Exploratory Data Analysis - IT 462

Assignment - 2

MCAR Test

Group:10

Team Members:

1. Hammad Farid - 202312108
2. Anmol Rangwani - 202312027
3. Yuvraj Bhadoriya - 202411002

INTRODUCTION:
Challenges of Missing Data in Exploratory Data Analysis

Missing data occurs when no value is stored for a variable in a dataset, a common issue in data analysis and machine learning. It can arise due to various reasons, such as survey non-responses, data entry errors, equipment failures, or skipped observations.

Types of Missing Data:

1. **Missing Completely at Random (MCAR)**: The missingness is entirely random and independent of the data itself.
2. **Missing at Random (MAR)**: The probability of missingness is related to the observed data but not the missing values themselves.
3. **Missing Not at Random (MNAR)**: The missingness is related to the unobserved (missing) data, making it the most challenging to address.

Challenges of Missing Data

1. **Bias**: Missing data can skew results if related to the target variable.
2. **Loss of Information**: Dropping missing values reduces data size and quality.
3. **Impact on Models**: Machine learning models struggle with missing data, affecting performance.
4. **Imputation Complexity**: Incorrect imputation can distort data and lead to errors.

The missingpy Library:

The `missingpy` library is a Python package that provides a variety of methods to fill in missing values while preserving the integrity of the dataset.

Key features include:

- KNNImputer: Uses k-nearest neighbors to estimate missing values based on similar data points.
- MissForest: An ensemble-based method that employs random forests for imputation, suitable for both numerical and categorical data.
- IterativeImputer: Models each feature with missing values as a function of other features, iteratively refining imputations.

In data science, software compatibility and dependencies can limit the use of certain libraries. Although `missingpy` provides effective imputation methods, installation challenges such as dependency conflicts or version mismatches can hinder its use. In our analysis, these issues made `missingpy` impractical, highlighting the value of alternatives like `fancyimpute`, which often has fewer installation barriers and offers robust imputation techniques.

Missing Completely at Random (MCAR)

When data is MCAR, the missingness does not depend on the data itself or any unobserved factors. This means that the missing values are essentially a random subset of the data, allowing for straightforward analysis without introducing bias.

Little's MCAR Test

Little's MCAR Test is a statistical test used to determine whether data is MCAR. It assesses the patterns of missing data and provides a chi-square statistic and corresponding p-value. The null hypothesis states that the data is MCAR, while the alternative hypothesis suggests that it is not.

Procedure

1. **Data Preparation**: Organize the dataset and identify missing values.
2. **Apply Little's MCAR Test**: Conduct the test using a statistical software package or a dedicated function in Python (e.g., using the `missingpy` or **`fancyimpute`** library).
3. **Interpret Results**:
   - If the p-value is greater than 0.05, we fail to reject the null hypothesis, indicating that the data can be considered MCAR.
   - If the p-value is less than or equal to 0.05, we reject the null hypothesis, suggesting that the missingness may not be completely random.

```
# Using the test on the filtered deteset
simple_chi_square_test(df_filtered)
```

```
~~~MCAR Test~~~
Chi-square statistic: 3.0832
P-value: 0.6872
The null hypothesis cannot be rejected. The data is MCAR.
```

For our dataset, the p-value is 0.6872 which is greater than 0.05. This shows that we fail to reject the null hypothesis, indicating that the data can be considered MCAR.

Imputation using mean, median and mode:

Mean Imputation:

- **When to Use**: For normally distributed numerical data without outliers.
- **Considerations**: Sensitive to outliers, which can skew results.

Median Imputation:

- **When to Use**: For skewed numerical data or data with outliers.
- **Considerations**: More robust against outliers, providing a better central measure.

Mode Imputation:

- **When to Use**: For categorical data or when the most frequent value matters.
- **Considerations**: Useful for nominal data but may not reflect overall distribution well.

KNN Imputation

KNN Imputation (K-Nearest Neighbors) is a method used in the `fancyimpute` library to fill in missing values by considering the values of the nearest neighbors in the dataset.

Steps for performing kNN:

1. **Install FancyImpute**: Use `pip install fancyimpute` to install the library.
2. **Prepare Dataset**: Load your dataset into a Pandas DataFrame and ensure non-numeric columns are handled if needed.KNN does not work for categorical data so we have to encode it first.
3. **Instantiate KNN Imputer**: Create a KNN imputer instance, specifying the number of neighbors (k).
4. **Fit and Transform**: Apply the KNN imputation using `fit_transform`.
5. **Convert to DataFrame**: Convert the imputed array back to a Pandas DataFrame.

`Here we are only taking the first 1000 rows for KNN imputation as it is very heavy for running on such a large data set (100514 rows).`

Iterative Imputation

Iterative Imputation is a method for filling in missing values by modeling each feature with missing values as a function of other features in the dataset. This approach uses an iterative process to refine imputations.

When to Use

- Ideal for datasets with complex interrelationships among features.
- Useful when missing values are prevalent and patterns are not easily captured.

The **IterativeImputer** uses a multivariate approach to impute missing values by modeling each feature as a function of the others. This method iteratively predicts and refines the missing values for each variable.

- **Creating the Imputer:** By instantiating `IterativeImputer()`, you set up the imputer to use regression models for each feature based on other features in the dataset.
- **Fitting and Transforming:** The method `.fit_transform(numeric_df)` first trains the imputation model on the provided data, then fills in the missing values. The output, `iterative_filled`, is a complete DataFrame where all missing values have been accurately imputed.

MissForest Imputation Method

MissForest is an imputation technique that uses a random forest algorithm to fill in missing values. It is particularly effective for datasets with mixed types of data (numerical and categorical) and leverages the power of ensemble learning.

Key Features

- **Non-parametric:** Does not assume any specific distribution for the data, making it versatile.
- **Handles Mixed Data Types:** Can impute both numerical and categorical missing values effectively.
- **Robustness:** Combines the predictions from multiple trees to improve accuracy and reduce overfitting.

When to Use

- Ideal for datasets with both numerical and categorical variables.
- Useful when there are complex relationships among features.

Steps for Implementing MissForest Imputation

1. **Install FancyImpute:** Ensure you have the library installed.
2. **Prepare Your Dataset:** Load your dataset into a Pandas DataFrame, including both numerical and categorical features.
3. **Instantiate MissForest Imputer:** Create an instance of the `MissForest` class.
4. **Fit and Transform the Data:** Use the `fit_transform` method to apply MissForest imputation on your DataFrame with missing values.
5. **Convert to DataFrame:** Convert the imputed data back to a Pandas DataFrame for analysis.

Github link:

https://github.com/yuvraj-daiict/exploratory-data-analysis/tree/main/assignment-2

ColabLink:

Group_10_Assignment2