(continued from previous page)

```
999 2002-09-26  11.440539 -50.553522  21.762509  7.829262

[1000 rows x 5 columns]
```

Gotchas

If you are attempting to perform an operation you might see an exception like:

```
>>> if pd.Series([False, True, False]):
...     print("I was true")
Traceback
    ...
ValueError: The truth value of an array is ambiguous. Use a.empty, a.any() or a.all().
```

See *Comparisons* for an explanation and what to do.

See *Gotchas* as well.

1.4.4 Getting started tutorials

What kind of data does pandas handle?

I want to start using pandas

```
In [1]: import pandas as pd
```

To load the pandas package and start working with it, import the package. The community agreed alias for pandas is `pd`, so loading pandas as `pd` is assumed standard practice for all of the pandas documentation.

Pandas data table representation

I want to store passenger data of the Titanic. For a number of passengers, I know the name (characters), age (integers) and sex (male/female) data.

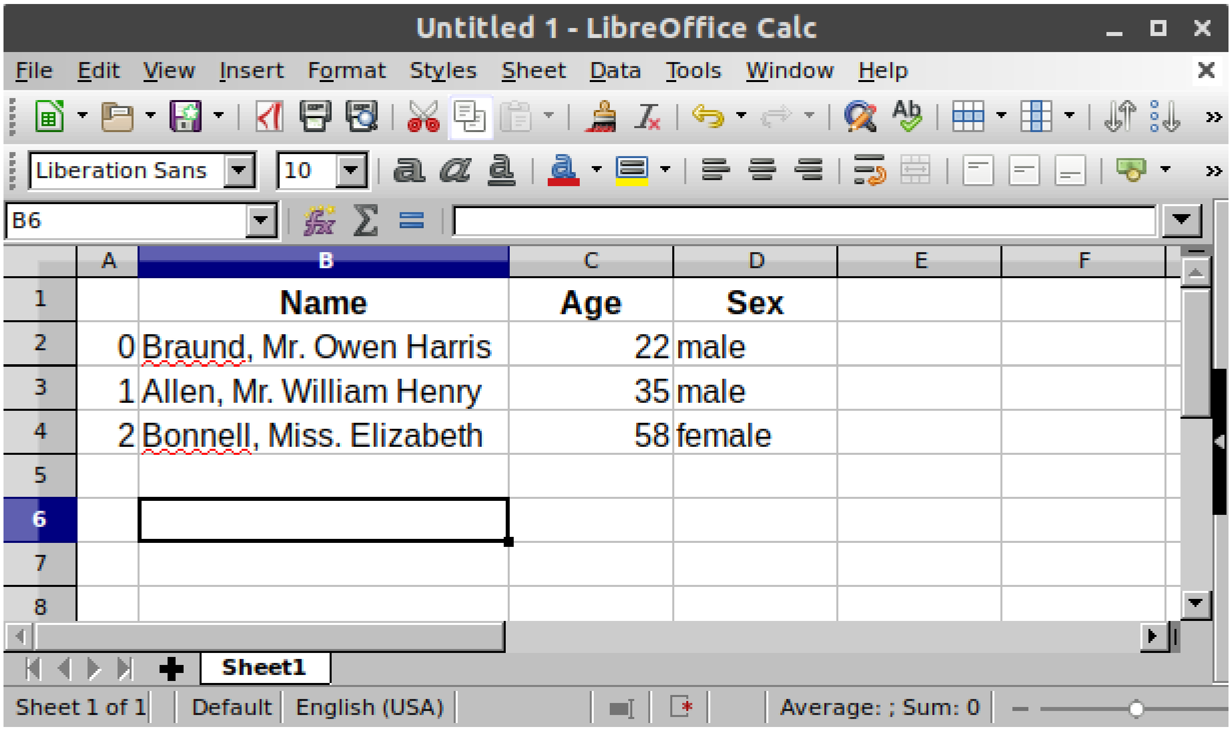
```
In [2]: df = pd.DataFrame({
   ...:     "Name": ["Braund, Mr. Owen Harris",
   ...:              "Allen, Mr. William Henry",
   ...:              "Bonnell, Miss. Elizabeth"],
   ...:     "Age": [22, 35, 58],
   ...:     "Sex": ["male", "male", "female"]}
   ...: )
   ...:

In [3]: df
Out[3]:
                       Name  Age     Sex
0   Braund, Mr. Owen Harris   22    male
1  Allen, Mr. William Henry   35    male
2  Bonnell, Miss. Elizabeth   58  female
```

To manually store data in a table, create a `DataFrame`. When using a Python dictionary of lists, the dictionary keys will be used as column headers and the values in each list as rows of the `DataFrame`.

A *`DataFrame`* is a 2-dimensional data structure that can store data of different types (including characters, integers, floating point values, categorical data and more) in columns. It is similar to a spreadsheet, a SQL table or the `data.frame` in R.

- The table has 3 columns, each of them with a column label. The column labels are respectively `Name`, `Age` and `Sex`.
- The column `Name` consists of textual data with each value a string, the column `Age` are numbers and the column `Sex` is textual data.

In spreadsheet software, the table representation of our data would look very similar:


Untitled 1 - LibreOffice Calc
File Edit View Insert Format Styles Sheet Data Tools Window Help
Liberation Sans
10
B6
Sheet1
Sheet 1 of 1
Default
English (USA)
Average: ; Sum: 0

	A	B	C	D	E	F
1		Name	Age	Sex		
2	0	Braund, Mr. Owen Harris	22	male		
3	1	Allen, Mr. William Henry	35	male		
4	2	Bonnell, Miss. Elizabeth	58	female		
5						
6						
7						
8						

Each column in a `DataFrame` is a `Series`

I'm just interested in working with the data in the column `Age`

```
In [4]: df["Age"]
Out[4]:
0    22
1    35
2    58
Name: Age, dtype: int64
```

When selecting a single column of a pandas *`DataFrame`*, the result is a pandas *`Series`*. To select the column, use the column label in between square brackets `[]`.

Note: If you are familiar to Python dictionaries, the selection of a single column is very similar to selection of dictionary values based on the key.

You can create a `Series` from scratch as well:

```
In [5]: ages = pd.Series([22, 35, 58], name="Age")

In [6]: ages
Out[6]:
0    22
1    35
2    58
Name: Age, dtype: int64
```

A pandas `Series` has no column labels, as it is just a single column of a `DataFrame`. A Series does have row labels.

Do something with a DataFrame or Series

I want to know the maximum Age of the passengers

We can do this on the `DataFrame` by selecting the `Age` column and applying `max()`:

```
In [7]: df["Age"].max()
Out[7]: 58
```

Or to the `Series`:

```
In [8]: ages.max()
Out[8]: 58
```

As illustrated by the `max()` method, you can *do* things with a `DataFrame` or `Series`. pandas provides a lot of functionalities, each of them a *method* you can apply to a `DataFrame` or `Series`. As methods are functions, do not forget to use parentheses `()`.

I'm interested in some basic statistics of the numerical data of my data table

```
In [9]: df.describe()
Out[9]:
             Age
count   3.000000
mean   38.333333
std    18.230012
min    22.000000
25%    28.500000
50%    35.000000
75%    46.500000
max    58.000000
```

The *`describe()`* method provides a quick overview of the numerical data in a `DataFrame`. As the `Name` and `Sex` columns are textual data, these are by default not taken into account by the *`describe()`* method.

Many pandas operations return a `DataFrame` or a `Series`. The *`describe()`* method is an example of a pandas operation returning a pandas `Series`.

Check more options on `describe` in the user guide section about *aggregations with describe*

Note: This is just a starting point. Similar to spreadsheet software, pandas represents data as a table with columns and rows. Apart from the representation, also the data manipulations and calculations you would do in spreadsheet software are supported by pandas. Continue reading the next tutorials to get started!

- Import the package, aka `import pandas as pd`
- A table of data is stored as a pandas `DataFrame`
- Each column in a `DataFrame` is a `Series`
- You can do things by applying a method to a `DataFrame` or `Series`

A more extended explanation to `DataFrame` and `Series` is provided in the *introduction to data structures*.

```
In [1]: import pandas as pd
```

This tutorial uses the titanic data set, stored as CSV. The data consists of the following data columns:

- PassengerId: Id of every passenger.
- Survived: This feature have value 0 and 1. 0 for not survived and 1 for survived.
- Pclass: There are 3 classes: Class 1, Class 2 and Class 3.
- Name: Name of passenger.
- Sex: Gender of passenger.
- Age: Age of passenger.
- SibSp: Indication that passenger have siblings and spouse.
- Parch: Whether a passenger is alone or have family.
- Ticket: Ticket number of passenger.
- Fare: Indicating the fare.
- Cabin: The cabin of passenger.
- Embarked: The embarked category.

How do I read and write tabular data?

I want to analyse the titanic passenger data, available as a CSV file.

```
In [2]: titanic = pd.read_csv("data/titanic.csv")
```

pandas provides the *`read_csv()`* function to read data stored as a csv file into a pandas `DataFrame`. pandas supports many different file formats or data sources out of the box (csv, excel, sql, json, parquet, …), each of them with the prefix `read_*`.

Make sure to always have a check on the data after reading in the data. When displaying a `DataFrame`, the first and last 5 rows will be shown by default:

```
In [3]: titanic
Out[3]:
     PassengerId  Survived  Pclass                                               Name␣
↪    Sex  ...  Parch            Ticket     Fare Cabin  Embarked
```

(continues on next page)

(continued from previous page)

```
0              1         0       3                            Braund, Mr. Owen Harris␣
↪    male  ...      0         A/5 21171   7.2500   NaN          S
1              2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...␣
↪  female  ...      0          PC 17599  71.2833   C85          C
2              3         1       3                             Heikkinen, Miss. Laina␣
↪  female  ...      0  STON/O2. 3101282   7.9250   NaN          S
3              4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)␣
↪  female  ...      0            113803  53.1000  C123          S
4              5         0       3                           Allen, Mr. William Henry␣
↪    male  ...      0            373450   8.0500   NaN          S
..           ...       ...     ...                                                ...␣
↪     ...  ...    ...                ...      ...   ...       ...
886          887         0       2                              Montvila, Rev. Juozas␣
↪    male  ...      0            211536  13.0000   NaN          S
887          888         1       1                       Graham, Miss. Margaret Edith␣
↪  female  ...      0            112053  30.0000   B42          S
888          889         0       3           Johnston, Miss. Catherine Helen "Carrie"␣
↪  female  ...      2        W./C. 6607  23.4500   NaN          S
889          890         1       1                              Behr, Mr. Karl Howell␣
↪    male  ...      0            111369  30.0000  C148          C
890          891         0       3                                Dooley, Mr. Patrick␣
↪    male  ...      0            370376   7.7500   NaN          Q

[891 rows x 12 columns]
```

I want to see the first 8 rows of a pandas DataFrame.

```
In [4]: titanic.head(8)
Out[4]:
   PassengerId  Survived  Pclass                                                Name  ␣
↪   Sex  ...  Parch            Ticket     Fare Cabin  Embarked
0            1         0       3                            Braund, Mr. Owen Harris   ␣
↪  male   ...     0         A/5 21171   7.2500   NaN          S
1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th... ␣
↪female  ...      0          PC 17599  71.2833   C85          C
2            3         1       3                             Heikkinen, Miss. Laina  ␣
↪female  ...      0  STON/O2. 3101282   7.9250   NaN          S
3            4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)  ␣
↪female  ...      0           113803  53.1000  C123          S
4            5         0       3                           Allen, Mr. William Henry   ␣
↪  male   ...     0            373450   8.0500   NaN          S
5            6         0       3                                   Moran, Mr. James   ␣
↪  male   ...     0            330877   8.4583   NaN          Q
6            7         0       1                            McCarthy, Mr. Timothy J   ␣
↪  male   ...     0             17463  51.8625   E46          S
7            8         0       3                     Palsson, Master. Gosta Leonard   ␣
↪  male   ...     1            349909  21.0750   NaN          S

[8 rows x 12 columns]
```

To see the first N rows of a `DataFrame`, use the *`head()`* method with the required number of rows (in this case 8) as argument.

Note: Interested in the last N rows instead? pandas also provides a *`tail()`* method. For example, `titanic.tail(10)` will return the last 10 rows of the DataFrame.

A check on how pandas interpreted each of the column data types can be done by requesting the pandas dtypes attribute:

```
In [5]: titanic.dtypes
Out[5]: 
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

For each of the columns, the used data type is enlisted. The data types in this DataFrame are integers (int64), floats (float63) and strings (object).

Note: When asking for the dtypes, no brackets are used! dtypes is an attribute of a DataFrame and Series. Attributes of DataFrame or Series do not need brackets. Attributes represent a characteristic of a DataFrame/Series, whereas a method (which requires brackets) *do* something with the DataFrame/Series as introduced in the *first tutorial*.

My colleague requested the titanic data as a spreadsheet.

```
In [6]: titanic.to_excel('titanic.xlsx', sheet_name='passengers', index=False)
```

Whereas read_* functions are used to read data to pandas, the to_* methods are used to store data. The *to_excel()* method stores the data as an excel file. In the example here, the sheet_name is named *passengers* instead of the default *Sheet1*. By setting index=False the row index labels are not saved in the spreadsheet.

The equivalent read function *to_excel()* will reload the data to a DataFrame:

```
In [7]: titanic = pd.read_excel('titanic.xlsx', sheet_name='passengers')
```

```
In [8]: titanic.head()
Out[8]: 
   PassengerId  Survived  Pclass                                               Name   ␣
↪   Sex  ...  Parch            Ticket     Fare Cabin  Embarked
0            1         0       3                            Braund, Mr. Owen Harris   ␣
↪  male  ...      0         A/5 21171   7.2500   NaN        S
1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  ␣
↪female   ...      0          PC 17599  71.2833   C85         C
2            3         1       3                             Heikkinen, Miss. Laina  ␣
↪female   ...      0  STON/O2. 3101282   7.9250    NaN         S
3            4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)  ␣
↪female   ...      0            113803  53.1000  C123         S
4            5         0       3                           Allen, Mr. William Henry   ␣
↪  male  ...      0            373450   8.0500   NaN        S

[5 rows x 12 columns]
```

I'm interested in a technical summary of a DataFrame

```
In [9]: titanic.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

The method `info()` provides technical information about a `DataFrame`, so let's explain the output in more detail:

- It is indeed a `DataFrame`.
- There are 891 entries, i.e. 891 rows.
- Each row has a row label (aka the `index`) with values ranging from 0 to 890.
- The table has 12 columns. Most columns have a value for each of the rows (all 891 values are `non-null`). Some columns do have missing values and less than 891 `non-null` values.
- The columns `Name`, `Sex`, `Cabin` and `Embarked` consists of textual data (strings, aka `object`). The other columns are numerical data with some of them whole numbers (aka `integer`) and others are real numbers (aka `float`).
- The kind of data (characters, integers,. . .) in the different columns are summarized by listing the `dtypes`.
- The approximate amount of RAM used to hold the DataFrame is provided as well.
- Getting data in to pandas from many different file formats or data sources is supported by `read_*` functions.
- Exporting data out of pandas is provided by different `to_*`methods.
- The `head`/`tail`/`info` methods and the `dtypes` attribute are convenient for a first check.

For a complete overview of the input and output possibilites from and to pandas, see the user guide section about *reader and writer functions*.

```
In [1]: import pandas as pd
```

This tutorial uses the titanic data set, stored as CSV. The data consists of the following data columns:

- PassengerId: Id of every passenger.
- Survived: This feature have value 0 and 1. 0 for not survived and 1 for survived.
- Pclass: There are 3 classes: Class 1, Class 2 and Class 3.
- Name: Name of passenger.
- Sex: Gender of passenger.
- Age: Age of passenger.

- SibSp: Indication that passenger have siblings and spouse.
- Parch: Whether a passenger is alone or have family.
- Ticket: Ticket number of passenger.
- Fare: Indicating the fare.
- Cabin: The cabin of passenger.
- Embarked: The embarked category.

```
In [2]: titanic = pd.read_csv("data/titanic.csv")

In [3]: titanic.head()
Out[3]:
   PassengerId  Survived  Pclass                                               Name  ␣
↪  Sex  ...  Parch            Ticket     Fare Cabin  Embarked
0            1         0       3                            Braund, Mr. Owen Harris  ␣
↪ male  ...      0         A/5 21171   7.2500   NaN         S
1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th... ␣
↪female  ...      0          PC 17599  71.2833   C85         C
2            3         1       3                             Heikkinen, Miss. Laina ␣
↪female  ...      0  STON/O2. 3101282   7.9250   NaN         S
3            4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel) ␣
↪female  ...      0            113803  53.1000  C123         S
4            5         0       3                           Allen, Mr. William Henry  ␣
↪ male  ...      0            373450   8.0500   NaN         S

[5 rows x 12 columns]
```

How do I select a subset of a `DataFrame`?

How do I select specific columns from a `DataFrame`?

I'm interested in the age of the titanic passengers.

```
In [4]: ages = titanic["Age"]

In [5]: ages.head()
Out[5]:
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
Name: Age, dtype: float64
```

To select a single column, use square brackets `[]` with the column name of the column of interest.

Each column in a *`DataFrame`* is a *`Series`*. As a single column is selected, the returned object is a pandas *`DataFrame`*. We can verify this by checking the type of the output:

```
In [6]: type(titanic["Age"])
Out[6]: pandas.core.series.Series
```

And have a look at the `shape` of the output:

```
In [7]: titanic["Age"].shape
Out[7]: (891,)
```

DataFrame.shape is an attribute (remember *tutorial on reading and writing*, do not use parantheses for attributes) of a pandas Series and DataFrame containing the number of rows and columns: *(nrows, ncolumns)*. A pandas Series is 1-dimensional and only the number of rows is returned.

I'm interested in the age and sex of the titanic passengers.

```
In [8]: age_sex = titanic[["Age", "Sex"]]

In [9]: age_sex.head()
Out[9]: 
    Age     Sex
0  22.0    male
1  38.0  female
2  26.0  female
3  35.0  female
4  35.0    male
```

To select multiple columns, use a list of column names within the selection brackets [].

Note: The inner square brackets define a Python list with column names, whereas the outer brackets are used to select the data from a pandas DataFrame as seen in the previous example.

The returned data type is a pandas DataFrame:

```
In [10]: type(titanic[["Age", "Sex"]])
Out[10]: pandas.core.frame.DataFrame
```

```
In [11]: titanic[["Age", "Sex"]].shape
Out[11]: (891, 2)
```

The selection returned a DataFrame with 891 rows and 2 columns. Remember, a DataFrame is 2-dimensional with both a row and column dimension.

For basic information on indexing, see the user guide section on *indexing and selecting data*.

How do I filter specific rows from a DataFrame?

I'm interested in the passengers older than 35 years.

```
In [12]: above_35 = titanic[titanic["Age"] > 35]

In [13]: above_35.head()
Out[13]: 
    PassengerId  Survived  Pclass                                               Name ␣
↪   Sex  ...  Parch    Ticket     Fare Cabin  Embarked
1             2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th... ␣
↪female  ...      0  PC 17599  71.2833   C85         C
6             7         0       1                           McCarthy, Mr. Timothy J ␣
↪  male  ...      0     17463  51.8625   E46         S
11           12         1       1                          Bonnell, Miss. Elizabeth ␣
↪female  ...      0    113783  26.5500  C103         S
```

(continues on next page)

(continued from previous page)

```
13           14         0       3                        Andersson, Mr. Anders Johan ␣
↪   male  ...      5    347082  31.2750   NaN         S
15           16         1       2                   Hewlett, Mrs. (Mary D Kingcome)  ␣
↪female   ...      0    248706  16.0000   NaN         S

[5 rows x 12 columns]
```

To select rows based on a conditional expression, use a condition inside the selection brackets `[]`.

The condition inside the selection brackets `titanic["Age"] > 35` checks for which rows the `Age` column has a value larger than 35:

```
In [14]: titanic["Age"] > 35
Out[14]: 
0      False
1       True
2      False
3      False
4      False
       ...  
886    False
887    False
888    False
889    False
890    False
Name: Age, Length: 891, dtype: bool
```

The output of the conditional expression (>, but also ==, ! =, <, <=,… would work) is actually a pandas `Series` of boolean values (either `True` or `False`) with the same number of rows as the original `DataFrame`. Such a `Series` of boolean values can be used to filter the `DataFrame` by putting it in between the selection brackets `[]`. Only rows for which the value is `True` will be selected.

We now from before that the original titanic `DataFrame` consists of 891 rows. Let's have a look at the amount of rows which satisfy the condition by checking the `shape` attribute of the resulting `DataFrame` `above_35`:

```
In [15]: above_35.shape
Out[15]: (217, 12)
```

I'm interested in the titanic passengers from cabin class 2 and 3.

```
In [16]: class_23 = titanic[titanic["Pclass"].isin([2, 3])]

In [17]: class_23.head()
Out[17]: 
   PassengerId  Survived  Pclass                            Name     Sex   Age  SibSp␣
↪  Parch            Ticket     Fare Cabin Embarked
0            1         0       3         Braund, Mr. Owen Harris    male  22.0      1␣
↪      0         A/5 21171   7.2500   NaN        S
2            3         1       3          Heikkinen, Miss. Laina  female  26.0      0␣
↪      0  STON/O2. 3101282   7.9250   NaN        S
4            5         0       3        Allen, Mr. William Henry    male  35.0      0␣
↪      0            373450   8.0500   NaN        S
5            6         0       3                Moran, Mr. James    male   NaN      0␣
↪      0            330877   8.4583   NaN        Q
7            8         0       3  Palsson, Master. Gosta Leonard    male   2.0      3␣
↪      1            349909  21.0750   NaN        S
```

Similar to the conditional expression, the *isin()* conditional function returns a `True` for each row the values are in the provided list. To filter the rows based on such a function, use the conditional function inside the selection brackets `[]`. In this case, the condition inside the selection brackets `titanic["Pclass"].isin([2, 3])` checks for which rows the `Pclass` column is either 2 or 3.

The above is equivalent to filtering by rows for which the class is either 2 or 3 and combining the two statements with an `|` (or) operator:

```
In [18]: class_23 = titanic[(titanic["Pclass"] == 2) | (titanic["Pclass"] == 3)]

In [19]: class_23.head()
Out[19]:
   PassengerId  Survived  Pclass                            Name     Sex   Age  SibSp␣
↪  Parch            Ticket     Fare Cabin Embarked
0            1         0       3         Braund, Mr. Owen Harris    male  22.0      1␣
↪      0         A/5 21171   7.2500   NaN        S
2            3         1       3          Heikkinen, Miss. Laina  female  26.0      0␣
↪      0  STON/O2. 3101282   7.9250   NaN        S
4            5         0       3        Allen, Mr. William Henry    male  35.0      0␣
↪      0            373450   8.0500   NaN        S
5            6         0       3                Moran, Mr. James    male   NaN      0␣
↪      0            330877   8.4583   NaN        Q
7            8         0       3  Palsson, Master. Gosta Leonard    male   2.0      3␣
↪      1            349909  21.0750   NaN        S
```

Note: When combining multiple conditional statements, each condition must be surrounded by parentheses `()`. Moreover, you can not use `or`/`and` but need to use the `or` operator `|` and the `and` operator `&`.

See the dedicated section in the user guide about *boolean indexing* or about the *isin function*.

I want to work with passenger data for which the age is known.

```
In [20]: age_no_na = titanic[titanic["Age"].notna()]

In [21]: age_no_na.head()
Out[21]:
   PassengerId  Survived  Pclass                                               Name  ␣
↪   Sex  ...  Parch            Ticket     Fare Cabin  Embarked
0            1         0       3                            Braund, Mr. Owen Harris  ␣
↪  male  ...      0         A/5 21171    7.2500   NaN           S
1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th... ␣
↪female   ...      0          PC 17599  71.2833   C85           C
2            3         1       3                             Heikkinen, Miss. Laina ␣
↪female   ...      0  STON/O2. 3101282   7.9250   NaN           S
3            4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel) ␣
↪female   ...      0            113803  53.1000  C123           S
4            5         0       3                           Allen, Mr. William Henry  ␣
↪  male  ...      0            373450    8.0500   NaN           S

[5 rows x 12 columns]
```

The *notna()* conditional function returns a `True` for each row the values are not an `Null` value. As such, this can be combined with the selection brackets `[]` to filter the data table.

You might wonder what actually changed, as the first 5 lines are still the same values. One way to verify is to check if the shape has changed:

```
In [22]: age_no_na.shape
Out[22]: (714, 12)
```

For more dedicated functions on missing values, see the user guide section about *handling missing data*.

How do I select specific rows and columns from a `DataFrame`?

I'm interested in the names of the passengers older than 35 years.

```
In [23]: adult_names = titanic.loc[titanic["Age"] > 35, "Name"]

In [24]: adult_names.head()
Out[24]:
1     Cumings, Mrs. John Bradley (Florence Briggs Th...
6                               McCarthy, Mr. Timothy J
11                             Bonnell, Miss. Elizabeth
13                          Andersson, Mr. Anders Johan
15                     Hewlett, Mrs. (Mary D Kingcome)
Name: Name, dtype: object
```

In this case, a subset of both rows and columns is made in one go and just using selection brackets `[]` is not sufficient anymore. The `loc`/`iloc` operators are required in front of the selection brackets `[]`. When using `loc`/`iloc`, the part before the comma is the rows you want, and the part after the comma is the columns you want to select.

When using the column names, row labels or a condition expression, use the `loc` operator in front of the selection brackets `[]`. For both the part before and after the comma, you can use a single label, a list of labels, a slice of labels, a conditional expression or a colon. Using a colon specificies you want to select all rows or columns.

I'm interested in rows 10 till 25 and columns 3 to 5.

```
In [25]: titanic.iloc[9:25, 2:5]
Out[25]:
    Pclass                                 Name     Sex
9        2  Nasser, Mrs. Nicholas (Adele Achem)  female
10       3      Sandstrom, Miss. Marguerite Rut  female
11       1             Bonnell, Miss. Elizabeth  female
12       3       Saundercock, Mr. William Henry    male
13       3          Andersson, Mr. Anders Johan    male
..     ...                                  ...     ...
20       2                 Fynney, Mr. Joseph J    male
21       2                Beesley, Mr. Lawrence    male
22       3          McGowan, Miss. Anna "Annie"  female
23       1         Sloper, Mr. William Thompson    male
24       3        Palsson, Miss. Torborg Danira  female

[16 rows x 3 columns]
```

Again, a subset of both rows and columns is made in one go and just using selection brackets `[]` is not sufficient anymore. When specifically interested in certain rows and/or columns based on their position in the table, use the `iloc` operator in front of the selection brackets `[]`.

When selecting specific rows and/or columns with `loc` or `iloc`, new values can be assigned to the selected data. For example, to assign the name `anonymous` to the first 3 elements of the third column:

```
In [26]: titanic.iloc[0:3, 3] = "anonymous"

In [27]: titanic.head()
Out[27]:
   PassengerId  Survived  Pclass                                          Name  ␣
↪Sex  ...  Parch            Ticket     Fare Cabin  Embarked
0            1         0       3                                     anonymous  ␣
↪male  ...      0         A/5 21171   7.2500   NaN         S
1            2         1       1                                     anonymous ␣
↪female  ...      0          PC 17599  71.2833   C85         C
2            3         1       3                                     anonymous ␣
↪female  ...      0  STON/O2. 3101282   7.9250   NaN         S
3            4         1       1  Futrelle, Mrs. Jacques Heath (Lily May Peel) ␣
↪female  ...      0            113803  53.1000  C123         S
4            5         0       3                      Allen, Mr. William Henry  ␣
↪male  ...      0            373450   8.0500   NaN         S

[5 rows x 12 columns]
```

See the user guide section on *different choices for indexing* to get more insight in the usage of `loc` and `iloc`.

- When selecting subsets of data, square brackets `[]` are used.
- Inside these brackets, you can use a single column/row label, a list of column/row labels, a slice of labels, a conditional expression or a colon.
- Select specific rows and/or columns using `loc` when using the row and column names
- Select specific rows and/or columns using `iloc` when using the positions in the table
- You can assign new values to a selection based on `loc`/`iloc`.

A full overview about indexing is provided in the user guide pages on *indexing and selecting data*.

```
In [1]: import pandas as pd

In [2]: import matplotlib.pyplot as plt
```

For this tutorial, air quality data about NO_2 is used, made available by openaq and using the py-openaq package. The `air_quality_no2.csv` data set provides NO_2 values for the measurement stations *FR04014*, *BETR801* and *London Westminster* in respectively Paris, Antwerp and London.

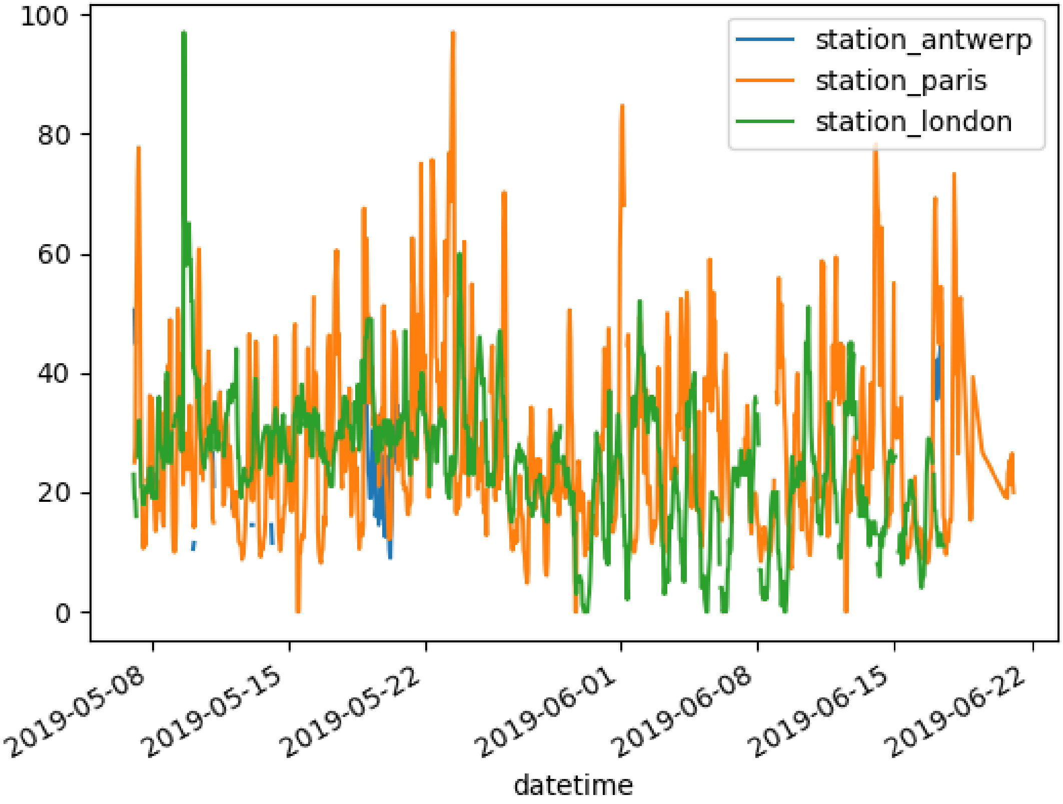
```
In [3]: air_quality = pd.read_csv("data/air_quality_no2.csv",
   ...:                           index_col=0, parse_dates=True)
   ...:

In [4]: air_quality.head()
Out[4]:
                     station_antwerp  station_paris  station_london
datetime
2019-05-07 02:00:00              NaN            NaN            23.0
2019-05-07 03:00:00             50.5           25.0            19.0
2019-05-07 04:00:00             45.0           27.7            19.0
2019-05-07 05:00:00              NaN           50.4            16.0
2019-05-07 06:00:00              NaN           61.9             NaN
```

Note: The usage of the `index_col` and `parse_dates` parameters of the `read_csv` function to define the first (0th) column as index of the resulting `DataFrame` and convert the dates in the column to `Timestamp` objects, respectively.

How to create plots in pandas?

I want a quick visual check of the data.

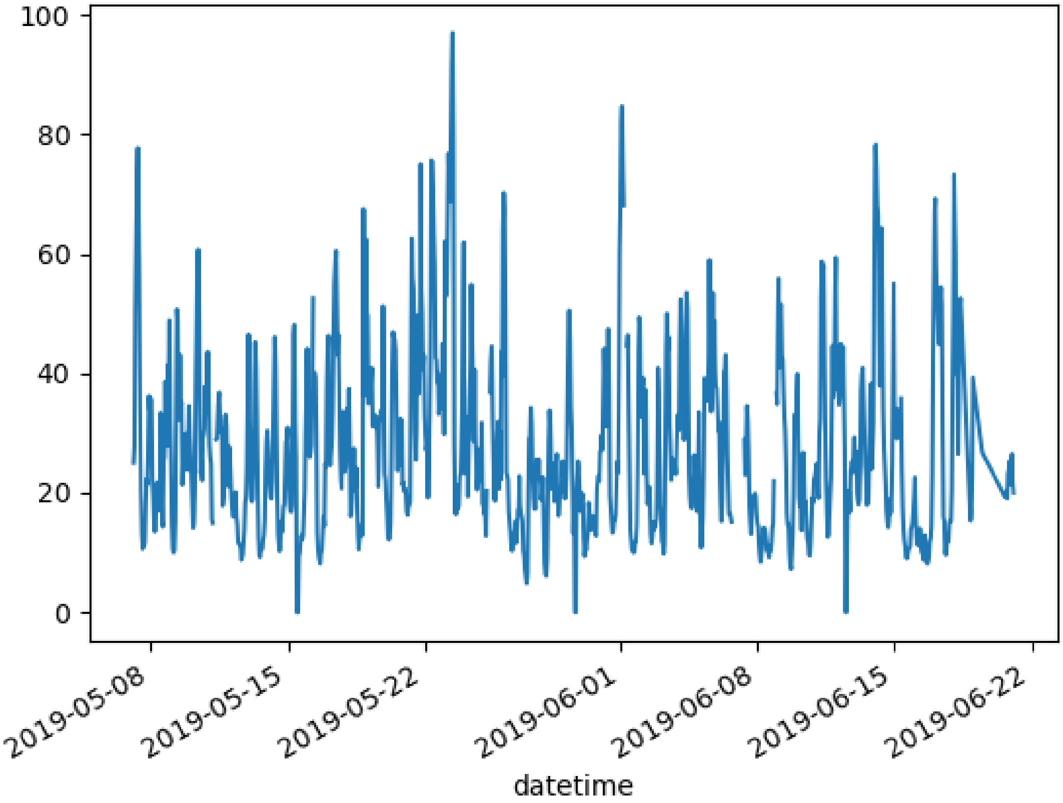
```
In [5]: air_quality.plot()
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7f534481f610>
```



With a DataFrame, pandas creates by default one line plot for each of the columns with numeric data.

I want to plot only the columns of the data table with the data from Paris.

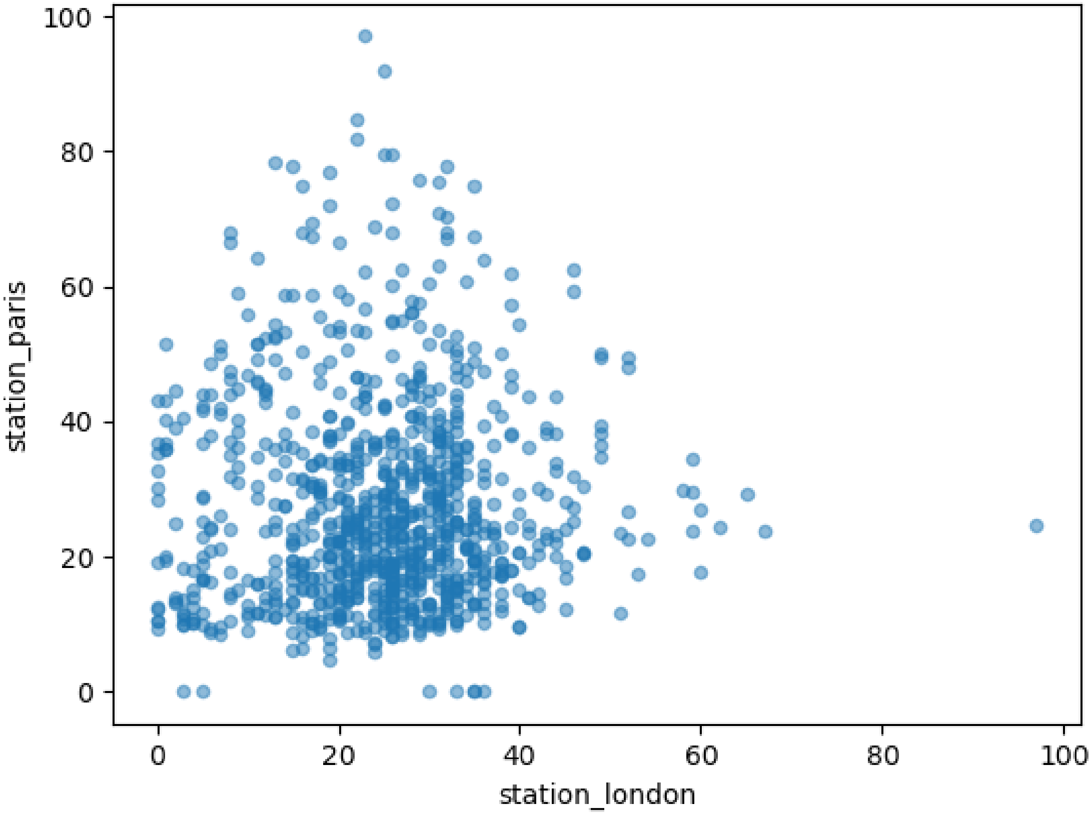
```
In [6]: air_quality["station_paris"].plot()
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f536afdf990>
```



To plot a specific column, use the selection method of the *subset data tutorial* in combination with the `plot()` method. Hence, the `plot()` method works on both `Series` and `DataFrame`.

I want to visually compare the $N0_2$ values measured in London versus Paris.

```
In [7]: air_quality.plot.scatter(x="station_london",
   ...:                          y="station_paris",
   ...:                          alpha=0.5)
   ...:
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f534481f710>
```



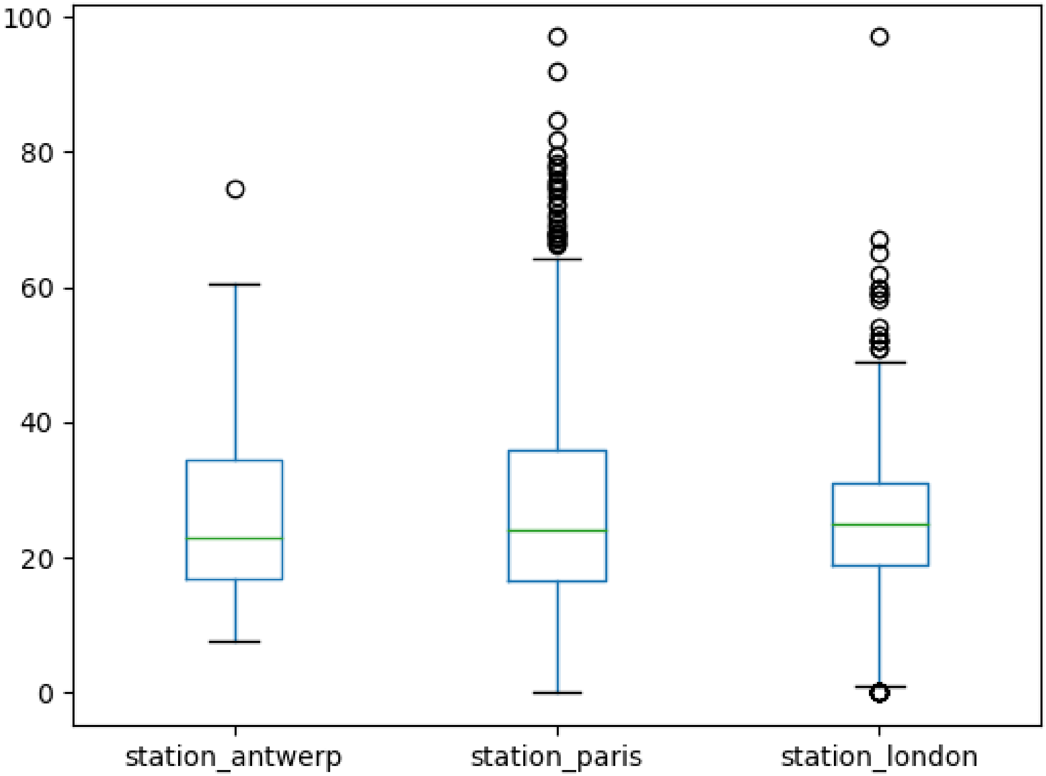
Apart from the default `line` plot when using the `plot` function, a number of alternatives are available to plot data. Let's use some standard Python to get an overview of the available plot methods:

```
In [8]: [method_name for method_name in dir(air_quality.plot)
   ...:     if not method_name.startswith("_")]
   ...:
Out[8]:
['area',
 'bar',
 'barh',
 'box',
 'density',
 'hexbin',
 'hist',
 'kde',
 'line',
 'pie',
 'scatter']
```

Note: In many development environments as well as ipython and jupyter notebook, use the TAB button to get an overview of the available methods, for example `air_quality.plot.` + TAB.

One of the options is *`DataFrame.plot.box()`*, which refers to a boxplot. The `box` method is applicable on the air quality example data:

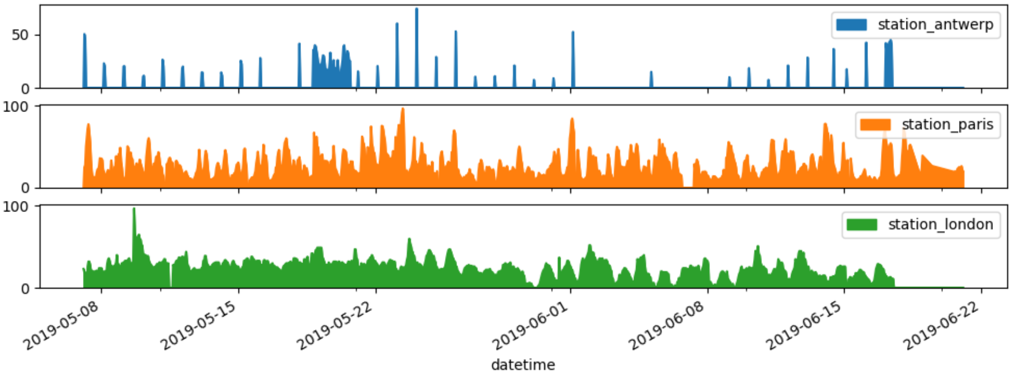
```
In [9]: air_quality.plot.box()
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5344381810>
```



For an introduction to plots other than the default line plot, see the user guide section about *supported plot styles*.

I want each of the columns in a separate subplot.

```
In [10]: axs = air_quality.plot.area(figsize=(12, 4), subplots=True)
```

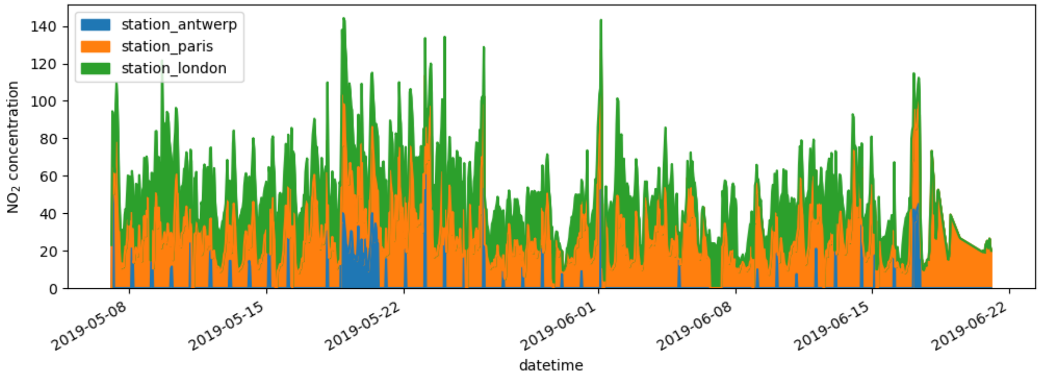


Separate subplots for each of the data columns is supported by the `subplots` argument of the `plot` functions. The builtin options available in each of the pandas plot functions that are worthwhile to have a look.

Some more formatting options are explained in the user guide section on *plot formatting*.

I want to further customize, extend or save the resulting plot.

```
In [11]: fig, axs = plt.subplots(figsize=(12, 4));

In [12]: air_quality.plot.area(ax=axs);

In [13]: axs.set_ylabel("NO$_2$ concentration");

In [14]: fig.savefig("no2_concentrations.png")
```



Each of the plot objects created by pandas are a matplotlib object. As Matplotlib provides plenty of options to customize plots, making the link between pandas and Matplotlib explicit enables all the power of matplotlib to the plot. This strategy is applied in the previous example:

```
fig, axs = plt.subplots(figsize=(12, 4))          # Create an empty matplotlib Figure␣
↪and Axes
air_quality.plot.area(ax=axs)                     # Use pandas to put the area plot on␣
↪the prepared Figure/Axes
axs.set_ylabel("NO$_2$ concentration")            # Do any matplotlib customization you␣
↪like
fig.savefig("no2_concentrations.png")             # Save the Figure/Axes using the␣
↪existing matplotlib method.
```

- The `.plot.*` methods are applicable on both Series and DataFrames
- By default, each of the columns is plotted as a different element (line, boxplot,…)
- Any plot created by pandas is a Matplotlib object.

A full overview of plotting in pandas is provided in the *visualization pages*.

```
In [1]: import pandas as pd
```

For this tutorial, air quality data about NO_2 is used, made available by openaq and using the py-openaq package. The `air_quality_no2.csv` data set provides NO_2 values for the measurement stations *FR04014*, *BETR801* and *London Westminster* in respectively Paris, Antwerp and London.

```
In [2]: air_quality = pd.read_csv("data/air_quality_no2.csv",
   ...:                           index_col=0, parse_dates=True)
   ...:

In [3]: air_quality.head()
```

(continues on next page)

(continued from previous page)

```
Out[3]:
                     station_antwerp  station_paris  station_london
datetime
2019-05-07 02:00:00              NaN            NaN            23.0
2019-05-07 03:00:00             50.5           25.0            19.0
2019-05-07 04:00:00             45.0           27.7            19.0
2019-05-07 05:00:00              NaN           50.4            16.0
2019-05-07 06:00:00              NaN           61.9             NaN
```

How to create new columns derived from existing columns?

I want to express the NO_2 concentration of the station in London in mg/m^3

(*If we assume temperature of 25 degrees Celsius and pressure of 1013 hPa, the conversion factor is 1.882*)

```
In [4]: air_quality["london_mg_per_cubic"] = air_quality["station_london"] * 1.882

In [5]: air_quality.head()
Out[5]:
                     station_antwerp  station_paris  station_london  london_mg_per_
↪cubic
datetime                                                                           ␣
↪
2019-05-07 02:00:00              NaN            NaN            23.0             43.
↪286
2019-05-07 03:00:00             50.5           25.0            19.0             35.
↪758
2019-05-07 04:00:00             45.0           27.7            19.0             35.
↪758
2019-05-07 05:00:00              NaN           50.4            16.0             30.
↪112
2019-05-07 06:00:00              NaN           61.9             NaN               ␣
↪NaN
```

To create a new column, use the `[]` brackets with the new column name at the left side of the assignment.

Note: The calculation of the values is done **element_wise**. This means all values in the given column are multiplied by the value 1.882 at once. You do not need to use a loop to iterate each of the rows!

I want to check the ratio of the values in Paris versus Antwerp and save the result in a new column

```
In [6]: air_quality["ratio_paris_antwerp"] = \
   ...:     air_quality["station_paris"] / air_quality["station_antwerp"]
   ...:

In [7]: air_quality.head()
Out[7]:
                     station_antwerp  station_paris  station_london  london_mg_per_
↪cubic  ratio_paris_antwerp
datetime                                                                           ␣
↪
```

(continues on next page)

(continued from previous page)

```
2019-05-07 02:00:00              NaN            NaN            23.0               43.
↪286                   NaN
2019-05-07 03:00:00             50.5           25.0            19.0               35.
↪758              0.495050
2019-05-07 04:00:00             45.0           27.7            19.0               35.
↪758              0.615556
2019-05-07 05:00:00              NaN           50.4            16.0               30.
↪112                   NaN
2019-05-07 06:00:00              NaN           61.9             NaN                  ␣
↪NaN                   NaN
```

The calculation is again element-wise, so the / is applied *for the values in each row*.

Also other mathematical operators (+, -, *, /) or logical operators (<, >, =,…) work element wise. The latter was already used in the *subset data tutorial* to filter rows of a table using a conditional expression.

I want to rename the data columns to the corresponding station identifiers used by openAQ

```
In [8]: air_quality_renamed = air_quality.rename(
   ...:     columns={"station_antwerp": "BETR801",
   ...:              "station_paris": "FR04014",
   ...:              "station_london": "London Westminster"})
   ...:
```

```
In [9]: air_quality_renamed.head()
Out[9]:
                     BETR801  FR04014  London Westminster  london_mg_per_cubic  ratio_
↪paris_antwerp
datetime                                                                              ␣
↪
2019-05-07 02:00:00      NaN      NaN                23.0               43.286        ␣
↪          NaN
2019-05-07 03:00:00     50.5     25.0                19.0               35.758        ␣
↪     0.495050
2019-05-07 04:00:00     45.0     27.7                19.0               35.758        ␣
↪     0.615556
2019-05-07 05:00:00      NaN     50.4                16.0               30.112        ␣
↪          NaN
2019-05-07 06:00:00      NaN     61.9                 NaN                  NaN        ␣
↪          NaN
```

The `rename()` function can be used for both row labels and column labels. Provide a dictionary with the keys the current names and the values the new names to update the corresponding names.

The mapping should not be restricted to fixed names only, but can be a mapping function as well. For example, converting the column names to lowercase letters can be done using a function as well:

```
In [10]: air_quality_renamed = air_quality_renamed.rename(columns=str.lower)

In [11]: air_quality_renamed.head()
Out[11]:
                     betr801  fr04014  london westminster  london_mg_per_cubic  ratio_
↪paris_antwerp
datetime                                                                              ␣
↪
2019-05-07 02:00:00      NaN      NaN                23.0               43.286        ␣
↪          NaN
```

(continues on next page)

(continued from previous page)

```
2019-05-07 03:00:00     50.5     25.0                19.0               35.758        ␣
↪      0.495050
2019-05-07 04:00:00     45.0     27.7                19.0               35.758        ␣
↪      0.615556
2019-05-07 05:00:00      NaN     50.4                16.0               30.112        ␣
↪           NaN
2019-05-07 06:00:00      NaN     61.9                 NaN                  NaN        ␣
↪           NaN
```

Details about column or row label renaming is provided in the user guide section on *renaming labels*.

- Create a new column by assigning the output to the DataFrame with a new column name in between the `[]`.
- Operations are element-wise, no need to loop over rows.
- Use `rename` with a dictionary or function to rename row labels or column names.

The user guide contains a separate section on *column addition and deletion*.

```
In [1]: import pandas as pd
```

This tutorial uses the titanic data set, stored as CSV. The data consists of the following data columns:

- PassengerId: Id of every passenger.
- Survived: This feature have value 0 and 1. 0 for not survived and 1 for survived.
- Pclass: There are 3 classes: Class 1, Class 2 and Class 3.
- Name: Name of passenger.
- Sex: Gender of passenger.
- Age: Age of passenger.
- SibSp: Indication that passenger have siblings and spouse.
- Parch: Whether a passenger is alone or have family.
- Ticket: Ticket number of passenger.
- Fare: Indicating the fare.
- Cabin: The cabin of passenger.
- Embarked: The embarked category.

```
In [2]: titanic = pd.read_csv("data/titanic.csv")

In [3]: titanic.head()
Out[3]:
   PassengerId  Survived  Pclass                                               Name   ␣
↪   Sex  ...  Parch            Ticket     Fare Cabin  Embarked
0            1         0       3                            Braund, Mr. Owen Harris   ␣
↪  male  ...      0         A/5 21171   7.2500   NaN         S
1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th... ␣
↪female   ...      0          PC 17599  71.2833   C85         C
2            3         1       3                             Heikkinen, Miss. Laina  ␣
↪female   ...      0  STON/O2. 3101282   7.9250   NaN         S
3            4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)  ␣
↪female   ...      0            113803  53.1000  C123         S
4            5         0       3                           Allen, Mr. William Henry   ␣
↪  male  ...      0            373450   8.0500   NaN         S
```

(continues on next page)

(continued from previous page)

```

[5 rows x 12 columns]
```

How to calculate summary statistics?

Aggregating statistics

What is the average age of the titanic passengers?

```
In [4]: titanic["Age"].mean()
Out[4]: 29.69911764705882
```

Different statistics are available and can be applied to columns with numerical data. Operations in general exclude missing data and operate across rows by default.

What is the median age and ticket fare price of the titanic passengers?

```
In [5]: titanic[["Age", "Fare"]].median()
Out[5]: 
Age     28.0000
Fare    14.4542
dtype: float64
```

The statistic applied to multiple columns of a `DataFrame` (the selection of two columns return a `DataFrame`, see the *subset data tutorial*) is calculated for each numeric column.

The aggregating statistic can be calculated for multiple columns at the same time. Remember the `describe` function from *first tutorial* tutorial?

```
In [6]: titanic[["Age", "Fare"]].describe()
Out[6]: 
              Age        Fare
count  714.000000  891.000000
mean    29.699118   32.204208
std     14.526497   49.693429
min      0.420000    0.000000
25%     20.125000    7.910400
50%     28.000000   14.454200
75%     38.000000   31.000000
max     80.000000  512.329200
```

Instead of the predefined statistics, specific combinations of aggregating statistics for given columns can be defined using the *`DataFrame.agg()`* method:

```
In [7]: titanic.agg({'Age': ['min', 'max', 'median', 'skew'],
   ...:              'Fare': ['min', 'max', 'median', 'mean']})
   ...: 
Out[7]: 
              Age        Fare
max     80.000000  512.329200
mean          NaN   32.204208
```

(continues on next page)

(continued from previous page)

```
median  28.000000   14.454200
min      0.420000    0.000000
skew     0.389108         NaN
```

Details about descriptive statistics are provided in the user guide section on *descriptive statistics*.

Aggregating statistics grouped by category

What is the average age for male versus female titanic passengers?

```
In [8]: titanic[["Sex", "Age"]].groupby("Sex").mean()
Out[8]:
              Age
Sex
female  27.915709
male    30.726645
```

As our interest is the average age for each gender, a subselection on these two columns is made first: `titanic[["Sex", "Age"]]`. Next, the *groupby()* method is applied on the `Sex` column to make a group per category. The average age *for each gender* is calculated and returned.

Calculating a given statistic (e.g. `mean` age) *for each category in a column* (e.g. male/female in the `Sex` column) is a common pattern. The `groupby` method is used to support this type of operations. More general, this fits in the more general `split-apply-combine` pattern:

- **Split** the data into groups
- **Apply** a function to each group independently
- **Combine** the results into a data structure

The apply and combine steps are typically done together in pandas.

In the previous example, we explicitly selected the 2 columns first. If not, the `mean` method is applied to each column containing numerical columns:

```
In [9]: titanic.groupby("Sex").mean()
Out[9]:
        PassengerId  Survived    Pclass        Age     SibSp     Parch       Fare
Sex
female   431.028662  0.742038  2.159236  27.915709  0.694268  0.649682  44.479818
male     454.147314  0.188908  2.389948  30.726645  0.429809  0.235702  25.523893
```

It does not make much sense to get the average value of the `Pclass`. if we are only interested in the average age for each gender, the selection of columns (rectangular brackets `[]` as usual) is supported on the grouped data as well:

```
In [10]: titanic.groupby("Sex")["Age"].mean()
Out[10]:
Sex
female    27.915709
male      30.726645
Name: Age, dtype: float64
```

Note: The *Pclass* column contains numerical data but actually represents 3 categories (or factors) with respectively the labels '1', '2' and '3'. Calculating statistics on these does not make much sense. Therefore, pandas provides a `Categorical` data type to handle this type of data. More information is provided in the user guide *Categorical data* section.

What is the mean ticket fare price for each of the sex and cabin class combinations?

```
In [11]: titanic.groupby(["Sex", "Pclass"])["Fare"].mean()
Out[11]:
Sex     Pclass
female  1         106.125798
        2          21.970121
        3          16.118810
male    1          67.226127
        2          19.741782
        3          12.661633
Name: Fare, dtype: float64
```

Grouping can be done by multiple columns at the same time. Provide the column names as a list to the *`groupby()`* method.

A full description on the split-apply-combine approach is provided in the user guide section on *groupby operations*.

Count number of records by category

What is the number of passengers in each of the cabin classes?

```
In [12]: titanic["Pclass"].value_counts()
Out[12]:
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

The *`value_counts()`* method counts the number of records for each category in a column.

The function is a shortcut, as it is actually a groupby operation in combination with counting of the number of records within each group:

```
In [13]: titanic.groupby("Pclass")["Pclass"].count()
Out[13]:
Pclass
1    216
2    184
3    491
Name: Pclass, dtype: int64
```

Note: Both `size` and `count` can be used in combination with `groupby`. Whereas `size` includes `NaN` values and just provides the number of rows (size of the table), `count` excludes the missing values. In the `value_counts` method, use the `dropna` argument to include or exclude the `NaN` values.

The user guide has a dedicated section on `value_counts` , see page on *discretization*.

- Aggregation statistics can be calculated on entire columns or rows
- `groupby` provides the power of the *split-apply-combine* pattern
- `value_counts` is a convenient shortcut to count the number of entries in each category of a variable

A full description on the split-apply-combine approach is provided in the user guide pages about *groupby operations*.

```
In [1]: import pandas as pd
```

This tutorial uses the titanic data set, stored as CSV. The data consists of the following data columns:

- PassengerId: Id of every passenger.
- Survived: This feature have value 0 and 1. 0 for not survived and 1 for survived.
- Pclass: There are 3 classes: Class 1, Class 2 and Class 3.
- Name: Name of passenger.
- Sex: Gender of passenger.
- Age: Age of passenger.
- SibSp: Indication that passenger have siblings and spouse.
- Parch: Whether a passenger is alone or have family.
- Ticket: Ticket number of passenger.
- Fare: Indicating the fare.
- Cabin: The cabin of passenger.
- Embarked: The embarked category.

```
In [2]: titanic = pd.read_csv("data/titanic.csv")

In [3]: titanic.head()
Out[3]:
   PassengerId  Survived  Pclass                                               Name  ␣
↪  Sex  ...  Parch            Ticket     Fare Cabin  Embarked
0            1         0       3                            Braund, Mr. Owen Harris  ␣
↪ male  ...      0         A/5 21171   7.2500   NaN         S
1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th... ␣
↪female  ...      0          PC 17599  71.2833   C85         C
2            3         1       3                             Heikkinen, Miss. Laina ␣
↪female  ...      0  STON/O2. 3101282   7.9250   NaN         S
3            4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel) ␣
↪female  ...      0            113803  53.1000  C123         S
4            5         0       3                           Allen, Mr. William Henry  ␣
↪ male  ...      0            373450   8.0500   NaN         S

[5 rows x 12 columns]
```

This tutorial uses air quality data about NO_2 and Particulate matter less than 2.5 micrometers, made available by openaq and using the py-openaq package. The `air_quality_long.csv` data set provides NO_2 and PM_{25} values for the measurement stations *FR04014*, *BETR801* and *London Westminster* in respectively Paris, Antwerp and London.

The air-quality data set has the following columns:

- city: city where the sensor is used, either Paris, Antwerp or London
- country: country where the sensor is used, either FR, BE or GB