Handling missing data is a crucial step in the data cleaning phase. Here is the theory and practical application using pandas to identify and manage these gaps.

## 1. Theory of Missing Data ❓

Missing data, often represented as **NaN** (Not a Number) in Python/pandas, refers to the absence of a value for a variable in an observation. Ignoring it can lead to biased statistics, faulty models, and incorrect conclusions.

| Concept | Description | Impact if Ignored |
| --- | --- | --- |
| **Missing At Random (MAR)** | The missingness is related to other observed data, but not to the missing value itself. | If not handled, it can reduce the precision of estimates. |
| **Missing Completely At Random (MCAR)** | The missingness has no relationship with any variables (observed or unobserved). | This is the "least problematic" type, often safe for simple deletion, as it doesn't introduce bias. |
| **Missing Not At Random (MNAR)** | The missingness is directly related to the missing value itself (e.g., people with very high incomes refuse to disclose it). | Most difficult to handle; simple deletion or imputation can lead to severe bias. |

## 2. Identifying Missing Data with Pandas 🔍

The pandas library provides simple, fast, and vectorized functions to detect missing values (NaNs, N/As, or Nulls).

### A. Core Detection Techniques

The functions isnull() and notnull() are the core of missing data detection. They return a DataFrame or Series of the same shape, filled with **Boolean values** (True or False).

| Pandas Function | Output | Practical Use |
| --- | --- | --- |
| **df.isnull()** | Returns True where a value is missing (NaN), False otherwise. | Used to filter or count the missing values. |
| **df.notnull()** | Returns True where a value is *present*, False where it's missing. | Used to quickly subset the DataFrame to only include complete rows. |

### B. Quantification and Summary

The most common workflow is to summarize the missing data by column.

| Python Code | Description | Rationale |
| --- | --- | --- |
| df.isnull().sum() | Applies isnull() to every cell, then sums the True values (which are treated as 1) down each column. | Gives the **total count** of missing values per column. Essential for a quick overview. |
| df.isnull().sum().sum() | Sums the result of the previous step. | Gives the **total number of missing values** in the entire DataFrame. |
| df.isnull().mean() \* 100 | Calculates the mean of the isnull() result (the count divided by total rows) and converts it to a percentage. | Provides the **percentage of missing data** per column, which helps decide on the treatment technique (e.g., impute if <5%, consider dropping the column if >70%). |

### Example Application

Assume you have a DataFrame df loaded from customer data:

Python

import pandas as pd  
import numpy as np # Used for creating NaN values  
  
data = {'Name': ['A', 'B', 'C', 'D'],   
 'Age': [25, 30, np.nan, 45],   
 'Income': [50000, 60000, 75000, np.nan]}  
df = pd.DataFrame(data)  
  
# 1. Check the DataFrame for missing values (returns a Boolean mask)  
print("Boolean Mask of Missing Data:")  
print(df.isnull())  
  
# 2. Get the count of missing values per column  
missing\_counts = df.isnull().sum()  
print("\nMissing Count per Column:")  
print(missing\_counts)  
  
# 3. Get the percentage of missing values per column  
total\_rows = len(df)  
missing\_percentage = (df.isnull().sum() / total\_rows) \* 100  
print("\nMissing Percentage per Column:")  
print(missing\_percentage)

Output Interpretation:

The output shows that the Age and Income columns each have 1 missing value, representing 25% of the total data for those columns, guiding the analyst toward an appropriate imputation or deletion strategy.