```
# EMILIO BERBER MALDONADO - A01640603
# ACT 2: Cartwheel and Iris
# SEMANA TEC
```

Data management using Pandas

Data management is a crucial component to statistical analysis and data science work.

This notebook will show you how to import, view, undertand, and manage your data using the Pandas data processing library, i.e., the notebook will demonstrates how to read a dataset into Python, and obtain a basic understanding of its content.

Note that **Python** by itself is a general-purpose programming language and does not provide high-level data processing capabilities. The **Pandas** library was developed to meet this need. **Pandas** is the most popular Python library for data manipulation, and we will use it extensively in this course. **Pandas** provides high-performance, easy-to-use data structures and data analysis tools.

The main data structure that **Pandas** works with is called a **Data Frame**. This is a two-dimensional table of data in which the rows typically represent cases and the columns represent variables (e.g. data used in this tutorial). Pandas also has a one-dimensional data structure called a **Series** that we will encounter when accesing a single column of a Data Frame.

Pandas has a variety of functions named `read_xxx` for reading data in different formats. Right now we will focus on reading `csv` files, which stands for comma-separated values. However the other file formats include `excel`, `json`, and `sql`.

There are many other options to `read_csv` that are very useful. For example, you would use the option `sep='\t'` instead of the default `sep=','` if the fields of your data file are delimited by tabs instead of commas. See here for the full documentation for `read_csv`.

Acknowledgments

- The dataset used in this tutorial is from https://www.coursera.org/ from the course "Understanding and Visualizing Data with Python" by University of Michigan

Importing libraries

```
# Import the packages that we will be using
import pandas as pd
```

Importing data

```
# Define where you are running the code: colab or local
RunInColab          = True     # (False: no  | True: yes)

# If running in colab:
if RunInColab:
    # Mount your google drive in google colab
    from google.colab import drive
    drive.mount('/content/drive')

    # Find location
    #!pwd
    #!ls
    #!ls "/content/drive/My Drive/Colab Notebooks/MachineLearningWithPython/"

    # Define path del proyecto
    Ruta            = "/content/drive/My Drive/Colab Notebooks/MachineLearningWithPython/"

else:
    # Define path del proyecto
    Ruta            = ""
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/conte
```

```
# url string that hosts our .csv file
url = "/content/drive/MyDrive/cartwheel.csv"
# Read the .csv file and store it as a pandas Data Frame
data = pd.read_csv(url)
data
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
0	1	56.0	F	1	Y	1	62.00	61.0	79	Y
1	2	26.0	F	1	Y	1	62.00	60.0	70	Y
2	3	33.0	F	1	Y	1	66.00	64.0	85	Y
3	4	39.0	F	1	N	0	64.00	63.0	87	Y
4	5	27.0	M	2	N	0	73.00	75.0	72	N
5	6	24.0	M	2	N	0	75.00	71.0	81	N
6	7	28.0	M	2	N	0	75.00	76.0	107	Y
7	8	22.0	F	1	N	0	65.00	62.0	98	Y
8	9	29.0	M	2	Y	1	74.00	73.0	106	N
9	10	33.0	F	1	Y	1	63.00	60.0	65	Y
10	11	30.0	M	2	Y	1	69.50	66.0	96	Y
11	12	28.0	F	1	Y	1	62.75	58.0	79	Y
12	13	25.0	F	1	Y	1	65.00	64.5	92	Y
13	14	23.0	F	1	N	0	61.50	57.5	66	Y
14	15	31.0	M	2	Y	1	73.00	74.0	72	Y
15	16	26.0	M	2	Y	1	71.00	72.0	115	Y
16	17	26.0	F	1	N	0	61.50	59.5	90	N
17	18	27.0	M	2	N	0	66.00	66.0	74	Y
18	19	23.0	M	2	Y	1	70.00	69.0	64	Y
19	20	24.0	F	1	Y	1	68.00	66.0	85	Y
20	21	23.0	M	2	Y	1	69.00	67.0	66	N
21	22	29.0	M	2	N	0	71.00	70.0	101	Y
22	23	25.0	M	2	N	0	70.00	68.0	82	Y
23	24	26.0	M	2	N	0	69.00	71.0	63	Y
24	25	23.0	F	1	Y	1	65.00	63.0	67	N
25	26	28.0	M	2	N	0	75.00	76.0	111	Y
26	27	24.0	M	2	N	0	78.40	71.0	92	Y
27	28	25.0	M	2	Y	1	76.00	73.0	107	Y
28	29	32.0	F	1	Y	1	63.00	60.0	75	Y
29	30	38.0	F	1	Y	1	61.50	61.0	78	Y
30	31	27.0	F	1	Y	1	62.00	60.0	72	Y
31	32	33.0	F	1	Y	1	65.30	64.0	91	Y
32	33	38.0	F	1	N	0	64.00	63.0	86	Y

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/cont
```

If we want to print the information about th output object type we would simply type the following: type(df)

```
type(data)
```

```
pandas.core.frame.DataFrame
```

41	42	26.0	M	2	Y	1	73.50	72.0	115	Y

Exploring the content of the data set

Use the `shape` method to determine the numbers of rows and columns in a data frame. This can be used to confirm that we have actually obtained the data the we are expecting.

Based on what we see below, the data set being read here has N_r rows, corresponding to N_r observations, and N_c columns, corresponding to N_c variables in this particular data file.

```
Nr = data.shape[0]
Nr
```

```
52
```

51	52	27.0	M	2	N	0	NaN	71.5	103	Y

```
Nc = data.shape[1]
Nc
```

```
12
```

```
data.shape
```

```
(52, 12)
```

If we want to show the entire data frame we would simply write the following:

```
data
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
0	1	56.0	F	1	Y	1	62.00	61.0	79	Y
1	2	26.0	F	1	Y	1	62.00	60.0	70	Y
2	3	33.0	F	1	Y	1	66.00	64.0	85	Y
3	4	39.0	F	1	N	0	64.00	63.0	87	Y
4	5	27.0	M	2	N	0	73.00	75.0	72	N
5	6	24.0	M	2	N	0	75.00	71.0	81	N
6	7	28.0	M	2	N	0	75.00	76.0	107	Y
7	8	22.0	F	1	N	0	65.00	62.0	98	Y
8	9	29.0	M	2	Y	1	74.00	73.0	106	N
9	10	33.0	F	1	Y	1	63.00	60.0	65	Y
10	11	30.0	M	2	Y	1	69.50	66.0	96	Y
11	12	28.0	F	1	Y	1	62.75	58.0	79	Y
12	13	25.0	F	1	Y	1	65.00	64.5	92	Y
13	14	23.0	F	1	N	0	61.50	57.5	66	Y
14	15	31.0	M	2	Y	1	73.00	74.0	72	Y
15	16	26.0	M	2	Y	1	71.00	72.0	115	Y
16	17	26.0	F	1	N	0	61.50	59.5	90	N
17	18	27.0	M	2	N	0	66.00	66.0	74	Y
18	19	23.0	M	2	Y	1	70.00	69.0	64	Y
19	20	24.0	F	1	Y	1	68.00	66.0	85	Y
20	21	23.0	M	2	Y	1	69.00	67.0	66	N
21	22	29.0	M	2	N	0	71.00	70.0	101	Y
22	23	25.0	M	2	N	0	70.00	68.0	82	Y
23	24	26.0	M	2	N	0	69.00	71.0	63	Y
24	25	23.0	F	1	Y	1	65.00	63.0	67	N
25	26	28.0	M	2	N	0	75.00	76.0	111	Y
26	27	24.0	M	2	N	0	78.40	71.0	92	Y
27	28	25.0	M	2	Y	1	76.00	73.0	107	Y
28	29	32.0	F	1	Y	1	63.00	60.0	75	Y
29	30	38.0	F	1	Y	1	61.50	61.0	78	Y
30	31	27.0	F	1	Y	1	62.00	60.0	72	Y
31	32	33.0	F	1	Y	1	65.30	64.0	91	Y
32	33	38.0	F	1	N	0	64.00	63.0	86	Y

As you can see, we have a 2-Dimensional object where each row is an independent observation and each coloum is a variable.

Now, use the the `head()` function to show the first 5 rows of our data frame

```
data.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y
3	4	39.0	F	1	N	0	64.0	63.0	87	Y
4	5	27.0	M	2	N	0	73.0	75.0	72	N

Also, you can use the the `tail()` function to show the last 5 rows of our data frame

```
data.tail()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
47	48	24.0	M	2	N	0	79.5	75.0	82	N
48	49	28.0	M	2	N	0	77.8	76.0	99	Y
49	50	30.0	F	1	N	0	74.6	NaN	71	Y
50	51	NaN	M	2	N	0	71.0	70.0	101	Y
51	52	27.0	M	2	N	0	NaN	71.5	103	Y

The columns in a Pandas data frame have names, to see the names, use the `columns` method:

To gather more information regarding the data, we can view the column names with the following function:

```
data.columns
```

```
Index(['ID', 'Age', 'Gender', 'GenderGroup', 'Glasses', 'GlassesGroup',
       'Height', 'Wingspan', 'CWDistance', 'Complete', 'CompleteGroup',
       'Score'],
      dtype='object')
```

Be aware that every variable in a Pandas data frame has a data type. There are many different data types, but most commonly you will encounter floating point values (real numbers), integers, strings (text), and date/time values. When Pandas reads a text/csv file, it guesses the data types based on what it sees in the first few rows of the data file. Usually it selects an appropriate type, but occasionally it does not. To confirm that the data types are consistent with what the variables represent, inspect the `dtypes` attribute of the data frame.

```
data.dtypes
```

```
ID                 int64
Age              float64
Gender            object
GenderGroup        int64
Glasses           object
GlassesGroup       int64
Height           float64
Wingspan         float64
CWDistance         int64
Complete          object
CompleteGroup    float64
Score              int64
dtype: object
```

Summary statistics, which include things like the mean, min, and max of the data, can be useful to get a feel for how large some of the variables are and what variables may be the most important.

```
# Summary statistics for the quantitative variables
data.Age.min()
```

```
22.0
```

```
# Drop observations with NaN values
#df.Age.dropna().describe()
data.Age.dropna().describe()
#df.Wingspan.dropna().describe()
```

```
count    51.000000
mean     28.411765
std       5.755611
min      22.000000
25%      25.000000
50%      27.000000
75%      30.000000
max      56.000000
Name: Age, dtype: float64
```

It is also possible to get statistics on the entire data frame or a column as follows

- `df.mean()` Returns the mean of all columns
- `df.corr()` Returns the correlation between columns in a data frame
- `df.count()` Returns the number of non-null values in each data frame column
- `df.max()` Returns the highest value in each column
- `df.min()` Returns the lowest value in each column
- `df.median()` Returns the median of each column
- `df.std()` Returns the standard deviation of each column

```
data.max()
```

```
ID                52
Age             56.0
Gender             M
GenderGroup        2
Glasses            Y
GlassesGroup       1
Height          79.5
Wingspan        76.0
CWDistance       115
Complete           Y
CompleteGroup    1.0
Score             10
dtype: object
```

How to write a data frame to a File

To save a file with your data simply use the `to_csv` attribute

Examples:

- df.to_csv('myDataFrame.csv')
- df.to_csv('myDataFrame.csv', sep='\t')

```
data.to_csv("myDataFrame.csv", sep='\t')
```

Rename columns

To change the name of a colum use the `rename` attribute

Example:

df = df.rename(columns={"Age": "Edad"})

df.head()

```
data = data.rename(columns={"Age":"Edad"})
data.head()
```

	ID	Edad	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y
3	4	39.0	F	1	N	0	64.0	63.0	87	Y
4	5	27.0	M	2	N	0	73.0	75.0	72	N

```
# Back to the original name
data = data.rename(columns={"Edad":"Age"})
data.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	C
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y	
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y	
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y	
3	4	39.0	F	1	N	0	64.0	63.0	87	Y	
4	5	27.0	M	2	N	0	73.0	75.0	72	N	

Selection of colums

As discussed above, a Pandas data frame is a rectangular data table, in which the rows represent observations or samples and the columns represent variables. One common manipulation of a data frame is to extract the data for one case or for one variable. There are several ways to do this, as shown below.

To extract all the values for one column (variable), use one of the following alternatives.

```
#a = df.Age
#b = df["Age"]
#c = df.loc[:, "Age"]
#d = df.iloc[:, 1]

a = data.Age
b = data["Age"]
c = data.loc[:, "Age"]
d = data.iloc[:, 1]

print(d)

#df[["Gender", "GenderGroup"]]
data[["Gender", "GenderGroup"]]
```

```
0     56.0
1     26.0
2     33.0
3     39.0
4     27.0
5     24.0
6     28.0
7     22.0
8     29.0
9     33.0
10    30.0
11    28.0
12    25.0
13    23.0
14    31.0
15    26.0
16    26.0
17    27.0
18    23.0
19    24.0
20    23.0
21    29.0
22    25.0
23    26.0
24    23.0
25    28.0
26    24.0
27    25.0
28    32.0
29    38.0
30    27.0
31    33.0
32    38.0
33    27.0
34    24.0
35    27.0
36    25.0
37    26.0
38    31.0
39    30.0
40    23.0
41    26.0
42    28.0
43    26.0
44    30.0
45    39.0
46    27.0
47    24.0
48    28.0
49    30.0
50     NaN
51    27.0
Name: Age, dtype: float64
```

	Gender	GenderGroup
0	F	1
1	F	1
2	F	1
3	F	1

Slicing a data set

As discussed above, a Pandas data frame is a rectangular data table, in which the rows represent cases and the columns represent variables. One common manipulation of a data frame is to extract the data for one observation or for one variable. There are several ways to do this, as shown below.

Lets say we would like to splice our data frame and select only specific portions of our data. There are three different ways of doing so.

1. .loc()
2. .iloc()
3. .ix()

We will cover the .loc() and .iloc() splicing functions.

The attibute **.loc()** uses labels/column names, in specific, it takes two single/list/range operator separated by ',', the first one indicates the rows and the second one indicates columns.

```
############ LOC
# Return all observations of CWDistance
data.loc[:,"CWDistance"]

# Return a subset of observations of CWDistance
data.loc[:9, "CWDistance"]

# Select all rows for multiple columns, ["Gender", "GenderGroup"]
data.loc[:,["Gender", "GenderGroup"]]

# Select multiple columns, ["Gender", "GenderGroup"]me
keep = ['Gender', 'GenderGroup']
data_gender = data[keep]
data_gender

# Select few rows for multiple columns, ["CWDistance", "Height", "Wingspan"]
data.loc[4:9, ["CWDistance", "Height", "Wingspan"]]

# Select range of rows for all columns
data.loc[10:15,:]
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
10	11	30.0	M	2	Y	1	69.50	66.0	96	Y
11	12	28.0	F	1	Y	1	62.75	58.0	79	Y

The attribute **iloc()** is an integer based slicing.

```
############ ILOC
# All rows from the 0 column to de 4th
#df.iloc[:, :4]
data.iloc[:,:4]

# All columns from the 0 row to the 3rth
#df.iloc[:4, :]
data.iloc[:4, :]

# All rows from the col 3 to 6
#df.iloc[:, 3:7]
data.iloc[:, 3:7]

# From the rows 4 to 8 and cols froms the 2 to the 3rd
#df.iloc[4:8, 2:4]
data.iloc[4:8, 2:4]

# This is incorrect:
#data.iloc[1:5, ["Gender", "GenderGroup"]] --> ILOC SOLO ACEPTA INDEX
```

	Gender	GenderGroup
4	M	2
5	M	2
6	M	2
7	F	1

Get unique existing values

List unique values in the one of the columns

df.Gender.unique()

```
# List unique values in the df['Gender'] column
data.Gender.unique()
```

```
array(['F', 'M'], dtype=object)
```

```
# Lets explore df["GenderGroup] as well
data.GenderGroup.unique()
```

```
array([1, 2])
```

```
data.Age.unique()
```

```
array([56., 26., 33., 39., 27., 24., 28., 22., 29., 30., 25., 23., 31.,
       32., 38., nan])
```

Filter, Sort and Groupby

With **Filter** you can use different conditions to filter columns. For example, df[df[year] > 1984] would give you only the column year is greater than 1984. You can use & (and) or | (or) to add different conditions to your filtering. This is also called boolean filtering.

df[df["Height"] >= 70]

```
data[data["Height"]>=70]
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
4	5	27.0	M	2	N	0	73.0	75.0	72	N
5	6	24.0	M	2	N	0	75.0	71.0	81	N
6	7	28.0	M	2	N	0	75.0	76.0	107	Y

With **Sort** is possible to sort values in a certain column in an ascending order using `df.sort_values("ColumnName")` or in descending order using `df.sort_values(ColumnName, ascending=False)`.

Furthermore, it's possible to sort values by Column1Name in ascending order then Column2Name in descending order by using `df.sort_values([Column1Name,Column2Name],ascending=[True,False])`

df.sort_values("Height")

df.sort_values("Height",ascending=False)

```
data.sort_values("Height", ascending = False)
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
47	48	24.0	M	2	N	0	79.50	75.0	82	N
26	27	24.0	M	2	N	0	78.40	71.0	92	Y
46	47	27.0	M	2	N	0	78.00	75.0	72	N
48	49	28.0	M	2	N	0	77.80	76.0	99	Y
33	34	27.0	M	2	N	0	77.00	75.0	100	Y
27	28	25.0	M	2	Y	1	76.00	73.0	107	Y
5	6	24.0	M	2	N	0	75.00	71.0	81	N
6	7	28.0	M	2	N	0	75.00	76.0	107	Y
25	26	28.0	M	2	N	0	75.00	76.0	111	Y
49	50	30.0	F	1	N	0	74.60	NaN	71	Y
8	9	29.0	M	2	Y	1	74.00	73.0	106	N
41	42	26.0	M	2	Y	1	73.50	72.0	115	Y
4	5	27.0	M	2	N	0	73.00	75.0	72	N
38	39	31.0	M	2	Y	1	73.00	74.0	72	Y
14	15	31.0	M	2	Y	1	73.00	74.0	72	Y
42	43	28.0	F	1	Y	1	72.50	72.0	81	Y
43	44	26.0	F	1	Y	1	72.00	72.0	92	Y
21	22	29.0	M	2	N	0	71.00	70.0	101	Y
50	51	NaN	M	2	N	0	71.00	70.0	101	Y
15	16	26.0	M	2	Y	1	71.00	72.0	115	Y
40	41	23.0	F	1	N	0	70.40	71.0	66	Y
18	19	23.0	M	2	Y	1	70.00	69.0	64	Y
22	23	25.0	M	2	N	0	70.00	68.0	82	Y
10	11	30.0	M	2	Y	1	69.50	66.0	96	Y
39	40	30.0	M	2	Y	1	69.50	66.0	96	Y
23	24	26.0	M	2	N	0	69.00	71.0	63	Y
20	21	23.0	M	2	Y	1	69.00	67.0	66	N
19	20	24.0	F	1	Y	1	68.00	66.0	85	Y
35	36	27.0	M	2	N	0	68.00	66.0	74	Y
34	35	24.0	F	1	N	0	67.80	62.0	98	Y
2	3	33.0	F	1	Y	1	66.00	64.0	85	Y
17	18	27.0	M	2	N	0	66.00	66.0	74	Y

The attribute **Groupby** involves splitting the data into groups based on some criteria, applying a function to each group independently and combining the results into a data structure. df.groupby(col) returns a groupby object for values from one column while df.groupby([col1,col2]) returns a groupby object for values from multiple columns.

df.groupby(['Gender'])

```
data.groupby(['Gender'])
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fd737f3e160>
```

28 29 32.0 F 1 Y 1 63.00 60.0 75 Y

Size of each group

df.groupby(['Gender']).size()

df.groupby(['Gender','GenderGroup']).size()

0 1 56.0 F 1 Y 1 62.00 61.0 70 Y

```
data.groupby(['Gender']).size()
data.groupby(['Gender','GenderGroup']).size()
```

```
Gender  GenderGroup
F       1              26
M       2              26
dtype: int64
```

29 30 38.0 F 1 Y 1 61.50 61.0 78 Y

This output indicates that we have two types of combinations.

- Case 1: Gender = F & Gender Group = 1
- Case 2: Gender = M & GenderGroup = 2.

This validates our initial assumption that these two fields essentially portray the same information.

Data Cleaning: handle with missing data

Before getting started to work with your data, it's a good practice to observe it thoroughly to identify missing values and handle them accordingly.

When reading a dataset using Pandas, there is a set of values including 'NA', 'NULL', and 'NaN' that are taken by default to represent a missing value. The full list of default missing value codes is in the '`read_csv`' documentation here. This document also explains how to change the way that '`read_csv`' decides whether a variable's value is missing.

Pandas has functions called `isnull` and `notnull` that can be used to identify where the missing and non-missing values are located in a data frame.

Below we use these functions to count the number of missing and non-missing values in each variable of the datasetr.

```
data.isnull()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Comple
0	False	False	False	False	False	False	False	False	False	Fal
1	False	False	False	False	False	False	False	False	False	Fal
2	False	False	False	False	False	False	False	False	False	Fal
3	False	False	False	False	False	False	False	False	False	Fal
4	False	False	False	False	False	False	False	False	False	Fal
5	False	False	False	False	False	False	False	False	False	Fal
6	False	False	False	False	False	False	False	False	False	Fal
7	False	False	False	False	False	False	False	False	False	Fal
8	False	False	False	False	False	False	False	False	False	Fal
9	False	False	False	False	False	False	False	False	False	Fal
10	False	False	False	False	False	False	False	False	False	Fal
11	False	False	False	False	False	False	False	False	False	Fal
12	False	False	False	False	False	False	False	False	False	Fal
13	False	False	False	False	False	False	False	False	False	Fal
14	False	False	False	False	False	False	False	False	False	Fal
15	False	False	False	False	False	False	False	False	False	Fal
16	False	False	False	False	False	False	False	False	False	Fal
17	False	False	False	False	False	False	False	False	False	Fal
18	False	False	False	False	False	False	False	False	False	Fal
19	False	False	False	False	False	False	False	False	False	Fal
20	False	False	False	False	False	False	False	False	False	Fal

Unfortunately, our output indicates that some of our columns contain missing values so we are no able to continue on doing analysis with those colums

```
data.notnull()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complet
0	True	True	True	True	True	True	True	True	True	Tru
1	True	True	True	True	True	True	True	True	True	Tru
2	True	True	True	True	True	True	True	True	True	Tru
3	True	True	True	True	True	True	True	True	True	Tru
4	True	True	True	True	True	True	True	True	True	Tru
5	True	True	True	True	True	True	True	True	True	Tru
6	True	True	True	True	True	True	True	True	True	Tru
7	True	True	True	True	True	True	True	True	True	Tru
8	True	True	True	True	True	True	True	True	True	Tru
9	True	True	True	True	True	True	True	True	True	Tru
10	True	True	True	True	True	True	True	True	True	Tru
11	True	True	True	True	True	True	True	True	True	Tru
12	True	True	True	True	True	True	True	True	True	Tru
13	True	True	True	True	True	True	True	True	True	Tru
14	True	True	True	True	True	True	True	True	True	Tru
15	True	True	True	True	True	True	True	True	True	Tru
16	True	True	True	True	True	True	True	True	True	Tru
17	True	True	True	True	True	True	True	True	True	Tru
18	True	True	True	True	True	True	True	True	True	Tru
19	True	True	True	True	True	True	True	True	True	Tru
20	True	True	True	True	True	True	True	True	True	Tru
21	True	True	True	True	True	True	True	True	True	Tru
22	True	True	True	True	True	True	True	True	True	Tru
23	True	True	True	True	True	True	True	True	True	Tru
24	True	True	True	True	True	True	True	True	True	Tru
25	True	True	True	True	True	True	True	True	True	Tru
26	True	True	True	True	True	True	True	True	True	Tru
27	True	True	True	True	True	True	True	True	True	Tru
28	True	True	True	True	True	True	True	True	True	Tru
29	True	True	True	True	True	True	True	True	True	Tru
30	True	True	True	True	True	True	True	True	True	Tru
31	True	True	True	True	True	True	True	True	True	Tru
32	True	True	True	True	True	True	True	True	True	Tru

```
#df.isnull().sum()
data.isnull().sum()
```

```
ID               0
Age              1
Gender           0
GenderGroup      0
Glasses          0
GlassesGroup     0
Height           1
Wingspan         1
CWDistance       0
Complete         0
CompleteGroup    1
Score            0
dtype: int64
```

```
#df.notnull().sum()
data.notnull().sum()
```

```
ID               52
Age              51
Gender           52
GenderGroup      52
Glasses          52
GlassesGroup     52
Height           51
Wingspan         51
CWDistance       52
Complete         52
CompleteGroup    51
Score            52
dtype: int64
```

Now we use these functions to count the number of missing and non-missing values in a single variable in the dataset

print(df.Height.notnull().sum())

print(pd.isnull(df.Height).sum())

```
data.Height.isnull().sum()
```

```
1
```

```
# Extract all non-missing values of one of the columns into a new variable
x = data.Age.dropna().describe()
x
```

```
count    51.000000
mean     28.411765
std       5.755611
min      22.000000
```

```
25%      25.000000
50%      27.000000
75%      30.000000
max      56.000000
Name: Age, dtype: float64
```

Add and eliminate columns

In some cases it is useful to create or eiminate new columns

```
#df.head()
data.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	C
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y	
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y	
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y	
3	4	39.0	F	1	N	0	64.0	63.0	87	Y	
4	5	27.0	M	2	N	0	73.0	75.0	72	N	

```
# Add a new column with new data

# Create a column data
#NewColumnData = df.Age/df.Age
NewColumnData = data.Wingspan/data.CWDistance
# Insert that column in the data frame
#df.insert(12, "ColumnInserted", NewColumnData, True)
data.insert(12, "ColumnInserted", NewColumnData, True)
#df.head()
data.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	C
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y	
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y	
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y	
3	4	39.0	F	1	N	0	64.0	63.0	87	Y	
4	5	27.0	M	2	N	0	73.0	75.0	72	N	

```
# # Eliminate inserted column
# df.drop("ColumnInserted", axis=1, inplace = True)
data.drop("ColumnInserted", axis=1, inplace = True)
# #df.drop(columns=['ColumnInserted'], inplace = True)
# # Remove three columns as index base
```

```
# #df.drop(df.columns[[12]], axis = 1, inplace = True)
#
# df.head()
data.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	C
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y	
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y	
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y	
3	4	39.0	F	1	N	0	64.0	63.0	87	Y	
4	5	27.0	M	2	N	0	73.0	75.0	72	N	

```
# # Add new column derived from existing columns
#
# # The new column is a function of another column
# df["AgeInMonths"] = df["Age"] * 12
data["AgeInMonths"] = data["Age"] * 12
# df.head()
data.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	C
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y	
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y	
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y	
3	4	39.0	F	1	N	0	64.0	63.0	87	Y	
4	5	27.0	M	2	N	0	73.0	75.0	72	N	

```
# # Eliminate inserted column
# df.drop("AgeInMonths", axis=1, inplace = True)
data.drop("AgeInMonths", axis=1, inplace = True)
# df.head()
data.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	C
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y	
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y	
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y	
3	4	39.0	F	1	N	0	64.0	63.0	87	Y	
4	5	27.0	M	2	N	0	73.0	75.0	72	N	

```
# Add a new column with text labels reflecting the code's meaning

# df["GenderGroupNew"] = df.GenderGroup.replace({1: "Female", 2: "Male"})
data["GenderGroupNew"] = data.GenderGroup.replace({1: "Female", 2: "Male"})
# Show the first 5 rows of the created data frame
data.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	C
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y	
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y	
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y	
3	4	39.0	F	1	N	0	64.0	63.0	87	Y	
4	5	27.0	M	2	N	0	73.0	75.0	72	N	

```
## Eliminate inserted column
# df.drop("GenderGroupNew", axis=1, inplace = True)
data.drop("GenderGroupNew", axis=1, inplace = True)
##df.drop(['GenderGroupNew'],vaxis='columns',vinplace=True)
data.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	C
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y	
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y	
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y	
3	4	39.0	F	1	N	0	64.0	63.0	87	Y	
4	5	27.0	M	2	N	0	73.0	75.0	72	N	

```
## Add a new column with strata based on these cut points
#
## Create a column data
#NewColumnData = df.Age/df.Age
NewColumnData = data.Age/data.Age

## Insert that column in the data frame
#df.insert(1, "ColumnStrata", NewColumnData, True)
data.insert(1, "ColumnStrata", NewColumnData, True)
data
#df["ColumnStrata"] = pd.cut(df.Height, [60., 63., 66., 69., 72., 75., 78.])
data["ColumnStrata"] = pd.cut(data.Height, [60., 63., 66., 69., 72., 75., 78.])

## Show the first 5 rows of the created data frame
data.head()
```

	ID	ColumnStrata	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistanc
0	1	(60.0, 63.0]	56.0	F	1	Y	1	62.0	61.0	7
1	2	(60.0, 63.0]	26.0	F	1	Y	1	62.0	60.0	7
2	3	(63.0, 66.0]	33.0	F	1	Y	1	66.0	64.0	8
3	4	(63.0, 66.0]	39.0	F	1	N	0	64.0	63.0	8
4	5	(72.0, 75.0]	27.0	M	2	N	0	73.0	75.0	7

```
## Eliminate inserted column
#df.drop("ColumnStrata", axis=1, inplace = True)
data.drop("ColumnStrata", axis=1, inplace = True)

#df.head()
data.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	C
0	1	56.0	F	1	Y	1	62.0	61.0	79	Y	
1	2	26.0	F	1	Y	1	62.0	60.0	70	Y	
2	3	33.0	F	1	Y	1	66.0	64.0	85	Y	
3	4	39.0	F	1	N	0	64.0	63.0	87	Y	
4	5	27.0	M	2	N	0	73.0	75.0	72	N	

```
# Drop several "unused" columns
#vars = ["ID", "GenderGroup", "GlassesGroup", "CompleteGroup"]
vars = ["ID", "GenderGroup", "GlassesGroup", "CompleteGroup"]
#df.drop(vars, axis=1, inplace = True)
data.drop(vars, axis=1, inplace = True)


data
```

	Age	Gender	Glasses	Height	Wingspan	CWDistance	Complete	Score
0	56.0	F	Y	62.00	61.0	79	Y	7
1	26.0	F	Y	62.00	60.0	70	Y	8
2	33.0	F	Y	66.00	64.0	85	Y	7
3	39.0	F	N	64.00	63.0	87	Y	10
4	27.0	M	N	73.00	75.0	72	N	4
5	24.0	M	N	75.00	71.0	81	N	3
6	28.0	M	N	75.00	76.0	107	Y	10
7	22.0	F	N	65.00	62.0	98	Y	9
8	29.0	M	Y	74.00	73.0	106	N	5
9	33.0	F	Y	63.00	60.0	65	Y	8
10	30.0	M	Y	69.50	66.0	96	Y	6
11	28.0	F	Y	62.75	58.0	79	Y	10
12	25.0	F	Y	65.00	64.5	92	Y	6
13	23.0	F	N	61.50	57.5	66	Y	4
14	31.0	M	Y	73.00	74.0	72	Y	9
15	26.0	M	Y	71.00	72.0	115	Y	6
16	26.0	F	N	61.50	59.5	90	N	10
17	27.0	M	N	66.00	66.0	74	Y	5
18	23.0	M	Y	70.00	69.0	64	Y	3
19	24.0	F	Y	68.00	66.0	85	Y	8
20	23.0	M	Y	69.00	67.0	66	N	2
21	29.0	M	N	71.00	70.0	101	Y	8
22	25.0	M	N	70.00	68.0	82	Y	4
23	26.0	M	N	69.00	71.0	63	Y	5
24	23.0	F	Y	65.00	63.0	67	N	3
25	28.0	M	N	75.00	76.0	111	Y	10
26	24.0	M	N	78.40	71.0	92	Y	7
27	25.0	M	Y	76.00	73.0	107	Y	8
28	32.0	F	Y	63.00	60.0	75	Y	8
29	38.0	F	Y	61.50	61.0	78	Y	7
30	27.0	F	Y	62.00	60.0	72	Y	8
31	33.0	F	Y	65.30	64.0	91	Y	7
32	38.0	F	N	64.00	63.0	86	Y	10

Add and eliminate rows

In some cases it is requiered to add new observations (rows) to the data set

```
# Print tail
data.tail()
```

	Age	Gender	Glasses	Height	Wingspan	CWDistance	Complete	Score
47	24.0	M	N	79.5	75.0	82	N	8
48	28.0	M	N	77.8	76.0	99	Y	9
49	30.0	F	N	74.6	NaN	71	Y	9
50	NaN	M	N	71.0	70.0	101	Y	8
51	27.0	M	N	NaN	71.5	103	Y	10

```
#df.loc[len(df.index)] = [26, 24, 'F', 1, 'Y', 1, 66, 'NaN', 68, 'N', 0, 3]
data.loc[len(data.index)] = [26, 24, 'F', 1, 'Y', 1, 66, 'NaN', 68, 'N', 0, 3]

#df.tail()
data.tail()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
48	49	28.0	M	2	N	0	77.8	76.0	99	Y
49	50	30.0	F	1	N	0	74.6	NaN	71	Y
50	51	NaN	M	2	N	0	71.0	70.0	101	Y
51	52	27.0	M	2	N	0	NaN	71.5	103	Y
52	26	24.0	F	1	Y	1	66.0	NaN	68	N

```
## Eliminate inserted row
#df.drop([28], inplace = True )
data.drop([28], inplace = True )

#df.tail()
data.tail()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
48	49	28.0	M	2	N	0	77.8	76.0	99	Y
49	50	30.0	F	1	N	0	74.6	NaN	71	Y
50	51	NaN	M	2	N	0	71.0	70.0	101	Y

```
data
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
0	1	56.0	F	1	Y	1	62.00	61.0	79	Y
1	2	26.0	F	1	Y	1	62.00	60.0	70	Y
2	3	33.0	F	1	Y	1	66.00	64.0	85	Y
3	4	39.0	F	1	N	0	64.00	63.0	87	Y
4	5	27.0	M	2	N	0	73.00	75.0	72	N
5	6	24.0	M	2	N	0	75.00	71.0	81	N
6	7	28.0	M	2	N	0	75.00	76.0	107	Y
7	8	22.0	F	1	N	0	65.00	62.0	98	Y
8	9	29.0	M	2	Y	1	74.00	73.0	106	N
9	10	33.0	F	1	Y	1	63.00	60.0	65	Y
10	11	30.0	M	2	Y	1	69.50	66.0	96	Y
11	12	28.0	F	1	Y	1	62.75	58.0	79	Y
12	13	25.0	F	1	Y	1	65.00	64.5	92	Y
13	14	23.0	F	1	N	0	61.50	57.5	66	Y
14	15	31.0	M	2	Y	1	73.00	74.0	72	Y
15	16	26.0	M	2	Y	1	71.00	72.0	115	Y
16	17	26.0	F	1	N	0	61.50	59.5	90	N
17	18	27.0	M	2	N	0	66.00	66.0	74	Y
18	19	23.0	M	2	Y	1	70.00	69.0	64	Y
19	20	24.0	F	1	Y	1	68.00	66.0	85	Y
20	21	23.0	M	2	Y	1	69.00	67.0	66	N
21	22	29.0	M	2	N	0	71.00	70.0	101	Y
22	23	25.0	M	2	N	0	70.00	68.0	82	Y
23	24	26.0	M	2	N	0	69.00	71.0	63	Y

Cleaning your data: drop out unused columns and/or drop out rows with any missing values

```
# Drop unused columns lo que quiero ELIMINAR
#vars = ["ID", "GenderGroup", "GlassesGroup", "CompleteGroup"]
#df.drop(vars, axis=1, inplace = True)
```

```
# Con lo que me quiero QUEDAR:
```

```
#vars = ["Age", "Gender", "Glasses", "Height", "Wingspan", "CWDistance", "Complete", "Score"]
#df = df[vars]

# Drop rows with any missing values ## BORRAR Valor NaN no los considere
#df = df.dropna()

# Drop unused columns and drop rows with any missing values
#vars = ["Age", "Gender", "Glasses", "Height", "Wingspan", "CWDistance", "Complete", "Score"]
vars = ["Age", "Gender", "Glasses", "Height", "Wingspan", "CWDistance", "Complete", "Score"]
#df = df[vars].dropna()
data2 = data[vars].dropna()

#df
data2
```

	Age	Gender	Glasses	Height	Wingspan	CWDistance	Complete	Score
0	56.0	F	Y	62.00	61.0	79	Y	7
1	26.0	F	Y	62.00	60.0	70	Y	8
2	33.0	F	Y	66.00	64.0	85	Y	7
3	39.0	F	N	64.00	63.0	87	Y	10
4	27.0	M	N	73.00	75.0	72	N	4
5	24.0	M	N	75.00	71.0	81	N	3
6	28.0	M	N	75.00	76.0	107	Y	10
7	22.0	F	N	65.00	62.0	98	Y	9
8	29.0	M	Y	74.00	73.0	106	N	5
9	33.0	F	Y	63.00	60.0	65	Y	8
10	30.0	M	Y	69.50	66.0	96	Y	6
11	28.0	F	Y	62.75	58.0	79	Y	10
12	25.0	F	Y	65.00	64.5	92	Y	6
13	23.0	F	N	61.50	57.5	66	Y	4
14	31.0	M	Y	73.00	74.0	72	Y	9
15	26.0	M	Y	71.00	72.0	115	Y	6
16	26.0	F	N	61.50	59.5	90	N	10
17	27.0	M	N	66.00	66.0	74	Y	5
18	23.0	M	Y	70.00	69.0	64	Y	3
19	24.0	F	Y	68.00	66.0	85	Y	8
20	23.0	M	Y	69.00	67.0	66	N	2
21	29.0	M	N	71.00	70.0	101	Y	8
22	25.0	M	N	70.00	68.0	82	Y	4
23	26.0	M	N	69.00	71.0	63	Y	5
24	23.0	F	Y	65.00	63.0	67	N	3

```
data
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete
0	1	56.0	F	1	Y	1	62.00	61.0	79	Y
1	2	26.0	F	1	Y	1	62.00	60.0	70	Y
2	3	33.0	F	1	Y	1	66.00	64.0	85	Y
3	4	39.0	F	1	N	0	64.00	63.0	87	Y
4	5	27.0	M	2	N	0	73.00	75.0	72	N
5	6	24.0	M	2	N	0	75.00	71.0	81	N
6	7	28.0	M	2	N	0	75.00	76.0	107	Y
7	8	22.0	F	1	N	0	65.00	62.0	98	Y
8	9	29.0	M	2	Y	1	74.00	73.0	106	N
9	10	33.0	F	1	Y	1	63.00	60.0	65	Y
10	11	30.0	M	2	Y	1	69.50	66.0	96	Y
11	12	28.0	F	1	Y	1	62.75	58.0	79	Y
12	13	25.0	F	1	Y	1	65.00	64.5	92	Y
13	14	23.0	F	1	N	0	61.50	57.5	66	Y
14	15	31.0	M	2	Y	1	73.00	74.0	72	Y
15	16	26.0	M	2	Y	1	71.00	72.0	115	Y
16	17	26.0	F	1	N	0	61.50	59.5	90	N
17	18	27.0	M	2	N	0	66.00	66.0	74	Y
18	19	23.0	M	2	Y	1	70.00	69.0	64	Y
19	20	24.0	F	1	Y	1	68.00	66.0	85	Y
20	21	23.0	M	2	Y	1	69.00	67.0	66	N
21	22	29.0	M	2	N	0	71.00	70.0	101	Y
22	23	25.0	M	2	N	0	70.00	68.0	82	Y
23	24	26.0	M	2	N	0	69.00	71.0	63	Y
24	25	23.0	F	1	Y	1	65.00	63.0	67	N
25	26	28.0	M	2	N	0	75.00	76.0	111	Y
26	27	24.0	M	2	N	0	78.40	71.0	92	Y
27	28	25.0	M	2	Y	1	76.00	73.0	107	Y
29	30	38.0	F	1	Y	1	61.50	61.0	78	Y
30	31	27.0	F	1	Y	1	62.00	60.0	72	Y
31	32	33.0	F	1	Y	1	65.30	64.0	91	Y
32	33	38.0	F	1	N	0	64.00	63.0	86	Y
33	34	27.0	M	2	N	0	77.00	75.0	100	Y

Final remarks

- The understanding of your dataset is essential
  - Number of observations
  - Variables
  - Data types: numerical or categorial
  - What are my variables of interest
- There are several ways to do the same thing
- Cleaning your dataset (dropping out rows with any missing values) is a good practice
- The **Pandas** library provides fancy, high-performance, easy-to-use data structures and data analysis tools

45	46	39.0	F	1	N	0	64.00	63.0	87	Y

Activity: work with the iris dataset

Repeat this tutorial with the iris data set and respond to the following inquiries

1. Calculate the statistical summary for each quantitative variables. Explain the results
   - Identify the name of each column
   - Identify the type of each column
   - Minimum, maximum, mean, average, median, standar deviation
2. Are there missing data? If so, create a new dataset containing only the rows with the non-missing data
3. Create a new dataset containing only the petal width and length and the type of Flower
4. Create a new dataset containing only the setal width and length and the type of Flower
5. Create a new dataset containing the setal width and length and the type of Flower encoded as a categorical numerical column

```
url = "/content/drive/MyDrive/iris.csv"

dfI = pd.read_csv(url, header = None)
dfI
```

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica

```
dfI.shape
```

```
(150, 5)
```

```
f = dfI.shape[0]
c = dfI.shape[1]

print("Refisters: "+str(f))
print ("Vars: "+str(c))
```

```
Refisters: 150
Vars: 5
```

```
dfI.columns
```

```
Int64Index([0, 1, 2, 3, 4], dtype='int64')
```

```
dfI.dtypes
```

```
0    float64
1    float64
2    float64
3    float64
4     object
dtype: object
```

```
# Media
dfI.mean()
```

```
<ipython-input-127-94118a9efb19>:2: FutureWarning: Dropping of nuisance columns in DataFrame red
  dfI.mean()
0    5.843333
1    3.057333
2    3.758000
3    1.199333
dtype: float64
```

```
# Correlación
dfI.corr()
```

	0	1	2	3
0	1.000000	-0.117570	0.871754	0.817941
1	-0.117570	1.000000	-0.428440	-0.366126
2	0.871754	-0.428440	1.000000	0.962865
3	0.817941	-0.366126	0.962865	1.000000

```
# No NULL
dfI.count()
```

```
0    150
1    150
2    150
3    150
4    150
dtype: int64
```

```
# Max y Min
print("Máximo: \n"+ str(dfI.max()))
print("Mínimo: \n"+ str(dfI.min()))
```

```
Máximo:
0               7.9
1               4.4
2               6.9
3               2.5
4    Iris-virginica
dtype: object
Mínimo:
0            4.3
1            2.0
2            1.0
3            0.1
4    Iris-setosa
dtype: object
```

```
# Mediana
dfI.median()
```

```
<ipython-input-130-68e9154072cf>:2: FutureWarning: Dropping of nuisance columns in DataFrame red
  dfI.median()
0    5.80
1    3.00
2    4.35
3    1.30
dtype: float64
```

```
# Desviación estándar
dfI.std()
```

```
<ipython-input-131-35e1ff6146d6>:2: FutureWarning: Dropping of nuisance columns in DataFrame red
  dfI.std()
0    0.828066
1    0.435866
2    1.765298
3    0.762238
dtype: float64
```

```
dfI.to_csv('myDataFrame.csv')
```

```
dfI = dfI.rename(columns={4: "Especie"})
```

```
dfI.head()
```

	0	1	2	3	Especie
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
a = dfI.Especie
b = dfI["Especie"]
c = dfI.loc[:, "Especie"]
d = dfI.iloc[:, 4]

print(d)
```

```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
              ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Especie, Length: 150, dtype: object
```

```
vars = ["Sepalo_Largo", "Sepalo_Ancho", "Petalo_Largo", "Petalo_Ancho", "Especie"]

dfI = dfI.rename(columns={0: "Sepalo_Largo"})
dfI = dfI.rename(columns={1: "Sepalo_Ancho"})
dfI = dfI.rename(columns={2: "Petalo_Largo"})
```

```
dfI = dfI.rename(columns={2: "Petalo_Largo"})
dfI = dfI.rename(columns={3: "Petalo_Ancho"})
dfI = dfI.rename(columns={4: "Especie"})

dfI.head()
```

	Sepalo_Largo	Sepalo_Ancho	Petalo_Largo	Petalo_Ancho	Especie
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
# 1
dfSum = dfI.iloc[:, 0:4].copy()
dfSum.describe()
```

	Sepalo_Largo	Sepalo_Ancho	Petalo_Largo	Petalo_Ancho
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
dfSum.dtypes
```

```
Sepalo_Largo    float64
Sepalo_Ancho    float64
Petalo_Largo    float64
Petalo_Ancho    float64
dtype: object
```

```
# 2
df2 = dfI.dropna()
df2
```

	Sepalo_Largo	Sepalo_Ancho	Petalo_Largo	Petalo_Ancho	Especie
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica

```
# 3
df3 = dfI[["Especie", "Petalo_Ancho", "Petalo_Largo"]].copy()
df3
```

	Especie	Petalo_Ancho	Petalo_Largo
0	Iris-setosa	0.2	1.4
1	Iris-setosa	0.2	1.4
2	Iris-setosa	0.2	1.3
3	Iris-setosa	0.2	1.5
4	Iris-setosa	0.2	1.4
...	...	...	...
145	Iris-virginica	2.3	5.2
146	Iris-virginica	1.9	5.0
147	Iris-virginica	2.0	5.2
148	Iris-virginica	2.3	5.4
149	Iris-virginica	1.8	5.1

150 rows × 3 columns

```
# 4
df4 = dfI[["Especie", "Sepalo_Ancho", "Sepalo_Largo"]].copy()
df4
```

	Especie	Sepalo_Ancho	Sepalo_Largo
0	Iris-setosa	3.5	5.1
1	Iris-setosa	3.0	4.9
2	Iris-setosa	3.2	4.7
3	Iris-setosa	3.1	4.6
4	Iris-setosa	3.6	5.0
...	...	...	...
145	Iris-virginica	3.0	6.7
146	Iris-virginica	2.5	6.3

```
#5
#1: Iris setosa
#2: Iris virginica
#3: Iris versicolor
df5 = dfI.copy()
df5["Especie"] = df5["Especie"].replace({"Iris-setosa":1,"Iris-virginica":2,"Iris-versicolor":3
df5
```

	Sepalo_Largo	Sepalo_Ancho	Petalo_Largo	Petalo_Ancho	Especie
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

Haz doble clic (o ingresa) para editar

✓ 0 s se ejecutó 17:25