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                # For data handling

#pd.set_option('display.max_columns', 100) # Show all columns when looking at dataframe
```

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/TC1002S/NotebooksProfessor"

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

```
 Mounted at /content/drive
```

```
# url string that hosts our .csv file
url = Ruta + "/datasets/cartwheel/cartwheel.csv"

# Read the .csv file and store it as a pandas Data Frame
df = pd.read_csv(url)
```

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

```
type(df)
```

```
pandas.core.frame.DataFrame
def __init__(data=None, index: Axes | None=None, columns: Axes | None=None,
dtype: Dtype | None=None, copy: bool | None=None) -> None
```

```
Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
pandas data structure.
```

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.

```
df.shape
```

```
(52, 12)
```

```
# Print the number of rows
Nrows = df.shape[0]
Nrows
```

```
52
```

```
# Print the number of columns
Ncols = df.shape[1]
Ncols
```

```
12
```

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

```
df
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistanc
0	1	56.0	F	1	Y	1	62.00	61.0	7
1	2	26.0	F	1	Y	1	62.00	60.0	7
2	3	33.0	F	1	Y	1	66.00	64.0	8
3	4	39.0	F	1	N	0	64.00	63.0	8
4	5	27.0	M	2	N	0	73.00	75.0	7
5	6	24.0	M	2	N	0	75.00	71.0	8
6	7	28.0	M	2	N	0	75.00	76.0	10
7	8	22.0	F	1	N	0	65.00	62.0	9
8	9	29.0	M	2	Y	1	74.00	73.0	10
9	10	33.0	F	1	Y	1	63.00	60.0	6

10	11	30.0	M	2	Y	1	69.50	66.0	9
11	12	28.0	F	1	Y	1	62.75	58.0	7
12	13	25.0	F	1	Y	1	65.00	64.5	9
13	14	23.0	F	1	N	0	61.50	57.5	6
14	15	31.0	M	2	Y	1	73.00	74.0	7
15	16	26.0	M	2	Y	1	71.00	72.0	11
16	17	26.0	F	1	N	0	61.50	59.5	9
17	18	27.0	M	2	N	0	66.00	66.0	7
18	19	23.0	M	2	Y	1	70.00	69.0	6
19	20	24.0	F	1	Y	1	68.00	66.0	8
20	21	23.0	M	2	Y	1	69.00	67.0	6
21	22	29.0	M	2	N	0	71.00	70.0	10
22	23	25.0	M	2	N	0	70.00	68.0	8
23	24	26.0	M	2	N	0	69.00	71.0	6
24	25	23.0	F	1	Y	1	65.00	63.0	6
25	26	28.0	M	2	N	0	75.00	76.0	11
26	27	24.0	M	2	N	0	78.40	71.0	9
27	28	25.0	M	2	Y	1	76.00	73.0	10
28	29	32.0	F	1	Y	1	63.00	60.0	7
29	30	38.0	F	1	Y	1	61.50	61.0	7
30	31	27.0	F	1	Y	1	62.00	60.0	7

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

34	35	24.0	F	1	N	0	67.80	62.0	9

```
#df.head()
df.head(10)
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance
0	1	56.0	F	1	Y	1	62.0	61.0	79
1	2	26.0	F	1	Y	1	62.0	60.0	70
2	3	33.0	F	1	Y	1	66.0	64.0	85

3	4	39.0	F	1	N	0	64.0	63.0	87
4	5	27.0	M	2	N	0	73.0	75.0	72
5	6	24.0	M	2	N	0	75.0	71.0	81
6	7	28.0	M	2	N	0	75.0	76.0	107
7	8	22.0	F	1	N	0	65.0	62.0	98
8	9	29.0	M	2	Y	1	74.0	73.0	106
9	10	33.0	F	1	Y	1	63.0	60.0	65
48	49	28.0	M	2	N	0	77.80	76.0	9

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

50	51	NaN	M	2	N	0	71.00	70.0	10

```
#df.tail()
df.tail(3)
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistanc
49	50	30.0	F	1	N	0	74.6	NaN	7
50	51	NaN	M	2	N	0	71.0	70.0	10
51	52	27.0	M	2	N	0	NaN	71.5	10

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:

```
df.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.

```
df.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
df.describe()
```

	ID	Age	GenderGroup	GlassesGroup	Height	Wingspan	CWDistan
count	52.000000	51.000000	52.000000	52.000000	51.000000	51.000000	52.0000
mean	26.500000	28.411765	1.500000	0.500000	68.971569	67.313725	85.5769
std	15.154757	5.755611	0.504878	0.504878	5.303812	5.624021	14.3531
min	1.000000	22.000000	1.000000	0.000000	61.500000	57.500000	63.0000
25%	13.750000	25.000000	1.000000	0.000000	64.500000	63.000000	72.0000
50%	26.500000	27.000000	1.500000	0.500000	69.000000	66.000000	85.0000
75%	39.250000	30.000000	2.000000	1.000000	73.000000	72.000000	96.5000
max	52.000000	56.000000	2.000000	1.000000	79.500000	76.000000	115.0000

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

```
count    51.000000
mean     67.313725
std       5.624021
min      57.500000
25%      63.000000
50%      66.000000
```

```
75%      72.000000
max      76.000000
Name: Wingspan, 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

```
df.mean()
```

```
<ipython-input-38-c61f0c8f89b5>:1: FutureWarning: The default value of numeric_only
  df.mean()
ID               26.500000
Age              28.411765
GenderGroup       1.500000
GlassesGroup      0.500000
Height           68.971569
Wingspan         67.313725
CWDistance       85.576923
CompleteGroup     0.843137
Score             7.173077
dtype: float64
```

How to write a data frame to a File

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

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

Rename columns

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

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

df.head()
```

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

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

df.head()
```

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

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]

print(d)

df[["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

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.

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

```
0      79
1      70
2      85
3      87
4      72
5      81
6     107
7      98
8     106
9      65
10     96
11     79
12     92
13     66
14     72
15    115
16     90
17     74
18     64
19     85
20     66
21    101
22     82
```

```
23     63
24     67
25    111
26     92
27    107
28     75
29     78
30     72
31     91
32     86
33    100
34     98
35     74
36     92
37     90
38     72
39     96
40     66
41    115
42     81
43     92
44     85
45     87
46     72
47     82
48     99
49     71
50    101
51    103
Name: CWDistance, dtype: int64
```

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

```
0     79
1     70
2     85
3     87
4     72
5     81
6    107
7     98
8    106
9     65
Name: CWDistance, dtype: int64
```

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

	Gender	GenderGroup
0	F	1
1	F	1

2	F	1
3	F	1
4	M	2
5	M	2
6	M	2
7	F	1
8	M	2
9	F	1
10	M	2
11	F	1
12	F	1
13	F	1
14	M	2
15	M	2
16	F	1
17	M	2
18	M	2
19	F	1
20	M	2
21	M	2
22	M	2
23	M	2
24	F	1
25	M	2
26	M	2
27	M	2
28	F	1
29	F	1
30	F	1

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

```
df_gender = df[keep]
```

```
      33      M           2
```

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

	CWDistance	Height	Wingspan
4	72	73.0	75.0
5	81	75.0	71.0
6	107	75.0	76.0
7	98	65.0	62.0
8	106	74.0	73.0
9	65	63.0	60.0

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

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistanc
10	11	30.0	M	2	Y	1	69.50	66.0	9(
11	12	28.0	F	1	Y	1	62.75	58.0	7!
12	13	25.0	F	1	Y	1	65.00	64.5	9:
13	14	23.0	F	1	N	0	61.50	57.5	6(
14	15	31.0	M	2	Y	1	73.00	74.0	7:
15	16	26.0	M	2	Y	1	71.00	72.0	11!

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

```
# .
#df.iloc[:, :4]

# .
#df.iloc[:4, :]

# .
#df.iloc[:, 3:7]

# .
df.iloc[4:8, 2:4]

# This is incorrect:
```

```
#df.iloc[1:5, ["Gender", "GenderGroup"]]
```

	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

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

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

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

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

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]
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistanc
4	5	27.0	M	2	N	0	73.0	75.0	7
5	6	24.0	M	2	N	0	75.0	71.0	8
6	7	28.0	M	2	N	0	75.0	76.0	10
8	9	29.0	M	2	Y	1	74.0	73.0	10
14	15	31.0	M	2	Y	1	73.0	74.0	7
15	16	26.0	M	2	Y	1	71.0	72.0	11

18	19	23.0	M	2	Y	1	70.0	69.0	6
21	22	29.0	M	2	N	0	71.0	70.0	10
22	23	25.0	M	2	N	0	70.0	68.0	8
25	26	28.0	M	2	N	0	75.0	76.0	11
26	27	24.0	M	2	N	0	78.4	71.0	9
27	28	25.0	M	2	Y	1	76.0	73.0	10
33	34	27.0	M	2	N	0	77.0	75.0	10
38	39	31.0	M	2	Y	1	73.0	74.0	7
40	41	23.0	F	1	N	0	70.4	71.0	6
41	42	26.0	M	2	Y	1	73.5	72.0	11
42	43	28.0	F	1	Y	1	72.5	72.0	8
43	44	26.0	F	1	Y	1	72.0	72.0	9
46	47	27.0	M	2	N	0	78.0	75.0	7
47	48	24.0	M	2	N	0	79.5	75.0	8
48	49	28.0	M	2	N	0	77.8	76.0	9
49	50	30.0	F	1	N	0	74.6	NaN	7
50	51	NaN	M	2	N	0	71.0	70.0	10

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)
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistanc
47	48	24.0	M	2	N	0	79.50	75.0	8
26	27	24.0	M	2	N	0	78.40	71.0	9
46	47	27.0	M	2	N	0	78.00	75.0	7

48	49	28.0	M	2	N	0	77.80	76.0	9
33	34	27.0	M	2	N	0	77.00	75.0	10
27	28	25.0	M	2	Y	1	76.00	73.0	10
5	6	24.0	M	2	N	0	75.00	71.0	8
6	7	28.0	M	2	N	0	75.00	76.0	10
25	26	28.0	M	2	N	0	75.00	76.0	11
49	50	30.0	F	1	N	0	74.60	NaN	7
8	9	29.0	M	2	Y	1	74.00	73.0	10
41	42	26.0	M	2	Y	1	73.50	72.0	11
4	5	27.0	M	2	N	0	73.00	75.0	7
38	39	31.0	M	2	Y	1	73.00	74.0	7
14	15	31.0	M	2	Y	1	73.00	74.0	7
42	43	28.0	F	1	Y	1	72.50	72.0	8
43	44	26.0	F	1	Y	1	72.00	72.0	9
21	22	29.0	M	2	N	0	71.00	70.0	10
50	51	NaN	M	2	N	0	71.00	70.0	10
15	16	26.0	M	2	Y	1	71.00	72.0	11
40	41	23.0	F	1	N	0	70.40	71.0	6
18	19	23.0	M	2	Y	1	70.00	69.0	6
22	23	25.0	M	2	N	0	70.00	68.0	8
10	11	30.0	M	2	Y	1	69.50	66.0	9
39	40	30.0	M	2	Y	1	69.50	66.0	9
23	24	26.0	M	2	N	0	69.00	71.0	6
20	21	23.0	M	2	Y	1	69.00	67.0	6
19	20	24.0	F	1	Y	1	68.00	66.0	8
35	36	27.0	M	2	N	0	68.00	66.0	7
34	35	24.0	F	1	N	0	67.80	62.0	9
2	3	33.0	F	1	Y	1	66.00	64.0	8

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'])
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x786de3762a10>
```

Size of each group

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

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

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

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.

```
df.isnull()
```

	Sepal_length	Sepal_width	Petal_length	Petal_width	Class
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
145	False	False	False	False	False
146	False	False	False	False	False
147	False	False	False	False	False
148	False	False	False	False	False
149	False	False	False	False	False

150 rows × 5 columns

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

```
df.notnull()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDista
0	True	True	True	True	True	True	True	True	
1	True	True	True	True	True	True	True	True	
2	True	True	True	True	True	True	True	True	
3	True	True	True	True	True	True	True	True	
4	True	True	True	True	True	True	True	True	
5	True	True	True	True	True	True	True	True	
6	True	True	True	True	True	True	True	True	
7	True	True	True	True	True	True	True	True	
8	True	True	True	True	True	True	True	True	
9	True	True	True	True	True	True	True	True	
10	True	True	True	True	True	True	True	True	
11	True	True	True	True	True	True	True	True	

11	True	True	True	True	True	True	True	True	
12	True	True	True	True	True	True	True	True	
13	True	True	True	True	True	True	True	True	
14	True	True	True	True	True	True	True	True	
15	True	True	True	True	True	True	True	True	
16	True	True	True	True	True	True	True	True	
17	True	True	True	True	True	True	True	True	
18	True	True	True	True	True	True	True	True	
19	True	True	True	True	True	True	True	True	
20	True	True	True	True	True	True	True	True	
21	True	True	True	True	True	True	True	True	
22	True	True	True	True	True	True	True	True	
23	True	True	True	True	True	True	True	True	
24	True	True	True	True	True	True	True	True	
25	True	True	True	True	True	True	True	True	
26	True	True	True	True	True	True	True	True	
27	True	True	True	True	True	True	True	True	
28	True	True	True	True	True	True	True	True	
29	True	True	True	True	True	True	True	True	
30	True	True	True	True	True	True	True	True	

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

40	True	True	True	True	True	True	True	True	

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

variable in the dataset

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

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

```
51
1
```

47 True True True True True True True True

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

```
count     8.000000
mean     30.645922
std      16.044470
min       5.755611
25%      24.250000
50%      27.705882
75%      35.250000
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()
```

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

```
# Add a new column with new data

# Create a column data
NewColumnData = df.Age/df.Age

# Insert that column in the data frame
df.insert(12, "ColumnInserted", NewColumnData, True)
```

```
df.head()
```

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

```
# Eliminate inserted column
df.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()
```

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

```
# Add new column derived from existing columns

# The new column is a function of another column
df["AgeInMonths"] = df["Age"] * 12

df.head()
```

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

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

df.head()
```

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

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

df["GenderGroupNew"] = df.GenderGroup.replace({1: "Female", 2: "Male"})

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

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

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

df.head()
```

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

```
# Add a new column with strata based on these cut points

# Create a column data
NewColumnData = df.Age/df.Age

# Insert that column in the data frame
df.insert(1, "ColumnStrata", NewColumnData, True)

df["ColumnStrata"] = pd.cut(df.Height, [60., 63., 66., 69., 72., 75., 78.])

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

	ID	ColumnStrata	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspa
0	1	(60.0, 63.0]	56.0	F	1	Y	1	62.0	61
1	2	(60.0, 63.0]	26.0	F	1	Y	1	62.0	60
2	3	(63.0, 66.0]	33.0	F	1	Y	1	66.0	64
3	4	(63.0, 66.0]	39.0	F	1	N	0	64.0	63
4	5	(72.0, 75.0]	27.0	M	2	N	0	73.0	75

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

df.head()
```

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

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

Add and eliminate rows

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

```
df.tail()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistanc
47	48	24.0	M	2	N	0	79.5	75.0	8
48	49	28.0	M	2	N	0	77.8	76.0	9
49	50	30.0	F	1	N	0	74.6	NaN	7
50	51	NaN	M	2	N	0	71.0	70.0	10
51	52	27.0	M	2	N	0	NaN	71.5	10

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

df.tail()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistanc
48	49	28.0	M	2	N	0	77.8	76.0	9
49	50	30.0	F	1	N	0	74.6	NaN	7
50	51	NaN	M	2	N	0	71.0	70.0	10
51	52	27.0	M	2	N	0	NaN	71.5	10
52	26	24.0	F	1	Y	1	66.0	NaN	6

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

df.tail()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistanc
48	49	28.0	M	2	N	0	77.8	76.0	9
49	50	30.0	F	1	N	0	74.6	NaN	7
50	51	NaN	M	2	N	0	71.0	70.0	10
51	52	27.0	M	2	N	0	NaN	71.5	10
52	26	24.0	F	1	Y	1	66.0	NaN	6

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

drop out rows with any missing values

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

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

# Drop rows with any missing values
#df = df.dropna()

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

df
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistanc
0	1	56.0	F	1	Y	1	62.00	61.0	7
1	2	26.0	F	1	Y	1	62.00	60.0	7
2	3	33.0	F	1	Y	1	66.00	64.0	8
3	4	39.0	F	1	N	0	64.00	63.0	8
4	5	27.0	M	2	N	0	73.00	75.0	7
5	6	24.0	M	2	N	0	75.00	71.0	8
6	7	28.0	M	2	N	0	75.00	76.0	10
7	8	22.0	F	1	N	0	65.00	62.0	9
8	9	29.0	M	2	Y	1	74.00	73.0	10
9	10	33.0	F	1	Y	1	63.00	60.0	6
10	11	30.0	M	2	Y	1	69.50	66.0	9
11	12	28.0	F	1	Y	1	62.75	58.0	7
12	13	25.0	F	1	Y	1	65.00	64.5	9
13	14	23.0	F	1	N	0	61.50	57.5	6
14	15	31.0	M	2	Y	1	73.00	74.0	7
15	16	26.0	M	2	Y	1	71.00	72.0	11
16	17	26.0	F	1	N	0	61.50	59.5	9
17	18	27.0	M	2	N	0	66.00	66.0	7

18	19	23.0	M	2	Y	1	70.00	69.0	6
19	20	24.0	F	1	Y	1	68.00	66.0	8
20	21	23.0	M	2	Y	1	69.00	67.0	6
21	22	29.0	M	2	N	0	71.00	70.0	10
22	23	25.0	M	2	N	0	70.00	68.0	8
23	24	26.0	M	2	N	0	69.00	71.0	6
24	25	23.0	F	1	Y	1	65.00	63.0	6
25	26	28.0	M	2	N	0	75.00	76.0	11
26	27	24.0	M	2	N	0	78.40	71.0	9
27	28	25.0	M	2	Y	1	76.00	73.0	10
29	30	38.0	F	1	Y	1	61.50	61.0	7
30	31	27.0	F	1	Y	1	62.00	60.0	7
31	32	33.0	F	1	Y	1	65.30	64.0	9

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

43	44	26.0	F	1	Y	1	72.00	72.0	9

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

```
# Dataset url
url = Ruta + "/datasets/iris/iris.csv"

# Load the dataset
newHeader=["Sepal_length", "Sepal_width", "Petal_length", "Petal_width", "Class"]
df = pd.read_csv(url, header=None, names=newHeader )

#dataset = dataset.rename(columns={"Sepal_length": 0, "Sepal_width": 1, "Petal_length": 2

# Print the dataset
df.head()
```

	Sepal_length	Sepal_width	Petal_length	Petal_width	Class
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. Calculate the statistical summary for each quantitative variables. Explain the results

Identify the name of each column

```
list(df)
```

```
['Sepal_length', 'Sepal_width', 'Petal_length', 'Petal_width', 'Class']
```

```
['Sepal_length', 'Sepal_Width', 'Petal_length', 'Petal_Width', 'Class']
```

Identify the type of each column

```
df.dtypes
```

```
Sepal_length    float64
Sepal_width     float64
Petal_length    float64
Petal_width     float64
Class            object
dtype: object
```

Minimum, maximum, mean, average, median, standar deviation

```
df.describe()
```

	Sepal_length	Sepal_width	Petal_length	Petal_width
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

```
df.median()
```

```
<ipython-input-11-6d467abf240d>:1: FutureWarning: The default value of numeric_only
  df.median()
Sepal_length    5.80
Sepal_width     3.00
Petal_length    4.35
Petal_width     1.30
dtype: float64
```

The results show that there are 150 observations (rows), and that the biggest mean is sepal length.

All values are measured in cms.

Sepal_length: The maximum value is 7.9 and the minimum is 4.3, while the mean is 5.84

Sepal_width: The maximum value is 4.4 and the minimum is 2.3, while the mean is 3.05

Petal_length: The maximum value is 6.9 and the minimum is 1, while the mean is 3.75

Sepal_width: The maximum value is 2.5 and the minimum is 0.1, while the mean is 1.19

This results show that the value that tends to be the longest is the sepal length on the flowers observed

2. Are there missing data? If so, create a new dataset containing only the rows with the non-missing data

```
df.isnull().sum()
```

```
Sepal_length    0
Sepal_width     0
Petal_length    0
Petal_width     0
Class           0
dtype: int64
```

There is no missing data, therefore no need to make a new table

3. Create a new dataset containing only the petal width and length and the type of Flower

```
d1=df.loc[:,["Petal_width", "Petal_length", "Class"]]
d1
```

	Petal_width	Petal_length	Class
0	0.2	1.4	Iris-setosa
1	0.2	1.4	Iris-setosa
2	0.2	1.3	Iris-setosa
3	0.2	1.5	Iris-setosa
4	0.2	1.4	Iris-setosa
...	...	...	...
145	2.3	5.2	Iris-virginica

146	1.9	5.0	Iris-virginica
147	2.0	5.2	Iris-virginica
148	2.3	5.4	Iris-virginica
149	1.8	5.1	Iris-virginica

150 rows × 3 columns

4. Create a new dataset containing only the setal width and length and the type of Flower

```
d2=df.loc[:,["Sepal_width", "Sepal_length", "Class"]]
d2
```

	Sepal_width	Sepal_length	Class
0	3.5	5.1	Iris-setosa
1	3.0	4.9	Iris-setosa
2	3.2	4.7	Iris-setosa
3	3.1	4.6	Iris-setosa
4	3.6	5.0	Iris-setosa
...	...	...	...
145	3.0	6.7	Iris-virginica
146	2.5	6.3	Iris-virginica
147	3.0	6.5	Iris-virginica
148	3.4	6.2	Iris-virginica
149	3.0	5.9	Iris-virginica

150 rows × 3 columns

5. Create a new dataset containing the setal width and length and the type of Flower encoded as a categorical numerical column

```
#df["ClassNew"] = df.Class.replace({1: "Iris Setosa", 2: "Iris Versicolour", 3:"Iris Vir
#df["ClassNew"] = df.Class.replace({["Iris Setosa",1],["Iris Versicolour",2],["Iris Virg
```

```
d3=df.loc[:,["Sepal_width", "Sepal_length", "ClassNew"]]
d3.replace(to_replace="Iris Setosa",
           value=1)
d3.replace(to_replace="Iris Versicolour",
           value=2)
d3.replace(to_replace="Iris Virginica",
           value=3)
d3
```

	Sepal_width	Sepal_length	ClassNew
0	3.5	5.1	Iris-setosa
1	3.0	4.9	Iris-setosa
2	3.2	4.7	Iris-setosa
3	3.1	4.6	Iris-setosa
4	3.6	5.0	Iris-setosa
...	...	...	...
145	3.0	6.7	Iris-virginica
146	2.5	6.3	Iris-virginica
147	3.0	6.5	Iris-virginica
148	3.4	6.2	Iris-virginica
149	3.0	5.9	Iris-virginica

150 rows × 3 columns