Dealing with missing data is a common and inherent issue in data collection, especially when working with large datasets. There are various reasons for missing data, such as incomplete information provided by participants, non-response from those who decline to share information, poorly designed surveys, or removal of data for confidentiality reasons.

When not appropriately handled, missing data can bias the conclusions of all the statistical analyses on the data, leading the business to make wrong decisions.

This article will focus on some techniques to efficiently handle missing values and their implementations in Python. We will illustrate the benefits and drawbacks of each technique to help you choose the right one for a given situation.

**Identifying Missing Data**

Missing data occurs in different formats. This section explains the different types of missing data and how to identify them.

**Types of missing data**

There are three main types of missing data: (1) Missing Completely at Random (MCAR), (2) Missing at Random (MAR), and (3) Missing Not at Random (MNAR).

It is important to have a better understanding of each one for choosing the appropriate methods to handle them.

**1) MCAR - Missing completely at random**

This happens if all the variables and observations have the same probability of being missing. Imagine providing a child with Lego of different colors to build a house. Each Lego represents a piece of information, like shape and color. The child might lose some Legos during the game. These lost legos represent missing information, just like when they can’t remember the shape or the color of the Lego they had. That information was lost randomly, but they do not change the information the child has on the other Legos.

**2) MAR - Missing at random**

For MAR, the probability of the value being missing is related to the value of the variable or other variables in the dataset. This means that not all the observations and variables have the same chance of being missing. An example of MAR is a survey in the Data community where data scientists who do not frequently upgrade their skills are more likely not to be aware of new state-of-the-art algorithms or technologies, hence skipping certain questions. The missing data, in this case, is related to how frequently the data scientist upskills.

**3) MNAR- Missing not at random**

MNAR is considered to be the most difficult scenario among the three types of missing data. It is applied when neither MAR nor MCAR apply. In this situation, the probability of being missing is completely different for different values of the same variable, and these reasons can be unknown to us. An example of MNAR is a survey about married couples. Couples with a bad relationship might not want to answer certain questions as they might feel embarrassed to do so.

**Methods for identifying missing data**

There are multiple methods that can be used to identify missing data in pandas. Below are the most recurrent ones.

|  |  |
| --- | --- |
| **Functions** | **Descriptions** |
| .isnull() | This function returns a pandas dataframe, where each value is a boolean value True if the value is missing, False otherwise. |
| .notnull() | Similarly to the previous function, the values for this one are False if either NaN or None value is detected. |
| .info() | This function generates three main columns, including the “Non-Null Count” which shows the number of non-missing values for each column. |
| .isna() | This one is similar to isnull and notnull. However it shows True only when the missing value is NaN type. |

**Real-life examples of Missing Values in Data**

**Predicting Survival on the Titanic**

The Titanic dataset is a widely-known example of missing values in data. By analyzing the probability of survival based on attributes like gender, age, and social status, predictions can be made on which passengers would have survived. Some groups were more likely to survive, revealing society’s priorities and privileges at the time.

**Peer-to-peer lending: Finance**

The Lending Club dataset is another instance where missing values play a significant role. The dataset contains complete loan data for all loans issued through 2007–2015, including loan status and payment information. However, missing data can occur in areas like credit scores, finance inquiries, addresses, and collections.

To work with these datasets, you can download the Titanic dataset [here](https://www.kaggle.com/c/titanic/data) and the Lending Club dataset [here](https://www.lendingclub.com/info/download-data.action).

Understanding and handling missing values is crucial in data analysis. The mechanisms of missing data — MCAR, MAR, MNAR — provide a roadmap to deal with missing values, enabling more accurate and comprehensive data analyses. As the field of data science continues to evolve, so too will the techniques and strategies for handling missing values.

**How to Handle Missing Values**

Identifying the mechanism of missing data is a critical step towards deciding how to manage those missing values. Depending on the underlying mechanism, different strategies can be employed.

1. Deletion: If the data is Missing Completely at Random (MCAR), then one way to handle missing values is to ignore those cases with missing data. This method, however, might lead to reduced statistical power, biased estimates, and lower accuracy if the missing data is not MCAR.

2. Imputation: This involves replacing missing data with substituted values. These could be a mean, median, or mode for numerical data and the most common category for categorical data. This method might introduce bias in the data, so it is essential to understand the nature of the missing data and assumptions about its distribution before proceeding with imputation.

3. Prediction Models: Regression, machine learning, or deep learning models can be used to predict and replace missing values based on other data. This is a more sophisticated way to handle missing data but might also lead to overfitting if not appropriately applied.

4. Multiple Imputation: This involves creating multiple filled-in copies of the dataset, analyzing each one separately, and then combining the results. This method can provide more accurate estimates and confidence intervals than simple imputation.

Missing values in a [dataset](https://www.analyticsvidhya.com/blog/2024/05/how-to-improve-dataset-selection-with-chatgpt/) can be represented in various ways, depending on the source of the data and the conventions used. Here are some common representations:

* **NaN (Not a Number)**: In many programming languages and data analysis tools, missing values are represented as NaN. This is the default for libraries like [Pandas](https://www.analyticsvidhya.com/blog/2024/02/how-to-make-pandas-faster/) in Python.
* **NULL or None**: In databases and some programming languages, missing values are often represented as NULL or None. For instance, in [SQL databases](https://www.analyticsvidhya.com/blog/2022/04/data-engineering-sql-vs-nosql-databases/), a missing value is typically recorded as NULL.
* **Empty Strings**: Sometimes, missing values are denoted by empty strings (""). This is common in text-based data or CSV files where a field might be left blank.
* **Special Indicators**: Datasets might use specific indicators like -999, 9999, or other unlikely values to signify missing data. This is often seen in older datasets or specific industries where such conventions were established.
* **Blanks or Spaces**: In some cases, particularly in fixed-width text files, missing values might be represented by spaces or blank fields.

How to Handle Missing Data?

Missing data is a common headache in any field that deals with datasets. It can arise for various reasons, from human error during data collection to limitations of data gathering methods. Luckily, there are strategies to address missing data and minimize its impact on your analysis. Here are two main approaches:

* **Deletion:** This involves removing rows or columns with missing values. This is a straightforward method, but it can be problematic if a significant portion of your data is missing. Discarding too much data can affect the reliability of your conclusions.
* **Imputation:** This replaces missing values with estimates. There are various imputation techniques, each with its strengths and weaknesses. Here are some common ones:
  + **Mean/Median/Mode Imputation:** Replace missing entries with the average (mean), middle value (median), or most frequent value (mode) of the corresponding column. This is a quick and easy approach, but it can introduce bias if the missing data is not randomly distributed.
  + **K-Nearest Neighbors (KNN Imputation):** This method finds the closest data points (neighbors) based on available features and uses their values to estimate the missing value. KNN is useful when you have a lot of data and the missing values are scattered.
  + **Model-based Imputation:** This involves creating a statistical model to predict the missing values based on other features in the data. This can be a powerful technique, but it requires more expertise and can be computationally expensive.pen\_spark.

Why Do We Need to Care About Handling Missing Data?

It is important to handle the [missing values](https://www.analyticsvidhya.com/blog/2021/10/a-complete-guide-to-dealing-with-missing-values-in-python/) appropriately.

* Many machine learning algorithms fail if the dataset contains missing values. However, algorithms like K-nearest and Naive Bayes support data with missing values.
* You may end up building a biased machine learning model, leading to incorrect results if the missing values are not handled properly.
* Missing data can lead to a lack of precision in the statistical analysis.

Checking for Missing Values in Python

The first step in handling missing values is to carefully look at the complete data and find all the missing values. The following code shows the total number of missing values in each column. It also shows the total number of missing values in the entire data set.

**import** pandas **as** pd

train\_df = pd.read\_csv("train.csv")

#Find the missing values from each column

print(train\_df.isnull().sum())

From the above output, we can see that there are 6 columns – Gender, Married, Dependents, Self\_Employed, LoanAmount, Loan\_Amount\_Term, and Credit\_History having missing values.

#Find the total number of missing values from the entire dataset

train\_df.isnull().sum().sum()

#Output

149

There are 149 missing values in total.

List of Methods to Handle Missing Values in a Dataset

Here is a list of popular strategies to handle missing values in a dataset

* Deleting the Missing Values
* Imputing the Missing Values
* Imputing the Missing Values for Categorical Features
* Imputing the Missing Values using Sci-kit Learn Library
* Using “Missingness” as a Feature

Handling Missing Values

Analyze each column with missing values carefully to understand the reasons behind the missing of those values, as this information is crucial to choose the strategy for handling the missing values.

There are 2 primary ways of handling missing values:

* Deleting the Missing values
* Imputing the Missing Values

Deleting the Missing value

Generally, this approach is not recommended. It is one of the quick and dirty techniques one can use to deal with missing values. If the missing value is of the type Missing Not At Random (MNAR), then it should not be deleted.

If the missing value is of type Missing At Random (MAR) or Missing Completely At Random (MCAR) then it can be deleted (In the analysis, all cases with available data are utilized, while missing observations are assumed to be completely random (MCAR) and addressed through pairwise deletion.)

The disadvantage of this method is one might end up deleting some useful data from the dataset.

There are 2 ways one can delete the missing data values:

Deleting the entire row (listwise deletion)

If a row has many missing values, you can drop the entire row. If every row has some (column) value missing, you might end up deleting the whole data. The code to drop the entire row is as follows:

df = train\_df.dropna(axis=0)

df.isnull().sum()

Deleting the entire column

If a certain column has many missing values, then you can choose to drop the entire column. The code to drop the entire column is as follows:

df = train\_df.drop(['Dependents'],axis=1)

df.isnull().sum()

Imputing the Missing Value

There are many imputation methods for replacing the missing values. You can use different python libraries such as Pandas, and Sci-kit Learn to do this. Let’s go through some of the ways of replacing the missing values.

Replacing with an arbitrary value

If you can make an educated guess about the missing value, then you can replace it with some arbitrary value using the following code. E.g., in the following code, we are replacing the missing values of the ‘Dependents’ column with ‘0’.

#Replace the missing **value** **with** '0' **using** 'fiilna' **method**

train\_df['Dependents'] = train\_df['Dependents'].fillna(0)

train\_df[‘Dependents'].isnull().sum()

Replacing with the mean

This is the most common method of imputing missing values of numeric columns. If there are outliers, then the mean will not be appropriate. In such cases, outliers need to be treated first. You can use the ‘fillna’ method for imputing the columns ‘LoanAmount’ and ‘Credit\_History’ with the mean of the respective column values.

#Replace the missing values for numerical columns with mean

train\_df['LoanAmount'] = train\_df['LoanAmount'].fillna(train\_df['LoanAmount'].mean())

train\_df['Credit\_History'] = train\_df[‘Credit\_History'].fillna(train\_df['Credit\_History'].mean())

Replacing with the mode

Mode is the most frequently occurring value. It is used in the case of categorical features. You can use the ‘fillna’ method for imputing the categorical columns ‘Gender,’ ‘Married,’ and ‘Self\_Employed.’

#Replace the missing values for categorical columns with mode

train\_df['Gender'] = train\_df['Gender'].fillna(train\_df['Gender'].mode()[0])

train\_df['Married'] = train\_df['Married'].fillna(train\_df['Married'].mode()[0])

train\_df['Self\_Employed'] = train\_df[‘Self\_Employed'].fillna(train\_df['Self\_Employed'].mode()[0])

train\_df.isnull().sum()

Replacing with the median

The median is the middlemost value. It’s better to use the median value for imputation in the case of outliers. You can use the ‘fillna’ method for imputing the column ‘Loan\_Amount\_Term’ with the median value.

train\_df['Loan\_Amount\_Term']= train\_df['Loan\_Amount\_Term'].fillna(train\_df['Loan\_Amount\_Term'].median())

Replacing with the previous value – forward fill

In some cases, imputing the values with the previous value instead of the mean, mode, or median is more appropriate. This is called forward fill. It is mostly used in time series data. You can use the ‘fillna’ function with the parameter ‘method = ffill’

IN:

**import** pandas **as** pd

**import** numpy **as** np

test = pd.Series(range(6))

test.loc[2:4] = np.nan

test

**IN:**

# Forward-Fill

test.fillna(method=‘ffill')

Replacing with the next value – backward fill

In backward fill, the missing value is imputed using the next value.

**IN:**

# Backward-Fill

test.fillna(method=‘bfill')

**Interpolation**

Missing values can also be imputed using interpolation. Pandas’ interpolate method can be used to replace the missing values with different interpolation methods like ‘polynomial,’ ‘linear,’ and ‘quadratic.’ The default method is ‘linear.’

IN:

test.interpolate()

How to Impute Missing Values for Categorical Features?

There are two ways to impute missing values for categorical features as follows:

Impute the Most Frequent Value

We will use ‘SimpleImputer’ in this case, and as this is a non-numeric column, we can’t use mean or median, but we can use the most frequent value and constant.

**import** pandas **as** pd

**import** numpy **as** np

X = pd.**DataFrame**({'Shape':['square', 'square', 'oval', 'circle', np.nan]})

X

from sklearn.impute **import** SimpleImputer

imputer = SimpleImputer(strategy='most\_frequent')

imputer.fit\_transform(X)

As you can see, the missing value is imputed with the most frequent value, ’square.’

Impute the Value “Missing”

We can impute the value “missing,” which treats it as a separate category.

imputer = SimpleImputer(strategy='constant', fill\_value='missing')

imputer.fit\_transform(X)

In any of the above approaches, you will still need to OneHotEncode the data (or you can also use another encoder of your choice). After One Hot Encoding, in case 1, instead of the values ‘square,’ ‘oval,’ and’ circle,’ you will get three feature columns. And in case 2, you will get four feature columns (4th one for the ‘missing’ category). So it’s like adding the missing indicator column in the data. There is another way to add a missing indicator column, which we will discuss further.

How to Impute Missing Values Using Sci-kit Learn Library?

We can impute missing values using the sci-kit library by creating a model to predict the observed value of a variable based on another variable which is known as regression imputation.

Univariate Approach

In a Univariate approach, only a single feature is taken into consideration. You can use the class SimpleImputer and replace the missing values with mean, mode, median, or some constant value.

Let’s see an example:

import numpy as np

from sklearn.impute import SimpleImputer

imp = SimpleImputer(missing\_values=np.nan, strategy='mean')

imp.fit([[1, 2], [np.nan, 3], [7, 6]])

**Output:**

OUT: SimpleImputer()

IN:

X = [[np.nan, 2], [6, np.nan], [7, 6]]

print(imp.transform(X))

**Output:**

OUT:

[[4. 2. ]

[6. 3.666...]

[7. 6. ]]

Multivariate Approach

In a multivariate approach, more than one feature is taken into consideration. There are two ways to impute missing values considering the multivariate approach. Using KNNImputer or IterativeImputer classes.

Let’s take an example of a titanic dataset.

Suppose the feature ‘age’ is well correlated with the feature ‘Fare’ such that people with lower fares are also younger and people with higher fares are also older. In that case, it would make sense to impute low age for low fare values and high age for high fare values. So here, we are taking multiple features into account by following a multivariate approach.

import pandas as pd

df = pd.read\_csv('http://bit.ly/kaggletrain', nrows=6)

cols = ['SibSp', 'Fare', 'Age']

X = df[cols]

X

|  |
| --- |
|  |
| **SibSp** | **Fare** | **Age** |
| **0** | 1 | 7.2500 | 22.0 |
| **1** | 1 | 71.2833 | 38.0 |
| **2** | 0 | 7.9250 | 26.0 |
| **3** | 1 | 53.1000 | 35.0 |
| **4** | 0 | 8.0500 | 35.0 |
| **5** | 0 | 8.4583 | NaN |

**from** sklearn.experimental **import** enable\_iterative\_imputer

**from** sklearn.impute **import** **IterativeImputer**

impute\_it = **IterativeImputer**()

impute\_it.**fit\_transform**(X)

**Output:**

OUT:

array([[ 1. , 7.25 , 22. ],

[ 1. , 71.2833 , 38. ],

[ 0. , 7.925 , 26. ],

[ 1. , 53.1 , 35. ],

[ 0. , 8.05 , 35. ],

[ 0. , 8.4583 , 28.50639495]])

Let’s see how IterativeImputer works. For all rows in which ‘Age’ is not missing, sci-kit learn runs a regression model. It uses ‘Sib sp’ and ‘Fare’ as the features and ‘Age’ as the target. And then, for all rows for which ‘Age’ is missing, it makes predictions for ‘Age’ by passing ‘Sib sp’ and ‘Fare’ to the training model. So it actually builds a regression model with two features and one target and then makes predictions on any places where there are missing values. And those predictions are the imputed values.

Nearest Neighbors Imputations (KNNImputer)

Missing values are imputed using the k-Nearest Neighbors approach, where a Euclidean distance is used to find the nearest neighbors. Let’s take the above example of the titanic dataset to see how it works.

from sklearn.impute **import** KNNImputer

impute\_knn = KNNImputer(n\_neighbors=2)

impute\_knn.fit\_transform(X)

**Output:**

OUT:

array([[ 1. , 7.25 , 22. ],

[ 1. , 71.2833, 38. ],

[ 0. , 7.925 , 26. ],

[ 1. , 53.1 , 35. ],

[ 0. , 8.05 , 35. ],

[ 0. , 8.4583, 30.5 ]])

In the above example, the n\_neighbors=2. So sci-kit learn finds the two most similar rows measured by how close the ‘Sib sp’ and ‘Fare’ values are to the row which has missing values. In this case, the last row has a missing value. And the third row and the fifth row have the closest values for the other two features. So the average of the ‘Age’ feature from these two rows is taken as the imputed value.

How to Use “Missingness” as a Feature?

In some cases, while imputing missing values, you can preserve information about which values were missing and use that as a feature. This is because sometimes, there may be a relationship between the reason for missing values (also called the “missingness”) and the target variable you are trying to predict. In such cases, you can add a missing indicator to encode the “missingness” as a feature in the imputed data set.

**Where can we use this?**

Suppose you are predicting the presence of a disease. Now, imagine a scenario where a missing age is a good predictor of the disease because we don’t have records for people in poverty. The age values are not missing at random. They are missing for people in poverty, and poverty is a good predictor of disease. Thus, missing age or “missingness” is a good predictor of disease.

**import** pandas **as** pd

**import** numpy **as** np

X = pd.**DataFrame**({'Age':[20, 30, 10, np.nan, 10]})

X

|  |
| --- |
|  |
| **Age** |
| **0** | 20.0 |
| **1** | 30.0 |
| **2** | 10.0 |
| **3** | NaN |
| **4** | 10.0 |

**from** sklearn.impute

**import** SimpleImputer

# impute the mean

imputer = SimpleImputer()

imputer.fit\_transform(X)

**Output:**

OUT:

array([[20. ],

[30. ],

[10. ],

[17.5],

[10. ]])

imputer = SimpleImputer(add\_indicator=True)

imputer.fit\_transform(X)

**Output:**

OUT:

array([[20. , 0. ],

[30. , 0. ],

[10. , 0. ],

[17.5, 1. ],

[10. , 0. ]])

In the above example, the second column indicates whether the corresponding value in the first column was missing or not. ‘1’ indicates that the corresponding value was missing, and ‘0’ indicates that the corresponding value was not missing.

If you don’t want to impute missing values but only want to have the indicator matrix, then you can use the ‘MissingIndicator’ class from scikit learn.

[Effective Strategies to Handle Missing Values in Data Analysis (analyticsvidhya.com)](https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/)