Few data science projects are exempt from the necessity of cleaning data. Data cleaning encompasses the initial steps of preparing data. Its specific purpose is that only the relevant and useful information underlying the data is retained, be it for its posterior analysis, to use as inputs to an AI or machine learning model, and so on. Unifying or converting data types, dealing with missing values, eliminating noisy values stemming from erroneous measurements, and removing duplicates are some examples of typical processes within the data cleaning stage.

As you might think, the more complex the data, the more intricate, tedious, and time-consuming the data cleaning can become, especially when implementing it manually.

This article delves into the functionalities offered by the Pandas library to automate the process of cleaning data.

Handling Data

Automating data cleaning processes with pandas boils down to systematizing the combined, sequential application of several data cleaning functions to encapsulate the sequence of actions into a single data cleaning pipeline. Before doing this, let's introduce some typically used pandas functions for diverse data cleaning steps. In the sequel, we assume an example python variable `df` that contains a dataset encapsulated in a pandas `DataFrame` object.

- **Filling missing values:** pandas provides methods for automatically dealing with missing values in a dataset, be it by replacing missing values with a “default” value using the `df.fillna()` method, or by removing any rows or columns containing missing values through the `df.dropna()` method.

- **Removing duplicated instances:** automatically removing duplicate entries (rows) in a dataset could not be easier thanks to the `df.drop_duplicates()` method, which allows the removal of extra instances when either a specific attribute value or the entire instance values are duplicated to another entry.

Representing Data

- **Manipulating strings**: some pandas functions are useful to make the format of string attributes uniform. For instance, if there is a mix of lowercase, sentence case, and uppercase values for an `'column'` attribute and we want them all to be lowercase, the `df['column'].str.lower()`method does the job.

  For removing accidentally introduced leading and trailing whitespaces, try the `df['column'].str.strip()` method.

- **Manipulating date and time:** the `pd.to_datetime(df['column'])` converts string columns containing date-time information, e.g. in the dd/mm/yyyy format, into Python datetime objects, thereby easing their further manipulation.

- **Column renaming:** automating the process of renaming columns can be particularly useful when there are multiple datasets seggregated by city, region, project, etc., and we want to add prefixes or suffixes to all or some of their columns for easing their identification. The `df.rename(columns={old_name: new_name})` method makes this possible.