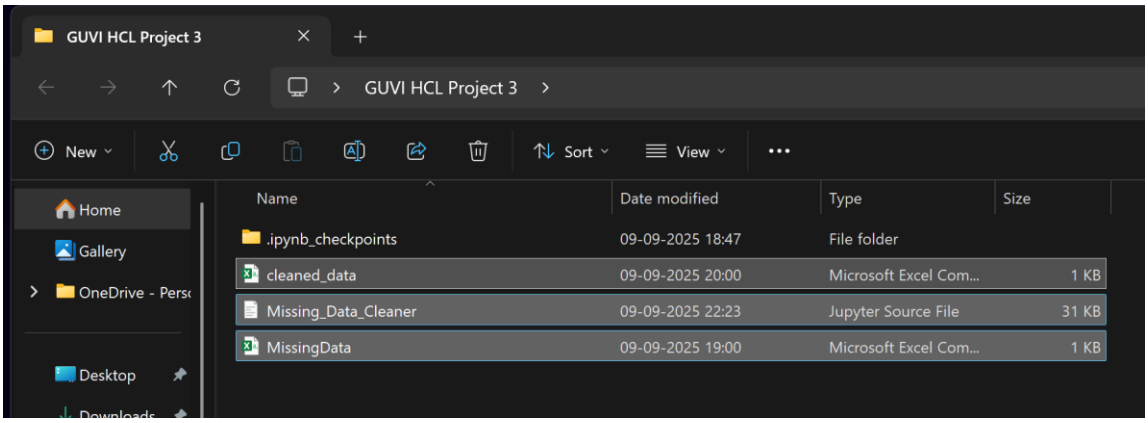
d. **Example**:

```
Welcome to the Missing Data Cleaner!
This tool cleans missing values in a dataset using mean, median, or mode.

Choose how to provide your dataset:
1. Enter a CSV file path (e.g., 'sample.csv' or 'C:/Users/YourName/data.csv')
2. Use a CSV file from the current directory (e.g., 'sample.csv')
3. Enter data manually in the terminal
4. Exit
Enter choice (1, 2, 3, or 4): C:/User/Ratnesh/Data.csv
Invalid choice! Please enter 1, 2, 3, or 4. Try again.
```

```
Welcome to the Missing Data Cleaner!
This tool cleans missing values in a dataset using mean, median, or mode.

Choose how to provide your dataset:
1. Enter a CSV file path (e.g., 'sample.csv' or 'C:/Users/YourName/data.csv')
2. Use a CSV file from the current directory (e.g., 'sample.csv')
3. Enter data manually in the terminal
4. Exit
Enter choice (1, 2, 3, or 4): 1
Enter the path to your CSV file (e.g., 'sample.csv') or type 'back' to return to main menu: C:/User/Ratnesh/Data.csv
Error: File not found. Please check the file path and try again.
Enter the path to your CSV file (e.g., 'sample.csv') or type 'back' to return to main menu: sample.csv
Error: File not found. Please check the file path and try again.
```

Deliverables

1. **Python Script**: missing_data_cleaner.ipynb containing the complete program with input handling, cleaning, analysis, visualization, and output generation.
2. **Output File**: cleaned_data.csv, containing the cleaned dataset with missing values filled in numeric columns.
3. **Optional Input File**: sample.csv with sample data (e.g., columns: Name,Age,Salary) for testing.



Conclusion

The Missing Data Cleaner project provides a robust, user-friendly solution for handling missing data in datasets, a critical step in data science workflows. Its flexible input methods (CSV file path, automatic directory selection, manual terminal input, and exit option) cater to diverse user needs. The program's retry mechanisms and clear error messages ensure accessibility for beginners, while its use of pandas for efficient data handling and statistical imputation (mean, median, mode) meets data science standards. The text-based visualization and automatic saving to cleaned_data.csv make the output accessible and reusable. Future improvements could include support for non-numeric column imputation (e.g., filling with most frequent string) or graphical visualizations using matplotlib.

Source Code GitHub Repository link :

https://github.com/ratneshranjan484/GUVI-HCL-Project-3-Missing-Data-Cleaner

Note :

- **Source file (Missing_Data_Cleaner.ipynb) is a Jupiter NoteBook file. So, Use Jupiter NoteBook run the program.**
- **Ensure pandas is installed (install pandas) before running the program.**
- **Place a sample CSV file (e.g., sample.csv) in the same directory for Option 2, or use a full file path for Option 1.**