Introduction to pandas

TOC 8 minutes reading

In practice, data is often stored in the form of a table, for example, an Excel spreadsheet, a CSV file, or an SQL database. Imagine that you need to analyze this tabular data and get some useful insights from it. Let's think of the task you will need to perform.

First, you will need to load data from different formats preserving its tabular structure and probably join several tables together. Then, to perform the actual data analysis, you will definitely want to access different columns, rows, and cells of the tables, compute some overall statistics, create pivot tables and maybe even make basic plots. Is there a tool in Python that combines all these functionalities? The answer is yes!

This topic will introduce you to `pandas`, a powerful open-source library for data manipulation and analysis. You will learn how to install the library and get an idea of its main functionality.

Installing pandas

`pandas` is not included in the standard Python library, so you might

need to install it separately, for example, using `pip`. Type the following in your command line:

```
pip install pandas
```

Note that `pandas` is built on top of `NumPy` which will be installed as well. Besides, there are many optional dependencies. For instance, if you want to use `pandas` data visualization functionality, you need to install `matplotlib`, a plotting library in Python. You can install those libraries using `pip` as well.

Once the installation is complete, you will be able to import it in your code. Since `pandas` is quite a long name, it is commonly abbreviated and imported as `pd`:

```
import pandas as pd
```

Note that new versions of pandas are released once in a while, with new functionalities and fixed bugs. So it would be a good idea to keep an eye on the updates. You can easily upgrade the version of pandas on your machine with this command: `pip install --upgrade pandas`.

Inside pandas

As stated in its guidelines, `pandas` aims to become "the most powerful and flexible open-source data analysis/manipulation tool available in any language".

The name `pandas` is derived from **'panel data'**, a term that is used in statistics and econometrics to refer to data sets containing observations over multiple time periods for the same individuals. This library will be helpful if you are working with tabular data, such as data stored in spreadsheets or databases. Apart from that, `pandas` offers great support for time series and provides extensive functionality for working with dates, times, and time-indexed data.

With the help of `pandas`, one can easily perform the most typical data

processing steps. In particular, the package makes it convenient to load and save data, as it supports out-of-the-box integration with many commonly-used tabular formats such as `.csv`, `.xlsx`, as well as SQL databases.

Let's look at what else we can get from pandas!

- Intuitive merging and joining of data sets allow for easily combining data from different sources, while flexible reshaping tools help construct statistical data summaries.
- With `pandas` we can edit, sort, reshape and explore tabular data. For example, you could get all unique values from a column just with one command.

- Missing values in the data are represented as NaN and can be easily handled, for example, they can be replaced by some value, using built-in functionality.
- If you install `matplotlib`, a Python plotting library, you can use the `pandas` built-in plotting functionality to make a basic plot from your data to better understand it.

- You can get basic statistical information about your data with literally one line of code.
- It can be integrated with other libraries for machine learning, such as `sklearn`.

Finally, `pandas` is open-source software, which makes it very popular in both academic and commercial domains. To discover the full potential of `pandas`, take a look at the documentation.

Data structures in pandas

The two data structures of `pandas` are `Series` (1D) and `DataFrame` (2D). You will get more familiar with them in the dedicated topics — we'll just provide an overview.

`Series` is a one-dimensional array that stores elements of the same data type.

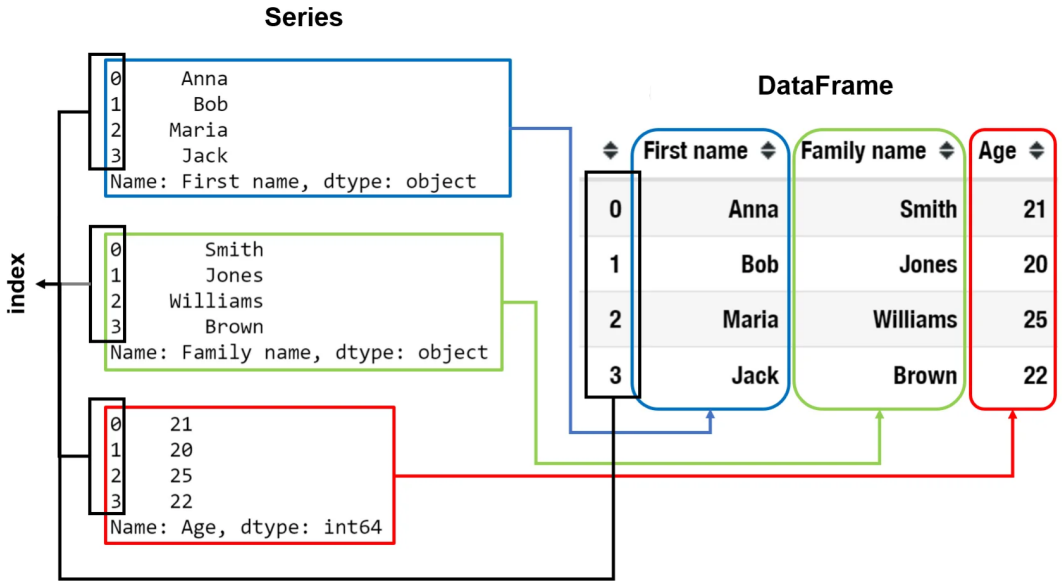
Each element stored in a `Series` is associated with a label called

index. By default, this index is just a sequence (0, 1, 2, ...) . However, you can use any custom values. For example, when analyzing time series, timestamps are typically set as indexes.

Indexes, as well as automatic and explicit data alignment based on them, are the core of `pandas`.

`DataFrame`, in turn, is a two-dimensional data structure that is used to represent tabular data with columns of potentially different data types. You can see `DataFrame` as a table, each column of which is a `Series` object. In other words, `DataFrame` is a container for `Series`, while `Series` is a container for scalars.

This is illustrated with the example below. The three `Series` objects store names, surnames, and ages of students respectively, while the `DataFrame` combines this information in a single table.



Note that each row in the `DataFrame` is also associated with an index. It is worth mentioning that even though `DataFrame` is a 2D data structure, one can still use it to represent higher-dimensional data by using the so-called **multi-index**, that is, assigning several indexes, or labels, to each row. However, this is rarely used in practice.

Tables represented as `DataFrame` slightly differ from spreadsheets to optimize computations. `Series` (= columns of a `DataFrame`) can store data of one type only to perform operations on them faster (`Series` uses NumPy arrays under the hood, and the same data type

limitation is imposed due to computation optimization).

It is also not possible to add elements to `Series` (that is, addition is possible, but a new object will be created instead of modifying the original `Series` in place), which helps `pandas` efficiently store `Series` objects in memory. When it comes to removing the values from a `Series`, by default, a new object will be created, but it's also possible to modify the existing object when removing the values (although, in both cases of a new object and changing the object in place, the object copy is generated anyway).

Note that since `pandas` is designed to preserve data immutability, it is generally not recommended to perform operations in place, because modifying the original object instead of creating a new object might lead to unexpected results and issues with debugging the code.

Conclusion

Here is what you should know about `pandas`:

- `pandas` is a flexible tool for data analysis.
- `pandas` is a perfect tool to work with heterogeneous data.
- There are two data structures in `pandas`: `Series` (1D) and `DataFrame` (2D).
- `Series` stores values of the same data type, while columns in a `DataFrame` can be of different types.

Read more on this topic in Exploring Pandas Library for Python on Hyperskill Blog.

Related topics

Topic prerequisites:

1. Importing modules
2. Type casting
3. Loop control: break, continue, pass