Data Cleaning and Preparing

Data Science with Python

Data Processing in Python

- data processing consists of gathering and manipulating data elements to return useful, potentially valuable information. Different encoding types will have various processing formats. The most known formats for encodings are XML, CSV, JSON, and HTML.
- **Why is Data processing essential?**
- Data processing is a vital part of data science. Having inaccurate and bad-quality data can be damaging to processes and analysis. Good clean data will boost productivity and provide great quality information for your decision-making.

Exploratory Data Analysis

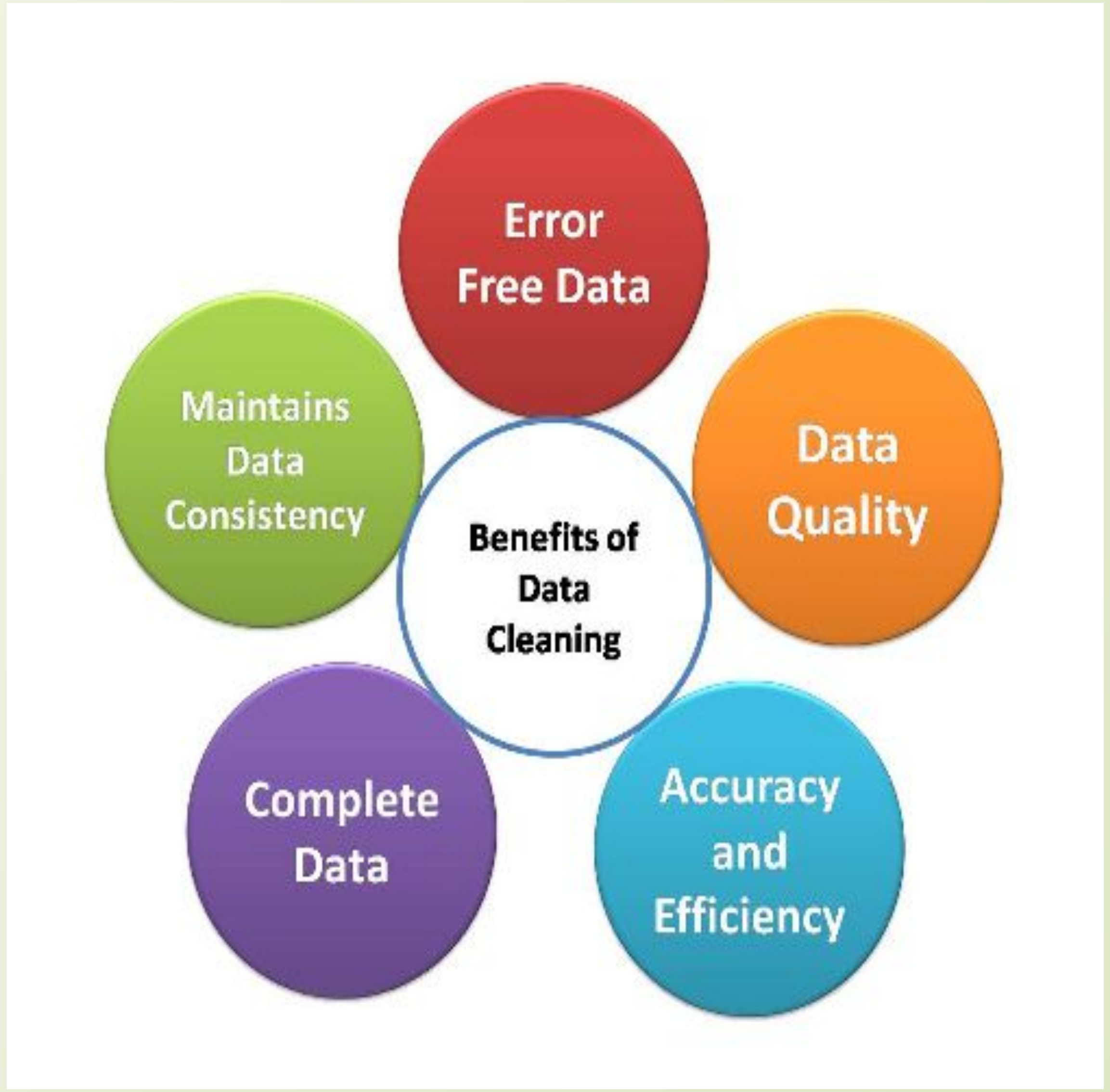
- Exploratory Data Analysis or (EDA) is understanding the data sets by summarizing their main characteristics often plotting them visually. This step is very important especially when we arrive at modeling the data in order to apply Machine learning.
- it depends on the data set that you are working. There is no one method in order to perform EDA, you can understand some common methods and plots that would be used in the EDA process.

10 Steps to your Exploratory data analysis

1. Import Dataset & Headers
2. Identify Missing Data
3. Replace Missing Data
4. Evaluate Missing Data
5. Dealing with Missing Data
6. Correct Data Formats
7. Data standardization
8. Data Normalization
9. Binning
10. Indicator variable

Why Is Data Cleaning Essential?

- Data Cleaning using Pandas in Python is the most important task that a data science professional should do.
- Wrong or bad-quality data can be detrimental to processes and analysis.
- Clean data will ultimately increase overall productivity and permit the very best quality information in decision-making.



What is Data Cleaning?

- Data cleaning is the **process of preparing data for analysis by removing or fixing data that is incorrect, incomplete, irrelevant, or duplicated within a dataset**. It's one of the important stages of machine learning, Data Science.
- Data cleaning can seem as a tedious task, but it's a fundamental block for any data analytics problem solving.
- Good and clean data can be used to produce accurate and trustworthy insights. Data scientists/engineers spend 60-80% of their time carrying out data cleaning activities.

How to clean data?

- Data cleaning puts data into the right shape and quality for analysis.

It includes many different steps, for example:
Basics (select, filter, removal of duplicates, …)
Sampling (balanced, stratified, …)
Data Partitioning (create training + validation + test data set, …)
Transformations (normalization, standardization, scaling, pivoting, …)
Binning (count-based, handling of missing values as its own group, …)
Data Replacement (cutting, splitting, merging, …)
Weighting and Selection (attribute weighting, automatic optimization, …)
Attribute Generation (ID generation, …)
Imputation (replacement of missing observations by using statistical algorithms)

Dataset/collected data have different values

- *The answer is no*. Because the data collected is unprocessed. Unprocessed data must be containing some
- **Missing values**
- **Outliers**
- **Unstructured manner**
- **Categorical data(which needs to be converted to numerical variables)**

When and Why Is Data Missed?

- **Sources of Missing Values and** some typical reasons why data is missing:
- User forgot to fill in a field.
- Data was lost while transferring manually from a legacy database.
- There was a programming error.
- Users chose not to fill out a field tied to their beliefs about how the results would be used or interpreted.
- **How to mark invalid/ corrupt values as missing**
- **Visible errors**: blank cells, special symbols like **NA** (Not Available), **NaN** (Not a Number), etc.
- **Obscure errors**: non-corrupt but **invalid values**. For example, a negative salary or a number for a name.
- You would notice values like **NA**, **NaN**, **?** and also **blank cells**. These represent missing values in our dataset.

Check for Missing Values

- To make detecting missing values easier (and across different array dtypes), Pandas provides the **isnull()** and **notnull()** functions, which are also methods on Series and DataFrame objects –
- **isnull()** function to mark all of the NaN values in the dataset as True
- **notnull()** to mark all of the NaN values in the dataset as False.
- Example:-
- print(df['Gender'].isnull().head(10)) # NaN values are marked True
- print(df['Gender'].notnull().head(10)) # non-NaN values are marked True

Calculations with Missing Data

- When summing data, NA will be treated as Zero
- If the data are all NA, then the result will be NA
- **df.isnull().sum():-** it will give the count of missing values of each feature or column it have.

Cleaning / Filling Missing Data

- Pandas provides various methods for cleaning the missing values. The fillna function can "fill in" NA values with non-null data in a couple of ways.
- **Replace NaN with a Scalar Value**
- program shows how you can replace "NaN" with "0".
- instead we can also fill with any other value.
- **Other Missing Value Examples**
- Pandas recognize "NA" as a missing value, but there could be other unexpected missing values you encounter in other datasets.
- **Making a list of missing value types**
- missing_values = ["n/a", "na", "missing"]
- df = pd.read_csv("dataset.csv", na_values = missing_values)
- **Replacing Missing Data**
- In Some dataset, missing data is indicated as "?"
  We'll replace "?" values to NaN" using the numpy's replace() method.
- df.replace("?", np.nan, inplace = True)

Filling Missing Data

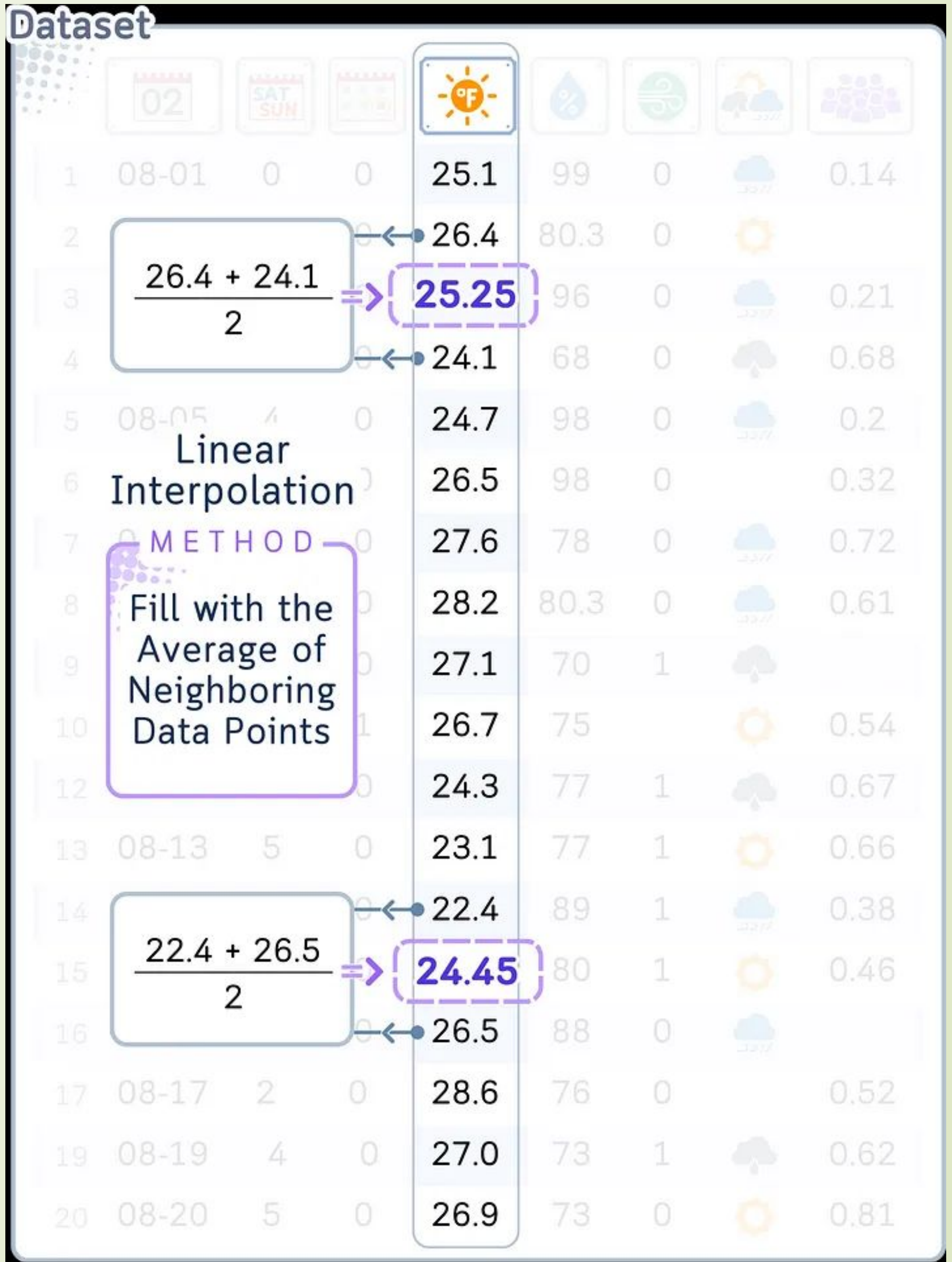
- So, let’s begin with the methods to solve the problem.
- Fillna
- Interpolate
- Dropna

Replacing With Arbitrary Value/Forward fill

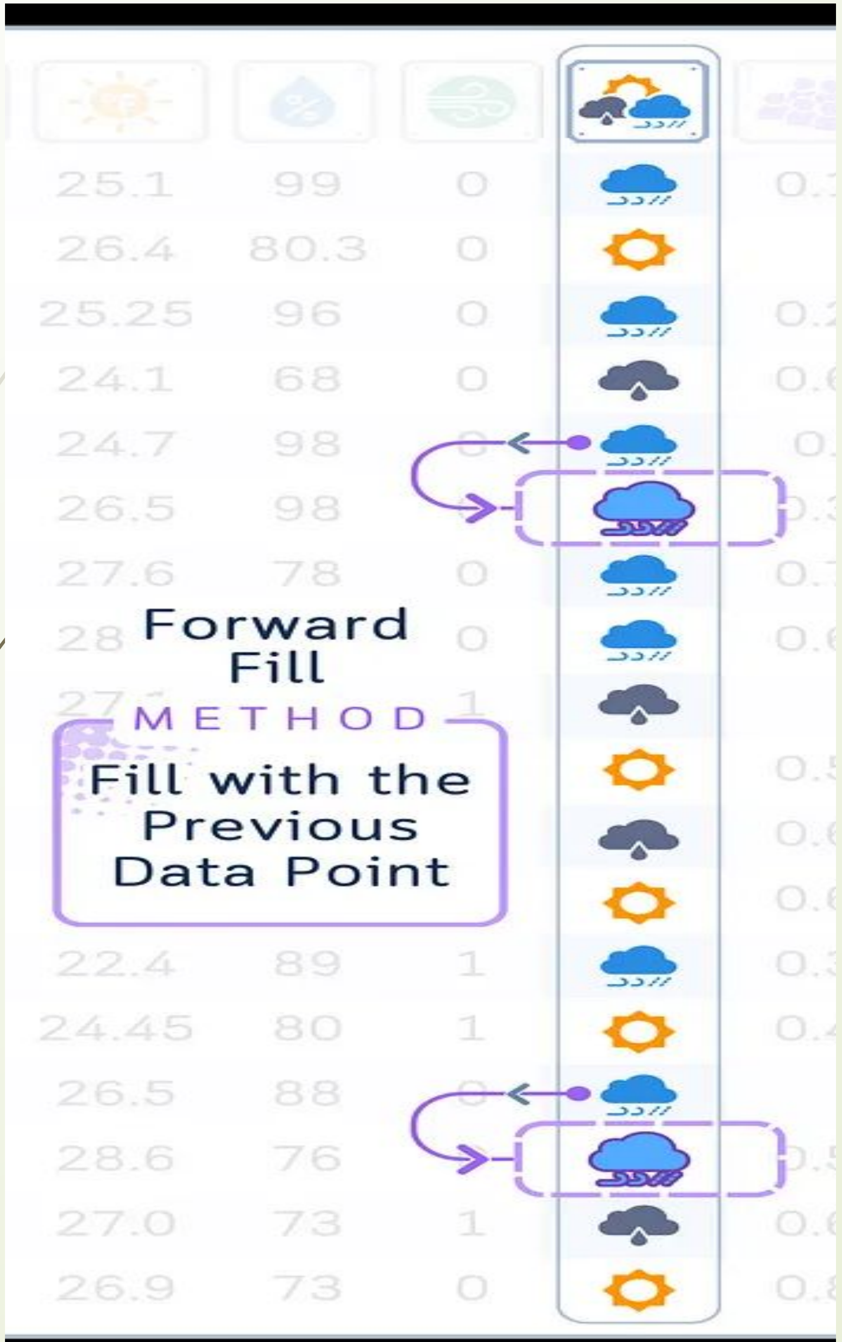
- If you can replace the missing value with some arbitrary value using fillna().
- Ex. In the below code, we are replacing the missing values with ‘0’.As well you can replace any particular column missing values with some arbitrary value also.
- **Replacing with previous value – Forward fill**
- We can impute the values with the previous value by using forward fill. It is mostly used in time series data.
- Syntax: df.fillna(method=’ffill’)
- **Replacing with next value – Backward fill**
- In backward fill, the missing value is imputed using the next value. It is mostly used in time series data.
- **3. 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’, ‘quadratic’. The default method is ‘linear’.
- Syntax: **df.interpolate(*method=’linear’*)**
- For the time-series dataset variable, it makes sense to use the interpolation of the variable before and after a timestamp for a missing value. Interpolation in most cases supposed to be the best technique to fill missing values

We're using linear interpolation for Temperature. Since temperature tends to change gradually over time and our data is ordered by date

- **df_clean['Temp'] = df_clean['Temp'].interpolate(method='linear')**

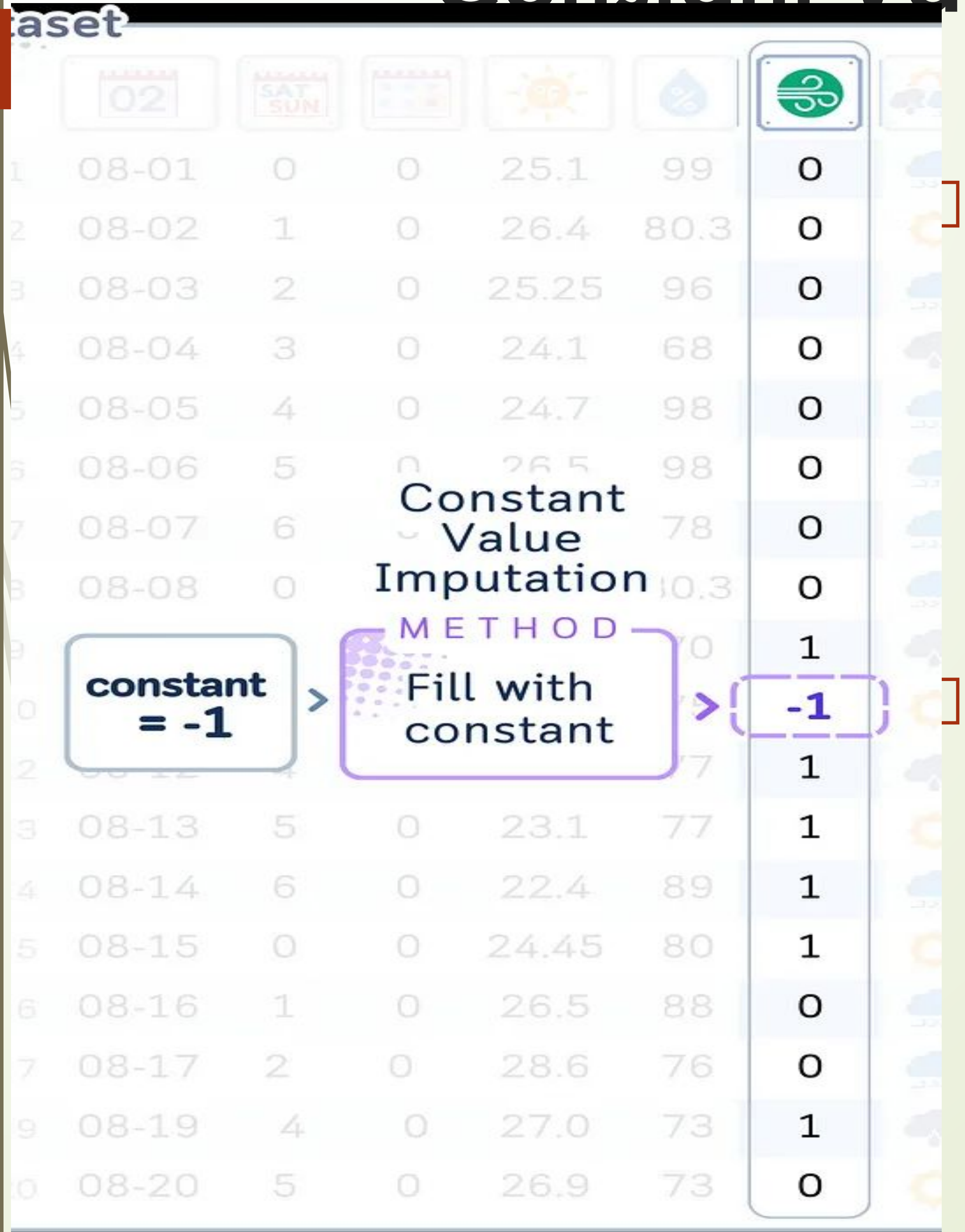


Forward/Backward Fill



- We’re using a combination of forward and backward fill for Outlook.
- Weather conditions often persist for several days, so it’s reasonable to assume that a missing Outlook value might be similar to the Outlook of the previous or following day.
- **df_clean['Outlook'] = df_clean['Outlook'].fillna(method='ffill').fillna(method='bfill')**

Constant Value Imputation



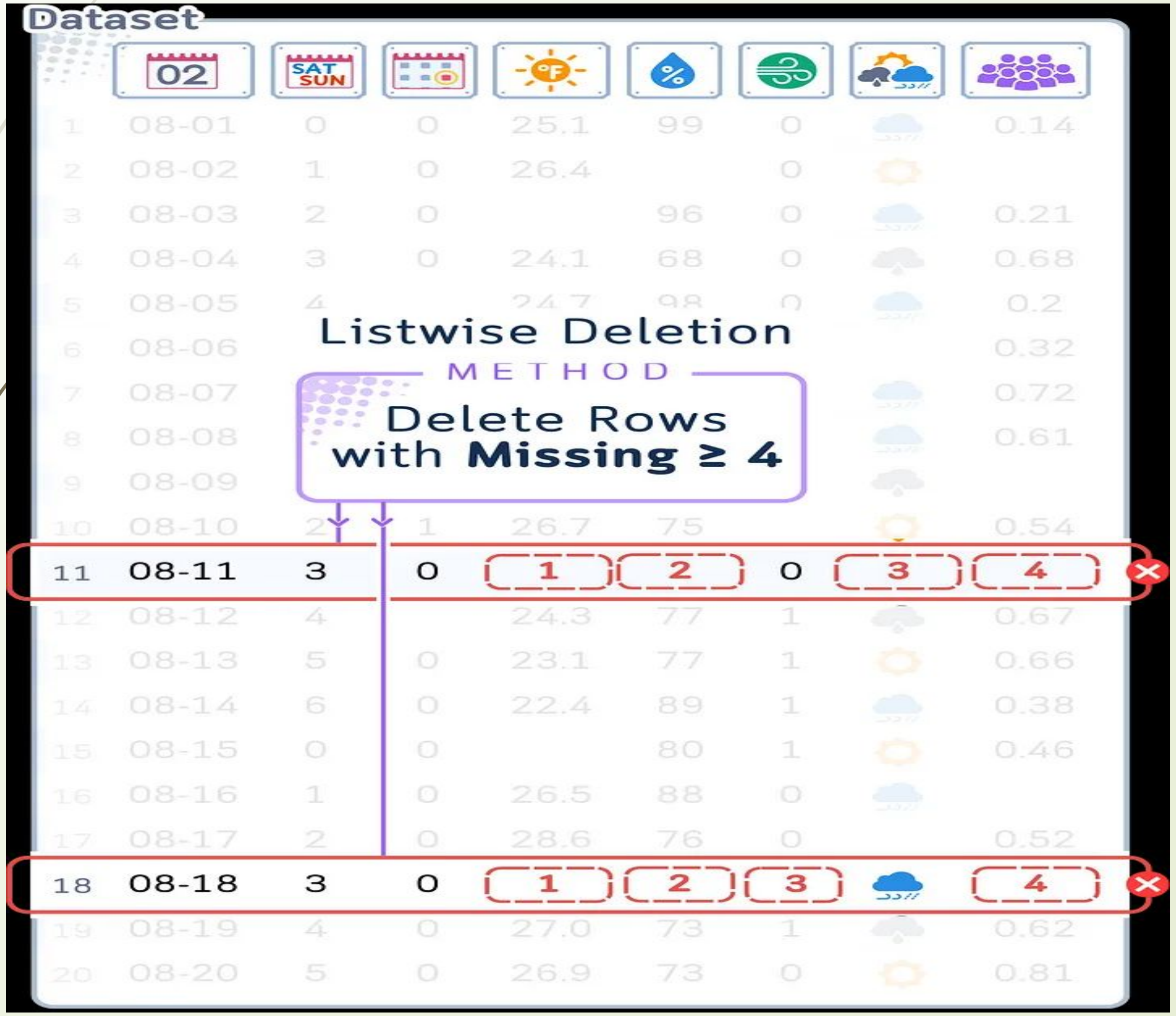
08-01	0	0	25.1	99	0
08-02	1	0	26.4	80.3	0
08-03	2	0	25.25	96	0
08-04	3	0	24.1	68	0
08-05	4	0	24.7	98	0
08-06	5	0	26.5	98	0
08-07	6	[illegible]	[illegible]	78	0
08-08	0	[illegible]	[illegible]	[illegible]	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	-1
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1
08-13	5	0	23.1	77	1
08-14	6	0	22.4	89	1
08-15	0	0	24.45	80	1
08-16	1	0	26.5	88	0
08-17	2	0	28.6	76	0
08-19	4	0	27.0	73	1
08-20	5	0	26.9	73	0

- We’re using constant value imputation for the Wind column, replacing missing values with -1. This approach explicitly flags imputed values (since -1 is outside the normal 0–1 range for Wind) and it preserves
- df_clean['Wind'] = df_clean['Wind'].fillna(-1)

Delete Rows with Missing Values

- One way of handling missing values is the deletion of the rows or columns having null values. If any columns have more than half of the values as null then you can drop the entire column. In the same way, rows can also be dropped if having one or more columns values as null. Before using this method one thing we have to keep in mind is that we should not be losing information.
- **When to delete the rows/column in a dataset?**
- If a certain column has many missing values then you can choose to drop the entire column.
- When you have a huge dataset. Deleting for e.g. 2-3 rows/columns will not make much difference.
- Output results do not depend on the Deleted data.

Example



	02	SAT SUN		°F	%			
1	08-01	0	0	25.1	99	0	☁	0.14
2	08-02	1	0	26.4		0	☀	
3	08-03	2	0		96	0	☁	0.21
4	08-04	3	0	24.1	68	0	☁	0.68
5	08-05	4		24.7	98	0	☁	0.2
6	08-06							0.32
7	08-07						☁	0.72
8	08-08						☁	0.61
9	08-09						☁	
10	08-10	2	1	26.7	75		☀	0.54
11	08-11	3	0	1	2	0	3	4
12	08-12	4		24.3	77	1	☁	0.67
13	08-13	5	0	23.1	77	1	☀	0.66
14	08-14	6	0	22.4	89	1	☁	0.38
15	08-15	0	0		80	1	☀	0.46
16	08-16	1	0	26.5	88	0	☁	
17	08-17	2	0	28.6	76	0		0.52
18	08-18	3	0	1	2	3	☁	4
19	08-19	4	0	27.0	73	1	☁	0.62
20	08-20	5	0	26.9	73	0	☀	0.81

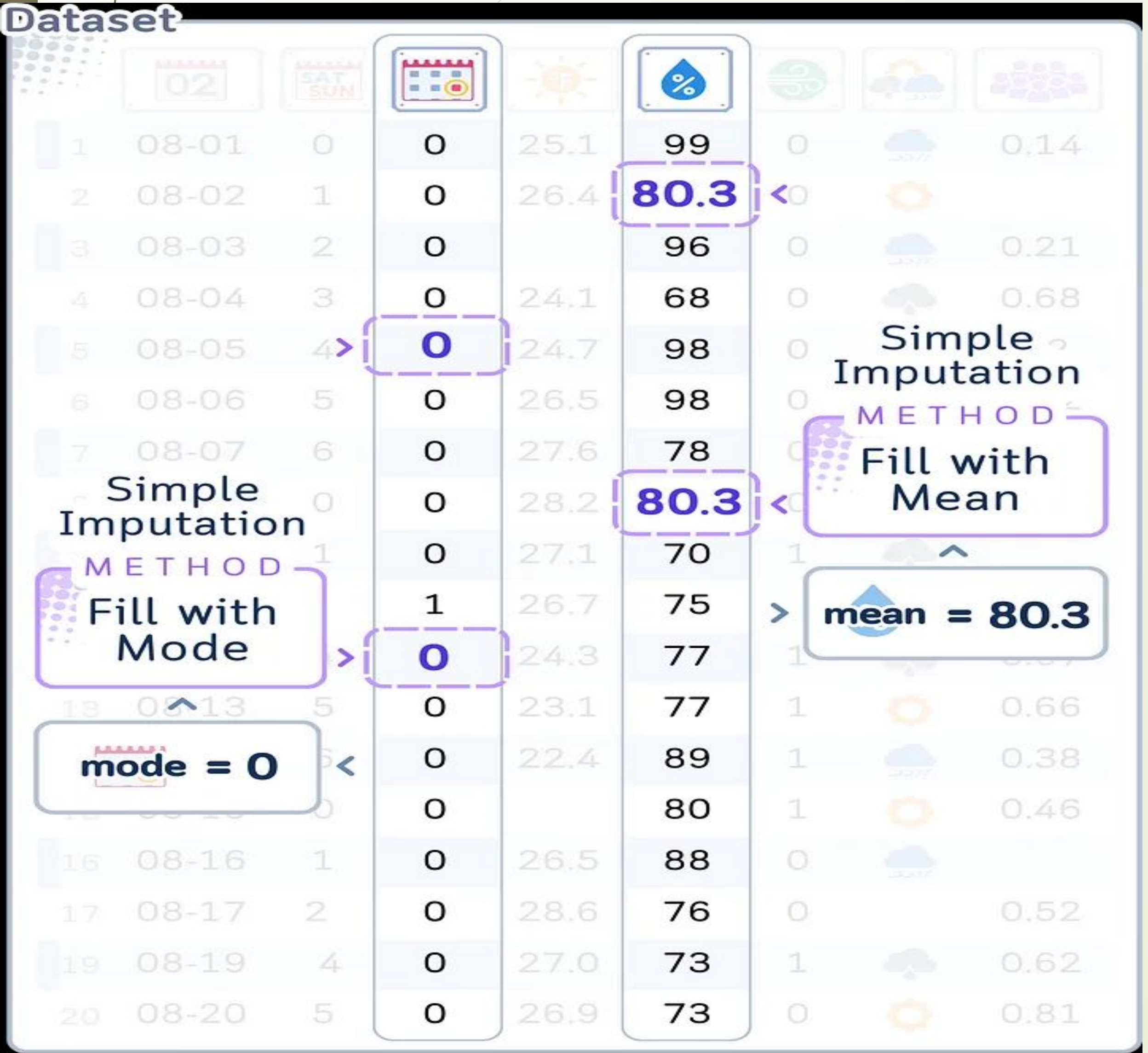
```
# Count missing values in each row
missing_count = df.isnull().sum(axis=1)

# Keep only rows with less than 4 missing value
df_clean = df[missing_count < 4].copy()
```

Drop Missing Values

- If you want to simply exclude the missing values, then use the **dropna** function along with the **axis** argument. By default, axis=0, i.e., along row,
- which means that if any value within a row is NA then the whole row is excluded.
- Pandas library provides the dropna() function that can be used to drop either columns or rows with missing data.
- **We can also use the how parameter.**
- how = 'any': at least one value must be null.
- how = 'all': all values must be null.
- **# drop all rows with atleast one NaN**
- new_df = df.dropna(axis = 0, how ='any')

Replacing NaNs with a value/Simple Imputation — Mean and Mode



- **Example:**
- df['Salary'].fillna(0)
- **Replacing NaNs using Median/Mean of the column**
- # **Mean imputation for Humidity**
- df_clean['Humidity'] = df_clean['Humidity'].fillna(df_clean['Humidity'].mean())

- **# Mode imputation for Holiday**
- df_clean['Holiday'] = df_clean['Holiday'].fillna(df_clean['Holiday'].mode()[0])

Steps to Processing Data using Pandas with Example data Set.

- We will use a simple dataset for this tutorial i.e. Highest grossing movies dataset. You can download this and other datasets from "kaggle.com."
- **1.Import dataset**
- This dataset contains data on the highest-grossing movies of each year. When working with datasets it is important to consider:first understand the data you are working with.
- **2. Exploring the data**
- The ***head()*** function is a built-in function in pandas for the dataframe used to display the rows of the dataset by default;
- We can also see how the last five rows look using the ***tail()*** function.
- The function ***memory_usage()*** returns a pandas series having the memory usage(in bytes) in a pandas dataframe.
- ***loc[:]*** can be used to access specific rows and columns as per what you require.
- ***sort_values()*** is used to sort values in a column in ascending or descending order.
- The 'inplace' attribute here is False but by specifying it to be True you can make a change in the original dataframe.

Finding and Rebuilding Missing Data

- basic statistics from your data using the simple data frame function i.e. ***describe(),*** this helps to better understand your data.
- ***value_counts()*** returns a Pandas Series containing the counts of unique values. ***value_counts()*** helps in identifying the number of occurrences of each unique value in a Series. It can be applied to columns containing data.
- **3. Finding and Rebuilding Missing Data**
- ***isnull()*** function, ***isna()*** function: This function provides the boolean value for the complete dataset to know if any null value is present or not.
- ***isna().sum()*** function: null values preset in the dataset column-wise
- ***isna().any().sum()*** function: This function gives output in a single value if any null is present or not.
- When there is a null value present in the dataset the ***fillna()*** function will fill the missing values with NA/NaN or 0

4.De-Duplicate

- This is removing all duplicate values. When analyzing data, duplicate values affect the accuracy and efficiency of the results. To find duplicate values the function ***duplicated()*** is used as seen below.
- if a dataset contains duplicate values it can be removed using the ***drop_duplicates()*** function.

THANK YOU