Daniel Estrada Ocaña A01369854

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](https://pandas.pydata.org/) 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/MyDrive/Sistemas/4to_semestre/semanaTec/TC1002S"

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

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

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
```

```
(150, 5)
```

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

```
150
```

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

```
5
```

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

```
df
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

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

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

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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

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

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

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

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(['Ms_1', 'Ms_2', 'Ms_3', 'Ms_4', 'Type'], 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
```

```
Ms_1    float64
Ms_2    float64
Ms_3    float64
Ms_4    float64
Type     object
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()

```

	Ms_1	Ms_2	Ms_3	Ms_4
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

```
# Drop observations with NaN values
df.Ms_2.dropna().describe()
#df.Wingspan.dropna().describe()
```

```
count    150.000000
mean       3.057333
std        0.435866
min        2.000000
25%        2.800000
50%        3.000000
75%        3.300000
max        4.400000
Name: Ms_2, dtype: float64
```

```
df.Ms_3.dropna().describe()

```

```
count    150.000000
mean       3.758000
std        1.765298
min        1.000000
25%        1.600000
50%        4.350000
75%        5.100000
max        6.900000
Name: Ms_3, 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-45-c61f0c8f89b5>:1: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future version
  df.mean()
```

```
Ms_1    5.843333
Ms_2    3.057333
Ms_3    3.758000
Ms_4    1.199333
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()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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

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

df.head()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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

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.Ms_1
b = df["Ms_1"]
c = df.loc[:, "Ms_1"]
d = df.iloc[:, 1]

print(a,b,c,d)



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

```
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: Ms_1, Length: 150, dtype: float64 0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: Ms_1, Length: 150, dtype: float64 0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: Ms_1, Length: 150, dtype: float64 0      3.5
1      3.0
2      3.2
3      3.1
4      3.6
      ...
145    3.0
146    2.5
147    3.0
148    3.4
149    3.0
Name: Ms_2, Length: 150, dtype: float64
```

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 Ms_1
df.loc[:,"Ms_1"]

# Return a subset of observations of Ms_1
df.loc[:9, "Ms_1"]

# Select all rows for multiple columns, ["Ms_2", "Type"]]
df.loc[:,["Ms_2", "Type"]]

# Select multiple columns, ['Ms_3', 'Ms_4']me
keep = ['Ms_3', 'Ms_4']
df_gender = df[keep]

# Select few rows for multiple columns, ["Ms_1", "Ms_2", "Ms_4"]
df.loc[4:9, ["Ms_1", "Ms_2", "Ms_4"]]

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

	Ms_2	Type
0	3.5	Iris-setosa
1	3.0	Iris-setosa
2	3.2	Iris-setosa
3	3.1	Iris-setosa
4	3.6	Iris-setosa
...	...	...
145	3.0	Iris-virginica
146	2.5	Iris-virginica
147	3.0	Iris-virginica
148	3.4	Iris-virginica
149	3.0	Iris-virginica

150 rows × 2 columns

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"]]
```

Get unique existing values

List unique values in the one of the columns

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

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```
# Lets explore df["Ms_4] as well
df.Ms_4.unique()
```

```
array([0.2, 0.4, 0.3, 0.1, 0.5, 0.6, 1.4, 1.5, 1.3, 1.6, 1. , 1.1, 1.8,
       1.2, 1.7, 2.5, 1.9, 2.1, 2.2, 2. , 2.4, 2.3])
```

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.

```
1 df[df["Ms_1"] >= 3.5]
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

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])`

```
1 df.sort_values("Ms_3")
2 #df.sort_values("Height",ascending=False)
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
22	4.6	3.6	1.0	0.2	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
35	5.0	3.2	1.2	0.2	Iris-setosa
36	5.5	3.5	1.3	0.2	Iris-setosa
...	...	...	...	...	...
131	7.9	3.8	6.4	2.0	Iris-virginica
105	7.6	3.0	6.6	2.1	Iris-virginica
117	7.7	3.8	6.7	2.2	Iris-virginica
122	7.7	2.8	6.7	2.0	Iris-virginica
118	7.7	2.6	6.9	2.3	Iris-virginica

150 rows × 5 columns

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(['Ms_2'])

```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7d4d679f1060>
```

Size of each group

```
df.groupby(['Type']).size()

df.groupby(['Type','Ms_3']).size()
```

```
Type             Ms_3
Iris-setosa      1.0      1
                 1.1      1
                 1.2      2
                 1.3      7
                 1.4     13
                 1.5     13
                 1.6      7
                 1.7      4
                 1.9      2
Iris-versicolor  3.0      1
                 3.3      2
                 3.5      2
                 3.6      1
                 3.7      1
                 3.8      1
                 3.9      3
                 4.0      5
                 4.1      3
                 4.2      4
                 4.3      2
                 4.4      4
                 4.5      7
                 4.6      3
                 4.7      5
                 4.8      2
                 4.9      2
                 5.0      1
                 5.1      1
Iris-virginica   4.5      1
                 4.8      2
                 4.9      3
                 5.0      3
                 5.1      7
                 5.2      2
                 5.3      2
                 5.4      2
                 5.5      3
                 5.6      6
                 5.7      3
                 5.8      3
                 5.9      2
                 6.0      2
                 6.1      3
                 6.3      1
                 6.4      1
                 6.6      1
                 6.7      2
                 6.9      1
dtype: int64
```

This output indicates that we have 49 types of combinations.

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()

```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
0	True	True	True	True	True
1	True	True	True	True	True
2	True	True	True	True	True
3	True	True	True	True	True
4	True	True	True	True	True
...	...	...	...	...	...
145	True	True	True	True	True
146	True	True	True	True	True
147	True	True	True	True	True
148	True	True	True	True	True
149	True	True	True	True	True

150 rows × 5 columns

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

```
Ms_1    150
Ms_2    150
Ms_3    150
Ms_4    150
Type    150
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.Ms_2.notnull().sum() )

print( pd.isnull(df.Ms_2).sum() )
```

```
150
0
```

```
# Extract all non-missing values of one of the columns into a new variable
x = df.Ms_1.dropna().describe()
x.describe()
```

```
count      8.000000
mean      23.271425
std       51.247113
min        0.828066
25%        4.900000
50%        5.821667
75%        6.775000
max      150.000000
Name: Ms_1, dtype: float64
```

Add and eliminate columns

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

```
df.head()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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

```
# Add a new column with new data

# Create a column data
NewColumnData = df.Ms_1/df.Ms_2

# Insert that column in the data frame
df.insert(5, "ColumnInserted", NewColumnData, True)

df.head()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type	ColumnInserted
0	5.1	3.5	1.4	0.2	Iris-setosa	1.457143
1	4.9	3.0	1.4	0.2	Iris-setosa	1.633333
2	4.7	3.2	1.3	0.2	Iris-setosa	1.468750
3	4.6	3.1	1.5	0.2	Iris-setosa	1.483871
4	5.0	3.6	1.4	0.2	Iris-setosa	1.388889

```
# 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()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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

```
# Add new column derived from existing columns

# The new column is a function of another column
df["Ms_5"] = df["Ms_4"] * 12

df.head()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type	ColumnInserted	Ms_5
0	5.1	3.5	1.4	0.2	Iris-setosa	1.457143	2.4
1	4.9	3.0	1.4	0.2	Iris-setosa	1.633333	2.4
2	4.7	3.2	1.3	0.2	Iris-setosa	1.468750	2.4
3	4.6	3.1	1.5	0.2	Iris-setosa	1.483871	2.4
4	5.0	3.6	1.4	0.2	Iris-setosa	1.388889	2.4

```
# Eliminate inserted column
df.drop("Ms_5", axis=1, inplace = True)

df.head()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type	ColumnInserted
0	5.1	3.5	1.4	0.2	Iris-setosa	1.457143
1	4.9	3.0	1.4	0.2	Iris-setosa	1.633333
2	4.7	3.2	1.3	0.2	Iris-setosa	1.468750
3	4.6	3.1	1.5	0.2	Iris-setosa	1.483871
4	5.0	3.6	1.4	0.2	Iris-setosa	1.388889

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

df["Ms_6"] = df.Type.replace({1: "Iris-setosa", 2: "Purple"})

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

	Ms_1	Ms_2	Ms_3	Ms_4	Type	ColumnInserted	Ms_6
145	6.7	3.0	5.2	2.3	Iris-virginica	2.233333	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica	2.520000	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica	2.166667	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica	1.823529	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica	1.966667	Iris-virginica

```
# Eliminate inserted column
df.drop("Ms_6", axis=1, inplace = True)
#df.drop(['GenderGroupNew'],vaxis='columns',vinplace=True)

df.head()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type	ColumnInserted
0	5.1	3.5	1.4	0.2	Iris-setosa	1.457143
1	4.9	3.0	1.4	0.2	Iris-setosa	1.633333
2	4.7	3.2	1.3	0.2	Iris-setosa	1.468750
3	4.6	3.1	1.5	0.2	Iris-setosa	1.483871
4	5.0	3.6	1.4	0.2	Iris-setosa	1.388889

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

# Create a column data
NewColumnData = df.Ms_1/df.Ms_2

# Insert that column in the data frame
df.insert(1, "Ms_7", NewColumnData, True)

df["Ms_7"] = pd.cut(df.Ms_3, [60., 63., 66., 69., 72., 75., 78.])

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

	Ms_1	Ms_7	Ms_2	Ms_3	Ms_4	Type
0	5.1	NaN	3.5	1.4	0.2	Iris-setosa
1	4.9	NaN	3.0	1.4	0.2	Iris-setosa
2	4.7	NaN	3.2	1.3	0.2	Iris-setosa
3	4.6	NaN	3.1	1.5	0.2	Iris-setosa
4	5.0	NaN	3.6	1.4	0.2	Iris-setosa

```
# Eliminate inserted column
df.drop("Ms_7", axis=1, inplace = True)

df.head()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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

```
# Drop several "unused" columns
#vars = ["Ms_3", "Ms_4", "Type"]
#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()
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
df.loc[len(df.index)] = [26, 24, 2.3, 1, 'Iris-virginica', ]

df.tail()

```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica
150	26.0	24.0	2.3	1.0	Iris-virginica

```
# Eliminate inserted row
df.drop([27], inplace = True )

df.tail()

```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica
150	26.0	24.0	2.3	1.0	Iris-virginica

Cleaning your data: drop out unused columns and/or 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", "Score"]
#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", "Score"]
#df = df[vars].dropna()

df

```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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
...	...	...	...	...	...
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica
150	26.0	24.0	2.3	1.0	Iris-virginica

149 rows × 5 columns

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

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
        - **Ms_1**
        - **Ms_2**
        - **Ms_3**
        - **Ms_4**
    - Identify the type of each column
        - **Ms_1: Float**
        - **Ms_2: Float**
        - **Ms_3: Float**
        - **Ms_4: Float**
    - Minimum, maximum, mean, average, median, standar deviation
        - Ms_1 Ms_2 Ms_3 Ms_4
        - count 149.000000 149.000000 149.000000 149.000000
        - mean 5.987248 3.192617 3.779195 1.211409
        - std 1.846373 1.770368 1.754772 0.756015
        - min 4.300000 2.000000 1.000000 0.100000
        - 25% 5.100000 2.800000 1.600000 0.300000
        - 50% 5.800000 3.000000 4.400000 1.300000
        - 75% 6.400000 3.300000 5.100000 1.800000
        - max 26.000000 24.000000 6.900000 2.500000

```
df.describe()
```

	Ms_1	Ms_2	Ms_3	Ms_4
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

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

- **There is no missing data**

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

```
Ms_1    0
Ms_2    0
Ms_3    0
Ms_4    0
Type    0
dtype: int64
```

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

```
new_df = df.copy()
vars = ["Ms_3", "Ms_4"]
new_df.drop(vars, axis=1, inplace = True)
new_df.head()
```

	Ms_1	Ms_2	Type
0	5.1	3.5	Iris-setosa
1	4.9	3.0	Iris-setosa
2	4.7	3.2	Iris-setosa
3	4.6	3.1	Iris-setosa
4	5.0	3.6	Iris-setosa

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

```
new2_df = df.copy()
vars = ["Ms_1", "Ms_2"]
new2_df.drop(vars, axis=1, inplace = True)
new2_df.head()
```

	Ms_3	Ms_4	Type
0	1.4	0.2	Iris-setosa
1	1.4	0.2	Iris-setosa
2	1.3	0.2	Iris-setosa
3	1.5	0.2	Iris-setosa
4	1.4	0.2	Iris-setosa

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

```
df["Type encoded"] = df.Type.replace({"Iris-setosa": 1,"Iris-versicolor": 2, "Iris-virginica": 3})


df_new5 = df.copy()

vars = ["Ms_1", "Ms_2", "Type"]
```