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/NotebooksProfessor"

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).
```

```
# 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)

```
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
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py
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
```

```
(52, 12)
```

```
df.shape[0]
```

```
52
```

```
df.shape[1]
```

```
12
```

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

```
df
```

20	21	23.0	M	2	Y	1	69.00	67.0	66
21	22	29.0	M	2	N	0	71.00	70.0	101
22	23	25.0	M	2	N	0	70.00	68.0	82
23	24	26.0	M	2	N	0	69.00	71.0	63
24	25	23.0	F	1	Y	1	65.00	63.0	67
25	26	28.0	M	2	N	0	75.00	76.0	111
26	27	24.0	M	2	N	0	78.40	71.0	92
27	28	25.0	M	2	Y	1	76.00	73.0	107
28	29	32.0	F	1	Y	1	63.00	60.0	75
29	30	38.0	F	1	Y	1	61.50	61.0	78
30	31	27.0	F	1	Y	1	62.00	60.0	72
31	32	33.0	F	1	Y	1	65.30	64.0	91
32	33	38.0	F	1	N	0	64.00	63.0	86
33	34	27.0	M	2	N	0	77.00	75.0	100
34	35	24.0	F	1	N	0	67.80	62.0	98
35	36	27.0	M	2	N	0	68.00	66.0	74
36	37	25.0	F	1	Y	1	65.00	64.5	92
37	38	26.0	F	1	N	0	61.50	59.5	90
38	39	31.0	M	2	Y	1	73.00	74.0	72
39	40	30.0	M	2	Y	1	69.50	66.0	96
40	41	23.0	F	1	N	0	70.40	71.0	66
41	42	26.0	M	2	Y	1	73.50	72.0	115
42	43	28.0	F	1	Y	1	72.50	72.0	81
43	44	26.0	F	1	Y	1	72.00	72.0	92
44	45	30.0	F	1	Y	1	66.00	64.0	85
45	46	39.0	F	1	N	0	64.00	63.0	87

46	47	27.0	M	2	N	0	78.00	75.0	72
47	48	24.0	M	2	N	0	79.50	75.0	82
48	49	28.0	M	2	N	0	77.80	76.0	99
49	50	30.0	F	1	N	0	74.60	NaN	71
50	51	NaN	M	2	N	0	71.00	70.0	101
51	52	27.0	M	2	N	0	NaN	71.5	103

Pasos siguientes: Generar código con df | Ver gráficos recomendados | New interactive sheet

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(5)
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Co
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	

Pasos siguientes: Generar código con df | Ver gráficos recomendados | New interactive sheet

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

```
df.tail(5)
```

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

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
```

```
                     0
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  CWDistance
count  52.000000  51.000000    52.000000     52.000000  51.000000  51.000000   52.000000
mean   26.500000  28.411765     1.500000      0.500000  68.971569  67.313725   85.576923
std    15.154757   5.755611     0.504878      0.504878   5.303812   5.624021   14.353173
min     1.000000  22.000000     1.000000      0.000000  61.500000  57.500000   63.000000
25%    13.750000  25.000000     1.000000      0.000000  64.500000  63.000000   72.000000
50%    26.500000  27.000000     1.500000      0.500000  69.000000  66.000000   85.000000
75%    39.250000  30.000000     2.000000      1.000000  73.000000  72.000000   96.500000
max    52.000000  56.000000     2.000000      1.000000  79.500000  76.000000  115.000000
```

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

```
        Gender
count       52
unique       2
top          F
freq        26

dtype: object
```

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.Height.mean()
```

```
68.97156862745098
```

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')

```
df.to_csv(Ruta + "/datasets/cartwheel/cartwheel2.csv")
```

Rename columns

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

Example:

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

df.head()

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

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

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

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

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"]]

a2= df[["Gender", "GenderGroup"]]
a2
```

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
31	F	1
32	F	1
33	M	2
34	F	1
35	M	2
36	F	1
37	F	1
38	M	2
39	M	2
40	F	1
41	M	2
42	F	1
43	F	1
44	F	1
45	F	1

```
46    M    2
47    M    2
48    M    2
49    F    1
50    M    2
51    M    2
```

Pasos siguientes: Generar código con a2 | Ver gráficos recomendados | New interactive sheet

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"]

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

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

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

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

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

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

df.Gender.unique()

```
# 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]

```
idx = df.Age >= 35
df2 = df[idx]
df2
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	C
0	1	56.0	F	1	Y	1	62.0	61.0	79	
3	4	39.0	F	1	N	0	64.0	63.0	87	
29	30	38.0	F	1	Y	1	61.5	61.0	78	
32	33	38.0	F	1	N	0	64.0	63.0	86	
45	46	39.0	F	1	N	0	64.0	63.0	87	

Pasos siguientes: Generar código con `df2` Ver gráficos recomendados New interactive sheet

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)

```
df3 = df2.sort_values("CWDistance", ascending = True)
df3
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	C
29	30	38.0	F	1	Y	1	61.5	61.0	78	
0	1	56.0	F	1	Y	1	62.0	61.0	79	
32	33	38.0	F	1	N	0	64.0	63.0	86	
3	4	39.0	F	1	N	0	64.0	63.0	87	
45	46	39.0	F	1	N	0	64.0	63.0	87	

Pasos siguientes: Generar código con `df3` Ver gráficos recomendados New interactive sheet

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'])

```
a = df.groupby(['Gender'])
```

```
a.size()
```

```
          0
Gender
  F      26
  M      26

dtype: int64
```

Size of each group

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

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

```
Empieza a programar o a crear código con IA.
```

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.

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

```
Empieza a programar o a crear código con IA.
```

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

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())

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

```
1
```

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

	Age
count	8.000000
mean	30.645922
std	16.044470
min	5.755611
25%	24.250000
50%	27.705882
75%	35.250000
max	56.000000

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	Co
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	

Pasos siguientes: Generar código con df Ver gráficos recomendados New interactive sheet

```
# 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	Co
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	

Pasos siguientes: Generar código con df Ver gráficos recomendados New interactive sheet

```
# # 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  Co
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
```

Pasos siguientes: Generar código con df | Ver gráficos recomendados | New interactive sheet

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

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

```
# 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
```

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

```
## 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()
```

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

```
# 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

```
# Print tail
```

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

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

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
```

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
   - 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

```
# Import necessary libraries
import seaborn as sns

# Load the Iris dataset from seaborn
df = sns.load_dataset('iris')

# Check the first few rows of the dataset
df
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

Pasos siguientes: Generar código con df | Ver gráficos recomendados | New interactive sheet

Column Names and Types

```
# Check column names
print(df.columns)

# Check the type of each column
df.dtypes
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

	0
sepal_length	float64
sepal_width	float64
petal_length	float64
petal_width	float64
species	object

dtype: object