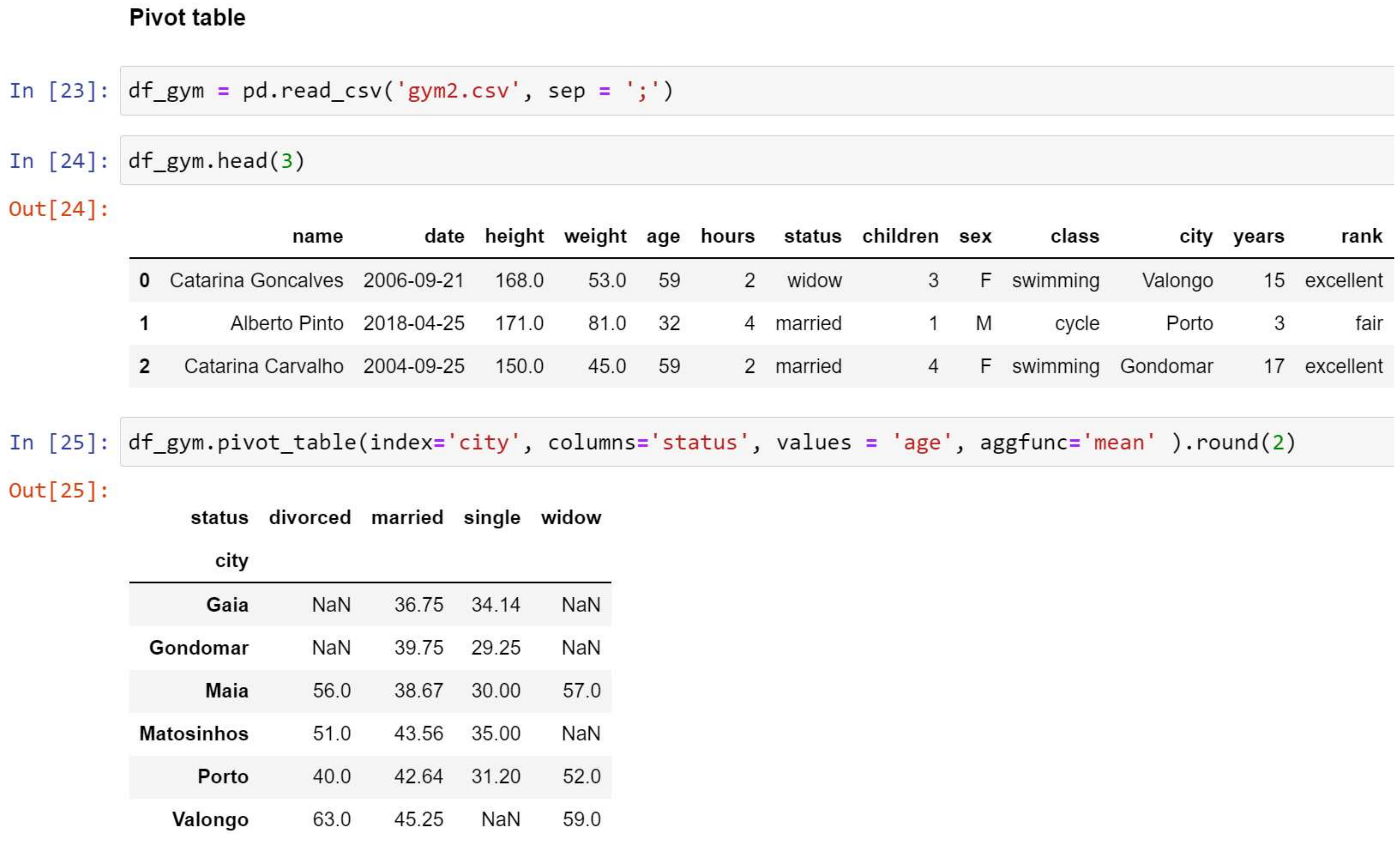
Pandas – pivot_table()

Create a spreadsheet-style pivot table as a DataFrame.

pandas.pivot_table(**data**, **values**=None, **index**=None, **columns**=None, **aggfunc**='mean', **fill_value**=None, **margins**=False, **dropna**=True, **margins_name**='All', **observed**=False, **sort**=True)

- **data**: a DataFrame object.
- **values**: a column or a list of columns to aggregate.
- **index**: a column, Grouper, array which has the same length as data, or list of them. Keys to group by on the pivot table index. If an array is passed, it is being used as the same manner as column values.
- **columns**: a column, Grouper, array which has the same length as data, or list of them. Keys to group by on the pivot table column. If an array is passed, it is being used as the same manner as column values.
- **aggfunc**: function to use for aggregation, defaulting to numpy.mean
- **fill_value**: value to replace missing values
- **margins**: add all row / columns (e.g. for subtotal / grand totals).
- **dropna**: do not include columns whose entries are all NaN.
- **margins_name**: name of the row / column that will contain the totals when margins is True
- **observed**: this only applies if any of the groupers are Categoricals
- **sort**: specifies if the result should be sorted

Pandas – pivot_table() - example



Pivot table

```
In [23]: df_gym = pd.read_csv('gym2.csv', sep = ';')
```

```
In [24]: df_gym.head(3)
```

Out[24]:

	name	date	height	weight	age	hours	status	children	sex	class	city	years	rank
0	Catarina Goncalves	2006-09-21	168.0	53.0	59	2	widow	3	F	swimming	Valongo	15	excellent
1	Alberto Pinto	2018-04-25	171.0	81.0	32	4	married	1	M	cycle	Porto	3	fair
2	Catarina Carvalho	2004-09-25	150.0	45.0	59	2	married	4	F	swimming	Gondomar	17	excellent

```
In [25]: df_gym.pivot_table(index='city', columns='status', values = 'age', aggfunc='mean' ).round(2)
```

Out[25]:

status	divorced	married	single	widow
city				
Gaia	NaN	36.75	34.14	NaN
Gondomar	NaN	39.75	29.25	NaN
Maia	56.0	38.67	30.00	57.0
Matosinhos	51.0	43.56	35.00	NaN
Porto	40.0	42.64	31.20	52.0
Valongo	63.0	45.25	NaN	59.0

Pandas – groupby()

A groupby operation involves some combination of splitting the object, applying a function, and combining the results

DataFrame.groupby(**by**=None, **axis**=0, **level**=None, **as_index**=True, **sort**=True, **group_keys** =True, **squeeze**=NoDefault.no_default, **observed**=False, **dropna**=True)

by : mapping, function, label, or list of labels - Used to determine the groups for the groupby.

axis : {0 or 'index', 1 or 'columns'}, default 0 - Split along rows (0) or columns (1).

level : int, level name, or sequence of such, default None - If the axis is a MultiIndex (hierarchical), group by a particular level or levels.

as_index : bool, default True - For aggregated output, return object with group labels as the index.

sort : bool, default True - Sort group keys. Get better performance by turning this off.

group_keys : bool, default True - When calling apply, add group keys to index to identify pieces.

squeeze : bool, default False - Reduce the dimensionality of the return type if possible, otherwise return a consistent type.

observed : bool, default False - This only applies if any of the groupers are Categoricals.

dropna : bool, default True - If True, and if group keys contain NA values, NA values together with row/column will be dropped. If False, NA values will also be treated as the key in groups

Pandas – groupby()

By "group by" we are referring to a process involving one or more of the following steps:

- **Splitting** the data into groups based on some criteria.
- **Applying** a function to each group independently.
- **Combining** the results into a data structure.

Out of these, the split step is the most straightforward. In fact, in many situations we may wish to split the data set into groups and do something with those groups. In the apply step, we might wish to do one of the following:

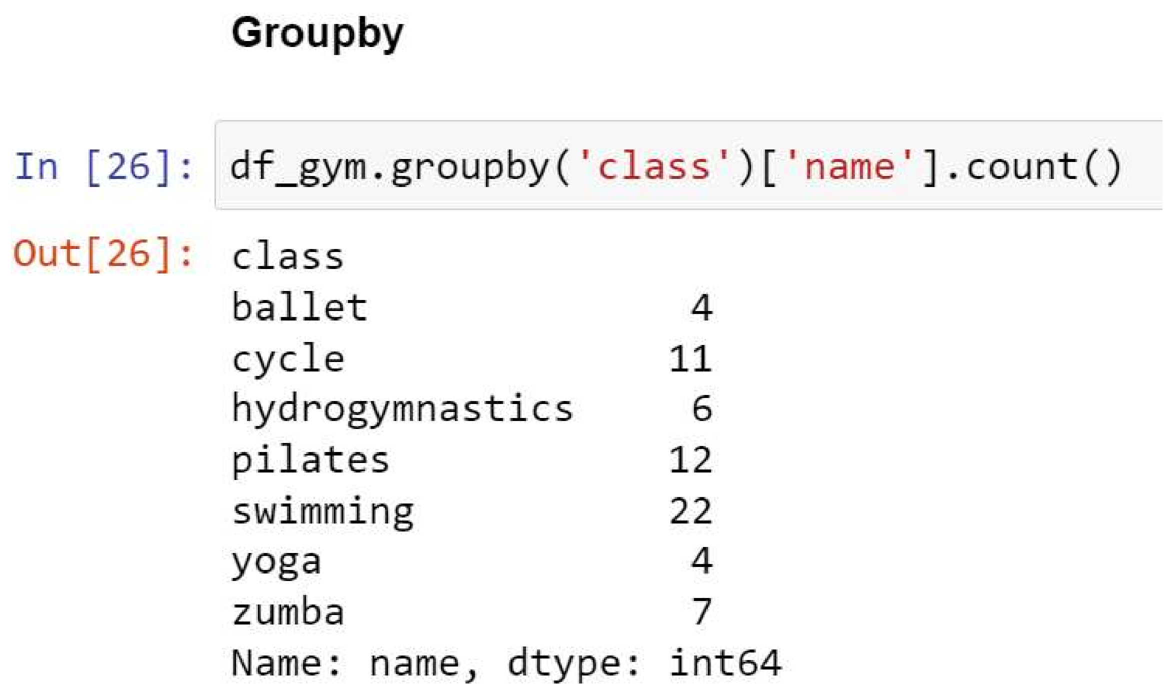
- **Aggregation**: compute a summary statistic (or statistics) for each group. Some examples:
  - Compute group sums or means.
  - Compute group sizes / counts.
- **Transformation**: perform some group-specific computations and return a like-indexed object. Some examples:
  - Standardize data (zscore) within a group.
  - Filling NAs within groups with a value derived from each group.
- **Filtration**: discard some groups, according to a group-wise computation that evaluates True or False. Some examples:
  - Discard data that belongs to groups with only a few members.
  - Filter out data based on the group sum or mean.
- Some combination of the above: GroupBy will examine the results of the apply step and try to return a sensibly combined result if it doesn't fit into either of the above two categories.

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Pandas – groupby() – aggregating functions

Function	Description
mean()	Compute mean of groups
sum()	Compute sum of group values
size()	Compute group sizes
count()	Compute count of group
std()	Standard deviation of groups
var()	Compute variance of groups
sem()	Standard error of the mean of groups
describe()	Generates descriptive statistics
first()	Compute first of group values
last()	Compute last of group values
nth()	Take nth value, or a subset if n is a list
min()	Compute min of group values
max()	Compute max of group values

Pandas – groupby() - example



Groupby

```
In [26]: df_gym.groupby('class')['name'].count()

Out[26]: class
         ballet              4
         cycle              11
         hydrogymnastics     6
         pilates            12
         swimming           22
         yoga                4
         zumba               7
         Name: name, dtype: int64
```