HANDLING NULL VALUES

NULL VALUES:

Null Values are values that appear when some fields are left blank for some records of your data**.**

NULL VALUES FOR FIELDS:

Depending on the data type of the field, there are different options to select or deselect the null values for a field.

TEXT FIELDS:

It represents a list of all values in the field editor. If some records have NULL values, NULL will be displayed as the first result in the list.

DATES:

If applied filter on a data it may be static or dynamic and NULL values will be excluded.

NUMERIC FIELDS:

For numeric fields you can exclude or include NULL values by adding a condition. Click on "**Add criteria"** in the field editor. In the drop-down list, you have two options for handling null values, "IS NULL" or "IS NOT NULL".

HANDLING NULL VALUES:

- Removal or deletion of missing value.
- Impute missing value with Mean/Median/Mode.
- Prediction Model (Regression and Classification)
- Use sklearn impute model (Simple Imputer, Iterative Imputer, KNNImputer)
- Imputation using Deep Learning library

NEED FOR HANDLING NULL VALUES:

- Missing data can reduce the accuracy of the model.
- While doing pre-processing, the visualization that we get for a particular feature can be misleading because of the presence of null values.
- The model created at the end can be biased.
- These null values can create problems in real life.

NULL VALUE HANDLING FUNCTIONS:

1. **Don't Overcomplicate Things:**

Handling nulls can be a complicated problem on its own and therefore we should make it as clean and as obvious as possible.

```
if (Optional.ofNullable(myVariable).isPresent()) // bad
if (Objects.nonNull(myVariable)) // better, but still bad
if (myVariable != null) // good
```

2. **Use Objects Methods as Stream Predicates:**

Although Objects.isNull and Objects.nonNull are not the best fit for typical null checks, they are a perfect fit to use with stream.

```
myStream.filter(Objects::nonNull)
myStream.anyMatch(Objects::isNull)
```

3. **Never Pass Null as an Argument:**

This is one of the most important principles of good coding, but if you don't know it already, you deserve an explanation. Passing null to indicate that there's no value for a given argument might seem like a viable option.

```
void kill() {
   kill(self);
}

void kill(Person person) {
   person.setDeathTime(now());
}
```

4. **Return Empty Collections Instead of Null:**

We know that null is not the best return value for a method and that we can make use of the Optional class to indicate that the value might be missing. But things are a bit different when we're talking about collections. Since a collection can contain any number of elements, it can also contain… 0 elements! There are even special empty XX methods in the Collections class that return such collections.

5. **Use Exceptions Over Nulls:**

One strange case is using null is exceptional situations. This is an inherently error prone practice, as critical errors can be omitted or resurface in different places of the system,

causing debugging to be a pain. Therefore, always throw an exception instead of returning null if something went wrong.

6.Test Your Code:

Testing your code thoroughly using an environment similar to production is a great way to prevent NPEs. Never release a piece of code without making sure it works. There's no such thing as "a quick, simple fix that doesn't require testing."

7.Double Check:

This is especially important when dealing with huge legacy databases or external providers. With the former, take the time to check if the columns you're meaning to use do not contain any null values, and if they do, check if those rows can make it into your system. In case of external providers, rely on contracts, documentation, and if you're not sure, send an email or call someone to make sure your assumptions are right