HandlingMissingValues

October 11, 2024

0.1 Missing Values

Missing values occurs in dataset when some of the informations is not stored for a variable There are 3 mechanisms

0.1.1 1. Missing Completely at Random, MCAR:

Missing completely at random (MCAR) is a type of missing data mechanism in which the probability of a value being missing is unrelated to both the observed data and the missing data. In other words, if the data is MCAR, the missing values are randomly distributed throughout the dataset, and there is no systematic reason for why they are missing.

For example, in a survey about the prevalence of a certain disease, the missing data might be MCAR if the survey participants with missing values for certain questions were selected randomly and their missing responses are not related to their disease status or any other variables measured in the survey.

0.1.2 2. Missing at Random MAR:

Missing at Random (MAR) is a missing data mechanism where the probability of missing data is related to the observed data, but not to the values that are actually missing. In other words, the missingness is systematically related to some of the other variables that are present (observed), but not to the missing values themselves.

Key Characteristics of MAR: Missingness is Conditional on Observed Data: The probability that a value is missing can be explained by other observed variables in the dataset. For example, the likelihood of a person not reporting their income may depend on their age, gender, or education level, but not on their actual income itself. No Dependence on the Missing Values: The missingness does not depend on the values of the missing data. For example, in the case of income data, participants' decision to not report their income is influenced by other observed factors (like age or gender), but not by their true income. Systematic but Predictable: Because the missingness is related to other observed data, MAR data introduces bias, but this bias can be accounted for if the relationships between the observed data and missingness are properly modeled. This allows for more advanced methods to handle missing data, such as multiple imputation or likelihood-based techniques.

Examples of MAR:

Income Data: In a survey collecting income data, some individuals might not report their income. If the missingness is related to the person's age or gender (e.g., younger individuals or women may be more likely to skip the income question), but not to their actual income, the data is missing at random. In this case, the missingness can be explained by age or gender, both of which are observed data, but the actual income values (the missing data) do not influence the missingness.

Medical Data: Suppose you are collecting blood pressure readings, but some patients do not provide this information. If younger patients or those with healthier lifestyles are more likely to have missing blood pressure values, but the missingness is unrelated to their actual blood pressure levels, the data is considered to be MAR. Here, the missingness is related to observable characteristics like age or lifestyle, but it is not directly tied to the patients' true blood pressure values

0.2 3. Missing data not at random (MNAR)

Missing Not at Random (MNAR) is a type of missing data mechanism where the probability of missing data depends on the value of the missing data itself. This means that the reason data is missing is related to the actual values that are missing, or to other unobserved factors. In this case, the missingness introduces bias because it is directly tied to the variable of interest or unmeasured factors, making it the most challenging type of missing data to handle.

Key Characteristics of MNAR: Missingness Depends on Unobserved Data: The likelihood of a value being missing is related to the missing data itself or some other unobserved factors. In other words, the missingness is not random, nor is it explainable by the observed data alone. Bias Introduced by Missing Data: Because the missingness is related to the missing data, there is a high risk of bias in the analysis if the missing data is not accounted for properly. Complex to Handle: Unlike MCAR or MAR, MNAR data is more difficult to handle using traditional statistical methods because you can't rely on the observed data to model the missingness. More advanced and sophisticated techniques, such as sensitivity analysis or model-based approaches, are required to deal with MNAR data.

Example of MNAR:

Income and Job Satisfaction: Imagine you are collecting data on employees' income and job satisfaction in a company. If employees with lower job satisfaction are more likely to refuse to report their income, then the missingness depends on the unmeasured factor of job satisfaction. In this case, the missing data (income) is not random because the likelihood of an employee not reporting their income is related to their job satisfaction, which is not directly observed or measured. This creates a situation where the data is MNAR.

Medical Data: Consider a clinical study where patients are asked to report their symptoms. Patients with more severe symptoms might be less likely to report their symptoms due to embarrassment or discomfort. Here, the missingness depends on the severity of the symptoms (unreported data), making it MNAR

0.3 Examples

[6]:
```python
import seaborn as sns
import numpy as np
import pandas as pd

df = sns.load_dataset('titanic')
df
```

[6]:
```
     survived  pclass     sex   age  sibsp  parch     fare embarked   class  \
0           0       3    male  22.0      1      0   7.2500        S   Third
1           1       1  female  38.0      1      0  71.2833        C   First
2           1       3  female  26.0      0      0   7.9250        S   Third
3           1       1  female  35.0      1      0  53.1000        S   First
4           0       3    male  35.0      0      0   8.0500        S   Third
..        …       …     …    …    …    …       …      …     …
886         0       2    male  27.0      0      0  13.0000        S  Second
887         1       1  female  19.0      0      0  30.0000        S   First
888         0       3  female   NaN      1      2  23.4500        S   Third
889         1       1    male  26.0      0      0  30.0000        C   First
890         0       3    male  32.0      0      0   7.7500        Q   Third

       who  adult_male deck  embark_town alive  alone
0      man        True  NaN  Southampton    no  False
1    woman       False    C    Cherbourg   yes  False
2    woman       False  NaN  Southampton   yes   True
3    woman       False    C  Southampton   yes  False
4      man        True  NaN  Southampton    no   True
..     …         …   …          …    …    …
886    man        True  NaN  Southampton    no   True
887  woman       False    B  Southampton   yes   True
888  woman       False  NaN  Southampton    no  False
889    man        True    C    Cherbourg   yes   True
890    man        True  NaN   Queenstown    no   True

[891 rows x 15 columns]
```

[3]:
```python
# To check missing values in dataset
df.isnull()
```

[3]:
```
     survived  pclass    sex    age  …   deck  embark_town  alive  alone
0       False   False  False  False  …   True        False  False  False
1       False   False  False  False  …  False        False  False  False
2       False   False  False  False  …   True        False  False  False
3       False   False  False  False  …  False        False  False  False
4       False   False  False  False  …   True        False  False  False
..        …     …    …    …  …    …          …     …     …
886     False   False  False  False  …   True        False  False  False
887     False   False  False  False  …  False        False  False  False
888     False   False  False   True  …   True        False  False  False
889     False   False  False  False  …  False        False  False  False
890     False   False  False  False  …   True        False  False  False

[891 rows x 15 columns]
```

[5]:
```python
df.isnull().sum()
```

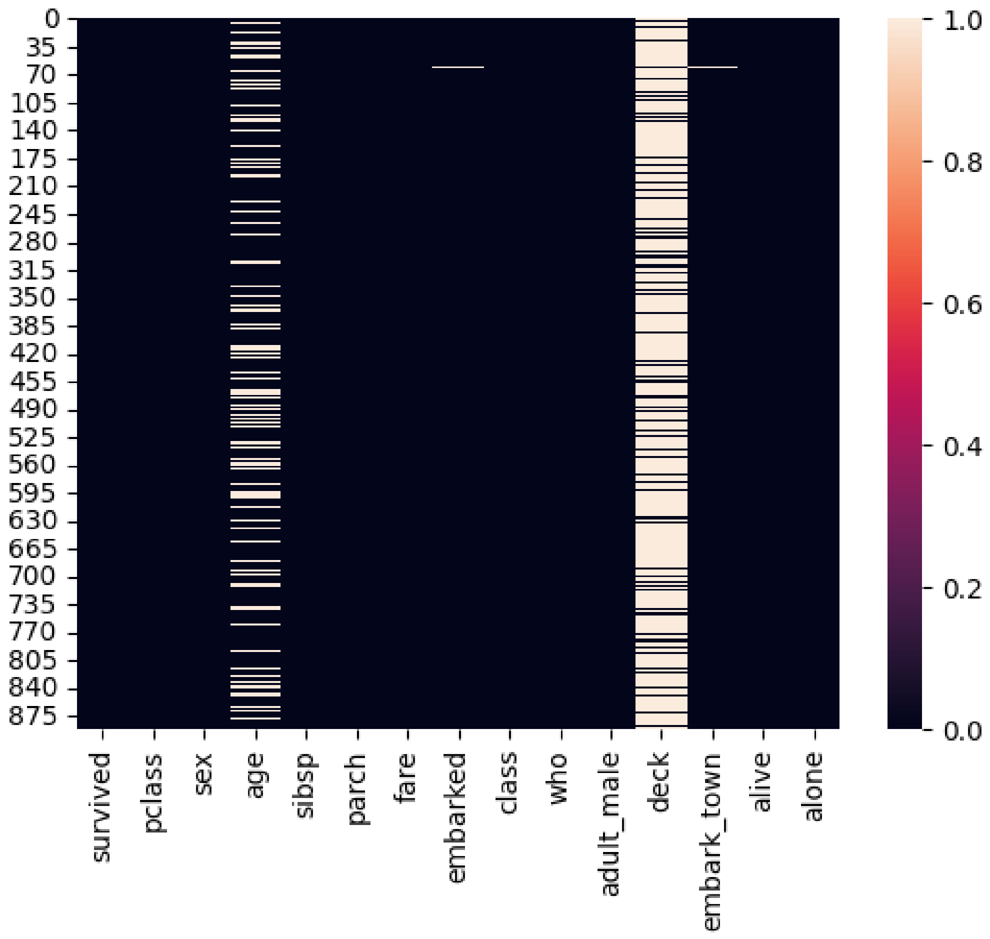
```
[5]: survived         0
     pclass           0
     sex              0
     age            177
     sibsp            0
     parch            0
     fare             0
     embarked         2
     class            0
     who              0
     adult_male       0
     deck           688
     embark_town      2
     alive            0
     alone            0
     dtype: int64
```

df.isnull(): This checks for missing values in the DataFrame df. It returns a DataFrame of the same shape as df, where each element is either True (if the corresponding value is missing) or False (if the value is not missing).

.sum(): When you apply .sum() to the result of df.isnull(), it calculates the sum of True values (which are treated as 1) for each column. This gives you the total count of missing values in each column

```
[7]: sns.heatmap(df.isnull())
```

```
[7]: <Axes: >
```



0.3.1 What is a Heatmap?

A heatmap is a graphical representation of data where individual values contained in a matrix (or 2D array) are represented by colors. Heatmaps are a popular way to visualize relationships or patterns in data, such as correlations between variables, missing data, or hierarchical data structures.

In Python, Seaborn and Matplotlib are commonly used libraries to create heatmaps. A heatmap assigns a color gradient to a range of values, making it easy to see high and low values, or other patterns, in large datasets

sns.heatmap(): This is a function from the Seaborn library that plots a heatmap. When you pass a binary DataFrame like df.isnull(), it visually represents missing data. The heatmap highlights the True (missing) values differently from the False (non-missing) values, allowing you to quickly see where the missing data is concentrated.

Interpretation: The heatmap will display a matrix where each cell represents a value in the DataFrame: Cells where data is missing (True values from isnull()) will be shown in one color. Cells with no missing data (False values) will be shown in another color. You can customize the color scheme by passing the cmap argument (e.g., 'viridis', 'coolwarm', etc.).

0.3.2 Key Points:

Color Representation: In this binary visualization (missing vs. non-missing data), one color will correspond to missing values (True), and another color will correspond to non-missing values (False). By default, sns.heatmap() often uses light (e.g., white) to indicate missing values and dark (e.g., black) for present values, but you can customize this using the cmap argument to use different color schemes if needed. #### Axes: X-axis: Represents the columns of your DataFrame. Y-axis: Represents the index (rows) of your DataFrame. So, each cell in the heatmap corresponds to an individual value in the DataFrame, positioned exactly as it is in the DataFrame. This means the heatmap is effectively a visual representation of your DataFrame's structure, showing the layout of missing and non-missing values in their actual positions.

[15]:
```
## Handling missing values by deleting rows
print(df.shape)
df.dropna()
print(df.dropna().shape)
```

```
(891, 15)
(182, 15)
```

[17]:
```
df.dropna()
```

[17]:
```
     survived  pclass     sex   age  …  deck  embark_town  alive  alone
1           1       1  female  38.0  …     C    Cherbourg    yes  False
3           1       1  female  35.0  …     C  Southampton    yes  False
6           0       1    male  54.0  …     E  Southampton     no   True
10          1       3  female   4.0  …     G  Southampton    yes  False
11          1       1  female  58.0  …     C  Southampton    yes   True
..          …       …       …     …  …   …        …    …      …
871         1       1  female  47.0  …     D  Southampton    yes  False
872         0       1    male  33.0  …     B  Southampton     no   True
879         1       1  female  56.0  …     C    Cherbourg    yes  False
887         1       1  female  19.0  …     B  Southampton    yes   True
889         1       1    male  26.0  …     C    Cherbourg    yes   True

[182 rows x 15 columns]
```

[21]:
```
## Handling missing values by deleting columns

df.dropna(axis=1)
```

[21]:
```
     survived  pclass     sex  sibsp  …    who  adult_male alive  alone
0           0       3    male      1  …    man        True    no  False
1           1       1  female      1  …  woman       False   yes  False
2           1       3  female      0  …  woman       False   yes   True
3           1       1  female      1  …  woman       False   yes  False
4           0       3    male      0  …    man        True    no   True
..          …       …       …     …  …     …        …     …      …
886         0       2    male      0  …    man        True    no   True
```

```
887           1         1  female       0  …  woman        False   yes   True
888           0         3  female       1  …  woman        False    no  False
889           1         1    male       0  …    man         True   yes   True
890           0         3    male       0  …    man         True    no   True

[891 rows x 11 columns]
```

While deleting rows or columns with missing values is a straightforward approach, it can lead to significant issues like loss of information, bias, decreased sample size, and increased variability. It's essential to carefully consider the implications of this method and explore alternative strategies for handling missing data

0.3.3 Imputation Techniques

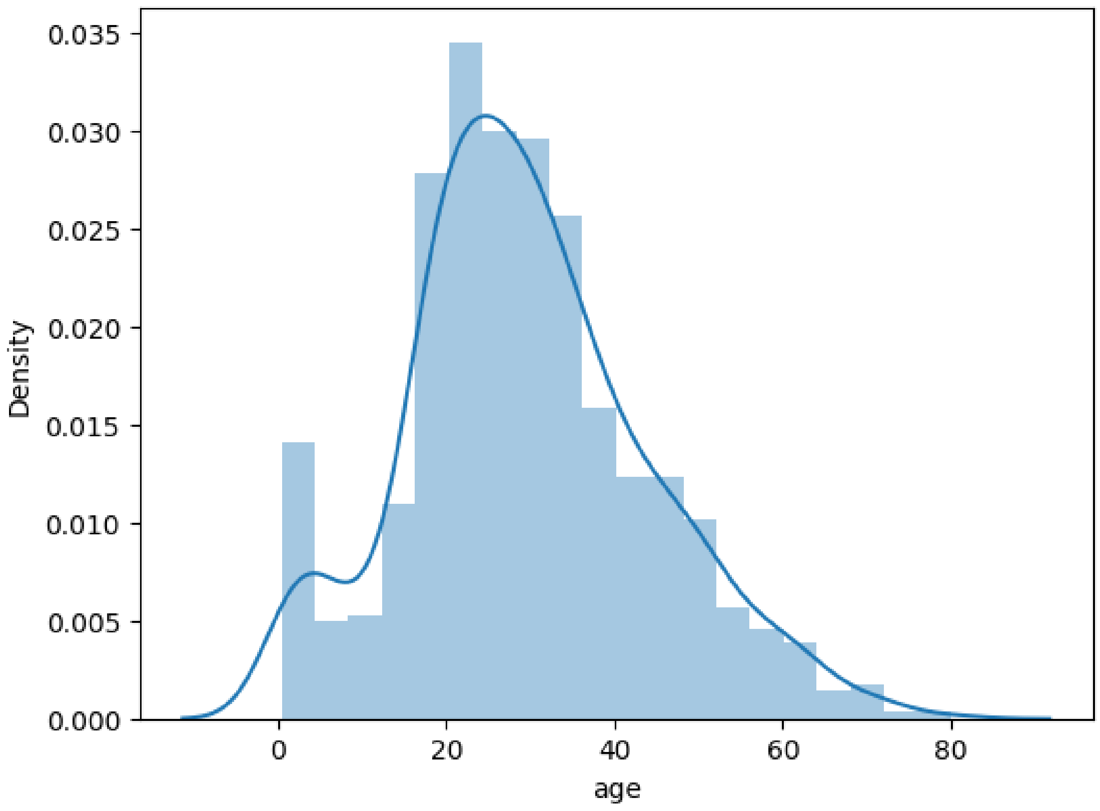
Imputation is a statistical technique used to replace missing data with substituted values. It's a crucial step in data preprocessing, particularly in fields like data science and machine learning, as many algorithms require complete datasets without any missing values. The goal of imputation is to preserve the dataset's integrity and improve the accuracy of analyses by minimizing the impact of missing values

Mean/Median/Mode Imputation: Mean Imputation: Replace missing values with the mean of the non-missing values in the same column. This is suitable for numerical data but can distort the distribution if there are outliers. Median Imputation: Replace missing values with the median of the non-missing values. This is more robust to outliers than mean imputation and is often used when the data is skewed. Mode Imputation: Replace missing values with the mode (most frequent value) for categorical data. This maintains the most common category in the dataset

[31]:
```python
## Imputation Techniques
## 1. Mean Value imputation - works well when data is NORMALLY DISTRIBUTED

sns.distplot(df['age'])
```

```
C:\Users\karth\AppData\Local\Temp\ipykernel_6588\2958496650.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['age'])
C:\Users\karth\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
[31]: <Axes: xlabel='age', ylabel='Density'>
```



0.3.4 Features of distplot():

Histogram: Shows the frequency distribution of the data by dividing it into bins and displaying the count of data points in each bin. Kernel Density Estimate (KDE): A smoothed representation of the distribution, which estimates the probability density function of the variable. This provides a continuous curve over the histogram. Flexibility: You could customize aspects like the number of bins, color, transparency, and whether to show the KDE.

```
[16]: df.age.isnull().sum()
```

```
[16]: 177
```

```
[20]: print('The mean of ages is: ', df['age'].mean(), '\n')

      df['age_mean'] = df['age'].fillna(df['age'].mean())

      df[['age', 'age_mean']]
```

```
The mean of ages is:  29.69911764705882
```

[20]:
```
      age   age_mean
0    22.0  22.000000
1    38.0  38.000000
2    26.0  26.000000
3    35.0  35.000000
4    35.0  35.000000
..    …         …
886  27.0  27.000000
887  19.0  19.000000
888   NaN  29.699118
889  26.0  26.000000
890  32.0  32.000000

[891 rows x 2 columns]
```

[22]:
```
## 2. Median Value Imputation - Incase of outliers
print('The median is: ', df['age'].median(), '\n')

df['age_median'] = df['age'].fillna(df['age'].median())

df[['age', 'age_median']]
```

```
The median is:  28.0
```

[22]:
```
      age  age_median
0    22.0        22.0
1    38.0        38.0
2    26.0        26.0
3    35.0        35.0
4    35.0        35.0
..    …          …
886  27.0        27.0
887  19.0        19.0
888   NaN        28.0
889  26.0        26.0
890  32.0        32.0

[891 rows x 2 columns]
```

[24]:
```
## 3. Mode Value Imputation - Categorical Values
df[df['embarked'].isnull()]
```

[24]:
```
     survived  pclass     sex   age  sibsp  parch  fare embarked  class  \
61          1       1  female  38.0      0      0  80.0      NaN  First
829         1       1  female  62.0      0      0  80.0      NaN  First

       who  adult_male deck embark_town alive  alone  age_mean  age_median
```

```
61   woman       False    B          NaN   yes   True      38.0         38.0
829  woman       False    B          NaN   yes   True      62.0         62.0
```

```
[26]: print('Unique values observed for the attribute "embarked" : ', df['embarked'].
        ↪unique(), '\n')

      mode = df[df['embarked'].notna()]['embarked'].mode()[0]
      print('Mode for the attribute "embarked": ', mode)

      df['embarked_mode'] = df['embarked'].fillna(mode)
      df[['embarked', 'embarked_mode']]
```

```
Unique values observed for the attribute "embarked" :  ['S' 'C' 'Q' nan]

Mode for the attribute "embarked":  S
```

```
[26]:     embarked embarked_mode
     0          S             S
     1          C             C
     2          S             S
     3          S             S
     4          S             S
     ..       …             …
     886        S             S
     887        S             S
     888        S             S
     889        C             C
     890        Q             Q

     [891 rows x 2 columns]
```

df['embarked'].notna(): This part creates a boolean mask (a Series of True and False values) that indicates which entries in the embarked column are not missing (i.e., not NaN). The result is a Series where True represents a valid (non-missing) entry.

df[df['embarked'].notna()]: This part uses the boolean mask from the previous step to filter the DataFrame df, returning a new DataFrame that only includes the rows where the embarked values are not missing. Essentially, it removes any rows where embarked is NaN.

['embarked']: After filtering the DataFrame, this selects the embarked column from the filtered DataFrame, which now contains only the non-missing values.

.mode(): This function calculates the mode of the embarked column. The mode is the value that appears most frequently in the Series. The mode() function returns a Series of modes because there can be multiple modes in the data.

[0]: Since mode() returns a Series of modes, the [0] index retrieves the first mode (the most frequently occurring value). This is useful when you know there is at least one mode and you want the most common one

```
[28]: df['embarked_mode'].isnull().sum()
```

```
[28]: 0
```

```
[ ]:
```