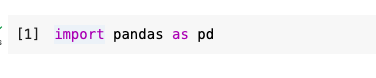
DATA MANIPULATION WITH PANDAS

Pandas is a powerful **data manipulation** and **analysis** library in Python. It provides flexible data structures to efficiently store and **manipulate large datasets**. It provides efficient data structures like **Series** (1D) and **DataFrame** (2D) to handle structured data.

Pandas is built on top of the NumPy package, plotting functions from Matplotlib, and machine learning algorithms in Scikit-learn.

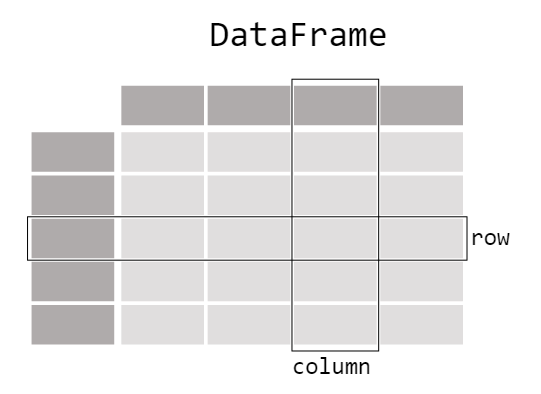
In order to use pandas in Python, it has to be imported to the platform as follows



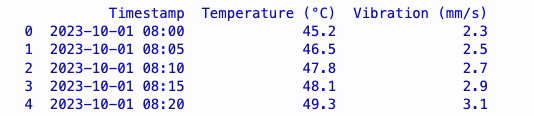
**Pandas Data Structures**

Pandas has two data structures namely Series and DataFrame.

It is represented in tabular form as shown below

pandas: Data Table Representation

This can be interpreted as table entries, could be from sensors, and their corresponding entries for different columns such as Timestamp Temperature (°C) and Vibration (mm/s) as shown below.

****

DataFrame representation of sensor data

**Core components of pandas: Series and DataFrames**

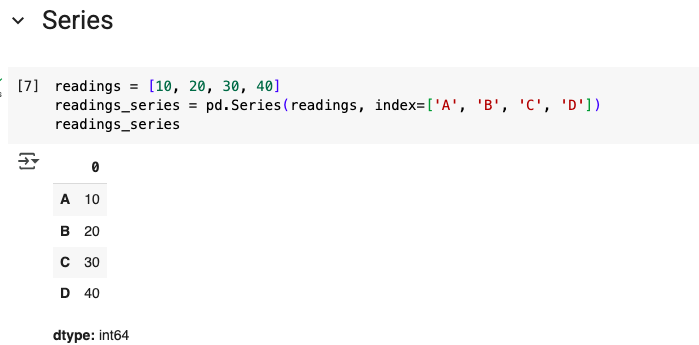
The primary two components of pandas are the **Series** and **DataFrame**.

**1. Series (1D Data Structure)**

A Pandas Series is a one-dimensional array capable of holding different data types. It is essentially a column.

* A Series consists of **data values** and an **index**.
* The index can be customized to provide meaningful labels.

In order to create a Series we first import pandas then use the inbuilt function Series to convert the data to a Series as shown below



**2. DataFrame (2D Data Structure)**

DataFrame is a 2-dimensional table made up of a collection of Series. The data is aligned in a tabular fashion in rows and columns. It is a collection of 2 or more Series.

* Each **column** is represented as a **Series**.
* The columns can be of different types
* Can perform arithmetic operations on rows and columns
* The axes (*rows* and *columns*) are labeled.

| **Data Structure** | **Dimensions** | **Description** |
| --- | --- | --- |
| Series | 1 | 1D labeled homogeneous array with immutable size |
| Data Frames | 2 | General 2D labeled, size mutable tabular structure  with potentially heterogeneously typed columns. |

**pandas.DataFrame**

**DataFrame** is primary pandas data structure.

**pandas.DataFrame(data, index, columns, dtype, copy)**

It has the following parameters

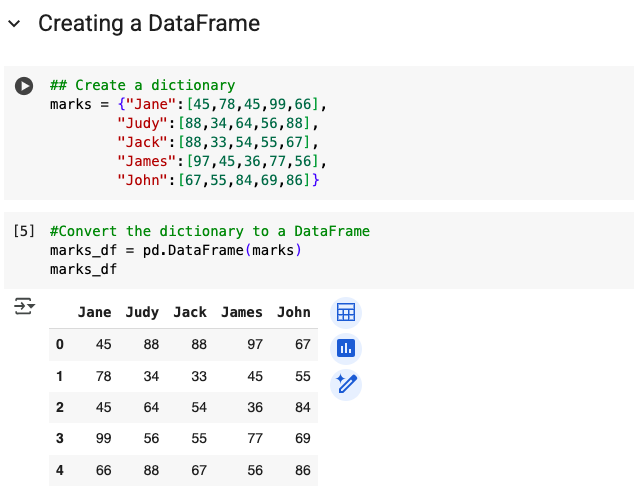
* **data**: data takes various forms like *ndarray*, *series*, *map*, *lists*, *dict*, constants and also another *DataFrame*.
* **index**: For the **row labels**, that are to be used for the resulting frame, Optional, Default is *np.arrange(n)* if no index is passed.
* **columns:** For **column labels**, the optional default syntax is - *np.arrange(n)*. This is only true if no index is passed.
* **dtype:** Data type of each column.
* **copy:** This command (or whatever it is) is used for copying of data, if the default is False.

**Creating a DataFrame**

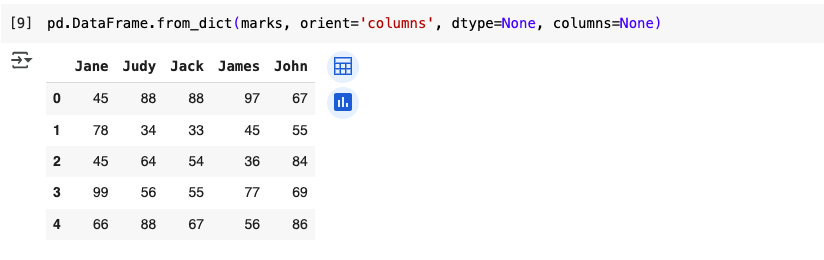
A DataFrame can be created using multiple methods, including dictionaries, lists, csv files, and more.

1. Creating a dataFrame from a dictionary

To convert a dictionary into DataFrame, we use **pd.DataFrame**

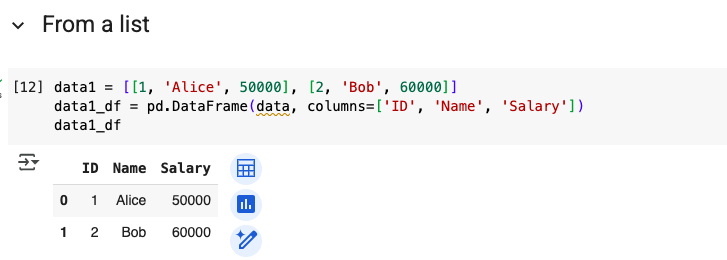
****

Another option is to use **from\_dict** function.

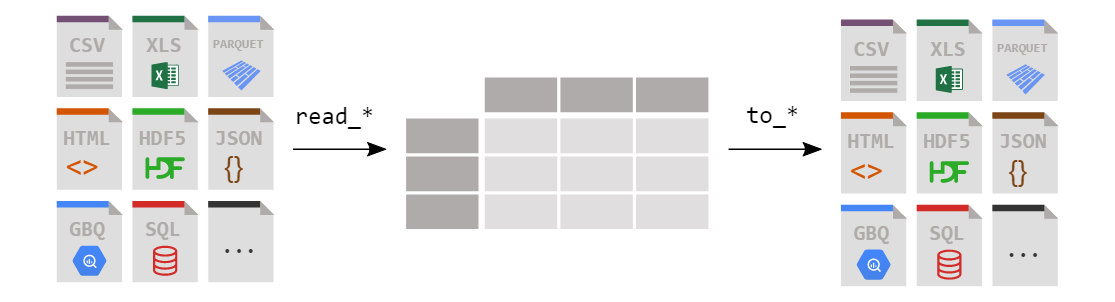
****

1. Creating a dataFrame from a list of lists

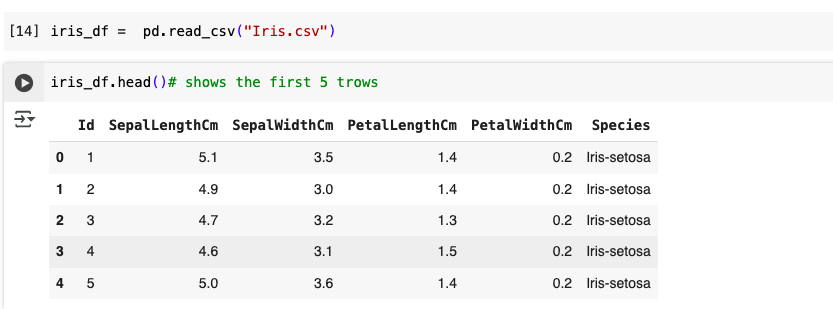
To convert a list of list into DataFrame, we use **pd.DataFrame**



1. Creating a dataFrame from a CSV file

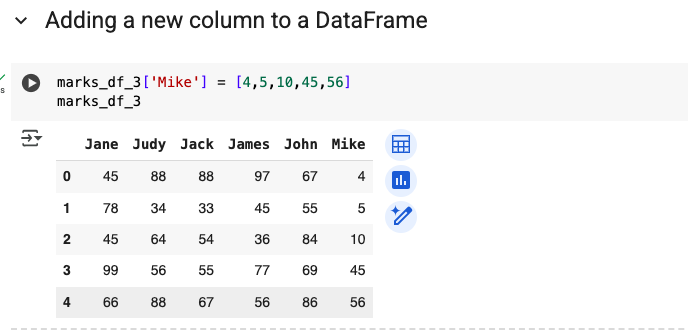
To read a CSV or Excel file into a pandas DataFrame, we use the **pd.read\_csv()** and **pd.read\_excel()** respectively.

The statement below will read a dataset called ‘iris.csv’ and load it as a DataFrame.

****

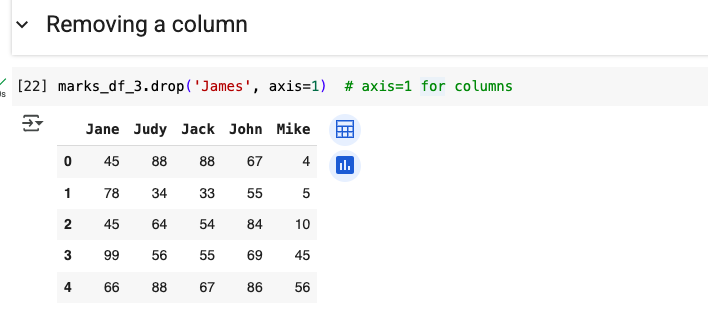
**Manipulating DataFrame**

Pandas dataFrame has inbuilt functions to add a new column/row, delete a column/row or alter column.



* **Removing a column**

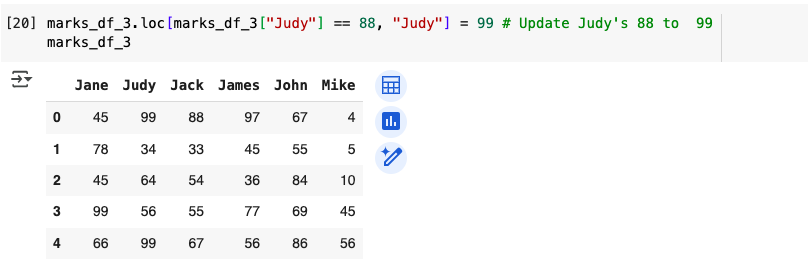
To delete a column in a DataFrame, you can use the drop function or del function.



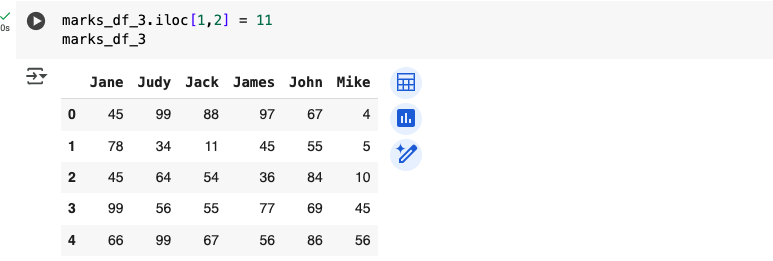
* **Updating values in a DataFrame**

Using loc function for a single entry;

It follows the syntax df.loc[df['column\_name'] == condition, 'column\_name'] = new\_value

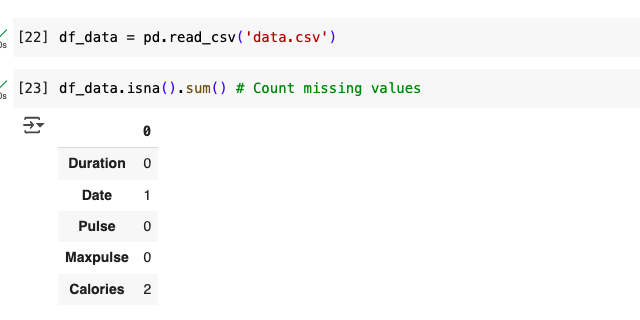
****

Using iloc

****

**Handling missing data**

* Checking for missing values

****

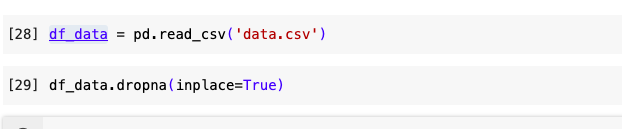
* Filling missing values

**df.fillna(0, inplace=True) # Replace NaN with 0**

****

* Dropping Rows with Missing Values

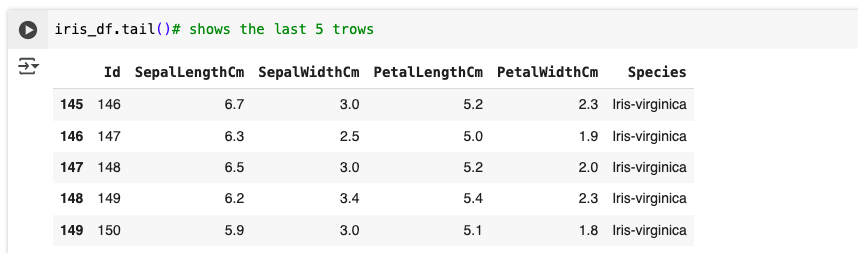
**df.dropna(inplace=True)**

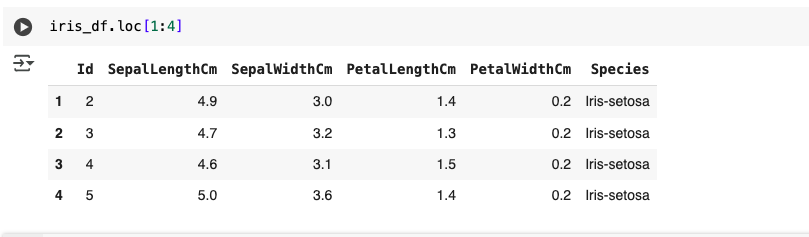
****

**Viewing your data**

In order to understand how your dataset looks, you need to view the first few rows.. We accomplish this with  **.head()**:

**head()** outputs the first five rows of your DataFrame by default, but we could also pass a number as well like **df\_purchases.head(10**) to output the top ten rows.



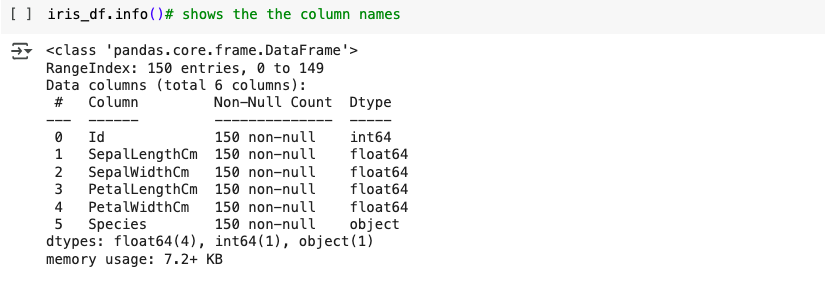


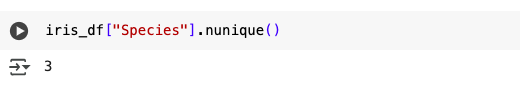
* + To view the last five rows use .tail() that also accepts a number.

**df\_purchases.tail()**

**Getting info about your data**

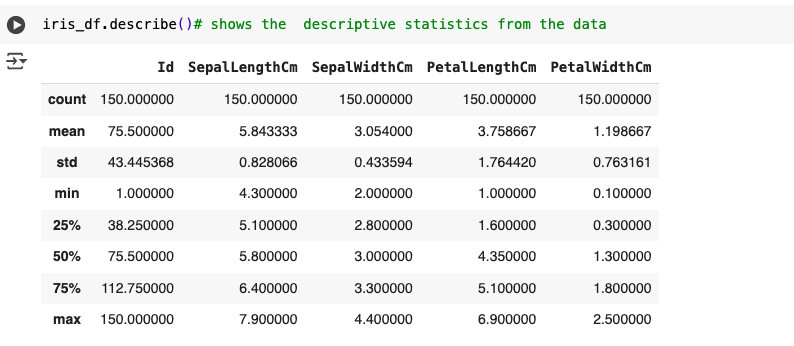
**info()** provides the essential details about the dataset, such as the number of rows and columns, the number of non-null values and what type of data is in each column.

****

****

**Understanding your variables**

The **.describe()** function in Pandas provides summary statistics for numerical columns in a DataFrame. It helps in quick data exploration by showing key statistical measures.

****

**Saving data to a file**

To save a DataFrame into csv or Excel, we use **to\_csv** and **to\_excel** respectively.

****

**Exercise 1: Creating and manipulating Data frame from dictionary**

Given the following sample data

**data = {**

**'Material': ['Steel', 'Aluminum', 'Titanium', 'Copper', 'Brass'],**

**'Tensile\_Strength\_MPa': [400, 310, 900, 210, None], # Strength in Megapascals**

**'Hardness\_HB': [150, 95, 330, 89, 170], # Brinell Hardness**

**'Density\_g\_cm3': [7.8, 2.7, 4.5, None, 8.4] # Density in g/cm³**

**}**

1. Create a DataFrame called **data\_df** from this dictionary
2. Identify and handle missing values. Replace missing values with the mean of the respective columns.

### **Exercise 2: Data Cleaning and Transformation**

#### Given the following Dataset

# Sample data

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],

'Math': [78, 85, 95, None, 88],

'Science': [88, 92, 90, 70, None],

'English': [None, 78, 85, 80, 83]

}

)

1. Identify and handle missing values:
   * Replace missing values with the mean of the respective columns.
2. Add a new column:
   * Calculate the average score for each student across all subjects and add it as a new column named 'Average'.
3. Filter the data:
   * Select students with an average score above 80