# **Data Wrangling**

Data Wrangling is the process of cleaning, transforming, and structuring data for use in analysis. It is the phase in which the necessary tasks are performed to prepare the data for further analysis. This includes deduplicating data, transforming data between different formats, filling in missing data, normalizing data, and converting categorical to numeric data, among other tasks.

## Basic tasks in data analysis

Data Wrangling, also known as data preparation, is the process of cleaning and transforming data to make it more useful and easier to analyze. Within Data Wrangling, there are basic tasks that are performed to begin exploring and analyzing the imported dataset.

One of the most important tasks in data preparation is the removal of missing values. Often the data that is collected may have missing values ​​that need to be handled before it can be analysed. This can be done by removing rows or columns that have missing values, or by imputing missing values ​​with a mean, median, or mode.

Another common task is data formatting to standardize and make it consistent. For example, if a column has values ​​in different units (for example, kilometers and miles), you can convert all values ​​to a common unit (for example, kilometers) to make analysis easier.

The conversion of categorical variables into quantitative numerical variables is also common. For example, if a column has categories such as "high", "medium", and "low", these categories can be converted to numeric values ​​(for example, 3 for "high", 2 for "medium", and 1 for "low") ) to facilitate statistical analysis.

Also, before conducting more detailed data analysis, it is important to understand the structure and content of the data. This involves conducting exploratory data analysis (EDA), which involves visually and statistically examining the data for patterns, relationships, and potential errors.

Finally, it is important to be able to handle data in different formats, such as text, CSV, Excel, etc. Depending on the data format, different Data Wrangling techniques and tools can be used.

In summary, basic Data Wrangling tasks are essential to begin exploring and analyzing imported data, and are critical to ensuring the quality and reliability of analysis results.

### Handling missing values ​​in data

Handling missing data is an important task in data analysis, as incomplete data can affect the accuracy and reliability of the results. Missing data can be the result of various reasons, such as data collection errors, failures in the data entry process, or simply because the information is not available.

There are various strategies for handling missing data, some of which are described below:

#### Elimination of missing data

This strategy consists of removing the rows or columns that contain missing data. However, this strategy can be risky, as it can lead to the loss of valuable information and biased analysis.

Example:

| import pandas as pd data = {'A': [1, 2, 3, None, 5],  'B': [6, 7, None, 9, 10],  'C': [11, 12, 13, 14, 15]} df = pd.DataFrame(data) # Delete rows with missing data df = df.dropna() print(df) |
| --- |
| A B C 0 1.0 6.0 11 1 2.0 7.0 12 4 5.0 10.0 15 |

#### Imputation of missing values

Another technique for handling outliers is value imputation. This involves replacing the outliers with an estimated value. Imputation can be done in different ways, depending on the type of data and the objective of the analysis. Some common imputation techniques are:

* Mean Replacement: In this approach, outliers are replaced by the mean of non-outliers of the same variable. This approach works well when the outliers are random and do not represent a trend in the data.
* Median replacement: In this approach, outliers are replaced by the median of non-outliers of the same variable. This approach works well when the outliers are a representation of the central tendency of the data.
* Model-based imputation: This approach involves using a model to estimate missing values ​​based on available variables. For example, a regression model can be used to predict missing values ​​in a variable based on other available variables.

Let's look at an example of how value imputation can be done using the scikit-learn library in Python:

| from sklearn.impute import SimpleImputer import numpy as e.g. import pandas as pd  # Create a DataFrame with missing values df = pd.DataFrame({'A': [1, 2, np.in,4, 5],  'B': [np.in,7, 8, 9, 10],  'C': [11, 12, 13, 14, 15]}) print(df) |
| --- |
| A B C 0 1.0 NaN11 1 2.0 7.0 12 2 NaN8.0 13 3 4.0 9.0 14 4 5.0 10.0 15 |
| # Create a SimpleImputer object to replace the missing values ​​with the mean impute = SimpleImputer(strategy='mean')  # Impute missing values df\_imputed = pd.DataFrame(imputer.fit\_transform(df), columns=df.columns) print(df\_imputed) |
| A B C 0 1.0 8.5 11.0 1 2.0 7.0 12.0 2 3.0 8.0 13.0 3 4.0 9.0 14.0 4 5.0 10.0 15.0 |

In this example, we create a DataFrame with some missing values ​​in columns A and B. Next, we create a SimpleImputer object that uses the mean as the imputation strategy. Finally, we impute the missing values ​​and save the result in a new DataFrame called df\_imputed. The output shows that the missing values ​​have been replaced by the mean of each column.

#### Eliminate outliers using Z-score

The Z-score method is a common way to detect and remove outliers in a normal distribution. The method is to calculate the Z score for each value in the data set and remove any value that is above a certain threshold.

Here's an example of how to remove outliers using Z-score in Python:

| import pandas as pd import numpy as e.g.  # Create an example DataFrame df = pd.DataFrame({'A': [1, 2, 3, 4, 5],  'B': [10, 20, 30, 40, 50]}) # Add an outlier df = df.append({'A': 6, 'B': 500}, ignore\_index=True) # Calculate the Z score for each value in column 'B' z\_scores = (df['B'] - df['B'].mean()) / df['B'].std() # We remove any value that is above a threshold of 3 df = df.drop(df[z\_scores > 3].index)  print(df) |
| --- |
| A B 0 1 10 1 2 20 2 3 30 3 4 40 4 5 50 5 6 500 |

This code will create a sample DataFrame with a column named "B" that has one outlier (500 instead of values ​​close to 30). We then calculate the Z score for each value in column "B" and remove any values ​​that are above a 3 standard deviation threshold. In this case, the outlier will be removed and the resulting DataFrame will only have the normal values ​​of "B".

#### Remove outliers using the interquartile range (IQR)

The interquartile range (IQR) method is another common way to detect and remove outliers. This method uses the difference between the third quartile (Q3) and the first quartile (Q1) of the data to calculate a threshold for outliers.

Here's an example of how to remove outliers using IQR in Python:

| import pandas as pd import numpy as e.g.  # Create an example DataFrame df = pd.DataFrame({'A': [1, 2, 3, 4, 5],  'B': [10, 20, 30, 40, 50]}) # Add an outlier df = df.append({'A': 6, 'B': 500}, ignore\_index=True) # Calculate the interquartile range (IQR) for column 'B' q1 = df['B'].quantile(0.25) q3 = df['B'].quantile(0.75) iqr = q3 - q1 # Calculate the threshold for outliers upper\_bound = q3 + (1.5 \* iqr) lower\_bound = q1 - (1.5 \* iqr) # We remove any value that is above the upper threshold or below the lower threshold df = df.drop(df[(df['B'] > upper\_bound) | (df['B'] < lower\_bound)].index)  print(df) |
| --- |
| A B 0 1 10 1 2 20 2 3 30 3 4 40 4 5 50 |

In summary, handling missing data is an important step in data analysis and can be approached in different ways, depending on the data set and the objective of the analysis. It is important to carefully evaluate the available options to ensure that informed decisions are made and that accurate and reliable results are obtained.

### data formatting

Data formatting is a fundamental task in data analysis, since it allows to standardize and normalize the data to make them consistent and comparable. Standardization refers to the conversion of the data to a common scale, while normalization refers to the transformation of the data so that it has a normal distribution.

#### Standardization:

Data standardization is a transformation process in which the values ​​of a variable are scaled and shifted so that they have a mean of zero and a standard deviation of one. This process is useful when data for different variables have different scales and cannot be easily compared. Standardization allows variables to be compared on a common scale.

Here is a practical example of how to perform data standardization in Python using the NumPy library:

| import numpy as e.g.  # Create an array with random values data = np.array([10, 20, 30, 40, 50])  print("Original data:", data) |
| --- |
| > Original data: [10 20 30 40 50] |
| # Calculate the mean and standard deviation of the data mean = np.mean(data) std = np.std(datos) # We perform the standardization of the data standardized\_data = (data - mean) / std  print("Standardized data:", standardized\_data) |
| > Standardized data: [-1.41421356 -0.70710678 0. 0.70710678 1.41421356] |

In this example, we create an array with random values, and then calculate the mean and standard deviation of the data. Then, we apply the standardization formula to obtain a new array of standardized data. In this new array, we can see that the mean is zero and the standard deviation is one.

As an analogy, we can imagine that we have two people who measure their height in different units: one person measures their height in meters and the other in feet. To compare their heights fairly, we must convert both units to the same scale. In this way, we can standardize the data to be able to compare them properly.

#### Normalization:

Normalization is another common technique in data preprocessing. It consists of scaling the data to a common range, typically 0 to 1. Normalization is useful when feature values ​​have different ranges and units, and we want to make sure they all have the same impact on the model.

For example, if we have a data set that contains information about people's weight in pounds and their height in inches, we might want to normalize the data so that the weight and height values ​​have the same impact on the model. To do this, we can use the min-max normalization formula:

| x\_norm = (x - min(x)) / (max(x) - min(x)) |
| --- |

Where x is the value of the feature, min(x) is the minimum value of the feature, and max(x) is the maximum value of the feature.

Here is a practical example using Python and the Scikit-Learn library:

| from sklearn.preprocessing import MinMaxScaler import pandas as pd  # We create a DataFrame with some characteristics data = {'age': [25, 30, 35, 40, 45],  'income': [50000, 70000, 90000, 110000, 130000],  'debt': [10000, 20000, 30000, 40000, 50000]} df = pd.DataFrame(data)  # normalize the data scaler = MinMaxScaler() df\_norm = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)  print(df\_norm) |
| --- |
| age income debt 0 0.00 0.00 0.00 1 0.25 0.25 0.25 2 0.50 0.50 0.50 3 0.75 0.75 0.75 4 1.00 1.00 1.00 |

In this example, we create a DataFrame with three characteristics: age, income, and debt. Next, we use Scikit-Learn's MinMaxScaler object to normalize the data. The result is a new DataFrame df\_norm with the same data as df, but normalized between 0 and 1.

The normalization of the data can also be visualized using a graph. Here is an example showing the difference between the original data and the normalized data:

| import matplotlib.pyplot as plt  # We plot the original and normalized data fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 4))  df.plot(ax=ax1) ax1.set\_title('Original data')  df\_norm.plot(ax=ax2) ax2.set\_title('Normalized data') |
| --- |
|  |

In this example, we use the Matplotlib library to plot the original and normalized data. The result is a graph showing the difference between the two data sets. It can be seen that the normalized data are all in the range from 0 to 1, while the original data have different scales.

In summary, both standardization and normalization are useful techniques in data pre-processing that can help improve the performance of machine learning models. It is important to note that these techniques should be applied after cleaning and handling of missing or inconsistent data.

### Categorical variables and quantitative numerical variables.

### Raw data analysis.

### Data handling in different formats

When we work with data, we often come across different file formats that contain the information we need. It is important to know how to import these files and how to handle them in Python in order to work with them.

One of the most used libraries to import data in Python is pandas. Pandas allows us to read files in various formats such as text, CSV, Excel, and many others. Here are some examples of how we can import different types of files with pandas:

| import pandas as pd df = pd.read\_csv('data.txt', sep='\t') |
| --- |

In this example, we are importing a text file called "data.txt" that is separated by tabs. The pandas read\_csv method allows us to specify the separator to use in the file.

| import pandas as pd df = pd.read\_csv('data.csv') |
| --- |

In this example, we are importing a CSV file named "data.csv". Pandas automatically recognizes that the file is separated by commas.

| import pandas as pd df = pd.read\_excel('data.xlsx', sheet\_name='Hoja1') |
| --- |

In this example, we are importing an Excel file named "data.xlsx" and specifying that we want to import the sheet named "Sheet1".

Once we have imported the data, we can perform different data wrangling tasks to clean and transform the data. For example, we can remove rows or columns that are not relevant to our analysis, change the data type of a column, or join two DataFrames. Pandas has many functions and methods that allow us to perform these tasks easily and efficiently.

In short, knowing how to import and handle data in different formats is essential for any data analysis. Pandas is a very useful tool for working with data in Python, and it allows us to import and manipulate data in different formats easily and efficiently.

In conclusion, the Data Wrangling module could be compared to the task of preparing a dinner. Just like cooking a dinner, Data Wrangling requires some basic tasks to begin exploring and analyzing the imported dataset, such as cleaning, filtering, and selecting the relevant data.

Just like at a dinner party, sometimes ingredients are missing or damaged, meaning they can't be used in the dish. Similarly, the data may contain missing values ​​or errors, requiring careful manipulation to make it fit the data set. Also, just like in the kitchen, it is important to standardize and make the data consistent to ensure that the analysis is accurate and consistent.

In general, Data Wrangling is an important and critical process in data analysis, as it ensures that the data is clean, accurate, and suitable for use in analysis and decision making.