How to Use the Pandas Groupby for Grouping Data and Applying Functions

Reading time
4 min

Published
Nov 28, 2019

Updated
Nov 28, 2019

Applying refers to the function that you can use on these groups. Combining means that you form results in a data structure.

Contents

What is the Pandas groupby function?

Pandas `groupby` is a function for grouping data objects into **Series** (columns) or **DataFrames** (a group of Series) based on particular indicators. In simpler terms, group by in Python makes the **management of datasets** easier since you can put related records into **groups**.

> **Note:** essentially, it is a map of labels intended to make data easier to sort and analyze.

Using the groupby function: syntax rules

The basic Python `groupby` syntax typically consists of clauses such as

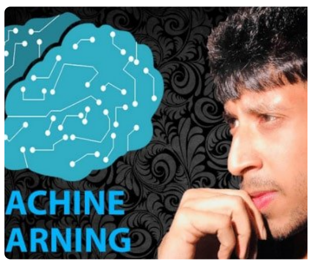
```
[illegible]
```

Which can be broken down into **these parts**:

- `Table_name` : this would be the name of the DataFrame, the source of the data you are working on.
- `groupby` : the group by in Python is for sorting data based on different criteria. In this case, the condition is `Group` .
- `Feature` : the part of the data or feature you want to be **inserted** in the computation.
- `aggregation()` : the specific **function name** or aggregation you wish to execute with this operation.

> **Note:** before using Python `groupby` function, you need to prepare the Pandas library. For instance, you can get Anaconda, and most of the necessary modules are already installed.

Splitting, applying and combining



How to split using Pandas groupby?

Splitting with `groupby` works by dividing a **DataFrame** into several categories and assigning labels to each one.

> **Note:** frequently, developers mention **split-apply-combine** technique. It means that you divide your data into groups

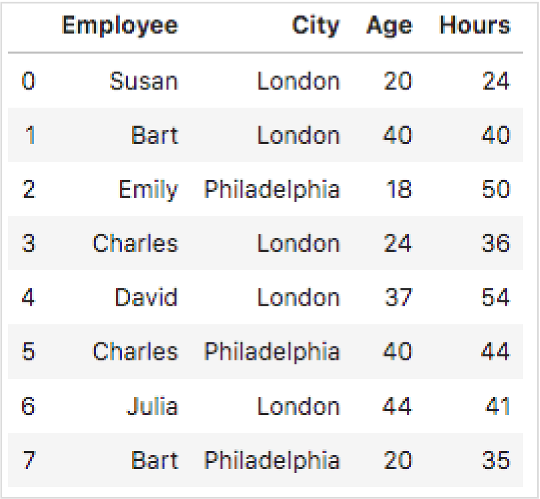
In the following example, we are creating a DataFrame with some information about employees (their **age**, **city**, and **hours** they have worked):

Example

Copy

```
import pandas as pd
import numpy as np
df = pd.DataFrame( {
"Employee" : ["Susan", "Bart", "Emily", "Charles", "David",
"Charles", "Julia", "Bart"] ,
"City" : ["London", "London", "Philadelphia", "London", "London",
"Philadelphia", "London", "Philadelphia"] ,
"Age" : [20, 40, 18, 24, 37, 40, 44, 20 ],
"Hours" : [24, 40, 50, 36, 54, 44, 41, 35]} )
df
```

In the next snapshot, you can see how the data looks **before** we start applying the Pandas `groupby` function:

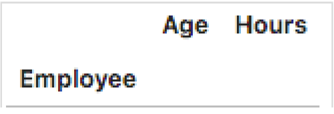


	Employee	City	Age	Hours
0	Susan	London	20	24
1	Bart	London	40	40
2	Emily	Philadelphia	18	50
3	Charles	London	24	36
4	David	London	37	54
5	Charles	Philadelphia	40	44
6	Julia	London	44	41
7	Bart	Philadelphia	20	35

Now, we can use the Pandas `groupby()` to arrange records in **alphabetical order**, **group similar records** and count the sums of hours and age:

```
df.groupby(['Employee']).sum()
```

Here is an outcome that will be presented to you:



	Age	Hours
Employee		

Categories

Search properties, snippets, c

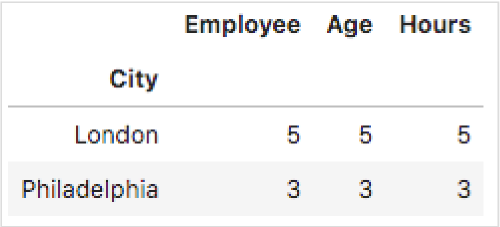
1. Python

Append
Array
Class
Command Line Arguments
Comment
Enumerate
Functions
Groupby
If... else
Map
Not Equal
Print
Queue
Random
Range
Round
Set
Sorting Lists
Split
Time

Applying functions with groupby

In this example, we will use this Python group by function to count how many employees are from the **same city**:

```
df.groupby('City').count()
```



	Employee	Age	Hours
City			
London	5	5	5
Philadelphia	3	3	3

In the following example, we add the values of **identical records** and present them in ascending order:

Example Copy

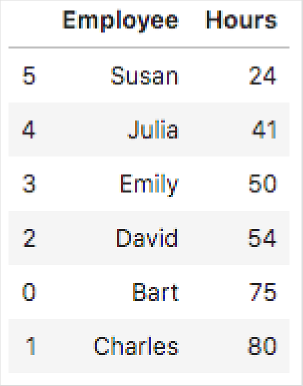
```
df.groupby('Employee')
['Hours'].sum().to_frame().reset_index().sort_values(by='Hours')
```

Train_test_split

Variables

While Loop

Here is the output:

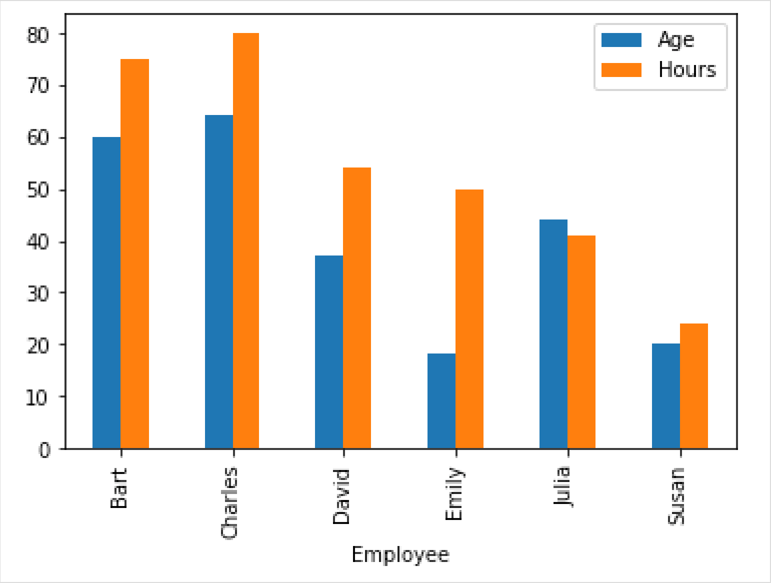


	Employee	Hours
5	Susan	24
4	Julia	41
3	Emily	50
2	David	54
0	Bart	75
1	Charles	80

You can also import **matplotlib.pyplot** to visualize your data in graphs. For instance, the following example visualizes the age and hours from the table. Add the following code to the first example before visualizing data:

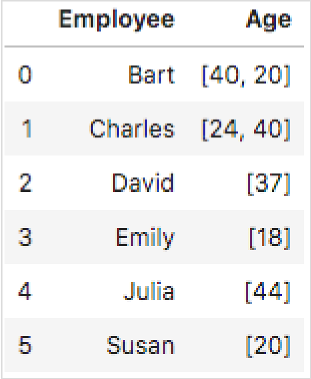
```
import matplotlib.pyplot as plt
```

```
plt.clf()
df.groupby('Employee').sum().plot(kind='bar')
plt.show()
```



The next example will display values of every group according to their ages:

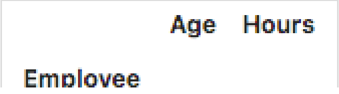
```
df.groupby('Employee')['Age'].apply(lambda group_series:
group_series.tolist()).reset_index()
```



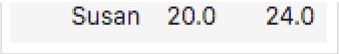
	Employee	Age
0	Bart	[40, 20]
1	Charles	[24, 40]
2	David	[37]
3	Emily	[18]
4	Julia	[44]
5	Susan	[20]

The following example shows how to use the collections you create with Pandas `groupby` and count their **average value**. It keeps the individual values unchanged.

```
df.groupby(['Employee']).mean()
```



	Age	Hours
Employee		

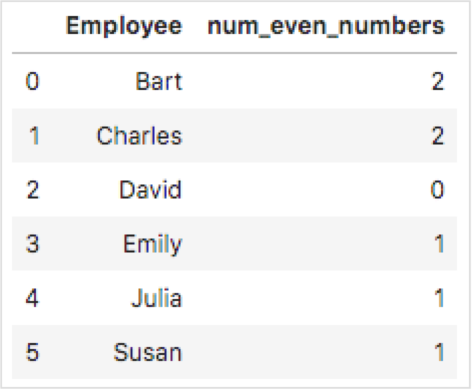


Susan	20.0	24.0

You can also find the number of **even numbers** in your groups. However, before you can complete this task with the Python group by function, you need to define the method for it to work:

Example Copy

```
def count_even_numbers(series):
  return len([elem for elem in series if elem % 2 == 0 ])
df.groupby('Employee')
['Age'].apply(count_even_numbers).reset_index(name='num_even_numbers')
```



	Employee	num_even_numbers
0	Bart	2
1	Charles	2
2	David	0
3	Emily	1
4	Julia	1
5	Susan	1

Filtering

You can filter data according to the age of people as well. For instance, this code will only include people that are **younger than 30**:

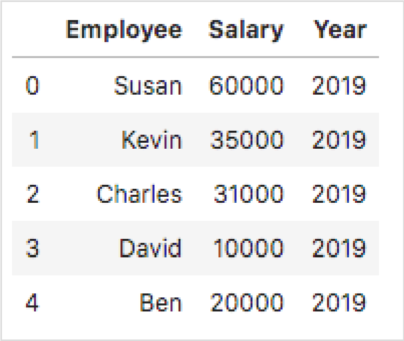
Example Copy

```
df_filtered = df.query('Age < 30')
print(df_filtered)
```

The filtering operation selects data groups by using true/false conditions. Here's another example of a simple **DataFrame** that consists of **employee**, **salary**, and **year** variables:

Example Copy

```
import pandas as pd
import numpy as np
df = pd.DataFrame( {
"Employee" : ["Susan", "Kevin", "Charles", "David", "Ben"] ,
"Salary" : [60000, 35000, 31000, 10000, 20000] ,
"Year" : [2019, 2019, 2019, 2019, 2019]} )
df
```



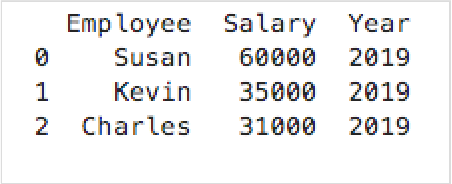
	Employee	Salary	Year
0	Susan	60000	2019
1	Kevin	35000	2019
2	Charles	31000	2019
3	David	10000	2019
4	Ben	20000	2019

The following code will **filter employees** according to their salary:

Example Copy

```
df_filtered = df.query('Salary > 30000')
print(df_filtered)
```

Here is the output after we apply the filtering function:



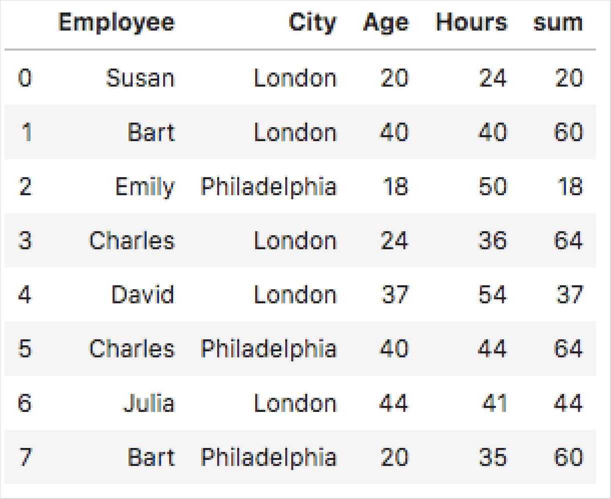
```
   Employee  Salary  Year
0     Susan   60000  2019
1     Kevin   35000  2019
2   Charles   31000  2019
```

Combining results

You might need to **group data**, **apply** a specific function on these collections, and then place it with the **original data**. For this purpose, you can use `transform()` . In the following example, we are grouping employees according to their age, adding values together and including a column of the **sum** to the table:

Example Copy

```
df['sum']=df.groupby(['Employee'])['Age'].transform('sum')
df
```



	Employee	City	Age	Hours	sum
0	Susan	London	20	24	20
1	Bart	London	40	40	60
2	Emily	Philadelphia	18	50	18
3	Charles	London	24	36	64
4	David	London	37	54	37
5	Charles	Philadelphia	40	44	64
6	Julia	London	44	41	44
7	Bart	Philadelphia	20	35	60

← Previous Topic

Next Topic →

Related Code Examples

Python

Python

While Loop Python Expressions →

Instantiating a Python Class Object →

Python

Python

Python print() for a String →

The Use of the Python If Else Statement →

Storing Multiple Variables Without Using Python Array →

Python Round Example 1 →

LEARN

Programming Languages

Get Certified

Sitemap

Code Examples

WEB DEVELOPMENT

HTML

CSS

JavaScript

SERVER SIDE

PHP

SQL

TOOLS

Code Editor

Color Picker

Devtools

Git

COURSES

Video Courses

Code Theory

Guides & Tutorials

Free Certifications Online

Online Learning Platforms

Learning Paths

Apply for a Scholarship

COMPANY

About BitDegree

Hiring

White Paper

Blog

Earn With Us

BitDegree Reviews

FAQ

Press Releases

LEGAL

Policies & Legal Documents

Privacy Policy

Cookie Policy

Money Back Guarantee & Refund Policy

Terms of Service

INSTRUCTORS & PARTNERS

Instructor Login

Become an Instructor

Become a Partner

TAP

Sponsor a Student

FOLLOW US

Learn to earn: BitDegree free online courses give you the best online education with a gamified experience. Gain knowledge and get your dream job: learn to earn.