**Three Common Data Analysis Operations with Three Common Tools**

Pandas, data.table and SQL



Photo by [Jack Hunter](https://unsplash.com/@jacktthunter?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) on [Unsplash](https://unsplash.com/s/photos/three?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText" \t "_blank)

We analyze data to extract insights, find valuable pieces of information, or discover what is unseen by just browsing. The complexity of data analysis processes vary depending on the characteristics and structure of the data.

However, there are some fundamental operations that are frequently done. These can be considered as the ABC of data analysis:

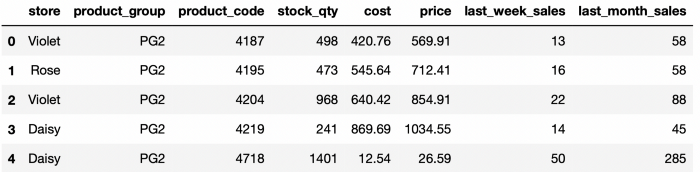
* Grouping
* Filtering
* Sorting

In this article, we will learn how to do these operations with 3 of the most commonly used tools for data analysis:

* Pandas for Python
* data.table for R
* SQL

The goal is not to compare these tools or classify one being superior to the others. You may need to use any or all of them in your data science career because these tools are used by numerous companies.

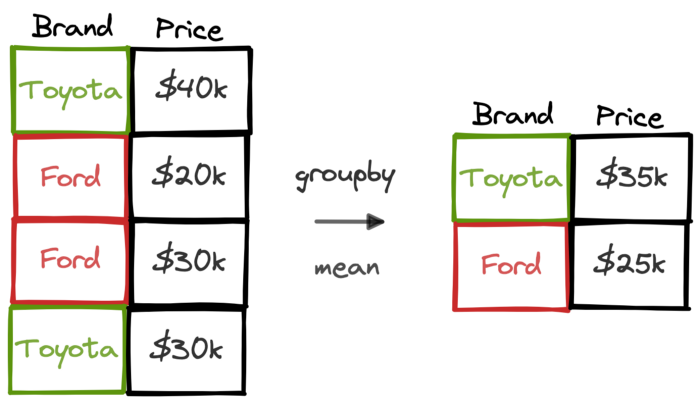
As always, we will learn by doing examples so we need a dataset to work with. I prepared a sales dataset with mock data. You can download it from the [datasets](https://github.com/SonerYldrm/datasets" \t "_blank) repository on my GitHub page. It’s called “sales\_data\_with\_stores”. Here are the first 5 rows of this dataset:



sales\_data\_with\_stores (image by author)

**Grouping**

Grouping data points (i.e. rows in tabular data) based on distinct values or categories in a column or columns is usually done in exploratory data analysis.



(image by author)

Some of the things that can be calculated with grouping:

* Average car price by brand
* Average revenue by month
* Day of week with highest sales quantity

Back to our dataset, we can find the average last week sales for each store as follows:

**Pandas**

We use the groupby and mean functions. First, the rows are grouped by the store column. Then, we select the column to be aggregated and apply the related function.

import pandas as pddf = pd.read\_csv("sales\_data\_with\_stores.csv")df.groupby("store")["last\_week\_sales"].mean()# output  
store  
Daisy 66.544681  
Rose 64.520000  
Violet 99.206061  
Name: last\_week\_sales, dtype: float64

**data.table**

The syntax of the data table package is a little simpler than that of Pandas. The actions to perform are written separated by comma inside square brackets as shown below:

https://miro.medium.com/max/700/1*3Rn_f6DCnXuX4bq0kMdASw.png

data table syntax structure (image by author)

library(data.table)dt <- fread("sales\_data\_with\_stores.csv")dt[, mean(last\_week\_sales), store]# output  
 store V1  
1: Violet 99.20606  
2: Rose 64.52000  
3: Daisy 66.54468

**SQL**

Assume we have a table called sales that contains the data in our dataset. We use the select and group by statements as below:

SELECT  
 store,  
 AVG(last\_week\_sales)  
FROM sales  
GROUP BY store

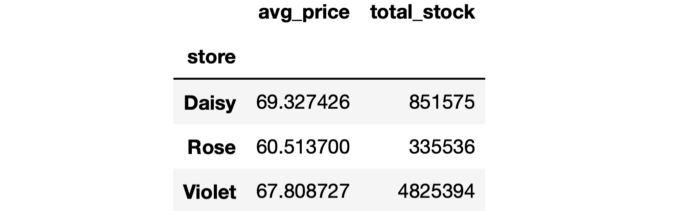
The output will be the same as in the other examples.

In all the examples, the aggregated columns do not have a self-explanatory column name, which is not the ideal case especially when working with other people. Let’s do another series of examples and find the average product price and total stock quantity for each store. We will also assign names to the aggregated columns.

**Pandas**

We will use the agg function. The column to be aggregated and the aggregate function are written inside a tuple as shown below:

df.groupby("store").agg(  
   
 avg\_price = ("price", "mean"),  
 total\_stock = ("stock\_qty", "sum"))# output



(image by author)

**data.table**

The structure of the syntax is the same but with a few small tweaks. The aggregations are written inside a parenthesis preceded by a dot.

dt[,   
 .(  
 avg\_price = mean(price),  
 total\_stock = sum(stock\_qty)  
 ),  
 store  
 ]

**SQL**

It is quite similar to the other SQL example. We just need to add the column names.

SELECT  
 store,  
 AVG(price) AS avg\_price,  
 SUM(stock\_qty) AS total\_stock  
FROM sales  
GROUP BY store

**Filtering**

Filtering is another frequent operation in data analysis. Most tools provide functions and methods to filter raw data based on string, numeric, and date values.

We will do an example that contains both string and numeric filters. Let’s select the data points (i.e. rows) in which:

* Store is Violet
* Product group is PG1, PG3, or PG5
* Last month sales is above 100

**Pandas**

We write the filtering conditions inside square brackets. In the case of multiple conditions, each one is written inside parenthesis and conditions are combined with the appropriate logical operator (e.g. & for end, | for or logic).

df[  
 (df["store"] == "Violet") &  
 (df["product\_group"].isin(["PG1","PG3","PG5"])) &   
 (df["last\_month\_sales"] > 100)  
]

The output of this code is a DataFrame with the rows that fit the given set of conditions.

**data.table**

The logic is similar. We combine multiple conditions using the and operator in this case.

dt[  
 store == "Violet" &  
 product\_group %in% c("PG1","PG3", "PG5") &  
 last\_month\_sales > 100  
 ]

**SQL**

The conditions are specified in the where statement. We use the and keyword to combine multiple conditions in this case.

SELECT \*  
FROM sales  
WHERE store = "Violet" AND  
 product\_group in ("PG1", "PG3", "PG5") AND  
 last\_month\_sales > 100

**Sorting**

We sometimes need to sort rows based on the values in a column or columns. For instance, we may want to sort the products based on the price in descending order.

**Pandas**

The sort\_values function is used for this task. We just need to write the columns that will be used for sorting. Pandas sorts in ascending order by default but this behavior can be changed using the ascending parameter.

df\_sorted = df.sort\_values(by="price", ascending=False)

**data.table**

The order function is used. To change from ascending to descending, we just need to add a minus sign in front of the column name.

dt\_sorted <- dt[order(-price)]

**SQL**

The order by statement is used in SQL to sort the rows. Like data.table and Pandas, the rows are sorted in ascending order. We can sort in descending order using the desc keyword.

SELECT \*  
FROM sales  
ORDER BY price DESC

We have learned how to do 3 fundamental data analysis operations with 3 commonly-used tools in the data science ecosystem. We have covered the simple cases but the functions and methods used are capable of doing more complicated tasks as well. It’s always better to get a good grasp of the basics before learning the details.

*You can become a*[*Medium member*](https://sonery.medium.com/membership)*to unlock full access to my writing, plus the rest of Medium. If you already are, don’t forget to*[*subscribe*](https://sonery.medium.com/subscribe)*if you’d like to get an email whenever I publish a new article.*