😊 Missing Data

What is Missing Data?

In the context of data analysis, **missing data** refers to the absence of a value where one is expected. In a dataset, this means that for a particular observation (row), the value for one or more features (columns) is not available. In real-world scenarios, missing data is common and arises due to various reasons such as human error, system glitches, incomplete data collection, or merging data from multiple sources.

Representation of Missing Data in Pandas

In Pandas, missing data is primarily represented in the following forms:

1. **NaN (Not a Number)**
   This is the most common representation of missing data in Pandas. It is a special floating-point value defined in the NumPy library and is used to denote missing numeric values.
2. **None**
   This is the Python built-in type for representing null values. Pandas automatically converts None to NaN when performing operations on dataframes.
3. **NaT (Not a Time)**
   This is a special case of missing data specific to datetime types in Pandas. It represents missing or null datetime values.

Why Does Missing Data Occur?

There are many practical situations in which missing data can appear:

- **Data Collection Errors**: Respondents may skip questions in surveys or forms.
- **System Failures**: Devices or systems might fail to record data due to technical issues.
- **Merging Datasets**: When combining multiple data sources, some rows may not have corresponding values in all sources.
- **Data Entry Issues**: Human errors during data entry can result in blank fields or incorrect values.
- **Irregular Data Sampling**: In time series or experimental data, samples might be recorded at inconsistent intervals.

Implications of Missing Data

Missing data can significantly affect the results and integrity of your analysis. Here are some key implications:

- **Statistical Bias**: Calculations like mean, median, standard deviation, and correlation may produce misleading results if missing values are not handled properly.
- **Algorithm Failures**: Many machine learning models and data processing algorithms do not support missing values and may throw errors if they are present.

- **Loss of Information**: Simply dropping missing data can lead to a reduction in dataset size and possibly the removal of valuable information.

Identifying Missing Data in Pandas

Pandas provides several methods to detect and examine missing data:

- **isnull() or isna()**: These functions return a Boolean DataFrame indicating where values are missing.
- **notnull() or notna()**: These are the logical opposites and indicate where data is present.

These methods are typically used to generate summaries or to filter the data for cleaning or analysis.

Handling Missing Data in Pandas

There are several strategies available for handling missing data, depending on the nature and context of the dataset:

1. **Removing Missing Data**
    - You can drop rows or columns that contain missing values.
    - This is simple but may lead to information loss if many values are missing.
2. **Filling Missing Data**
    - You can replace missing values with a constant (like 0 or "Unknown").
    - Alternatively, you can fill them using statistical measures like the mean, median, or mode of the column.
    - You can also fill forward or backward based on nearby values (especially in time series).
3. **Interpolating Missing Data**
    - Pandas allows for interpolation using various methods such as linear, polynomial, or spline interpolation.
    - This estimates the missing values based on existing data points, useful in continuous data like stock prices or sensor data.
4. **Flagging or Encoding Missing Data**
    - For machine learning models, missing data can be treated as a separate category or encoded with a special flag to retain the information about the absence of data.

Best Practices When Dealing with Missing Data

- **Understand the cause**: Try to find out why the data is missing. Is it random or systematic?
- **Analyze patterns**: Use visualization and summaries to assess the extent of missingness.

- **Choose a method that suits the data**: Use domain knowledge to decide whether to fill, drop, or model the missing data.
- **Be cautious with imputation**: Filling missing values can introduce bias or false patterns, especially if done without understanding the data distribution.

Conclusion

Missing data is an inevitable part of real-world data analysis. Pandas provides robust and flexible tools to detect, visualize, and handle missing values effectively. Proper treatment of missing data is crucial for building reliable models and drawing accurate insights. The choice of handling technique depends on the type of data, the amount of missing information, and the goals of the analysis.

😄 To begin with Lab

1. The code imports the NumPy and Pandas libraries and demonstrates three different types of missing or null values used in data analysis. `np.nan` is a special floating-point value from NumPy that represents "Not a Number" and is commonly used to indicate missing numerical data. `pd.NA` is pandas' own missing value marker that is more flexible and can represent missing values in various data types like integers, strings, or booleans. `pd.NaT` stands for "Not a Time" and is specifically used to represent missing or null datetime values in pandas.
2. These markers help handle and process incomplete or missing data in a DataFrame or Series.

```
[3]: import numpy as np
     import pandas as pd

[4]: np.nan

[4]: nan

[5]: pd.NA

[5]: <NA>

[6]: pd.NaT

[6]: NaT
```

3. The code demonstrates how missing values behave when compared in Python using NumPy and pandas. np.nan == np.nan returns False because np.nan is not equal to itself by design—this follows the IEEE standard for floating-point arithmetic.
4. np.nan in [np.nan] returns True because Python checks for object presence, not equality, in a list. np.nan is np.nan returns True because both sides refer to the same nan object in memory.
5. On the other hand, pd.NA == pd.NA returns pd.NA, not True or False, because pandas treats missing values as "unknown," and any comparison with pd.NA results in pd.NA, preserving the idea of uncertainty in comparisons.

```
[8]: np.nan == np.nan
[8]: False
[9]: np.nan in [np.nan]
[9]: True
[10]: np.nan is np.nan
[10]: True
[11]: pd.NA == pd.NA
[11]: <NA>
```

6. We created a data frame using the movie scores CSV file and then represented the data.

```
[14]: df = pd.read_csv('movie_scores.csv')
[15]: df
```

[15]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

7. Here you can see the data frame and using the df.isnull() function checks for missing or null values in a DataFrame.
8. It returns a DataFrame of the same shape as df, where each element is either True (if the value is missing, i.e., NaN, None, or pd.NA) or False (if the value is not missing).
9. This method is useful for identifying which values in the DataFrame are missing, allowing you to handle missing data appropriately (e.g., filling or dropping those values).

[17]:
```
df
```

[17]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

[18]:
```
df.isnull()
```

[18]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	False	False	False	False	False	False
1	True	True	True	True	True	True
2	False	False	False	False	True	True
3	False	False	False	False	False	False
4	False	False	False	False	False	False

10. The df.notnull() function is the opposite of df.isnull(). It checks for non-missing or valid values in a DataFrame.
11. It returns a DataFrame of the same shape as df, where each element is either True (if the value is not missing) or False (if the value is missing, i.e., NaN, None, or pd.NA).
12. This method is useful for identifying and working with the non-null values in the dataset.

[19]:
```
df.notnull()
```

[19]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	True	True	True	True	True	True
1	False	False	False	False	False	False
2	True	True	True	True	False	False
3	True	True	True	True	True	True
4	True	True	True	True	True	True

13. The code works with the "first_name" and "pre_movie_score" columns in the DataFrame. df['first_name'] selects the "first_name" column.
14. df[df['first_name'].notnull()] filters the DataFrame to include only rows where "first_name" is not null. df[(df['pre_movie_score'].isnull()) & df['sex'].notnull()] filters rows where "pre_movie_score" is null and "sex" is not null.
15. The notnull() and isnull() methods check for non-missing and missing values, respectively, and are used to filter data based on these conditions.

```
[20]: df['first_name']
```

```
[20]: 0      Tom
      1      NaN
      2     Hugh
      3    Oprah
      4     Emma
      Name: first_name, dtype: object
```

```
[21]: df[df['first_name'].notnull()]
```

[21]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

```
[22]: df[(df['pre_movie_score'].isnull()) & df['sex'].notnull()]
```

[22]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
2	Hugh	Jackman	51.0	m	NaN	NaN

16. The help(df.dropna) command in pandas provides detailed documentation and usage information for the dropna() method, which is used to remove missing (null) values from a DataFrame.
17. Specifically, it shows how to drop rows or columns containing NaN, None, or pd.NA values.
18. You can use dropna() to remove rows with any missing values or specify criteria to remove rows or columns with all or some missing values.

```
[25]: help(df.dropna)
```

```
Help on method dropna in module pandas.core.frame:

dropna(*, axis: 'Axis' = 0, how: 'AnyAll | NoDefault' = <no_default>, thresh: 'int | NoDefault' = <no_default>, subset: 'IndexLabel' = None, inplace: 'bo
ol' = False, ignore_index: 'bool' = False) -> 'DataFrame | None' method of pandas.core.frame.DataFrame instance
    Remove missing values.

    See the :ref:`User Guide <missing_data>` for more on which values are
    considered missing, and how to work with missing data.

    Parameters
    ----------
    axis : {0 or 'index', 1 or 'columns'}, default 0
        Determine if rows or columns which contain missing values are
        removed.

        * 0, or 'index' : Drop rows which contain missing values.
        * 1, or 'columns' : Drop columns which contain missing value.

        Pass tuple or list to drop on multiple axes.
        Only a single axis is allowed.

    how : {'any', 'all'}, default 'any'
        Determine if row or column is removed from DataFrame, when we have
        at least one NA or all NA.
```

19. The code uses the dropna() method to remove missing values from a DataFrame in various ways. df.dropna() removes rows with any missing values.
20. df.dropna(thresh=1) keeps rows with at least one non-null value, removing rows with fewer.
21. df.dropna(axis=1) removes columns with any missing values. df.dropna(thresh=4, axis=1) removes columns that have fewer than 4 non-null values.
22. The dropna() method is useful for cleaning data by removing incomplete rows or columns based on specified criteria.

[26]:
```
df.dropna()
```

[26]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

[27]:
```
df.dropna(thresh=1)
```

[27]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

[28]:
```
df.dropna(axis=1)
```

[28]:

0
1
2
3
4

[29]:
```
df.dropna(thresh=4,axis=1)
```

[29]:

	first_name	last_name	age	sex
0	Tom	Hanks	63.0	m
1	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m
3	Oprah	Winfrey	66.0	f
4	Emma	Stone	31.0	f

23. The code uses the fillna() method to replace missing values (NaN) in a DataFrame.
24. df.fillna("NEW VALUE!") fills all missing values with the string "NEW VALUE!".
25. df['first_name'].fillna("Empty") fills missing values in the "first_name" column with "Empty".
26. df['pre_movie_score'].mean() calculates the mean of the "pre_movie_score" column, and df['pre_movie_score'].fillna(df['pre_movie_score'].mean()) replaces missing values in that column with the mean.
27. df.fillna(df.mean(numeric_only=True)) fills missing values in numeric columns with their column means.

[31]:
```
df
```

[31]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

[32]:
```
df.fillna("NEW VALUE!")
```

[32]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	NEW VALUE!	NEW VALUE!	NEW VALUE!	NEW VALUE!	NEW VALUE!	NEW VALUE!
2	Hugh	Jackman	51.0	m	NEW VALUE!	NEW VALUE!
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

[33]:
```
df['first_name'].fillna("Empty")
```

[33]:
```
0      Tom
1    Empty
2     Hugh
3    Oprah
4     Emma
Name: first_name, dtype: object
```

[34]:
```
df['first_name'] = df['first_name'].fillna("Empty")
```

[35]:
```
df
```

[35]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	Empty	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

```
[36]: df['pre_movie_score'].mean()
```

```
[36]: 7.0
```

```
[37]: df['pre_movie_score'].fillna(df['pre_movie_score'].mean())
```

```
[37]: 0    8.0
      1    7.0
      2    7.0
      3    6.0
      4    7.0
      Name: pre_movie_score, dtype: float64
```

```
[50]: df.fillna(df.mean(numeric_only=True))
```

[50]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.00	m	8.0	10.0
1	Empty	NaN	52.75	NaN	7.0	9.0
2	Hugh	Jackman	51.00	m	7.0	9.0
3	Oprah	Winfrey	66.00	f	6.0	8.0
4	Emma	Stone	31.00	f	7.0	9.0

28. The code demonstrates the use of interpolation to fill missing values in a pandas Series and DataFrame.
29. First, a Series is created with some missing values (np.nan). ser.interpolate() fills the missing value using linear interpolation.
30. Then, the index is adjusted using pd.RangeIndex, and ser.interpolate(method='spline', order=2) performs spline interpolation to fill missing values with a smoother curve.
31. A DataFrame df is created from the Series, and df.interpolate() fills missing values using linear interpolation. df.interpolate(method='spline', order=2) applies spline interpolation to the DataFrame.

```
[52]: airline_tix = {'first':100,'business':np.nan,'economy-plus':50,'economy':30}
```

```
[54]: ser = pd.Series(airline_tix)
```

```
[56]: ser
```

```
[56]: first          100.0
      business         NaN
      economy-plus    50.0
      economy         30.0
      dtype: float64
```

```
[58]: ser.interpolate()
```

```
[58]: first          100.0
      business        75.0
      economy-plus    50.0
      economy         30.0
      dtype: float64
```

[62]:
```
ser.index = pd.RangeIndex(start=0, stop=len(ser), step=1)
ser.interpolate(method='spline', order=2)
```

[62]:
```
0    100.000000
1     73.333333
2     50.000000
3     30.000000
dtype: float64
```

[64]:
```
df = pd.DataFrame(ser,columns=['Price'])
```

[66]:
```
df
```

[66]:

	Price
0	100.0
1	NaN
2	50.0
3	30.0

[68]:
```
df.interpolate()
```

[68]:

	Price
0	100.0
1	75.0
2	50.0
3	30.0

[70]:
```
df = df.reset_index()
```

[72]:
```
df
```

[72]:

	index	Price
0	0	100.0
1	1	NaN
2	2	50.0
3	3	30.0

[74]:
```
df.interpolate(method='spline',order=2)
```

[74]:

	index	Price
0	0	100.000000
1	1	73.333333
2	2	50.000000
3	3	30.000000