**10 Quick Pandas Tricks to Energize your Analytics Project**

**Use these tricks to make your life easy as an analyst**

Pandas is the popular data analytics library in Python. It certainly provides you the flexibility and the tools you need to process the data.

However, You must know time-saving and easy-to-use tricks for efficient working. That’s why I’m sharing 10 quick yet highly useful tricks in pandas, which you can master in under 10 minutes.

Even if you are already well versed with pandas methods and functions, you will still find some of these tricks actionable. If you are an absolute beginner, then this article is right place to start your learning.

Here is a quick index for your reference —

· [Adjust the number of columns or rows to display](https://towardsdatascience.com/10-quick-pandas-tricks-to-energize-your-analytics-project-2333a144946b#b151)  
· [Randomly sample rows with sample()](https://towardsdatascience.com/10-quick-pandas-tricks-to-energize-your-analytics-project-2333a144946b#b002)  
· [Subset a dataframe with the query() method](https://towardsdatascience.com/10-quick-pandas-tricks-to-energize-your-analytics-project-2333a144946b#4a3d)  
· [Round values in pandas DataFrame with round()](https://towardsdatascience.com/10-quick-pandas-tricks-to-energize-your-analytics-project-2333a144946b#6107)  
· [Unpack the list values using the method explode()](https://towardsdatascience.com/10-quick-pandas-tricks-to-energize-your-analytics-project-2333a144946b#3208)   
· [Create quick plots with the plot() method](https://towardsdatascience.com/10-quick-pandas-tricks-to-energize-your-analytics-project-2333a144946b#4fa1)  
· [Return multiple DataFrames with display()](https://towardsdatascience.com/10-quick-pandas-tricks-to-energize-your-analytics-project-2333a144946b#de5e)  
· [Count unique values with nunique()](https://towardsdatascience.com/10-quick-pandas-tricks-to-energize-your-analytics-project-2333a144946b#bee2)  
· [Return a cumulative summary with cumsum()](https://towardsdatascience.com/10-quick-pandas-tricks-to-energize-your-analytics-project-2333a144946b#492c)  
· [Aggregate by multiple functions using agg()](https://towardsdatascience.com/10-quick-pandas-tricks-to-energize-your-analytics-project-2333a144946b#0360)

For the sake of examples, I’ll be using [Airbnb Open Data](https://www.kaggle.com/datasets/arianazmoudeh/airbnbopendata) which you can avail for free as it is publicly available under [ODbL Licence](https://opendatacommons.org/licenses/odbl/1-0/)!

Don’t forget to get the *Jupyter-Notebook* with all the examples, at the end of this read!

The data analytics task starts with importing the dataset into pandas DataFrame and the required dataset is often available in .csv format.

However, when you read a csv file with a large number of columns in the pandas DataFrame, you can see only few column names followed by **. . . . .** and few more column names as shown below.

import pandas as pd  
df = pd.read\_csv("Airbnb\_Open\_Data.csv")  
df.head()



Display limited number of columns in Jupyter-Notebook | Image by Author

The main purpose behind .head() is to take a sneak-peek into the data. With such hidden column names behind . . . . ., you are blind about the data in those columns. So the first trick here is to fix the number of columns and rows you would like to display.

**Adjust the number of columns or rows to display**

pandas by default displays only 60 rows and 20 columns on your screen. This can be reason for the few columns to hide behind . . . . ..

You can check this limit yourself by using pandas display options as below.

**pd.options.display.max\_rows**#Output  
60**pd.options.display.max\_columns**#Output  
20

You can use exactly same options to set a maximum number of rows and columns to be displayed on the screen.

For example, you can set the maximum number of columns to 30 as below.

pd.options.display.max\_columns = 30  
df = pd.read\_csv("Airbnb\_Open\_Data.csv")  
df.head()



Pandas setting number of maximum rows and columns | Image by Author

In the above picture, you can see all the columns in the dataset. The columns within red box were previously hidden behind . . . . .. and were not visible in the output.

Also, you can reset an option back to its default value using the function .reset\_option() as shown below.

pd.reset\_option('display.max\_rows')  
pd.options.display.max\_rows  
  
#Output  
60

When you change multiple default options in pandas, you can reset all the options back to the original values in single statement using ‘all’ argument in .reset\_options() function.

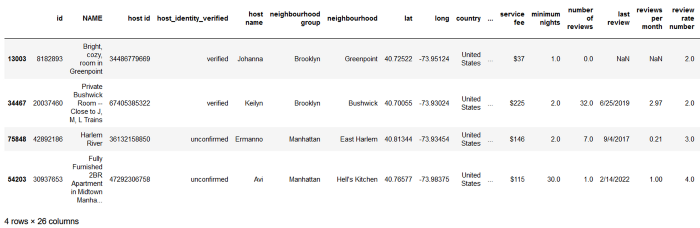
To take a sneak-peek into the dataset you can use .head() or .tail() methods on. This limits you to see only the first/last few rows. In pandas, you can always check data from any randomly selected record or even subset the data using random selection.

**Randomly sample rows with sample()**

The data may look well organized and clean from the first or last few rows. So, it is always good to take a look at any random records in the dataset to understand it better.

pandas.DataFrame.sample offers a flexibility to randomly select any record from the dataset with 7 optional and adjustable parameters.

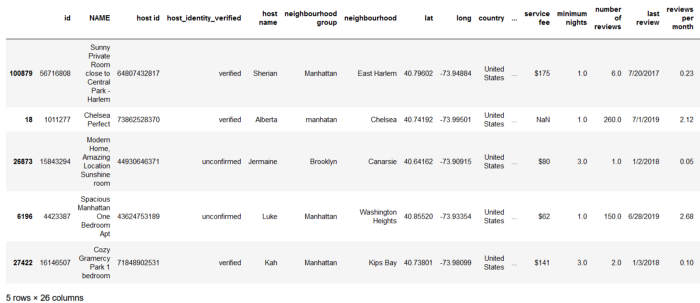
For example, suppose you would like to select 4 random records from the Airbnb dataset. You can simply type in — df.sample(4) — to get the below output.



Random selection of records from DataFrame | Image by Author

Alternatively, you can also use fraction to specify how much fraction of entire dataset you want to select randomly. Simply assign the fraction (less than 1) to the parameter frac in the sample() function, as shown below.

df.sample(frac = 0.00005)



Pandas random sample using fraction | Image by Author

Here you simply fetched *0.005%* of total number of rows in DataFrame.

However, the records are randomly choosen everytime when you execute this cell. So, to get the same output every time you can set the parameter random state to any integer.

Suppose you would like to retrive same three rows everytime, then you can use the sample() function like below.

df.sample(3, random\_state = 4)



Pandas random sample same output everytime | Image by Author

When you want to select different records you can simply change the random\_state parameter to another integer.

Once you check the data available in the dataset, the next task is to select the required subset of the dataset. Certainly you can use .loc and .iloc methods followed by series of square brackets to extract selected rows and columns.

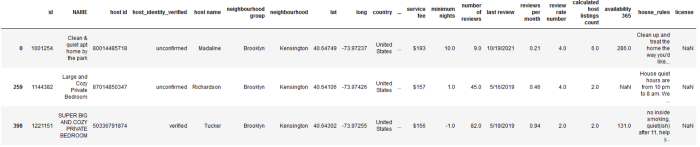
But, there is one more method .query() which can help you get rid of multiple opening and closing square brackets to subset a DataFrame.

**Subset a dataframe with the query() method**

The function pandas.DataFrame.query(expression)offers you the flexibility to conditionally select a subset of DataFrame. The expression provided within this function is a combination of one or more conditions. You can write it in an absolutely easy-going way without any square brackets.

For example, suppose you want to extract all the records from the dataset where *neighbourhood* is *Kensigton*. Using query() method this is quite simple, as shown below.

df.query("neighbourhood == 'Kensington'")



Subset pandas DataFrame using query() | Image by Author

You can also use multiple conditions on the same column and any logic such as AND, OR, NOT between them.

To learn more about this method, I strongly recommend reading —

**[10 Pandas Query Examples That Will Make You Use Pandas Query Easily](https://towardsdatascience.com/10-pandas-query-examples-that-will-make-you-use-pandas-query-easily-12a45b1e115" \t "_blank)**

**[Start filtering your data easily using pandas query function](https://towardsdatascience.com/10-pandas-query-examples-that-will-make-you-use-pandas-query-easily-12a45b1e115" \t "_blank)**

[towardsdatascience.com](https://towardsdatascience.com/10-pandas-query-examples-that-will-make-you-use-pandas-query-easily-12a45b1e115" \t "_blank)

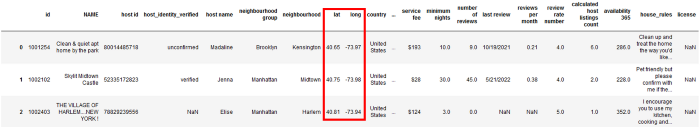
After selecting the subset, you’ll land into for data cleaning phase in the analytics. One of the common problem with the numerical columns is, they contain more than required number of digits after decimal point. Let’s see how you can deal with such columns.

**Round values in pandas DataFrame with round()**

Sometimes the numerical data in a column contains multiple digits after decimal and it is better to limit them up to 2–3 digits.

The method [pandas.DataFrame.round](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.round.html) can be very handy in such cases. All you need to do is mention the required number of digits within the method as shown below.

df.round(2)



Round a DataFrame to a variable number of decimal places | Image by Author

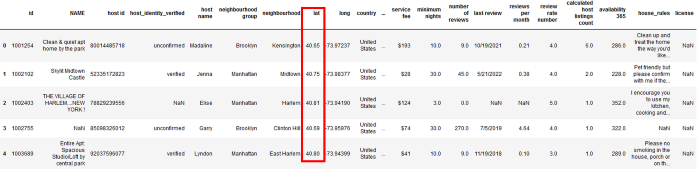
In the Airbnb dataset, only the columns *lat* and *long* contain values with 5 digits after the decimal. The DataFrame method round() simply rounds up the number of digits for all the columns in the dataset.

But suppose, you want to round the number of digits only for a single column — *lat* — in this dataset. The method pandas.Series.round can be useful in such scenario, as shown below.

df['lat'].round(2)  
  
#Output  
  
0 40.65  
1 40.75  
2 40.81  
 ...   
102596 40.68  
102597 40.75  
102598 40.77  
Name: lat, Length: 102599, dtype: float64

The output of pandas.Series.round is again a Series. To have it as part of the same DataFrame, you need to re-assign the changed column to the original column as shown below.

df['lat'] = df['lat'].round(2)  
df.head()



Round each value in a Series or column to the given number of decimals | Image by Author

It changed only the values in the column *lat* to have 2 digits after decimal, whereas the values in the column *long* remain unchanged.

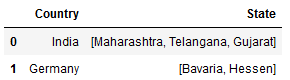
Continuing manipulating column data, let’s see another method explode() which has a more specific use-case — to transform each item of a list-like values to a separate row.

**Unpack the list values using the method explode()**

Sometimes you come across a dataset, where values in a column are lists. It is difficult to deal with such values in the long run and it is always better to create a single row for each value of the list.

To understand this concept better, let’s create a DataFrame.

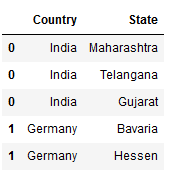
df = pd.DataFrame({"Country":["India","Germany"],  
 "State":[["Maharashtra", "Telangana", "Gujarat"],  
 ["Bavaria", "Hessen"]]})  
df



Sample dataset | Image by Author

The column *State* in the above DataFrame contains list as a value. To get a single row for each value in each list of *State* column, you can use the method pandas.DataFrame.explode, as shown below.

df.explode("State")



df.explode() in Python | Image by Author

All you need to do is mention the column name containing lists as its values within explode(). You may notice in the above output, it simply replicated the index values for each item in the list.

The common task after data cleaning is data visualization. Charts and graphs make it easy to identify underlying trends, patterns and correlations.

When you are using pandas for data analytics, you don’t need to import any other library for creating charts. pandas has its own methods and flexible options to create variety of charts quickly.

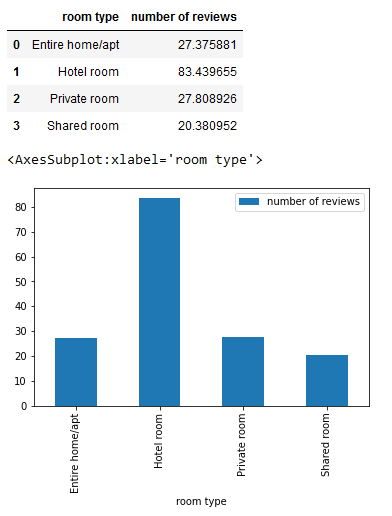
**Create quick plots with the plot() method**

Often, the purpose of your analytics task is not data visualization but you want to see simple charts/graphs from your data. Pandas is such a flexible package that it allows you to visualize the DataFrame contents using its own methods.

Suppose, you would like to see the average number of reviews for each type of room. You can achieve it with the method pandas.DataFrame.plot which makes plots of Series or DataFrame.

You can create a smaller and simpler DataFrame — *df\_room* — with only required two columns, as I did here.

df\_room = pd.DataFrame(df.groupby('room type')['number of reviews'].mean())  
df\_room.reset\_index(drop=False, inplace=True)  
  
display(df\_room)  
  
df\_room.plot(x='room type', y='number of reviews', kind='bar')



pandas.DataFrame.plot | Image by Author

Both — the newly created DataFrame and a chart — is displayed in the output.

You can always change the type of chart from bar to line using the parameter ‘*kind’* in **.plot()**. You can find a complete list of available chart types in this [Notebook](https://github.com/17rsuraj/data-curious/blob/master/TowardsDataScience/10%20Useful%20Python%20Tricks%20You%20Should%20Know.ipynb).

But how does pandas created the bar chart with no input about chart style?

pandas.DataFrame.plot uses the backend specified by the option plotting.backend . Plotting backend is the plotting library that pandas uses to create charts and it uses matplotlib as default library.

You can change it anytime by setting pd.options.plotting.backend or by using the option pd.set\_option(‘plotting\_backend’, ‘name\_of\_backend’).

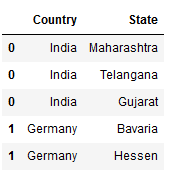
Moving back to dealing with the DataFrames as a whole let’s see how you can display multiple DataFrames simultaneously.

**Return multiple DataFrames with display()**

Often, you create multiple DataFrames, but when you mention their names or use .head() / .tail() method on them in the same cell, only the latest DataFrame is displayed in the output.

For an instance, let’s create two DataFrames and try to view them in the output.

df1 = pd.DataFrame({"Country":["India","Germany"],  
 "State":[["Maharashtra", "Telangana", "Gujarat"],  
 ["Bavaria", "Hessen"]]})  
  
df2 = df1.explode("State")  
  
# Get both DataFrames as output  
df1  
df2

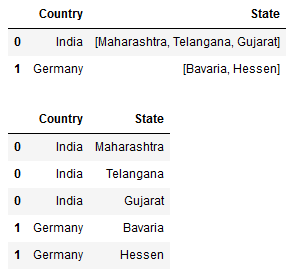


Cell output | Image by Author

Although, you mentioned df1 and df2 at the end of your code; it displayed only the df2 in the output.

But, you want to see both the DataFrames, one-below-other. That’s where the function display() is useful. You only need to pass the DataFrame to the display() function as shown below.

df1 = pd.DataFrame({"Country":["India","Germany"],  
 "State":[["Maharashtra", "Telangana", "Gujarat"],  
 ["Bavaria", "Hessen"]]})  
  
df2 = df1.explode("State")  
  
# Get both DataFrames as output  
display(df1)  
display(df2)



Display all DataFrames in output | Image by Author

Simple!

Now you can see both (or all) the DataFrames in the output — stacked one-over-another.

The previous trick is also a good example of the fucntion display() where you can see the DataFrame and the bar chart stacked over each other in the output.

Once you explore the dataset and investigate the trends, patterns in it, the next step is to do descriptive analysis. It can be achieved with the data transformation.

Starting with one of the basic data transformation — to investigate the distinct values in the categorical columns using different built-in functions.

**Count unique values with nunique()**

When you have categorical columns in the dataset, you sometime need to check how many different values are present in a column.

You can get it using the simplest function —**nunique()**. For an instance, suppose you would like to see how many different room types are there in the dataset, you can quickly check it using nunique().

df['room type'].nunique()  
  
#Output  
4

Well, it only tells you about how many unique values are available but to get the different values i.e. type of rooms you can use another function — **unique()**

df['room type'].unique()  
  
#Output  
array(['Private room', 'Entire home/apt', 'Shared room', 'Hotel room'],  
 dtype=object)

It returns an array with all the unique values.

After checking the unique values, it would also be interesting to check how many times each value appeared in the dataset i.e. how many times each type of room is recorded in the dataset.

You can get it using another method — value\_counts() — as shown below.

df.value\_counts('room type')  
  
#Output  
room type  
Entire home/apt 53701  
Private room 46556  
Shared room 2226  
Hotel room 116

In this way you can get the number of unique values, and the number of time they appeared using a single line of code.

The data transformation is never limited to categorical columns, infact most of the actionable insights are obtained from numerical columns.

Hence, let’s explore two commonly needed operations related to numerical columns. First thing is to see how you can get a cumulative summary of a column in the DataFrame.

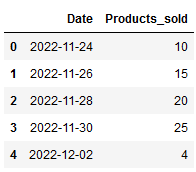
**Return a cumulative summary with cumsum()**

Cumulative sums are also called as running totals which are used to display the total sum of data as it grows with time. So at any point of time, it tells you total of all the values upto that point.

pandas DataFrame has its own method pandas.DataFrame.cumsum which returns cumulative sum of a DataFrame column.

Let’s create a simple DataFrame of dates and number of products sold.

daterange = pd.date\_range('2022-11-24', periods=5, freq='2D')  
df1 = pd.DataFrame({ 'Date': daterange,  
 'Products\_sold' : [10, 15, 20, 25, 4]})  
df1



Dummy dataset | Image by Author

This is a dummy dataset which has date range from 24.11.2022 to 02.12.2022 and respective number of products sold on each date.

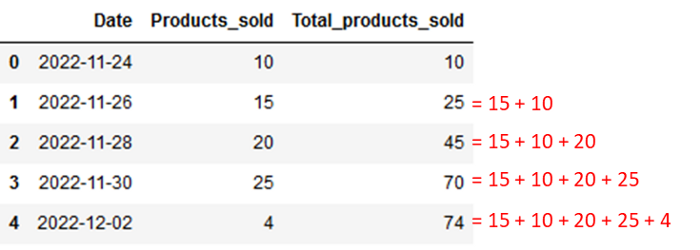
Now, suppose you want to see total number of products sold till 30.11.2022. You don’t need to manually calculate it, rather the method pandas.DataFrame.cumsum will get it for you in just one line of code.

df1["Products\_sold"].cumsum()  
  
#Output  
0 10  
1 25  
2 45  
3 70  
4 74

It simply returned the running total of a specific column. But, it is difficult to understand as you don’t see any dates or original values in the output.

Therefore, you should assign the cumulative sum to a new column in the same DataFrame as shown here.

df1["Total\_products\_sold"] = df1["Products\_sold"].cumsum()  
df1



Cumulative sum or running total in Pandas DataFrame | Image by Author

Bingo!

You got Running total of your column in just one line of code!!

The commonly observed use-cases for cumulative sum are to understand “*how much so far*” such as —

* How much water level increased in the river till now
* How much sales, products sold until specific time
* How much balance is remained in the account after every transaction

So, knowing how to get [cumsum](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.cumsum.html) in your dataset can be real savior in your analytics project.

Also, while dealing with the numerical data you must know how you can aggregate the data and present it in the summary form.

**Aggregate by multiple functions using agg()**

You can always aggregate the raw data to present statistical insights such as minimum, maximum, sum and count. But, you really don’t need to do it manually when you are using pandas for data analytics.

Pandas offers a function — agg()— which can be used on pandas DataFrame groupby object. This object is created when DataFrame method groupby() is used for grouping the data into categories.

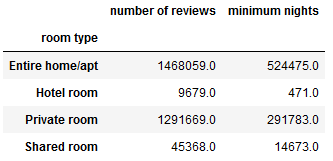
Using agg() function you can apply a aggregate function to all the numerical columns in the dataset.

For an instance, you can group the Airbnb dataset by room type to create a pandas.DataFrame.groupby object.

df\_room = df.groupby('room type')  
df\_room  
  
#Output  
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000029860741EB0>

Now, you can apply aggregate function — sum — on the columns *number of reviews* and *minimum nights*, as shown below.

df\_room.agg({'number of reviews': sum,  
 'minimum nights': sum})



Data aggregation in pandas | Image by Author

All you need to do is pass a dictionary to the function agg() in which keys are column names and values are aggregate function names such as sum, max, min.

You can also apply multiple functions on the same column or even different functions on different columns.

To understand data aggregation better, I highly recommend reading —

**[5 Pandas Group By Tricks You Should Know in Python](https://towardsdatascience.com/5-pandas-group-by-tricks-you-should-know-in-python-f53246c92c94" \t "_blank)**

**[All you need to know about Pandas DataFrame Group By, to use it efficiently!](https://towardsdatascience.com/5-pandas-group-by-tricks-you-should-know-in-python-f53246c92c94" \t "_blank)**

[towardsdatascience.com](https://towardsdatascience.com/5-pandas-group-by-tricks-you-should-know-in-python-f53246c92c94" \t "_blank)

I hope you finished this article quickly and found it useful to skill up data analytics using pandas.

I’m using Python since past 4+ years, and I found these tricks often time saving. These concepts are very useful while working on real projects.

*Interested in reading unlimited stories on Medium??*

💡 Consider [**Becoming a Medium Member**](https://medium.com/@17.rsuraj/membership) to **access unlimited** stories on medium and daily interesting Medium digest. I will get a small portion of your fee and no additional cost to you.

I’m also a Medium member & it is just Amazing. I can read & learn from each and every article posted here.

💡 Be sure to [**Sign-up to my Email list**](https://medium.com/subscribe/@17.rsuraj) to never miss another article on data science guides, tricks and tips, SQL and Python.