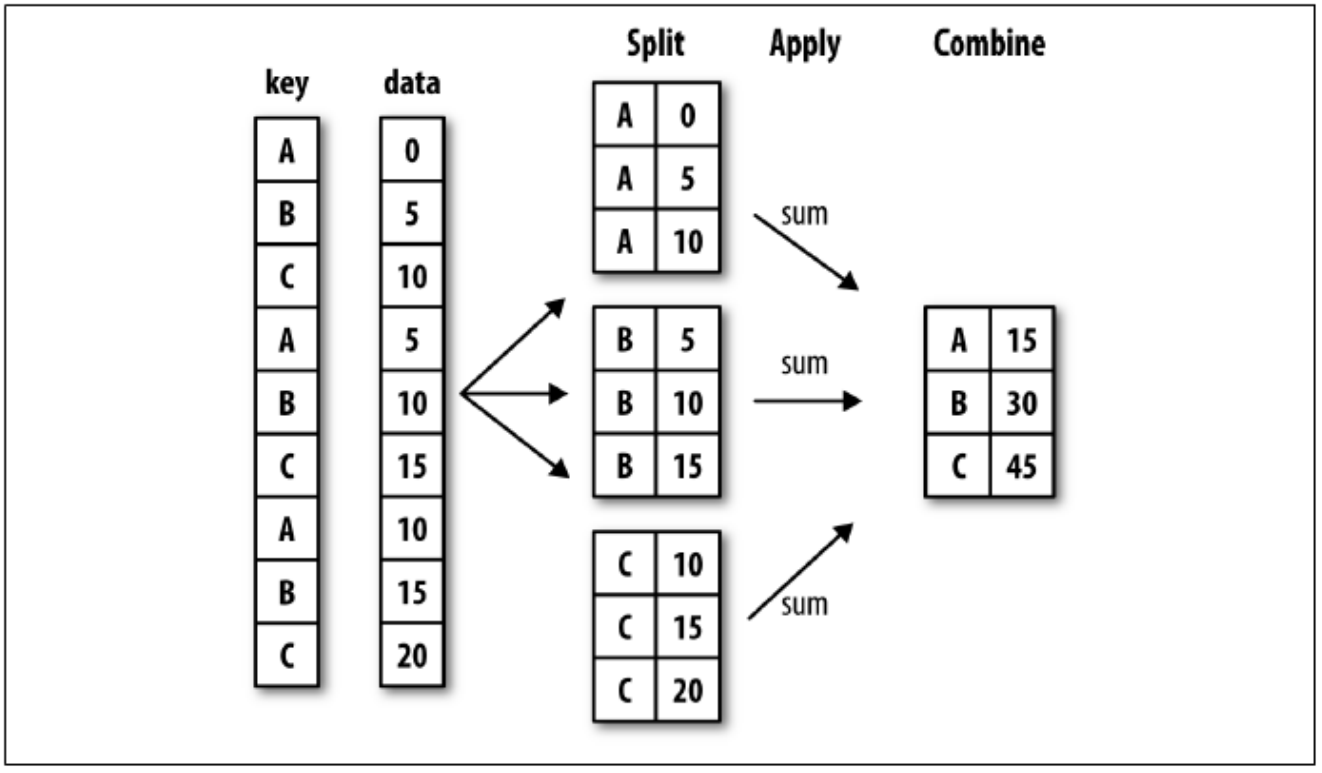
Groupby

Categorizing a dataset and applying a function to each group, whether an aggregation or transformation, is often a critical component of a data analysis workflow. After loading, merging, and preparing a dataset, you may need to compute group statistics or possibly pivot tables for reporting or visualization purposes. pandas provides a flexible groupby interface, enabling you to slice, dice, and summarize datasets in a natural way.

Group Operation : > split-apply-combine

Data-Frame is splitted into groups based on one or more keys that you provide.The splitting is performed on a particular axis of an object. For example, a DataFramecan be grouped on its rows (axis=0) or its columns (axis=1). Once this is done, a function is applied to each group, producing a new value. Finally, the results of all those function applications are combined into a result object.



In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [6]:
```
np.random.seed(0) # for keeping same values generted by random in every execution o
f code below:
```

In [7]:
```
#dic={'key1' : ['a', 'b','a', 'b', 'b', 'a','b','a'],
#                    'key2' : ['one', 'one','two', 'one', 'two', 'one','two','two'],
#                    'data1' : np.arange(8),
#                    'data2' : np.arange(5,13)}
#df = pd.DataFrame(dic)
##############################################

df = pd.DataFrame({'key1' : ['a', 'b','a', 'b', 'b', 'a','b','a'],
                   'key2' : ['one', 'one','two', 'one', 'two', 'one','two','two'],
                   'data1' : np.random.randint(1,20,8),
                   'data2' : np.random.randint(5,25,8)})
df
```

Out[7]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

Suppose you wanted to compute the mean of the data1 column using the labels from key1. There are a number of ways to do this. One is to access data1 and call groupby with the column (a Series) at key1:

In [10]:
```
grouped = df['data1'].groupby(df['key1'])
grouped
```

Out[10]:
```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x0000000008E1EC48>
```

In [11]:
```
grouped.mean()
```

Out[11]:
```
key1
a    10.25
b     8.50
Name: data1, dtype: float64
```

This grouped variable is now a GroupBy object. It has not actually computed anything yet except for some intermediate data about the group key df['key1']. The idea is that this object has all of the information needed to then apply some operation to each of the groups. For example, to compute group means we can call the GroupBy's mean method:

In [12]:
```
#grouped.mean()
df['data1'].groupby(df['key1']).mean()
```

Out[12]:
```
key1
a    10.25
b     8.50
Name: data1, dtype: float64
```

In [16]:
```
# similarly we can use .sum()
# grouped.sum()
df['data1'].groupby(df['key1']).sum()
```

Out[16]:
```
key1
a    41
b    34
Name: data1, dtype: int32
```

The data (a Series) has been aggregated according to the group key, producing a new Series that is now indexed by the unique values in the key1 column. The result index has the name 'key1' because the DataFrame column df['key1'] did

- We can use multiple keys for grouping as well:

In [18]:
```
df
```

Out[18]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

In [19]:
```
sumd = df['data2'].groupby([df['key1'], df['key2']]).sum()
sumd  ## You can use it in 'data1' as well
```

Out[19]:
```
key1  key2
a     one     21
      two     39
b     one     17
      two     30
Name: data2, dtype: int32
```

Here we grouped the data using two keys, and the resulting Series now has a hierarchical index consisting of the unique pairs of keys observed. Above result is stacked form, which can be unstacked as below:

In [20]:
```
aa=sumd.unstack()
aa
```

Out[20]:

key2	one	two
key1		
a	21	39
b	17	30

In [21]:
```
aa.stack() # again converted back to stack form
```

Out[21]:
```
key1  key2
a     one     21
      two     39
b     one     17
      two     30
dtype: int32
```

*** Applying on entire DataFrame**

In [22]:
```
df
```

Out[22]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

In [23]:
```
df.groupby('key1').sum()
```

Out[23]:

	data1	data2
key1		
a	41	60
b	34	47

In this case df.groupby('key1').sum() that there is no key2 column in the result. Because df['key2'] is not numeric data, it is said to be a nuisance column, which is therefore excluded from the result. By default, all of the numeric columns are aggregated.

- Particular data column can be sliced like given below (from entire grouping):

In [24]:
```
df.groupby('key1')['data2'].sum() ## Slicing like this is also possible
```

Out[24]:
```
key1
a    60
b    47
Name: data2, dtype: int32
```

- Grouping on both keys on entire dataframe is also possible:

In [101]:
```
df.groupby(['key1', 'key2']).sum()
```

Out[101]:

		data1	data2
key1	**key2**		
a	**one**	14	30
	two	12	46
b	**one**	11	23
	two	13	34

- Regardless of the objective in using groupby, a generally useful GroupBy method is size, which returns a Series containing group sizes:

In [102]:
```
df.groupby(['key1','key2']).size() ## counting of elements
```

Out[102]:
```
key1  key2
a     one     2
      two     2
b     one     2
      two     2
dtype: int64
```

Iterating Over Groups

In [25]:
```
df
```

Out[25]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

In [28]:
```
df.groupby('key1')
```

Out[28]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000000008E3F848>
```

The GroupBy object supports iteration, generating a sequence of 2-tuples containing the group name along with the chunk of data.

In [29]:
```
lisd=[]
for name,group in df.groupby('key1'):
    print(name)
    print(group)
    lisd.append(group)
```

```
a
  key1 key2  data1  data2
0    a  one     13      9
2    a  two      1     17
5    a  one      8     12
7    a  two     19     22
b
  key1 key2  data1  data2
1    b  one     16     11
3    b  one      4      6
4    b  two      4     11
6    b  two     10     19
```

In [30]:
```
type(group)
```

Out[30]:
```
pandas.core.frame.DataFrame
```

In [110]:
```
type(name)
```

Out[110]:
```
str
```

In [33]:
```
lisd # since we have append both dataframe in this list, each dataframe is a memebe
r of this list.
```

Out[33]:
```
[  key1 key2  data1  data2
 0    a  one     13      9
 2    a  two      1     17
 5    a  one      8     12
 7    a  two     19     22,
   key1 key2  data1  data2
 1    b  one     16     11
 3    b  one      4      6
 4    b  two      4     11
 6    b  two     10     19]
```

In [34]:
```
lisd[0] # Slicing of list for getting particular dataframe'
#lisd[1]
```

Out[34]:

	key1	key2	data1	data2
0	a	one	13	9
2	a	two	1	17
5	a	one	8	12
7	a	two	19	22

In [35]:
```
df
```

Out[35]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

In the case of multiple keys, the first element in the tuple will be a tuple of key values:

In [36]:
```
lisd=[]
for (k1, k2), group in df.groupby(['key1', 'key2']):
    print((k1, k2))
    print(group)
    lisd.append(group)
```

```
('a', 'one')
  key1 key2  data1  data2
0    a  one     13      9
5    a  one      8     12
('a', 'two')
  key1 key2  data1  data2
2    a  two      1     17
7    a  two     19     22
('b', 'one')
  key1 key2  data1  data2
1    b  one     16     11
3    b  one      4      6
('b', 'two')
  key1 key2  data1  data2
4    b  two      4     11
6    b  two     10     19
```

In [119]:
```
d1=lisd[0]
d1
```

Out[119]:

	key1	key2	data1	data2
0	a	one	10	17
5	a	one	4	13

In [116]:
```
lisd[1]
```

Out[116]:

	key1	key2	data1	data2
2	a	two	5	24
7	a	two	7	22

In [117]:
```
lisd[2]
```

Out[117]:

	key1	key2	data1	data2
1	b	one	9	12
3	b	one	2	11

In [118]:
```
lisd[3]
```

Out[118]:

	key1	key2	data1	data2
4	b	two	2	24
6	b	two	11	10

- You can choose to do whatever you want with the pieces of data. Computing a dict of the data pieces as a one-liner may be useful.

In [37]:
```
df
```

Out[37]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

In [125]:
```
df.groupby('key1')
```

Out[125]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000000000904FD08>
```

In [120]:
```
list(df.groupby('key1'))
```

Out[120]:
```
[('a',
    key1 key2  data1  data2
  0    a  one     10     17
  2    a  two      5     24
  5    a  one      4     13
  7    a  two      7     22),
 ('b',
    key1 key2  data1  data2
  1    b  one      9     12
  3    b  one      2     11
  4    b  two      2     24
  6    b  two     11     10)]
```

In [38]:
```
pieces = dict(list(df.groupby('key1')))
pieces
```

Out[38]:
```
{'a':   key1 key2  data1  data2
 0    a  one     13      9
 2    a  two      1     17
 5    a  one      8     12
 7    a  two     19     22,
 'b':   key1 key2  data1  data2
 1    b  one     16     11
 3    b  one      4      6
 4    b  two      4     11
 6    b  two     10     19}
```

In [39]:
```
pieces['a'] # getting values ( dataframe in this case) of dictionary by using keys
 of dictionary
```

Out[39]:

	key1	key2	data1	data2
0	a	one	13	9
2	a	two	1	17
5	a	one	8	12
7	a	two	19	22

In [40]:
```
pieces['b']
```

Out[40]:

	key1	key2	data1	data2
1	b	one	16	11
3	b	one	4	6
4	b	two	4	11
6	b	two	10	19

- By default groupby groups on axis=0, but you can group on any of the other axes.For example, we could group the columns of our example df here by dtype like so:

In [45]:
```
df
```

Out[45]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

In [42]:
```
df.dtypes
```

Out[42]:
```
key1     object
key2     object
data1     int32
data2     int32
dtype: object
```

In [52]:
```
grouped = df.groupby(df.dtypes, axis=1) # grouping of data based on data type
# Here axis=1 is used as data types are based on column
grouped
```

Out[52]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000000008E820C8>
```

In [53]:
```
for dtype, group in grouped: ## checking by iterating over grouped
    print(dtype)
    print(group)
```

```
int32
   data1  data2
0     13      9
1     16     11
2      1     17
3      4      6
4      4     11
5      8     12
6     10     19
7     19     22
object
  key1 key2
0    a  one
1    b  one
2    a  two
3    b  one
4    b  two
5    a  one
6    b  two
7    a  two
```

Selection :> Column or Subset of Columns

In [54]:
```
df
```

Out[54]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

In [60]:
```
a=df.groupby('key1')[['data1']].sum()  # Double square bracket -> results in DataFr
ame
a
#b=df.groupby('key1')['data1'].sum()  # Single square bracket -> results in series
#b
## notice single and double square bracket
```

Out[60]:

key1	data1
a	41
b	34

In [58]:
```
type(a)
```

Out[58]:
```
pandas.core.frame.DataFrame
```

Above operation is similar as operation given below

In [59]:
```
df['data1'].groupby(df['key1']).sum()   ## notice single and double square bracket
#df[['data2']].groupby(df['key1']).sum()
```

Out[59]:
```
key1
a    41
b    34
Name: data1, dtype: int32
```

- Notice the differences in below two syntax resulting the same ouptput:

In [65]:
```
a1=df.groupby(['key1', 'key2'])['data2'].mean()
a1
```

Out[65]:
```
key1  key2
a     one     10.5
      two     19.5
b     one      8.5
      two     15.0
Name: data2, dtype: float64
```

In [66]:
```
a1=df['data2'].groupby([df['key1'], df['key2']]).mean()
a1
```

Out[66]:
```
key1  key2
a     one     10.5
      two     19.5
b     one      8.5
      two     15.0
Name: data2, dtype: float64
```

Note : In above syntax --> If you slice the dataframe at first then, inside groupby function you have to also pass the keys as slice of dataframe.

2.Data Aggregation

Aggregations refer to any data transformation that produces scalar values from arrays. The preceding examples have used several of them, including mean, count, min, and sum.Many common aggregations, such as those found in Table below have optimized implementations. However, you are not limited to only this set of methods.

Function name	Description
`count`	Number of non-NA values in the group
`sum`	Sum of non-NA values
`mean`	Mean of non-NA values
`median`	Arithmetic median of non-NA values
`std, var`	Unbiased (n – 1 denominator) standard deviation and variance
`min, max`	Minimum and maximum of non-NA values
`prod`	Product of non-NA values
`first, last`	First and last non-NA values

In [67]:
```
df
```

Out[67]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

In [68]:
```
df.groupby('key1')['data1'].sum()
```

Out[68]:
```
key1
a    41
b    34
Name: data1, dtype: int32
```

In [69]:
```
df.groupby('key1')['data1'].agg('sum')
```

Out[69]:
```
key1
a    41
b    34
Name: data1, dtype: int32
```

To use your own aggregation functions, pass any function that aggregates an array to the aggregate or agg method:

*** Basically agg method is used for passing our own function for aggregation**

In [70]:
```
df
```

Out[70]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

In [71]:
```
def peak_to_peak(x):       ## We already know input argument type and nature,
    return x.max() - x.min()
```

In [74]:
```
peak_to_peak(10) # will  give error as this function acceptsonlt  array or series
as 'x'.
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-74-28a833ad74ee> in <module>
----> 1 peak_to_peak(10) # will  give error as this function acceptsonlt  array or
series as 'x'.

<ipython-input-71-865433b9cf9a> in peak_to_peak(x)
      1 def peak_to_peak(x):       ## We already know input argument type and natu
re
----> 2     return x.max() - x.min()

AttributeError: 'int' object has no attribute 'max'
```

In [75]:
```
peak_to_peak(np.array([7,1,2,3,4]))
```

Out[75]: 6

In [76]:
```
X=np.array([5,1,2,3,4])
```

In [77]:
```
peak_to_peak(X)
```

Out[77]: 4

Passing user defined 'peak_to_peak' function into .agg()

In [78]:
```
df
```

Out[78]:

	key1	key2	data1	data2
0	a	one	13	9
1	b	one	16	11
2	a	two	1	17
3	b	one	4	6
4	b	two	4	11
5	a	one	8	12
6	b	two	10	19
7	a	two	19	22

In [79]:
```
df.groupby('key1').agg(peak_to_peak)
```

Out[79]:

	data1	data2
key1		
a	18	13
b	12	13

Or we can store grouped result and then pass any function inside .agg()

In [80]:
```
grouped = df.groupby('key1')
```

In [81]:
```
grouped.agg(peak_to_peak)
```

Out[81]:

	data1	data2
key1		
a	18	13
b	12	13

In [82]:
```
df.groupby('key1').agg(peak_to_peak)
```

Out[82]:

	data1	data2
key1		
a	18	13
b	12	13

OR, using lambda functions as given below :>

In [83]:
```
f=lambda x:x.max()-x.min()
grouped.agg(f)
```

Out[83]:

	data1	data2
key1		
a	18	13
b	12	13

How about using describe on grouped result

In [84]:
```
grouped = df.groupby('key1')
grouped
```

Out[84]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000000008F3F548>
```

In [85]:
```
grouped.describe()
```

Out[85]:

	data1								data2					
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**	**std**	**min**	**25%**	**50%**
key1														
a	4.0	10.25	7.632169	1.0	6.25	10.5	14.5	19.0	4.0	15.00	5.715476	9.0	11.25	14.5
b	4.0	8.50	5.744563	4.0	4.00	7.0	11.5	16.0	4.0	11.75	5.377422	6.0	9.75	11.0

In [86]:
```
df.describe()
```

Out[86]:

	data1	data2
count	8.000000	8.000000
mean	9.375000	13.375000
std	6.323143	5.423165
min	1.000000	6.000000
25%	4.000000	10.500000
50%	9.000000	11.500000
75%	13.750000	17.500000
max	19.000000	22.000000

Example with more funtions :

In [91]:
```
tips = pd.read_csv('tips.csv')
tips.head()
```

Out[91]:

	total_bill	tip	smoker	day	time	size
0	16.99	1.01	No	Sun	Dinner	2
1	10.34	1.66	No	Sun	Dinner	3
2	21.01	3.50	No	Sun	Dinner	3
3	23.68	3.31	No	Sun	Dinner	2
4	24.59	3.61	No	Sun	Dinner	4

In [92]:
```
# Add tip percentage of total bill
tips['tip_pct'] = tips['tip']*100 / tips['total_bill']
tips.head()
```

Out[92]:

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	5.944673
1	10.34	1.66	No	Sun	Dinner	3	16.054159
2	21.01	3.50	No	Sun	Dinner	3	16.658734
3	23.68	3.31	No	Sun	Dinner	2	13.978041
4	24.59	3.61	No	Sun	Dinner	4	14.680765

For descriptive statistics we can pass the name of the function as a string:

In [93]:
```
tips.groupby([ 'smoker','day'])['total_bill'].agg('sum')
```

Out[93]:
```
smoker  day
No      Fri       73.68
        Sat      884.78
        Sun     1168.88
        Thur     770.09
Yes     Fri      252.20
        Sat      893.62
        Sun      458.28
        Thur     326.24
Name: total_bill, dtype: float64
```

In [94]:
```
tips.groupby([ 'smoker','day'])['total_bill'].sum() ## or simply --> .sum()
```

Out[94]:
```
smoker  day
No      Fri       73.68
        Sat      884.78
        Sun     1168.88
        Thur     770.09
Yes     Fri      252.20
        Sat      893.62
        Sun      458.28
        Thur     326.24
Name: total_bill, dtype: float64
```

In [95]:
```
grouped=tips.groupby([ 'smoker','day'])
```

In [96]:
```
grouped_tip= grouped['tip'] ## total_bill, tip_pct
grouped_tip.agg('mean')
```

Out[96]:
```
smoker  day
No      Fri     2.812500
        Sat     3.102889
        Sun     3.167895
        Thur    2.673778
Yes     Fri     2.714000
        Sat     2.875476
        Sun     3.516842
        Thur    3.030000
Name: tip, dtype: float64
```

In [97]:
```
def peak_to_peak(x):
    return x.max() - x.min()
```

In [98]:
```
f=lambda x:x.max()-x.min()
```

In [99]:
```
grouped = tips.groupby(['day', 'smoker'])
grouped_total_bill = grouped['total_bill']
```

In [100]:
```
grouped_total_bill.agg(['mean', 'std', peak_to_peak])
```

Out[100]:

		mean	std	peak_to_peak
day	**smoker**			
Fri	**No**	18.420000	5.059282	10.29
	Yes	16.813333	9.086388	34.42
Sat	**No**	19.661778	8.939181	41.08
	Yes	21.276667	10.069138	47.74
Sun	**No**	20.506667	8.130189	39.40
	Yes	24.120000	10.442511	38.10
Thur	**No**	17.113111	7.721728	33.68
	Yes	19.190588	8.355149	32.77

In [101]:
```
grouped_tip.agg(['mean', 'std',f])
```

Out[101]:

smoker	day	mean	std	<lambda_0>
No	Fri	2.812500	0.898494	2.00
	Sat	3.102889	1.642088	8.00
	Sun	3.167895	1.224785	4.99
	Thur	2.673778	1.282964	5.45
Yes	Fri	2.714000	1.077668	3.73
	Sat	2.875476	1.630580	9.00
	Sun	3.516842	1.261151	5.00
	Thur	3.030000	1.113491	3.00

You don't need to accept the names that GroupBy gives to the columns; notably, lambda functions have the name '', which makes them hard to identify (you can see for yourself by looking at a function's **name** attribute). Thus, if you pass a list of (name, function) tuples, the first element of each tuple will be used as the DataFrame column names (you can think of a list of 2-tuples as an ordered mapping):

In [102]:
```
grouped_pct= grouped['tip_pct']
```

In [103]:
```
grouped_pct.agg([('Average', 'mean'), ('standard dev', 'std'),('max-min',f)])
```

Out[103]:

day	smoker	Average	standard dev	max-min
Fri	No	15.165044	2.812295	6.734944
	Yes	17.478305	5.129267	15.992499
Sat	No	15.804766	3.976730	23.519300
	Yes	14.790607	6.137495	29.009476
Sun	No	16.011294	4.234723	19.322576
	Yes	18.725032	15.413424	64.468495
Thur	No	16.029808	3.877420	19.335021
	Yes	16.386327	3.938881	15.124046

In [104]:
```
grouped_tip.agg([('Average', 'mean'), ('standard dev', 'std'),('max-min', peak_to_p
eak)])
```

Out[104]:

smoker	day	Average	standard dev	max-min
No	Fri	2.812500	0.898494	2.00
	Sat	3.102889	1.642088	8.00
	Sun	3.167895	1.224785	4.99
	Thur	2.673778	1.282964	5.45
Yes	Fri	2.714000	1.077668	3.73
	Sat	2.875476	1.630580	9.00
	Sun	3.516842	1.261151	5.00
	Thur	3.030000	1.113491	3.00

In [194]:
```
tips.head()
```

Out[194]:

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	5.944673
1	10.34	1.66	No	Sun	Dinner	3	16.054159
2	21.01	3.50	No	Sun	Dinner	3	16.658734
3	23.68	3.31	No	Sun	Dinner	2	13.978041
4	24.59	3.61	No	Sun	Dinner	4	14.680765

In [105]:
```
grouped = tips.groupby([ 'smoker','day'])
```

In [106]:
```
#functions = [('Number of bills','count'), 'mean', 'max','sum']
result = grouped[['tip_pct', 'total_bill','size']].agg([('Number of bills','count'
), 'mean', 'max','sum'])#.agg(functions)
result
```

Out[106]:

		tip_pct				total_bill				size
		Number of bills	mean	max	sum	Number of bills	mean	max	sum	Number of bills
smoker	day									
No	Fri	4	15.165044	18.773467	60.660177	4	18.420000	22.75	73.68	4
	Sat	45	15.804766	29.198966	711.214459	45	19.661778	48.33	884.78	45
	Sun	57	16.011294	25.267250	912.643775	57	20.506667	48.17	1168.88	57
	Thur	45	16.029808	26.631158	721.341368	45	17.113111	41.19	770.09	45
Yes	Fri	15	17.478305	26.348039	262.174578	15	16.813333	40.17	252.20	15
	Sat	42	14.790607	32.573290	621.205474	42	21.276667	50.81	893.62	42
	Sun	19	18.725032	71.034483	355.775601	19	24.120000	45.35	458.28	19
	Thur	17	16.386327	24.125452	278.567563	17	19.190588	43.11	326.24	17

To save pandas table as excel file

In [107]:
```
result.to_excel(r'C:\Users\Ramendra\Desktop\ramen.xlsx') #  pass your own path
```

*** We can pass dictionary inside agg funtion, where keys are numerical columns name and values are opeartion to be done.**

In [108]:
```
grouped = tips.groupby([ 'smoker','day'])
```

In [109]:
```
grouped
```

Out[109]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000000007E38D08>
```

In [110]:
```
grouped.agg({'total_bill' : ['min', 'max', 'mean', 'std'],'size' : 'sum'})
```

Out[110]:

		total_bill				size
		min	max	mean	std	sum
smoker	day					
No	Fri	12.46	22.75	18.420000	5.059282	9
	Sat	7.25	48.33	19.661778	8.939181	115
	Sun	8.77	48.17	20.506667	8.130189	167
	Thur	7.51	41.19	17.113111	7.721728	112
Yes	Fri	5.75	40.17	16.813333	9.086388	31
	Sat	3.07	50.81	21.276667	10.069138	104
	Sun	7.25	45.35	24.120000	10.442511	49
	Thur	10.34	43.11	19.190588	8.355149	40

In [41]:
```
data_tip=tips.groupby(['day', 'smoker'], as_index=True).mean()
data_tip # by default it is True
```

Out[41]:

		total_bill	tip	size	tip_pct
day	smoker				
Fri	No	18.420000	2.812500	2.250000	15.165044
	Yes	16.813333	2.714000	2.066667	17.478305
Sat	No	19.661778	3.102889	2.555556	15.804766
	Yes	21.276667	2.875476	2.476190	14.790607
Sun	No	20.506667	3.167895	2.929825	16.011294
	Yes	24.120000	3.516842	2.578947	18.725032
Thur	No	17.113111	2.673778	2.488889	16.029808
	Yes	19.190588	3.030000	2.352941	16.386327

In [111]:
```
data_tip=tips.groupby(['day', 'smoker'], as_index=False).mean()
data_tip  # Notice while passing False, grouped keys don't appear as index.
```

Out[111]:

	day	smoker	total_bill	tip	size	tip_pct
0	Fri	No	18.420000	2.812500	2.250000	15.165044
1	Fri	Yes	16.813333	2.714000	2.066667	17.478305
2	Sat	No	19.661778	3.102889	2.555556	15.804766
3	Sat	Yes	21.276667	2.875476	2.476190	14.790607
4	Sun	No	20.506667	3.167895	2.929825	16.011294
5	Sun	Yes	24.120000	3.516842	2.578947	18.725032
6	Thur	No	17.113111	2.673778	2.488889	16.029808
7	Thur	Yes	19.190588	3.030000	2.352941	16.386327

This much for this module.

Feel Free to Share and Distribute.

Don't forget to follow me for more such stuff.

https://github.com/Rami-RK/Python_Starter (https://github.com/Rami-RK/Python_Starter)

https://www.linkedin.com/in/ramendra-kumar-57334478 (https://www.linkedin.com/in/ramendra-kumar-57334478)

Reference: Python for Data Analysis, McKinney

Thank You !