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

```
1 # Import the packages that we will be using
2 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/MyDrive/ITC/5toSem/semanaTecAn/"

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

```
# Import the packages that we will be using
import pandas as pd
# Dataset url
url="datasets/cartwheel/cartwheel.csv"
# Load the dataset
df= pd.read_csv(Ruta + url)
# Print data set
df
```

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

Next steps: Generate code with `df` View recommended plots New interactive sheet

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

```
1 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.

```
Nc, Nr = df.shape
print(f"El numero de columnas/mediciones es: {Nc}, El numero de filas/rows es: {Nr}")
```

```
El numero de columnas/mediciones es: 52, El numero de filas/rows es: 12
```

```
Ncol=df.shape[0]
print(f"El numero de columnas/mediciones es: {Ncol}")
```

```
El numero de columnas/mediciones es: 52
```

```
Nrow=df.shape[1]
print("El numero de filas/rows es: {Nrow}")
```

```
El numero de filas/rows es: {Nrow}
```

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

```
df
```

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

Next steps: Generate code with `df` View recommended plots 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

```
1 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	

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

```
1 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:

```
1 print(df.columns)
```

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

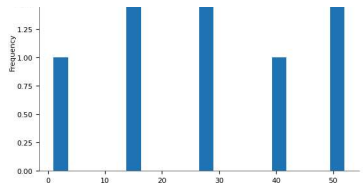
Be aware that every variable in a Pandas data frame has a data type. There are many different data types, but most commonly you will encounter floating point values (real numbers), integers, strings (text), and date/time values. When Pandas reads a text/csv file, it guesses the data types based on what 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.

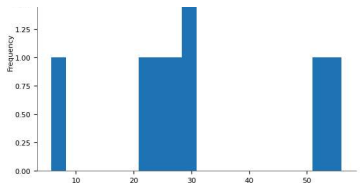
```
1 print(df.dtypes)
```

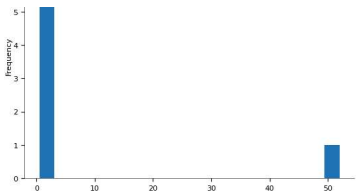
```
ID                int64
Age             float64
Gender           object
GenderGroup       int64
Glasses          object
GlassesGroup      int64
Height          float64
Wingspan        float64
CWDistance        int64
Complete         object
CompleteGroup   float64
Score             int64
dtype: object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

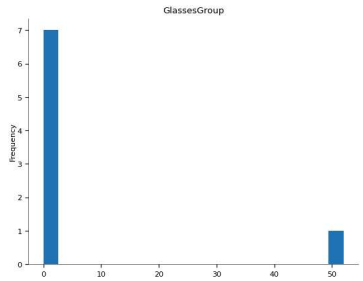
Summary statistics, which include things like the mean, min, and max of the data, can be useful to get a feel for how large some of the variables are and what variables may be the most important.

```
1 # Summary statistics for the quantitative variables
2 df.describe()
```

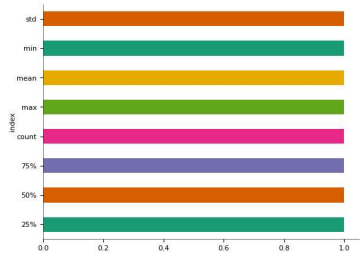




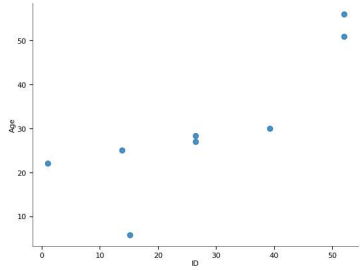


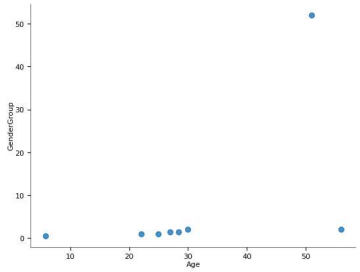


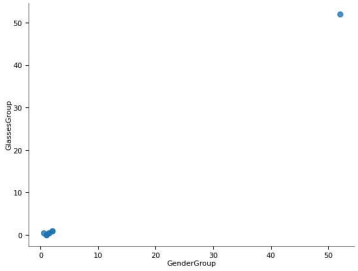
Categorical distributions

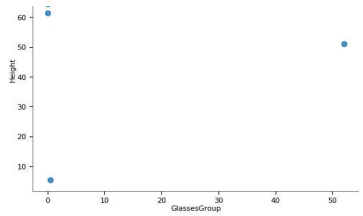


2-d distributions

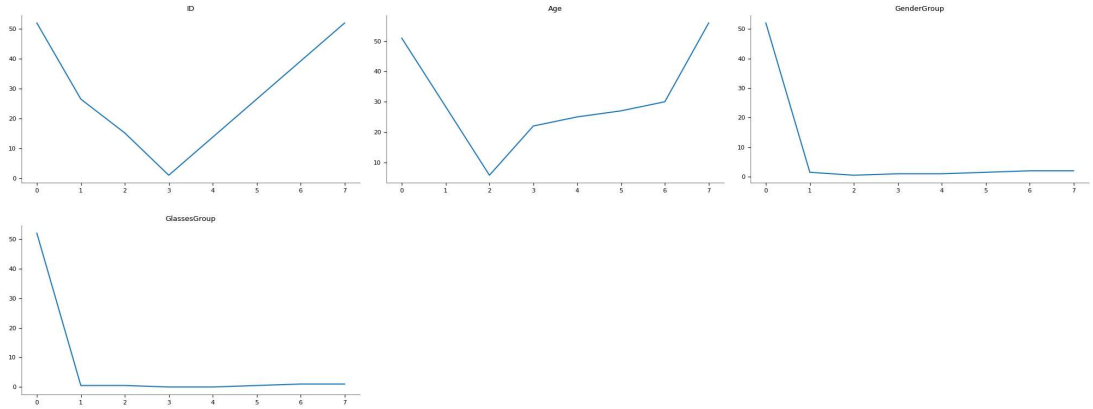








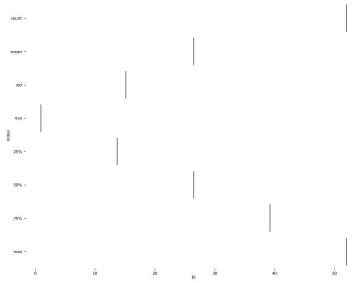
Values



Faceted distributions

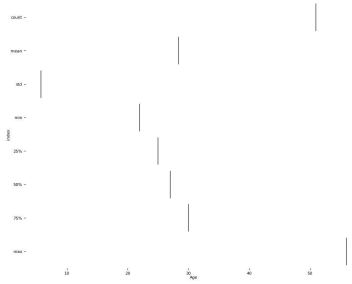
```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
```



```
                    | <string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
```



```
                    |   <string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
```

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

	Wingspan
count	51.000000
mean	67.313725
std	5.624021
min	57.500000
25%	63.000000
50%	66.000000
75%	72.000000
max	76.000000

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

```
# Select only numeric columns and apply statistical functions
numeric_df = df.select_dtypes(include=['number'])

# Apply statistical functions to the numeric data
print(numeric_df.mean())     # Mean of each numeric column
print(numeric_df.corr())     # Correlation matrix of numeric columns
print(numeric_df.count())    # Count of non-null values in each numeric column
print(numeric_df.max())      # Maximum value in each numeric column
print(numeric_df.min())      # Minimum value in each numeric column
print(numeric_df.median())   # Median of each numeric column
print(numeric_df.std())      # Standard deviation of each numeric column
```

```
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
                     ID       Age  GenderGroup  GlassesGroup    Height  \
ID             1.000000 -0.139088     0.066630     -0.184513  0.313570
Age           -0.139088  1.000000    -0.263563      0.153441 -0.321105
GenderGroup    0.066630 -0.263563     1.000000     -0.230769  0.731284
GlassesGroup  -0.184513  0.153441    -0.230769      1.000000 -0.251782
Height         0.313570 -0.321105     0.731284     -0.251782  1.000000
Wingspan       0.272402 -0.250976     0.763129     -0.212414  0.941516
CWDistance     0.120251 -0.030501     0.265168     -0.075762  0.315947
CompleteGroup  0.110208  0.229843    -0.224166      0.116313 -0.242616
Score          0.223190  0.317358    -0.272192     -0.149266 -0.042706

               Wingspan  CWDistance  CompleteGroup     Score
ID             0.272402    0.120251       0.110208  0.223190
Age           -0.250976   -0.030501       0.229843  0.317358
GenderGroup    0.763129    0.265168      -0.224166 -0.272192
GlassesGroup  -0.212414   -0.075762       0.116313 -0.149266
Height         0.941516    0.315947      -0.242616 -0.042706
Wingspan       1.000000    0.326148      -0.198542 -0.044400
CWDistance     0.326148    1.000000       0.175570  0.368652
CompleteGroup -0.198542    0.175570       1.000000  0.372424
Score         -0.044400    0.368652       0.372424  1.000000
ID               52
Age              51
GenderGroup      52
GlassesGroup     52
Height           51
Wingspan         51
CWDistance       52
CompleteGroup    51
Score            52
dtype: int64
ID               52.0
Age              56.0
GenderGroup       2.0
GlassesGroup      1.0
Height           79.5
Wingspan         76.0
CWDistance      115.0
CompleteGroup     1.0
Score            10.0
dtype: float64
ID               1.0
Age             22.0
GenderGroup      1.0
GlassesGroup     0.0
Height          61.5
Wingspan        57.5
```

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('DataManagment_Cartwheel.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"})
```

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

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"] #toda la columna
#d = df.iloc[:, 1] #segmentos dentro de la columna

#print(d)

df[["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
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

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,:]


```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	C
10	11	30.0	M	2	Y	1	69.50	66.0	96	
11	12	28.0	F	1	Y	1	62.75	58.0	79	
12	13	25.0	F	1	Y	1	65.00	64.5	92	
13	14	23.0	F	1	N	0	61.50	57.5	66	
14	15	31.0	M	2	Y	1	73.00	74.0	72	
15	16	26.0	M	2	Y	1	71.00	72.0	115	

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

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
print(df['GenderGroup'].unique())
```

```
[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]

```
# Filter rows where 'Height' is greater than or equal to 70
filtered_df = df[df["Height"] >= 70]
print(filtered_df)

# Filter rows where 'Height' is greater than or equal to 70 AND 'Age' is less than 30
filtered_df = df[(df["Height"] >= 70) & (df["Age"] < 30)]
print(filtered_df)

# Filter rows where 'Height' is greater than or equal to 70 OR 'Gender' is 'Female'
filtered_df = df[(df["Height"] >= 70) | (df["Gender"] == 'Female')]
print(filtered_df)
```

8	106	N	0.0	5
14	72	Y	1.0	9
15	115	Y	1.0	6
18	64	Y	1.0	3
21	101	Y	1.0	8
22	82	Y	1.0	4
25	111	Y	1.0	10
26	92	Y	1.0	7
27	107	Y	1.0	8
33	100	Y	1.0	8
38	72	Y	1.0	9
40	66	Y	1.0	4
41	115	Y	1.0	6
42	81	Y	1.0	10
43	92	Y	1.0	8
46	72	N	0.0	7
47	82	N	0.0	8
48	99	Y	1.0	9
49	71	Y	1.0	9
50	101	Y	NaN	8

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)

```
# Sort the DataFrame by the 'Height' column in ascending order
sorted_df = df.sort_values("Height")
print(sorted_df)

# Sort the DataFrame by the 'Height' column in descending order
sorted_df = df.sort_values("Height", ascending=False)
print(sorted_df)

# Sort the DataFrame by 'Height' in ascending order, then by 'Age' in descending order
sorted_df = df.sort_values(["Height", "Age"], ascending=[True, False])
print(sorted_df)
```

	CWDistance	Complete	CompleteGroup	Score
29	78	Y	1.0	7
16	90	N	0.0	10
37	90	Y	1.0	9
13	66	Y	1.0	4
0	79	Y	1.0	7
30	72	Y	1.0	8
1	70	Y	1.0	8
11	79	Y	1.0	10
9	65	Y	1.0	8
28	75	Y	1.0	8
3	87	Y	1.0	10
45	87	Y	1.0	10
32	86	Y	1.0	10
12	92	Y	1.0	6
36	92	Y	1.0	6
24	67	N	0.0	3
7	98	Y	1.0	9
31	91	Y	1.0	7
2	85	Y	1.0	7
44	85	Y	1.0	7
17	74	Y	1.0	5
34	98	Y	1.0	9
35	74	Y	1.0	5
19	85	Y	1.0	8
23	63	Y	1.0	5
20	66	N	0.0	2
10	96	Y	1.0	6
39	96	Y	1.0	6
22	82	Y	1.0	4
18	64	Y	1.0	3
40	66	Y	1.0	4
21	101	Y	1.0	8
15	115	Y	1.0	6
50	101	Y	NaN	8
43	92	Y	1.0	8
42	81	Y	1.0	10
14	72	Y	1.0	9
38	72	Y	1.0	9
4	72	N	0.0	4
41	115	Y	1.0	6
8	106	N	0.0	5
49	71	Y	1.0	9
6	107	Y	1.0	10
25	111	Y	1.0	10
5	81	N	0.0	3
27	107	Y	1.0	8
33	100	Y	1.0	8
48	99	Y	1.0	9
46	72	N	0.0	7
26	92	Y	1.0	7
47	82	N	0.0	8
51	103	Y	1.0	10

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'])

```
# Group the DataFrame by the 'Gender' column
grouped_df = df.groupby(['Gender'])

# Display grouped object (to view data, you usually apply an aggregate function)
print(grouped_df)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x79b1bb257eb0>
```

Size of each group

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

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

```
#Size of each group
df.groupby(['Gender']).size()
df.groupby(['Gender','GenderGroup']).size()
```

		0
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.

```
# Count the number of missing values in each column
missing_values_count = df.isnull().sum()
print("Missing values in each column:\n", missing_values_count)

# Count the number of non-missing values in each column
non_missing_values_count = df.notnull().sum()
print("Non-missing values in each column:\n", non_missing_values_count)
```

```
Missing values in each column:
 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
Non-missing values in each column:
 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
```

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

```
1 df.notnull().sum()
```

	0
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

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

	0
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

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())

```
#Now we use these functions to count the number of missing and non-missing values in a si
print( df.Height.notnull().sum() )
print( pd.isnull(df.Height).sum() )
```

```
51
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

```
1 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	

```
1 # Create new column data by dividing 'Age' by itself, resulting in a column of 1s
2 NewColumnData = df.Age / df.Age
3
4 # Insert the new column into the DataFrame at position 12 with the name "ColumnInserted"
5 df.insert(12, "ColumnInserted", NewColumnData, True)
6
7 # Display the DataFrame to verify the new column was added
8 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	

```
# # 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	

```
# # 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	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	

```
# # Eliminate inserted column
df.drop("AgeInMonths", 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	

```
# 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(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	

```
## 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()


```

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

```
## Eliminate inserted column
df.drop("ColumnStrata", 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	

```
# 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.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

```
# Create a new row with the correct number of values
print(len(df.columns))
# Make sure the list length matches the number of columns in the DataFrame
new_row = [26, 24, 'F', 1, 'Y', 1, 66, 'NaN', 68, 'N', 0, 3]  # Example list

# Ensure the list length matches the DataFrame's column count
if len(new_row) == len(df.columns):
    # Add the new row at the end of the DataFrame
    df.loc[len(df.index)] = new_row
else:
    print("Error: The new row does not match the number of columns in the DataFrame.")

# Print the last few rows to verify
print(df.tail())
```

```
8
Error: The new row does not match the number of columns in the DataFrame.
     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
```