**Pandas Sidetable Simplifies the Exploratory Data Analysis Process**

A practical add-on for Pandas

[Sidetable](https://github.com/chris1610/sidetable) is an open-source pandas utility library created by [Chris Moffitt](https://twitter.com/chris1610). It creates summary tables for pandas DataFrames, which is quite useful in exploratory data analysis.

What sidetable does can also be achieved with pandas but it simplifies the process. You can get a decent overview of the data with just one line of code.

As of writing this article, sidetable has 5 functions and it is being improved. Thus, we might expect more to come soon. We will do some examples for each function to understand what it does and how it contributes to an efficient exploratory data analysis.

You can install sidetable as follows:

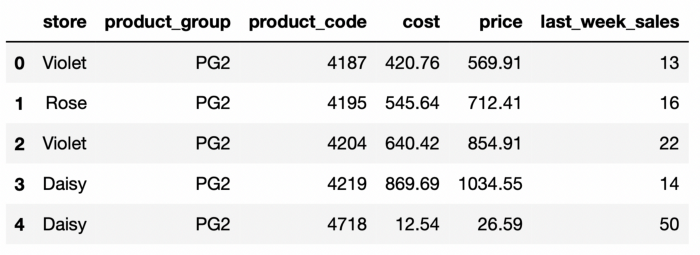
# terminal  
$ python -m pip install -U sidetable# jupyter notebook  
!pip install sidetable

Once installed, you can import sidetable and use it as an accessor on pandas DataFrames.

import pandas as pd  
import sidetable

We will be working on a sales dataset I created with mock data. You can download it from the [datasets](https://github.com/SonerYldrm/datasets" \t "_blank) repository on my GitHub page. The dataset we will use in this article is called sales data with stores.

sales = pd.read\_csv(  
 "sales\_data\_with\_stores.csv",  
 usecols = ["store", "product\_group", "product\_code", "cost",   
 "price", "last\_week\_sales"]  
)sales.head()



(image by author)

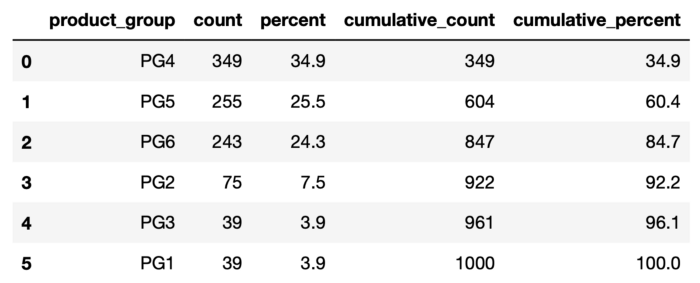
**Value counts on steroids**

Let’s say we want to get an overview from product group perspective and are looking for answers to the following questions:

* How many product groups exist?
* What is the size of each product group in terms of the number of products they contain?
* What is the cumulative coverage of the entire portfolio?

We can find the answer to all these questions with a single line of code using the **freq** function.

sales.stb.freq(["product\_group"])

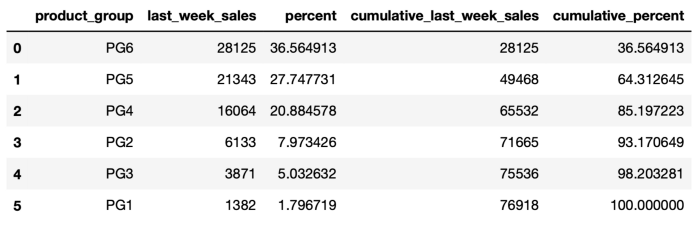


(image by author)

There are 6 product groups and we can see how many products each group contains. Furthermore, the resulting table has cumulative values. For instance, PG4, PG5, and PG6 represent 85% of our entire portfolio.

The table above is based on the row counts in the DataFrame. Let’s say we would like to see the distribution with regards to sales. This can easily be done by using the value parameter of the freq function.

sales.stb.freq(["product\_group"], value="last\_week\_sales")



(image by author)

PG3 is the third largest product group in terms of the number of products but it has the most sales. This is a valuable insight for our analysis.

We may want to take this analysis one step further and see the performance of product groups at each store separately. All we need to do is to write the product group and store columns in a Python list.

sales.stb.freq(["product\_group", "store"], value="last\_week\_sales")

**6 questions at once**

* How many observations (i.e. rows)?
* How many unique values?
* The most frequent value?
* How many observations for the most frequent value?
* The least frequent value?
* How many observations for the least frequent value?

All these questions can be answered for all the columns with a single line of code using the **counts** function.

We get the results for all the columns by default. However, these questions are usually asked for categorical columns. Thus, if we just need this data for categorical columns, we can exclude numerical columns using the exclude parameter.

sales.stb.counts(exclude="number")



(image by author)

Let’s do a quick result check on the store column:

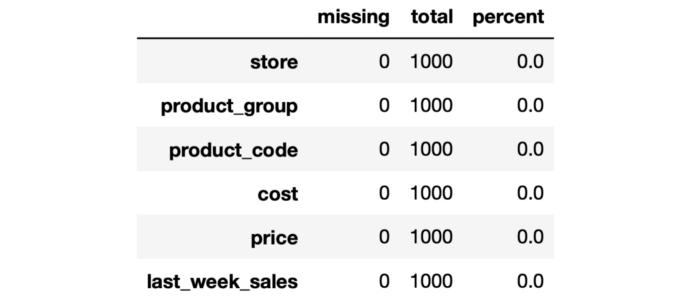
sales["store"].value\_counts()**# output**  
Daisy 470  
Violet 330  
Rose 200  
Name: store, dtype: int64

The most frequent value is daisy and it has 470 observations, which is what sidetable shows above.

**Missing values**

It’s not a complicated task to find missing values with pandas. The isna and sum functions can be used together to find the number of missing values in each column. What sidetable adds onto this is the percent of missing values.

sales.stb.missing()



(image by author)

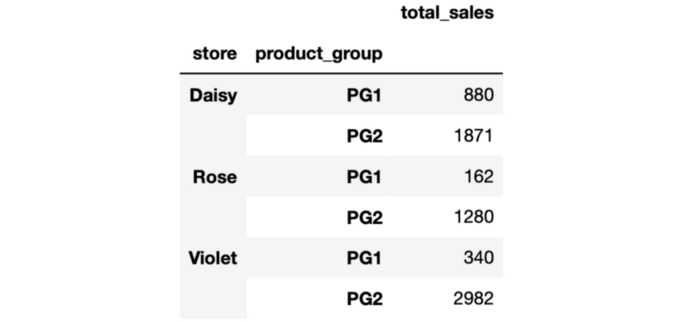
The missing function provides a quick overview of the missing values in the DataFrame.

If you only want the columns that contain missing values, set the value of the clip\_0 parameter as True.

**Subtotals for aggregated values**

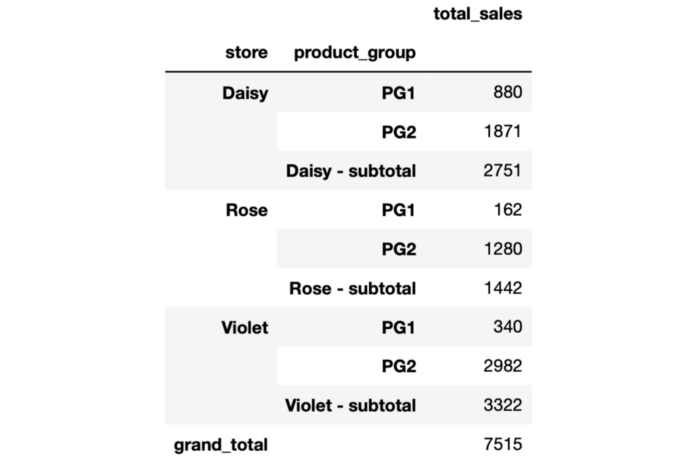
Sidetable makes it very simple to add subtotals to the results of the groupby function. Having subtotals might come in handy for some tasks. Let’s do examples with and without subtotals to see the difference.

sales\_filtered = sales[sales["product\_group"].isin(["PG1", "PG2"])]sales\_filtered.groupby(["store", "product\_group"]).agg(  
 total\_sales = ("last\_week\_sales", "sum")  
)



without subtotals (image by author)

sales\_filtered.groupby(["store", "product\_group"]).agg(  
 total\_sales = ("last\_week\_sales", "sum")  
).stb.subtotal()

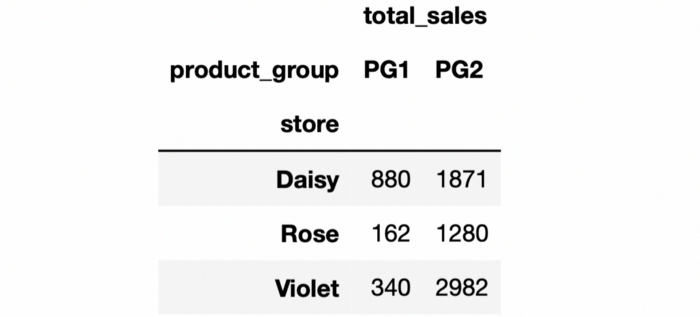


with subtotals (image by author)

It is important to note that subtotals will not be calculated if you make the groups shown as columns in the output of the groupby function (i.e. set the value of the as\_index parameter as False).

**Flatten DataFrames with multi-level column index**

Consider we are given a DataFrame as follows:



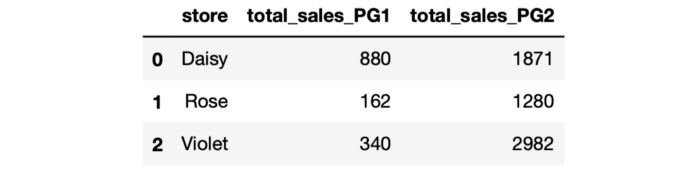
(image by author)

Let’s check the column names:

df.columns**# output**  
MultiIndex([('total\_sales', 'PG1'),  
 ('total\_sales', 'PG2')],  
 names=[None, 'product\_group'])

It’s a multi-index and each column name is a tuple. We can flatten the column names by using the **flatten** function of sidetable.

df.stb.flatten()



(image by author)

It looks better and easier to work with.

Sidetable is a highly practical tool to use in exploratory data analysis. The operations we have done in this article are not very complex but it does not harm to simplify them. Sidetable allows for performing all these operations with a single line of code.