Exploratory Data Analysis (EDA)

Missing Compeletly at Random (MCAR)

Instructor: Prof. Gopinath Panda

Group - 16

Contributing Members:

Name	**Student ID**
Dev Vyas	202201453
Dharmi Patel	202201467
Preksha Shah	202203004

Introduction

Missing data is a common issue in real-world datasets and can significantly impact the results of data analysis and model predictions. Data can be missing for various reasons, such as measurement errors, equipment failure, or respondents choosing not to answer particular questions in a survey. To deal with missing data effectively, it is crucial to understand the underlying mechanism that governs the missingness.

Missing data mechanisms are broadly classified into three categories:

- Missing Completely At Random (MCAR),
- Missing At Random (MAR),
- Missing Not At Random (MNAR).

In this paper, we will focus on the Missing Completely At Random (MCAR) mechanism, which is the most stringent and easiest to handle statistically. Specifically, we will describe the MCAR mechanism and introduce Little's MCAR test to assess whether data is missing completely at random.

Missing Completely At Random (MCAR)

The concept of MCAR refers to a situation where the probability of missing data is unrelated to both the observed and unobserved data. In other words, the missingness of data occurs purely by chance and is not influenced by the data itself.

Formally, if we denote the dataset as D, with D_{obs} representing the observed data and D_{mis} representing the missing data, the MCAR assumption can be stated mathematically as:

$$P(M = 1 | D_{\text{obs}}, D_{\text{mis}}) = P(M = 1)$$

where:

- M is the missingness indicator, where $M = 1$ indicates missing data, and $M = 0$ indicates observed data,
- D_{obs} is the observed portion of the dataset,
- D_{mis} is the missing portion of the dataset.

In the MCAR case, the probability of missingness does not depend on any variables in the dataset. As a result, the missing data can be ignored when performing analyses without introducing bias into the estimates, making it the most straightforward type of missingness to handle.

Little's MCAR Test

Little's MCAR test is a statistical test designed to evaluate whether missing data in a dataset is missing completely at random. The null hypothesis H_0 for the test is that the data is MCAR, while the alternative hypothesis H_1 suggests that the data is not MCAR. The test compares the observed patterns of missing data with the expected patterns under the assumption that the data is MCAR.

Mathematically, the null and alternative hypotheses can be expressed as:

$$H_0 : \text{Data are MCAR}$$

$$H_1 : \text{Data are not MCAR}$$

The test statistic used in Little's MCAR test is chi-square distributed and is calculated as:

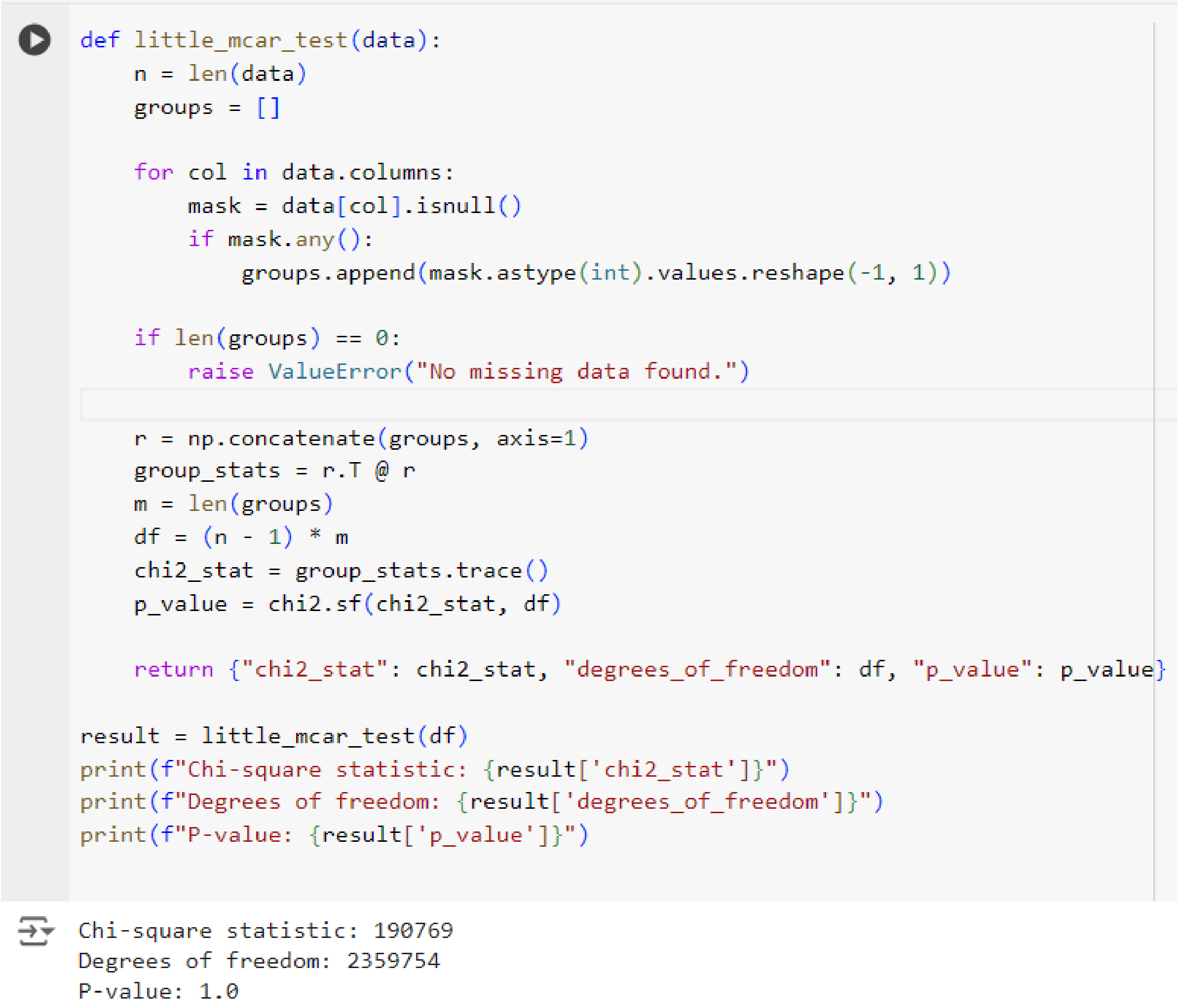
$$\chi^2 = \sum_i \sum_j \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

where:

- O_{ij} represents the observed frequencies of missing data in cell (i, j),
- E_{ij} represents the expected frequencies under the MCAR assumption.

If the p-value of the test is greater than a predefined significance level (typically 0.05), we fail to reject the null hypothesis and conclude that the data is MCAR. Conversely, if the p-value is less than the significance level, we reject the null hypothesis, implying that the data is not MCAR and may follow a different missingness mechanism, such as MAR or MNAR.

Code for Little's MCAR Test



```python
def little_mcar_test(data):
    n = len(data)
    groups = []

    for col in data.columns:
        mask = data[col].isnull()
        if mask.any():
            groups.append(mask.astype(int).values.reshape(-1, 1))

    if len(groups) == 0:
        raise ValueError("No missing data found.")

    r = np.concatenate(groups, axis=1)
    group_stats = r.T @ r
    m = len(groups)
    df = (n - 1) * m
    chi2_stat = group_stats.trace()
    p_value = chi2.sf(chi2_stat, df)

    return {"chi2_stat": chi2_stat, "degrees_of_freedom": df, "p_value": p_value}

result = little_mcar_test(df)
print(f"Chi-square statistic: {result['chi2_stat']}")
print(f"Degrees of freedom: {result['degrees_of_freedom']}")
print(f"P-value: {result['p_value']}")
```

```
Chi-square statistic: 190769
Degrees of freedom: 2359754
P-value: 1.0
```

The `little_mcar_test` function is used to perform Little's MCAR (Missing Completely

At Random) test on a dataset that contains missing values. The test checks whether the missing data is distributed randomly across the dataset. Below are the detailed steps:

- **Identifying Missing Data:** The function first checks each column for missing values. For each column that contains missing data, a binary mask is created, where:

$$\text{mask}(i, j) = \begin{cases} 1 & \text{if the value at row } i \text{ is missing in column } j \\ 0 & \text{otherwise} \end{cases}$$

  These binary masks are then concatenated into a matrix R, where each column represents a missing data pattern for a variable.

- **Chi-Square Statistic:** The matrix R is used to calculate the chi-square statistic χ^2, which measures the deviation of the observed missing data patterns from what would be expected if the data were MCAR. The chi-square statistic is given by:

$$\chi^2 = \text{trace}(R^T R)$$

  where $R^T R$ is the cross-product of the transpose of R and R, and trace refers to the sum of the diagonal elements of the resulting matrix.

- **Degrees of Freedom:** The degrees of freedom (df) for the chi-square test is calculated as:

$$df = (n - 1) \cdot m$$

  where n is the number of rows in the dataset, and m is the number of variables with missing data.

- **P-Value:** The p-value is obtained by comparing the calculated chi-square statistic to the chi-square distribution with the specified degrees of freedom:

$$p = P(\chi^2 \geq \text{observed value} | df)$$

  A high p-value (e.g., $p > 0.05$) supports the null hypothesis that the data is MCAR, indicating that the missing data is likely random. A low p-value suggests that the missing data is not random and may be dependent on other factors in the dataset.

missingpy Library

`missingpy` is a Python library designed for handling missing data through various imputation techniques. The library is built with an API consistent with `scikit-learn`, making it intuitive for users already familiar with the `scikit-learn` ecosystem. The primary goal of `missingpy` is to provide efficient and scalable methods for imputing missing data while maintaining ease of use.

Currently, `missingpy` supports the following imputation methods:

- k-Nearest Neighbors (KNN) based imputation
- Random Forest (MissForest) based imputation

KNN Imputer

The `KNNImputer` class performs missing value imputation by using the k-Nearest Neighbors (KNN) approach. Missing values in each sample are imputed by considering the values of the nearest neighbors identified in the training set. If a sample has multiple features with missing values, it may have different sets of neighbors for each missing feature, depending on the specific feature being imputed.

For each feature with missing values, imputation is done by calculating the average value of the neighboring samples. The average can be either weighted or unweighted, depending on the settings. If the number of available neighbors is smaller than `n_neighbors`, the average value from the training set for that feature is used for imputation.

The total number of samples available in the training set will always be greater than or equal to the number of nearest neighbors used for imputation. However, this depends on both the overall size of the dataset and the number of samples excluded from the neighbor calculation due to having too many missing features. The latter is controlled by the `row_max_missing` parameter.

MissForest

The `MissForest` algorithm imputes missing values using Random Forests in an iterative manner. The imputation process begins by selecting the column with the fewest missing values, referred to as the candidate column. Initially, any missing values in the remaining non-candidate columns are filled with a preliminary estimate. For numerical variables, the missing values are replaced by the column mean, while for categorical variables, the column mode is used. Categorical variables must be explicitly specified during the call to the `fit()` method.

Once the initial imputation is complete, a Random Forest model is trained using the non-missing rows of the candidate column as the outcome (target variable), and the remaining columns as predictors. The trained Random Forest is then used to predict and impute the missing values in the candidate column. The imputed rows of the candidate column are predicted based on the corresponding rows of the other columns, which serve as the input features for the model.

After imputing the missing values in the candidate column, the imputer moves on to the next column with missing data, this time selecting the column with the second fewest missing values. This process is repeated for all columns containing missing values. The imputation process may continue over multiple iterations, where the algorithm cycles through the columns until a stopping criterion is reached.

The iterative nature of the MissForest algorithm allows it to improve the accuracy of imputations over successive rounds. Each iteration refines the imputed values by updating the estimates based on the newly imputed data, thereby reducing potential biases and improving the overall quality of the imputed dataset.

MissForest is particularly advantageous because it does not assume any specific distribution for the data and can handle both numerical and categorical variables. The Random Forest-based approach allows the method to capture complex relationships between variables and provide robust imputation results. However, it is important to note that MissForest can be computationally intensive, especially for large datasets with many missing values.

Implementation Details: When using MissForest, ensuring that the data is appropriately preprocessed before imputation is necessary. This includes handling any extreme outliers and ensuring that categorical variables are properly encoded. Additionally, the choice of hyperparameters for the Random Forest, such as the number of trees and the maximum depth, can impact the performance of the imputation process.

In practice, evaluating the quality of the imputed data is crucial. This can be done by comparing the imputed values with known values in a subset of the data (if available) or by assessing the impact of imputation on subsequent analyses and models. Cross-validation techniques can also be employed to estimate the reliability of the imputation method.

Google Collab Link
Github Link

(The dataset is uploaded in the github folder)