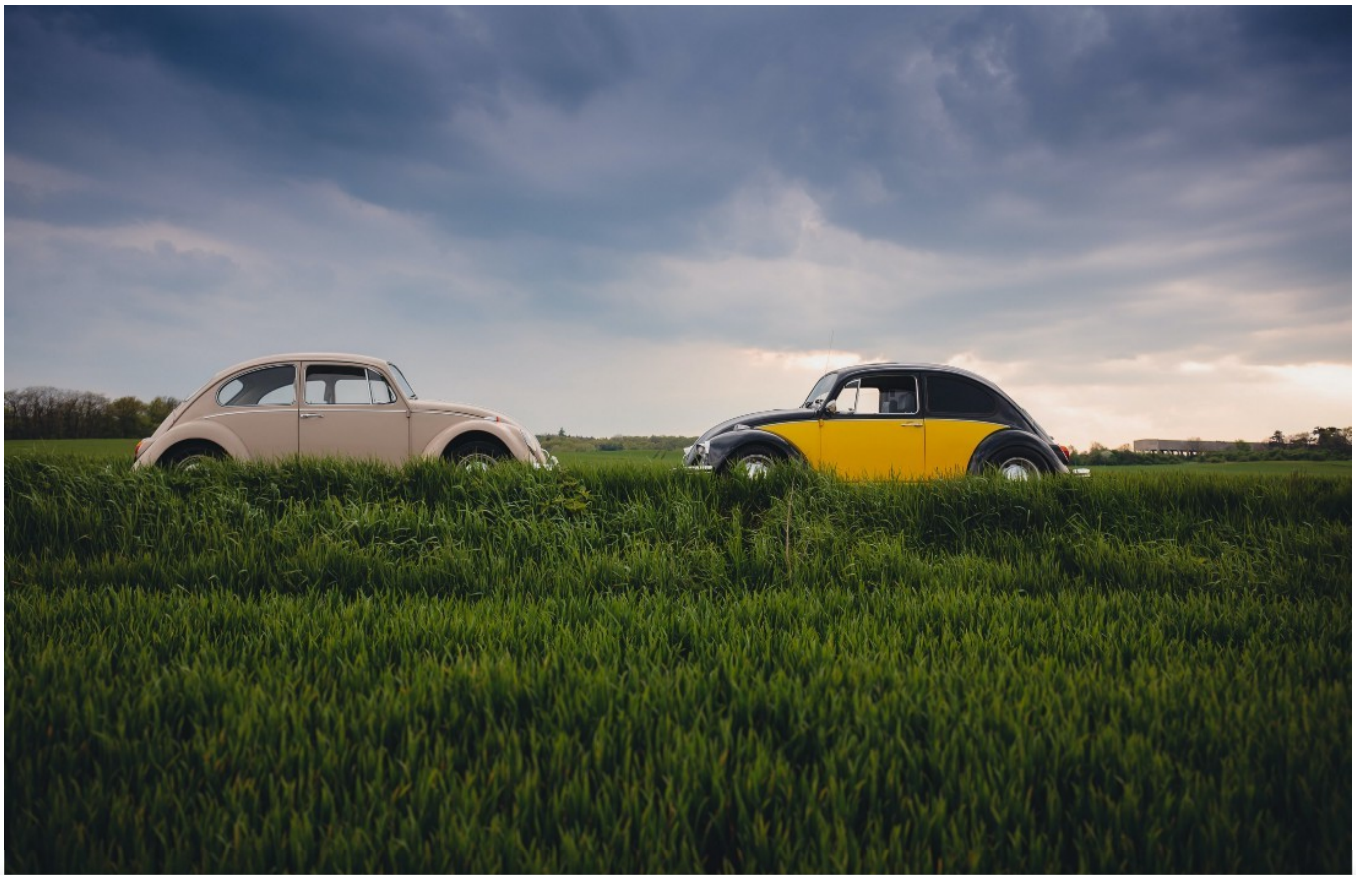
Published in Towards Data Science

Soner Yıldırım Follow

Apr 20, 2021 · 4 min read · ✨ · Listen

5 Examples to Compare Python Pandas and R data.table

A practical guide for both



workflow.

In this article, we will compare pandas and data.table, two popular data analysis and manipulation libraries for Python and R, respectively. We will not be trying to declare one as superior to the other. Instead, the focus is to demonstrate how both libraries provide efficient and flexible methods for data wrangling.

The examples we will cover are the common data analysis and manipulation operations. Thus, you are likely to use them a lot.

We will be using the Melbourne housing dataset available on Kaggle for the examples. I will be using Google Colab (for pandas) and RStudio (for data.table) as IDE. Let's first import the libraries and read the dataset.

```
# pandas
import pandas as pd
melb = pd.read_csv("/content/melb_data.csv")

# data.table
library(data.table)
melb <- fread("datasets/melb_data.csv")
```

Example 1

The first example is about creating a new column based on an existing one in the dataset. It is a common operation in feature engineering processes. Both pandas and data.table provide simple ways to complete this task.

```
# pandas
melb["Price_per_area"] = melb["Price"] / melb["Landsize"]

# data.table
```


0	1480000.0	202.0	7326.732673
1	1035000.0	156.0	6634.615385
2	1465000.0	134.0	10932.835821
3	850000.0	94.0	9042.553191
4	1600000.0	120.0	13333.333333

The first 5 rows of the Price, Landsize, and Price_per_area columns (image by author)

Example 2

For the second example, we create a subset of the original dataset by applying a couple of filters. The subset includes houses that cost more than 1 million and are of type h.

```
# pandas
subset = melb[(melb.Price > 1000000) & (melb.Type == "h")]

# data.table
subset <- melb[Price > 1000000 & Type == "h"]
```

For pandas, we provide the name of the data frame to select the columns used for filtering. On the other hand, it is enough for data.table to use only the column names.

Example 3

A highly common function used in data analysis is the groupby function. It allows to compare the distinct values in a categorical variable based on some numerical measures.


```
# pandas
melb[melb.Type == "u"].groupby("Regionname").agg(
   avg_price = ("Price", "mean")
)

# data.table
melb[Type == "u", .(avg_price = mean(Price)), by="Regionname"]
```

```
                  Regionname avg_price
1:      Northern Metropolitan  544403.8
2:       Western Metropolitan  488414.4
3:      Southern Metropolitan  664860.0
4:       Eastern Metropolitan  649314.4
5: South-Eastern Metropolitan  583364.9
6:           Eastern Victoria  461333.3
```

(image by author)

Pandas performs such operations using the groupby function. This operation is relatively simpler with data.table because we just need to use the by parameter.

Example 4

Let's take the previous example one step further. We find the average price of houses but we do not know the number of houses in each region.

Both libraries allow for applying multiple aggregations in one operation. We can also sort the results in ascending or descending order.

```
# pandas
melb[melb.Type == "u"].groupby("Regionname").agg(
   avg_price = ("Price", "mean"),
   number_of_houses = ("Price", "count")
```

```
    by="Regionname"
][order(-avg_price)]
```

```
                 Regionname avg_price number_of_houses
1:      Southern Metropolitan  664860.0             1549
2:       Eastern Metropolitan  649314.4              180
3: South-Eastern Metropolitan  583364.9               37
4:      Northern Metropolitan  544403.8              829
5:       Western Metropolitan  488414.4              419
6:           Eastern Victoria  461333.3                3
```

(image by author)

We use the count function to get the number of houses in each group. The ".N" can be used as the count function in data.table.

Both libraries sort the results in ascending order by default. This behavior is controlled with the ascending parameter in pandas. We just use a minus sign to get the results in descending order in data.table.

Example 5

In the last example, we will see how to change the column names. For instance, we can change the names of the type and distance columns.

- Type: HouseType
- Distance: DistanceCBD

The distance column in the dataset indicates the distance to the central business district (CBD) so it is better to provide this information in the column name.


```
# data.table
setnames(melb,
         c("Type", "Distance"),
         c("HouseType", "DistanceCBD"))
```

For pandas, we pass a dictionary that maps the changes to the rename function. The inplace parameter is used for saving the results in the original data frame.

For data.table, we use the setnames function. It takes three arguments which are the name of the table, column names to be changed, and new column names.

Conclusion

We have compared pandas and data.table based on 5 examples of common operations in data analysis and manipulation processes. Both libraries provide simple and efficient ways to accomplish these tasks.

In my opinion, the syntax of data.table is a little simpler than that of pandas. Data.table is more to the point in some cases.

It is important to mention that the examples we have done in this article only represent a very small portion of what these libraries are capable of. They provide numerous functions and methods to perform much more complex operations.

Thank you for reading. Please let me know if you have any feedback.