Theory: DataFrame

23 minutes 0 / 5 problems solved

You are already familiar with `Series`, a one-dimensional data structure in `pandas`. In this topic, you will learn about another key `pandas` data structure, the `DataFrame`.

Don't forget to import the `pandas` library:

```
1 import pandas as pd
```

§1. What is DataFrame

`DataFrame` is a table with columns. Each element of a `Series`, each row of a `DataFrame` is labeled with an index.

Here is an example of a `DataFrame` object `students` that stores information about four students:

	First name	Family name	Age
0	Anna	Smith	21
1	Bob	Jones	20
2	Maria	Williams	25
3	Jack	Brown	22

This `DataFrame` has three columns, namely 'First name', 'Family name', and 'Age'. The four rows are labeled with indexes 0, 1, 2, 3.

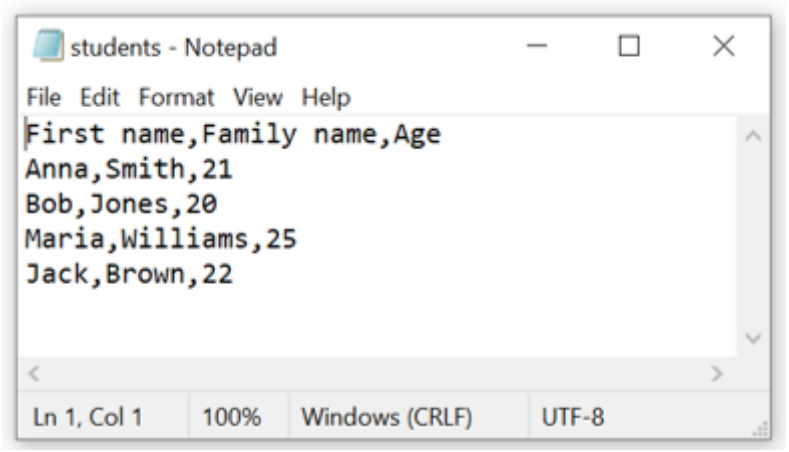
Alright, so how to create it?

§2. Creating a DataFrame: reading data from a file

Often you want to use the data from a file that is stored on your computer. `pandas` has functions that allow you to do it.

One of the most popular text formats is .csv, which stands for comma-separated values. This format can store tabular data; each row in a file represents a row in a table, and values corresponding to different columns are separated by commas.

Suppose the data about the students is stored in a students.csv file:



To transfer a `students` `DataFrame`, you can use a `read_csv()` function from `pandas`. This function takes the path to the file and some additional arguments that can be helpful to read the data correctly.

If we want to read the file as it is, we can simply write:

Current topic:

DataFrame

Topic depends on:

× Series

Table of contents:

```
students = pd.read_csv('students.csv')
students
```

	First name	Family name	Age
0	Anna	Smith	21
1	Bob	Jones	20
2	Maria	Williams	25
3	Jack	Brown	22

We won't list all additional parameters that `read_csv` can take here, but here are the most essential ones:

- *sep* — delimiter that is used with (default ',').
- *header* — row number that stores the column headers. By default, `pandas` tries to infer them from the first row.
- *names* — a list of column names. If you want to use other column names, set *header=0* and pass a list of new column names with *names*.
- *index_col* — columns in your file that are used as row labels of the DataFrame. It's set to None by default and the row numbers are used as indexes.
- *usecols* — a list of column numbers or column names to be read. By default, the dataframe reads every column.

Let's read the same file again, but this time we only use the first and the last column, giving them different names:

```
students = pd.read_csv('students.csv', usecols=[0,2],header=0, names=['name', 'age'])
students
```

	name	age
0	Anna	21
1	Bob	20
2	Maria	25
3	Jack	22

You can use the `read_excel()` function to read the data from a spreadsheet. It has a similar interface but it reads .xlsx files. To read a JSON file, use `read_json()` instead.

§3. Creating a DataFrame from other data structures

It's also possible to convert other data structures, e.g. dictionaries, lists, or numpy arrays, to a `DataFrame` object. You need to pass the data to the `DataFrame` constructor.

For instance, suppose you have a nested list containing information about students:

```
students_list = [['Anna', 'Smith', 21],
                 ['Bob', 'Jones', 20],
                 ['Maria', 'Williams', 25],
                 ['Jack', 'Brown', 22]]
```

We can easily turn it into a `DataFrame`:

```
students = pd.DataFrame(students_list, columns = ['First name', 'Family Name', 'Age'])
students
```

	First name	Family name	Age
0	Anna	Smith	21
1	Bob	Jones	20
2	Maria	Williams	25
3	Jack	Brown	22

We could additionally specify the `index` instead of the default 0, 1, 2, ... with the index argument. Let's try that:

```
students_number = [100, 200, 300, 400]
students = pd.DataFrame(students_list,
                        columns = ['First name', 'Family Name', 'Age'],
                        index = students_number)
students
```

	First name	Family Name	Age
100	Anna	Smith	21
200	Bob	Jones	20
300	Maria	Williams	25
400	Jack	Brown	22

Creating a `DataFrame` from a nested dictionary, index and column names will be automatically inferred from the dictionary keys. Take a look at the example:

```
# This is a nested dictionary representing the students table
students_dict = {'First name': {100: 'Anna',
                                200: 'Bob',
                                300: 'Maria',
                                400: 'Jack'},

                 'Family name': {100: 'Smith',
                                 200: 'Jones',
                                 300: 'Williams',

                                 400: 'Brown'},

                 'Age': {100: 21,

                         200: 20,

                         300: 25,

                         400: 22}}


students = pd.DataFrame(students_dict)

students
```

	First name	Family Name	Age
100	Anna	Smith	21
200	Bob	Jones	20
300	Maria	Williams	25
400	Jack	Brown	22

§4. First glance at the data

Imagine that you've just loaded your data into a `DataFrame` and you can't wait to start exploring it.

To check how many rows and columns a frame has, you can access the `shape` attribute. It contains a tuple with two values, the dimensions along the two axes. For example, in our `students` `DataFrame`, there're four rows and three columns:

```
students.shape
# (4, 3)
```

You might also want to take a look at your data. The `DataFrame` may be too large to print it out. In this case, use `head()` and `tail()` methods. They will print the first or the last five rows of the `DataFrame` respectively. If you want a different number of rows displayed, just specify it in the brackets. Let's print out just 2 first rows:

```
students.head(2)
```

	First name	Family name	Age
0	Anna	Smith	21
1	Bob	Jones	20

You can also access each of the `DataFrame`'s columns separately by putting the name of the column in the square brackets after the name of the `DataFrame`. Note that each column of a `DataFrame` is a `Series`:

```
students['Age']
# 0    21
# 1    20
# 2    25
# 3    22
# Name: Age, dtype: int64
```

If you need to access several columns at once, just put their names on a list. Let's take a look at the first and last columns only. Note that a resulting table is a `DataFrame` object:

```
students[['First name', 'Age']]
```

	First name	Age
1	Anna	21
2	Bob	20
3	Maria	25
4	Jack	22

Note that if you want to get a single column from a DataFrame as another DataFrame object but not Series, you should put the name of the columns in double square brackets:

```
students[['Age']]
```

	Age
1	21
2	20
3	25
4	22

If you need to access the data in a particular column itself without the indexes, you can use the `values` attribute. Then, you'll get a NumPy array instead of a `Series` or a `DataFrame`:

```
students['Age'].values
# array([21, 20, 25, 22], dtype=int64)

students[['First name', 'Age']].values
# array([['Anna', 21],
#        ['Bob', 20],
#        ['Maria', 25],
#        ['Jack', 22]], dtype=object)
```

§5. Saving a DataFrame to a file

Once you are done with a `DataFrame`, you can easily save it to a file on your computer. Just like with reading data from different file formats, `pandas` implements methods to save the `DataFrame` in various formats: `to_csv`, `to_excel` and `to_json`. They are alike so let's write a table in a .csv file.

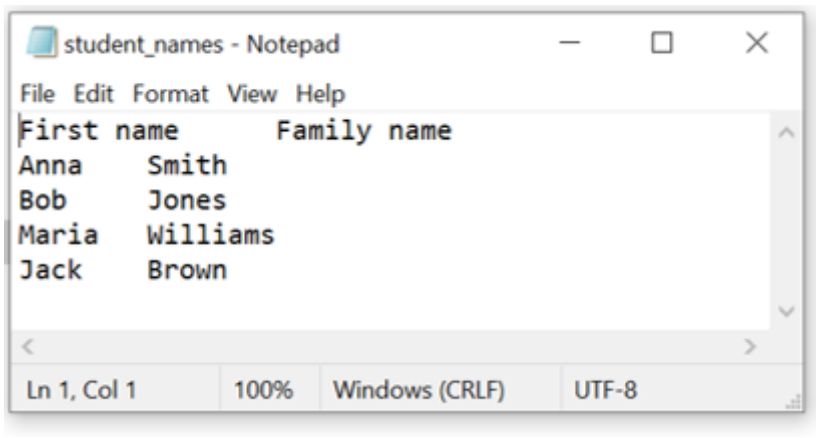
`to_csv()` method can take a lot of arguments, but the most important ones are the following:

- path to the file where the `DataFrame` should be stored.
- *sep* — delimiter to use (default ',')
- *header* — stores the column names (default `True`). You can also pass a list of column names different than the ones that the `DataFrame` has.
- *index* — whether to write index (default `True`)
- *columns* — columns to write. By default, all columns are used, but you can pass a list of column names to use only part of them.

If we want to write the first and the second columns of the `students` `DataFrame` to the *student_names.csv* file, without index and with tabulation as a delimiter. This can be done as follows:

```
students.to_csv('student_names.csv', sep='\t', columns=
['First name', 'Family name'], index=False)
```

Here is the resulting file:



§6. Conclusions

- `DataFrame` is a two-dimensional data structure. It's useful to store tabular data with columns of different data types.
- Row names in a `DataFrame` are called indexes.
- Each column of a `DataFrame` is a `Series`.

- A `DataFrame` can be created by reading data from a file (e.g., .csv), or by converting other data structures into a `DataFrame`.
- `head()` and `tail()` methods allow one to see the first and the last couple of rows of a `DataFrame`.

Report a typo

16 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

This content was created 8 months ago and updated 13 days ago. Share your feedback below in comments to help us improve it!

Comments (1) Hints (0) Useful links (0) Show discussion