Dataframe manipulation

Daniel García Hernández

Master Big Data

February 20, 2020

: Overview

1. Why to learn dataframe manipulation
2. Rearranging dataframes
3. Reshaping dataframes
4. Join, merge and concat

Why to learn dataframe manipulation: mastering pandas

It is rare that the data you have is the data you need -even less the data you deserve!

- In this situation, you need to apply certain modification to the available data you have, and adapt it to your needs
- Using `pandas` is a great way of handling datasets, as you already know, and it contains several functions and methods that, when combined, can extract a lot of information that was not present in the original dataset

In order to perform a good exploratory data analysis (EDA) you'll need to rearrange data, join different datasets, and switch from long to wide data and viceversa.

Rearranging dataframes: From NumPy to pandas

Since `pandas` has `NumPy` under the hood, and we've already learned how to rearrange and reshape `Numpy` arrays and how to deal with `pandas` DataFrames, it's about time to put it all together!

Rearranging dataframes: Rearranging dataframes (1)

Rearranging columns: every change in the `df.columns` object will be reflected in the original dataframe

- We can mutate the `df.columns` by sorting it or slicing it, and the dataframe will do that operation for every row in the affected columns:
- Sort the list with `df.columns`, mutate it with a list comprehension, filter it with `lambda` functions...
- Keep only some columns by using `df[columns_to_keep]`, where `columns_to_keep` is a list

Rearranging dataframes: Rearranging dataframes (2)

Rearranging rows: slicing, sorting

- Slice dataframe on the rows dimension in a similar way to `NumPy` arrays: `df.iloc[ri:rf, :]`
- Sort rows according to a certain column values by using `df.sort_values(by=column_name)`

Rearranging dataframes: Rearranging dataframes (3)

Changing the index:

- We can always set one of the columns as `df.index` by using `df.set_index(column_to_index`
- The df can also be sorted according to the index by using we can always set one of the columns as `df.index` by using `df.sort_index()`

Reshaping dataframes: Grouping

`df.groupby()` allows us to create a new dataframe based on the original, but grouping the info according to a categorical variable.

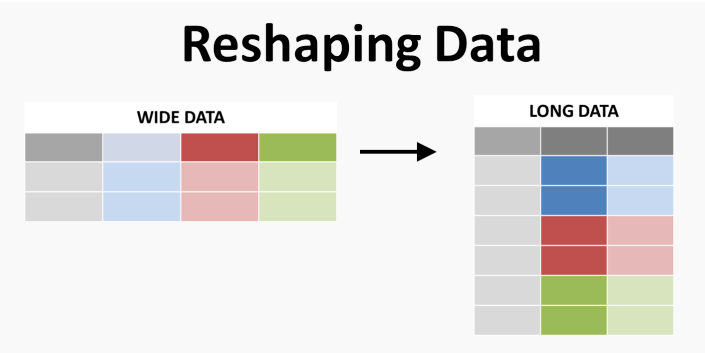
To use it, you have to specify the column(s) on which you want to group by.

Once the `groupby()` object is created, you can use methods like `.sum()`, `.mean()`, etc in order to get the sum, mean, of values for each group if the column accepts such operation.

Using `.agg()` after `groupby()` allows you to perform specific operations on the specified variables for each group

Reshaping dataframes: Wide vs. long data

Wide data is a format of DataFrames where the shape could be reduced in the columns by including a new column that would include the value, and therefore reducing the horizontal dimension[1]

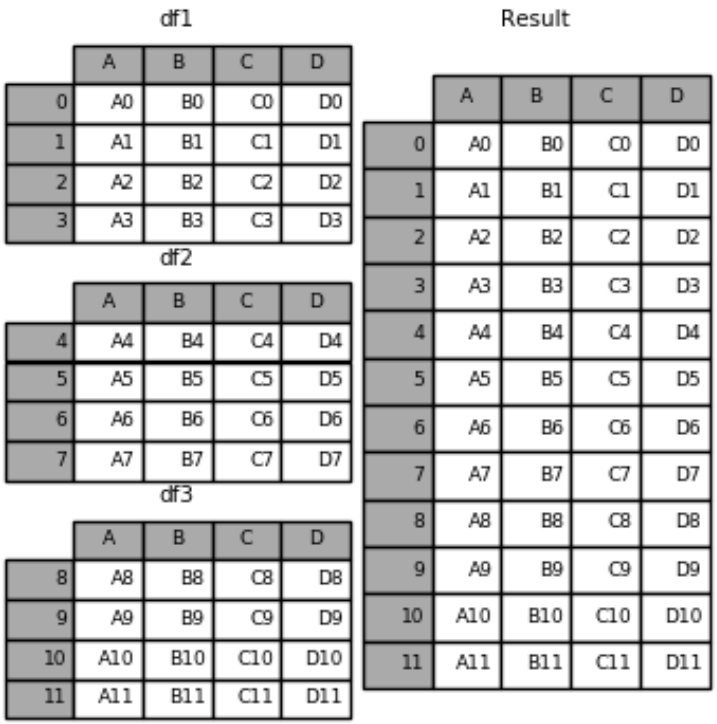


Changing from wide to long and viceversa can be done with two simple `pandas` functions: `melt()`, `pivot_table()`

[1] https://towardsdatascience.com/seven-clean-steps-to-reshape-your-data-with-pandas-or-how-i-use-python-where-excel-fails-62061f86ef9c

Join, merge and concat: concat (1)

- Concatenating dataframes is the most basic way of putting two or more dataframes (input as a list) as a single dataframe
- It can be done without explicit logic behind the concatenation, just putting the dataframes together in the specified axis[2]



	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

[2] Merge, join, and concatenate

Join, merge and concat: concat (2)

How to use `concat`

- Vertically: `pd.concat([df_up, df_down], axis=0)`
- Horizontally: `pd.concat([df_left, df_right], axis=1)`

Join, merge and concat: merge (1)

`merge` is recommended when the operation you want to perform with dataframes involves a SQL-like join logic

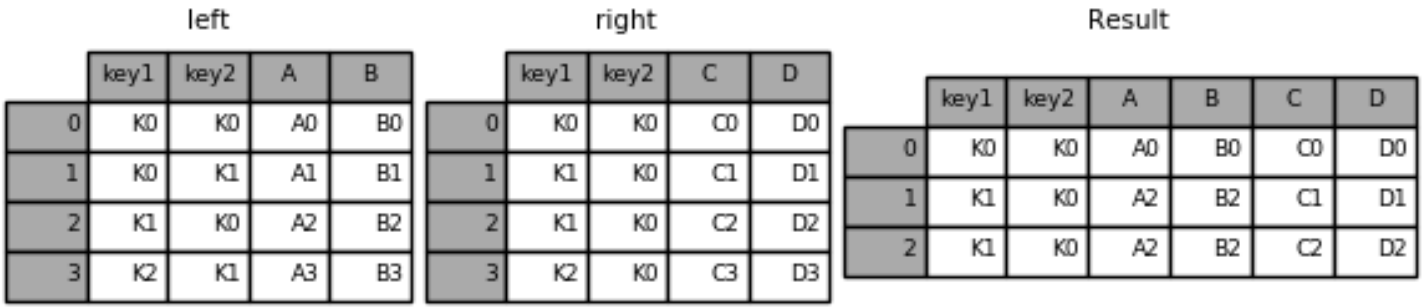
While `concat` receives a list of dataframes, `merge` can only be performed by pairs:

- `pd.merge(df_left, df_right, on, how)`
- `on` receives a single column or a list of columns on which to perform the join operation
- `how`: "inner" (default), "left", "right", "outer"[3]

[3] https://documentation.mindsphere.io/resources/html/predictive-learning/en-US/Types_of_Joins.htm

Join, merge and concat: merge (2)

The most common value for `how` argument is "inner", hence why it is the default value: choosing those observations that are BOTH on the left and right dataframes[4]



left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

Even though it is not native to merge, we can perform a `merge` operation on several datasets by using

```
reduce(lambda left, right: merge(left, right, on), list_df)
```

[4] Merge, join, and concatenate

The End