Group operations

Data aggregation, grouping, and pivoting

Kunal Khurana

2024-03-01

Table of contents

How to think about group operations

- Iterating over groups
- Selecting a column or subset of columns
- Grouping with dictionaries and series
- Grouping with functions
- Grouping by Index levels

Data Aggregation

- Column-wise and Multiple Function Application
- Returning aggregated Data without Row Indexes

Apply: General split-apply-combine

- Suppressing the Group keys
- Quantile and Bucked analysis
- Filling missing and group specific values
- Random sampling and permutation
- group Weighted Average and Correlation
- group wise linear regression

Group Transformations adn 'Unwrapped' GroupBys

Pivot tables adn Cross-tabulation

- cross-tabulations: crosstab

```
import numpy as np
import pandas as pd
```

```
df = pd.DataFrame({"key1": ['a', 'a', None, 'b', 'c'],
                  'key2' : pd.Series([1, 2, 3, None, 5],
                                     dtype='Int64'),
                   'data1' : np.random.standard_normal(5),
                   'data2' : np.random.standard_normal(5),


                                     })
```

```
df
```

	key1	key2	data1	data2
0	a	1	-0.949232	-0.859027
1	a	2	0.902370	0.987569

	key1	key2	data1	data2
2	None	3	0.805328	-0.892250
3	b	<NA>	0.269996	0.291597
4	c	5	-1.341779	-0.229023

```python
 # mean of data1 using labels from key1

grouped = df["data1"].groupby(df['key1'])
```

```python
grouped
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x0000018F93327910>
```

```python
means = df['data1'].groupby([df['key1'], df["key2"]]).mean()
```

```python
means
```

```
key1  key2
a     1       -0.949232
      2        0.902370
c     5       -1.341779
Name: data1, dtype: float64
```

```python
means.unstack()
```

key2 key1	1	2	5
a	-0.949232	0.90237	NaN
c	NaN	NaN	-1.341779

```python
# using Series as groupkeys
states = np.array(['PB', 'DL', 'UK', 'HP', 'UP',])
years = [2001, 2002, 2005, 2001, 2009,]

df['data1'].groupby([states, years]).mean()
```

```
DL  2002    0.902370
HP  2001    0.269996
PB  2001   -0.949232
UK  2005    0.805328
UP  2009   -1.341779
Name: data1, dtype: float64
```

```python
grouped.mean()
```

```
key1
a   -0.023431
b    0.269996
c   -1.341779
Name: data1, dtype: float64
```

```python
means2 = df['data1'].groupby([df['key1'], df['key2']]).mean()

means2
```

```
key1  key2
a     1      -0.949232
      2       0.902370
c     5      -1.341779
Name: data1, dtype: float64
```

```python
means2.unstack()
```

key2 key1	1	2	5
a	-0.949232	0.90237	NaN
c	NaN	NaN	-1.341779

```python
means2.unstack(0)
```

key1 key2	a	c
1	-0.949232	NaN
2	0.902370	NaN
5	NaN	-1.341779

```
df.groupby('key1').mean()
```

key1	key2	data1	data2
a	1.5	-0.023431	0.064271
b	<NA>	0.269996	0.291597
c	5.0	-1.341779	-0.229023

```
df
```

	key1	key2	data1	data2
0	a	1	-0.949232	-0.859027
1	a	2	0.902370	0.987569
2	None	3	0.805328	-0.892250
3	b	<NA>	0.269996	0.291597
4	c	5	-1.341779	-0.229023

```
df.groupby(['key1', 'key2']).mean()
```

key1	key2	data1	data2
a	1	-0.949232	-0.859027
	2	0.902370	0.987569
c	5	-1.341779	-0.229023

```
df.groupby(['key1', 'key2']).size()
```

```
key1  key2
```

```
a     1       1
      2       1
c     5       1
dtype: int64
```

```python
df.groupby('key1', dropna= False).size()
```

```
key1
a      2
b      1
c      1
NaN    1
dtype: int64
```

```python
df.groupby(['key1', 'key2'], dropna=False).size()
```

```
key1  key2
a     1       1
      2       1
b     <NA>    1
c     5       1
NaN   3       1
dtype: int64
```

```python
df.groupby('key1').count()
```

key1	key2	data1	data2
a	2	2	2
b	0	1	1
c	1	1	1

Iterating over groups

```
for name, group in df.groupby('key1'):
    print(name)
    print(group)
```

```
a
  key1  key2     data1     data2
0    a     1 -0.949232 -0.859027
1    a     2  0.902370  0.987569
b
  key1  key2     data1     data2
3    b  <NA>  0.269996  0.291597
c
  key1  key2     data1     data2
4    c     5 -1.341779 -0.229023
```

```
# in case of multiple key elements
for (k1, k2), group in df.groupby(['key1','key2']):
    print((k1, k2))
    print(group)
```

```
('a', 1)
  key1  key2     data1     data2
0    a     1 -0.949232 -0.859027
('a', 2)
  key1  key2    data1     data2
1    a     2  0.90237  0.987569
('c', 5)
  key1  key2     data1     data2
4    c     5 -1.341779 -0.229023
```

dictionary data in the form of pieces

```
pieces = {name : group for name, group in df.groupby('key1')}
pieces['b']
```

	key1	key2	data1	data2
3	b	<NA>	0.269996	0.291597

```
pieces['c']
```

	key1	key2	data1	data2
4	c	5	-1.341779	-0.229023

```
grouped = df.groupby({'key1': 'key','key2': 'key',
                      'data1': 'data', 'data2': 'data'})
```

```
grouped
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000018F9795F9A0>
```

```
for group_key, group_values in grouped:
    print(group_key)
    print(group_values)
```

Selecting a Column or Subset of Columns

```
df.groupby('key1')['data1']
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x0000018F978AF040>
```

```
df.groupby('key1')[['data2']]
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000018F97822580>
```

```
# this can also be called by-
df['data1'].groupby(df['key1'])
```

```
df['data2'].groupby(df['key1'])
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x0000018F978EE970>
```

```
df.groupby(['key1', 'key2'])[['data2']].mean()
```

		data2
key1	key2	
a	1	-0.859027
	2	0.987569
c	5	-0.229023

```
s_grouped = df.groupby(['key1', 'key2'])['data2']
```

```
s_grouped
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x0000018F97770520>
```

```
s_grouped.mean()
```

```
key1  key2
a     1      -0.859027
      2       0.987569
c     5      -0.229023
Name: data2, dtype: float64
```

Grouping with Dictionaries and Series

```
people = pd.DataFrame(np.random.standard_normal((5,5)),
                     columns=['a', 'b', 'c', 'd','e'],
                     index = ['Kunal', 'Rahul', 'Sachin', 'Sorav', 'Andrew'])
```

```
# add few NA values
people.iloc[2:3, [1, 2]] = np.nan
```

```python
people
```

	a	b	c	d	e
Kunal	0.168348	-1.118059	-0.800172	-0.866632	-0.197982
Rahul	0.200227	0.640015	1.108393	-0.085850	-0.888133
Sachin	0.847146	NaN	NaN	0.463212	-0.077719
Sorav	1.440603	0.431537	-0.670088	-0.121215	-1.211302
Andrew	-0.637420	0.237133	-0.426589	-0.631543	0.098273

```python
mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
           'd': 'blue', 'e': 'red', 'f': 'orange'}
```

```python
by_column = people.groupby(mapping, axis= 'columns')
by_column.sum()
```

	blue	red
Kunal	-1.666804	-1.147693
Rahul	1.022543	-0.047891
Sachin	0.463212	0.769427
Sorav	-0.791303	0.660838
Andrew	-1.058132	-0.302014

```python
# same functionality holds for Series

map_series = pd.Series(mapping)

map_series
```

```
a       red
b       red
c      blue
d      blue
e       red
f    orange
dtype: object
```

```
people.groupby(map_series, axis= 'columns').count()
```

	blue	red
Kunal	2	3
Rahul	2	3
Sachin	1	2
Sorav	2	3
Andrew	2	3

Grouping with Functions

```
people.groupby(len).sum()
```

	a	b	c	d	e
5	1.809178	-0.046507	-0.361866	-1.073697	-2.297416
6	0.209726	0.237133	-0.426589	-0.168331	0.020554

```
# mixing functions with arrays, dicitonaries, or Series

key_list = ['one', 'one', 'two', 'one', 'two']

people.groupby([len, key_list]).min()
```

		a	b	c	d	e
5	one	0.168348	-1.118059	-0.800172	-0.866632	-1.211302
6	two	-0.637420	0.237133	-0.426589	-0.631543	-0.077719

Grouping with Index levels

```
levels = pd.MultiIndex.from_arrays([['IN', 'US', 'CAN'],
                                   [1,2,3]],
                                   names=['fdk', 'goleala'])
```

```
hier_df = pd.DataFrame(np.random.standard_normal((4,3)),
                      columns = levels)

hier_df
```

fdk	IN	US	CAN
goleala	1	2	3
0	-0.973483	0.175499	-1.375942
1	-0.417278	-0.593448	-0.097291
2	-1.864057	-1.157927	0.219800
3	-0.062763	0.796581	-0.905844

```
# to groupby level, pass level keyword
```

```
hier_df.groupby(level = 'fdk', axis = 'columns').count()
```

fdk	CAN	IN	US
0	1	1	1
1	1	1	1
2	1	1	1
3	1	1	1

Data Aggregation

- transformation that produces scalar values
- optimised methods

```
df
```

	key1	key2	data1	data2
0	a	1	-0.949232	-0.859027
1	a	2	0.902370	0.987569
2	None	3	0.805328	-0.892250
3	b	<NA>	0.269996	0.291597
4	c	5	-1.341779	-0.229023

```
grouped = df.groupby('key1')
```

```
grouped['data1'].nsmallest(2)
```

```
key1
a     0   -0.949232
      1    0.902370
b     3    0.269996
c     4   -1.341779
Name: data1, dtype: float64
```

```
# aggregation function
def peak_to_peak(arr):
    return arr.max() - arr.min()
```

```
grouped.agg(peak_to_peak)
```

	key2	data1	data2
key1			
a	1	1.851602	1.846596
b	<NA>	0.000000	0.000000
c	0	0.000000	0.000000

```
grouped.describe()
```

	key2								data1			
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%
key1												
a	2.0	1.5	0.707107	1.0	1.25	1.5	1.75	2.0	2.0	-0.023431	...	0.43
b	0.0	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	1.0	0.269996	...	0.26
c	1.0	5.0	<NA>	5.0	5.0	5.0	5.0	5.0	1.0	-1.341779	...	-1.3

Column-Wise multiple function application

```python
iris = pd.read_csv(r"E:\pythonfordatanalysis\semainedu26fevrier\iris.csv")
```

```python
iris.head()
```

	Id	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```python
iris.columns
```

```
Index(['Id', 'Sepal Length (cm)', 'Sepal Width (cm)', 'Petal Length (cm)',
       'Petal Width (cm)', 'Species'],
      dtype='object')
```

```python
grouped2 = iris.groupby(['Sepal Length (cm)', 'Sepal Width (cm)', 'Petal Length (cm)',
       'Petal Width (cm)'])
```

```python
missing_values = iris.isna()

missing_values.sum = iris.isna().sum()

missing_values_percent = (iris.isna().sum() / len(iris)) * 100

iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Id                 150 non-null    int64
 1   Sepal Length (cm)  150 non-null    float64
```

```
 2   Sepal Width (cm)   150 non-null    float64
 3   Petal Length (cm)  150 non-null    float64
 4   Petal Width (cm)   150 non-null    float64
 5   Species            150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
grouped.agg(['mean', 'std', peak_to_peak])
```

	key2			data1			data2		
	mean	std	peak_to_peak	mean	std	peak_to_peak	mean	std	pea
key1									
a	1.5	0.707107	1	-0.023431	1.30928	1.851602	0.064271	1.305741	1.84
b	<NA>	<NA>	<NA>	0.269996	NaN	0.000000	0.291597	NaN	0.00
c	5.0	<NA>	0	-1.341779	NaN	0.000000	-0.229023	NaN	0.00

```
# dropping column as it gave traceback before
iris2 = iris.drop('Species', axis=1)
```

```
iris2.agg(['mean', 'std', peak_to_peak])
```

	Id	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
peak_to_peak	149.000000	3.600000	2.400000	5.900000	2.400000

Apply: General split-apply-combine

```
def top(iris2, n=5, column = 'Sepal Length (cm)'):
    return iris2.sort_values(column, ascending= False)[:n]
```

```
top(iris2, n=6)
```

	Id	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)
131	132	7.9	3.8	6.4	2.0
135	136	7.7	3.0	6.1	2.3
122	123	7.7	2.8	6.7	2.0
117	118	7.7	3.8	6.7	2.2
118	119	7.7	2.6	6.9	2.3
105	106	7.6	3.0	6.6	2.1

```
iris2.groupby('Sepal Length (cm)').apply(top)
```

		Id	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (c
Sepal Length (cm)						
4.3	13	14	4.3	3.0	1.1	0.1
	8	9	4.4	2.9	1.4	0.2
4.4	38	39	4.4	3.0	1.3	0.2
	42	43	4.4	3.2	1.3	0.2
4.5	41	42	4.5	2.3	1.3	0.3
...	...	...	...	...	...	...
	117	118	7.7	3.8	6.7	2.2
7.7	118	119	7.7	2.6	6.9	2.3
	122	123	7.7	2.8	6.7	2.0
	135	136	7.7	3.0	6.1	2.3
7.9	131	132	7.9	3.8	6.4	2.0

```
# recall by groupby object

result = iris2.groupby("Sepal Length (cm)").describe()

result
```

	Id								Sepal Width (cm)	
	count	mean	std	min	25%	50%	75%	max	count	mean
Sepal Length (cm)										
4.3	1.0	14.000000	NaN	14.0	14.00	14.0	14.00	14.0	1.0	3.00000
4.4	3.0	30.333333	18.583146	9.0	24.00	39.0	41.00	43.0	3.0	3.03333
4.5	1.0	42.000000	NaN	42.0	42.00	42.0	42.00	42.0	1.0	2.30000

	Id								Sepal Width (cm)	
	count	mean	std	min	25%	50%	75%	max	count	mean
Sepal Length (cm)										
4.6	4.0	20.500000	20.141168	4.0	6.25	15.0	29.25	48.0	4.0	3.32500
4.7	2.0	16.500000	19.091883	3.0	9.75	16.5	23.25	30.0	2.0	3.20000
4.8	5.0	25.400000	14.046352	12.0	13.00	25.0	31.00	46.0	5.0	3.18000
4.9	6.0	41.666667	37.866432	2.0	16.25	36.5	53.00	107.0	6.0	2.86666
5.0	10.0	39.200000	26.029044	5.0	26.25	38.5	48.50	94.0	10.0	3.12000
5.1	9.0	35.111111	28.117808	1.0	20.00	24.0	45.00	99.0	9.0	3.47777
5.2	4.0	37.500000	15.154757	28.0	28.75	31.0	39.75	60.0	4.0	3.42500
5.3	1.0	49.000000	NaN	49.0	49.00	49.0	49.00	49.0	1.0	3.70000
5.4	6.0	28.666667	29.001149	6.0	12.50	19.0	29.25	85.0	6.0	3.55000
5.5	7.0	67.000000	24.779023	34.0	45.50	81.0	86.00	91.0	7.0	2.84285
5.6	6.0	84.666667	22.060523	65.0	67.75	79.5	93.50	122.0	6.0	2.81666
5.7	8.0	72.250000	37.821951	16.0	46.75	88.0	97.75	114.0	8.0	3.10000
5.8	7.0	88.428571	40.265192	15.0	75.50	93.0	108.50	143.0	7.0	2.88571
5.9	3.0	94.333333	48.418316	62.0	66.50	71.0	110.50	150.0	3.0	3.06666
6.0	6.0	95.166667	28.435307	63.0	80.25	85.0	111.50	139.0	6.0	2.73333
6.1	6.0	94.166667	30.413265	64.0	72.50	83.0	119.00	135.0	6.0	2.85000
6.2	4.0	110.750000	34.798228	69.0	90.75	112.5	132.50	149.0	4.0	2.82500
6.3	9.0	107.222222	30.780586	57.0	88.00	104.0	134.00	147.0	9.0	2.85555
6.4	7.0	107.857143	32.328669	52.0	93.50	116.0	131.00	138.0	7.0	2.95714
6.5	5.0	107.200000	33.558903	55.0	105.00	111.0	117.00	148.0	5.0	3.00000
6.6	2.0	67.500000	12.020815	59.0	63.25	67.5	71.75	76.0	2.0	2.95000
6.7	8.0	112.125000	31.984092	66.0	84.75	117.0	142.00	146.0	8.0	3.05000
6.8	3.0	111.333333	33.531080	77.0	95.00	113.0	128.50	144.0	3.0	3.00000
6.9	4.0	114.000000	41.753243	53.0	104.00	130.5	140.50	142.0	4.0	3.12500
7.0	1.0	51.000000	NaN	51.0	51.00	51.0	51.00	51.0	1.0	3.20000
7.1	1.0	103.000000	NaN	103.0	103.00	103.0	103.00	103.0	1.0	3.00000
7.2	3.0	122.000000	10.583005	110.0	118.00	126.0	128.00	130.0	3.0	3.26666
7.3	1.0	108.000000	NaN	108.0	108.00	108.0	108.00	108.0	1.0	2.90000
7.4	1.0	131.000000	NaN	131.0	131.00	131.0	131.00	131.0	1.0	2.80000
7.6	1.0	106.000000	NaN	106.0	106.00	106.0	106.00	106.0	1.0	3.00000
7.7	4.0	124.000000	8.286535	118.0	118.75	121.0	126.25	136.0	4.0	3.05000
7.9	1.0	132.000000	NaN	132.0	132.00	132.0	132.00	132.0	1.0	3.80000

```
# invoking a method inside GroupBy
def f(group):
```

```
    return group.describe()
grouped.apply(f)
```

		key2	data1	data2
key1				
a	count	2.0	2.000000	2.000000
	mean	1.5	-0.023431	0.064271
	std	0.707107	1.309280	1.305741
	min	1.0	-0.949232	-0.859027
	25%	1.25	-0.486331	-0.397378
	50%	1.5	-0.023431	0.064271
	75%	1.75	0.439470	0.525920
	max	2.0	0.902370	0.987569
b	count	0.0	1.000000	1.000000
	mean	<NA>	0.269996	0.291597
	std	<NA>	NaN	NaN
	min	<NA>	0.269996	0.291597
	25%	<NA>	0.269996	0.291597
	50%	<NA>	0.269996	0.291597
	75%	<NA>	0.269996	0.291597
	max	<NA>	0.269996	0.291597
c	count	1.0	1.000000	1.000000
	mean	5.0	-1.341779	-0.229023
	std	<NA>	NaN	NaN
	min	5.0	-1.341779	-0.229023
	25%	5.0	-1.341779	-0.229023
	50%	5.0	-1.341779	-0.229023
	75%	5.0	-1.341779	-0.229023
	max	5.0	-1.341779	-0.229023

```
iris2.apply(f)
```

	Id	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000

	Id	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

Suppressing the Group Keys

```
iris2.groupby('Sepal Length (cm)', group_keys = False).apply(top)
```

	Id	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)
13	14	4.3	3.0	1.1	0.1
8	9	4.4	2.9	1.4	0.2
38	39	4.4	3.0	1.3	0.2
42	43	4.4	3.2	1.3	0.2
41	42	4.5	2.3	1.3	0.3
…	…	…	…	…	…
117	118	7.7	3.8	6.7	2.2
118	119	7.7	2.6	6.9	2.3
122	123	7.7	2.8	6.7	2.0
135	136	7.7	3.0	6.1	2.3
131	132	7.9	3.8	6.4	2.0

Quantile and Bucket analysis

```
frame = pd.DataFrame({'data1': np.random.standard_normal(1000),
                      'data2': np.random.standard_normal(1000)})

frame.head()
```

	data1	data2
0	-0.218315	-1.593930
1	2.851876	0.110652
2	-2.136658	-0.057209
3	1.380738	-1.014417
4	0.375207	0.910869

```
quartiles = pd.cut(frame['data1'],4)
```

```
quartiles.head()
```

```
0     (-1.6, 0.127]
1    (1.854, 3.581]
2    (-3.333, -1.6]
3    (0.127, 1.854]
4    (0.127, 1.854]
Name: data1, dtype: category
Categories (4, interval[float64, right]): [(-3.333, -1.6] < (-1.6, 0.127] < (0.127, 1.854] <
```

```
def get_stats(group):
    return pd.DataFrame({
        'min': group.min(), 'max':group.max(),
        'count':group.count(), "mean":group.mean()
    })
```

```
grouped3 = frame.groupby(quartiles)
```

```
grouped3.apply(get_stats)
```

		min	max	count	mean
data1					
(-3.333, -1.6]	data1	-3.326377	-1.615895	47	-2.008459
	data2	-2.449983	1.891096	47	-0.031678
(-1.6, 0.127]	data1	-1.592835	0.127111	499	-0.559453
	data2	-3.688831	2.891848	499	0.060442
(0.127, 1.854]	data1	0.129218	1.845841	419	0.767489
	data2	-3.107630	3.210868	419	0.063923
(1.854, 3.581]	data1	1.860336	3.580781	35	2.272007
	data2	-1.915169	1.595983	35	-0.040774

```
# the same result can also be computed with;
grouped3.agg({'min', 'max', 'count', 'mean'})
```

	data1				data2			
	max	count	min	mean	max	count	min	mean
data1								
(-3.333, -1.6]	-1.615895	47	-3.326377	-2.008459	1.891096	47	-2.449983	-0.031678
(-1.6, 0.127]	0.127111	499	-1.592835	-0.559453	2.891848	499	-3.688831	0.060442
(0.127, 1.854]	1.845841	419	0.129218	0.767489	3.210868	419	-3.107630	0.063923
(1.854, 3.581]	3.580781	35	1.860336	2.272007	1.595983	35	-1.915169	-0.040774

```python
# using pandas.qcut

quartiles_samp = pd.qcut(frame['data1'], 4, labels= False)

quartiles_samp.head()
```

```
0    1
1    3
2    0
3    3
4    2
Name: data1, dtype: int64
```

```python
grouped4 = frame.groupby(quartiles_samp)
```

```python
grouped4.apply(get_stats)
```

		min	max	count	mean
data1					
0	data1	-3.326377	-0.634348	250	-1.220039
	data2	-2.449983	2.891848	250	0.170686
1	data1	-0.629175	0.018952	250	-0.288017
	data2	-2.673280	2.854084	250	-0.041068
2	data1	0.023148	0.676789	250	0.326941
	data2	-3.688831	3.210868	250	0.070652
3	data1	0.681385	3.580781	250	1.291250
	data2	-3.107630	2.653283	250	0.015843

Ex: Filling missing values with group-spscific values

```
s = pd.Series(np.random.standard_normal(6))

s[::2] = np.nan

s
```

```
0         NaN
1   -0.860730
2         NaN
3    1.412749
4         NaN
5    0.419197
dtype: float64
```

```
s.fillna(s.mean())
```

```
0    0.323739
1   -0.860730
2    0.323739
3    1.412749
4    0.323739
5    0.419197
dtype: float64
```

```
states = ['MP', 'UP', 'MH', 'RJ',
         'HP', 'UK', 'PB', 'KR']

group_key = ['Centre', 'Centre', 'Centre', 'Centre',
            'North', 'North', "North", "North"]
```

```
data4 = pd.Series(np.random.standard_normal(8),
                 index= states)
```

```
data4
```

```
MP    0.712152
UP    0.211319
MH    0.925109
RJ   -0.204536
HP    0.293047
UK    0.497221
PB    0.034935
KR   -0.842300
dtype: float64
```

```
# missing data for states

data4[['MP', "PB", "KR"]] = np.nan
```

```
data4
```

```
MP         NaN
UP    0.211319
MH    0.925109
RJ   -0.204536
HP    0.293047
UK    0.497221
PB         NaN
KR         NaN
dtype: float64
```

```
data4.groupby(group_key).size()
```

```
Centre    4
North     4
dtype: int64
```

```
data4.groupby(group_key).mean()
```

```
Centre    0.310630
North     0.395134
dtype: float64
```

```python
data4.groupby(group_key).count()
```

```
Centre    3
North     2
dtype: int64
```

```python
data4.groupby(group_key).mean()
```

```
Centre    0.310630
North     0.395134
dtype: float64
```

```python
# filling NA values from group means
def fill_mean(group):
    return group.fillna(group.mean())

data.groupby(group_key).apply(fill_mean)
```

```
Centre  MP   -1.190913
        UP    0.136560
        MH   -0.046813
        RJ    0.962568
North   HP    0.477671
        UK   -0.695437
        PB    0.745012
        KR    2.058104
dtype: float64
```

```python
data
```

```
MP   -1.190913
UP    0.136560
MH   -0.046813
RJ    0.962568
HP    0.477671
UK   -0.695437
PB    0.745012
KR    2.058104
dtype: float64
```

```
data4
```

```
MP         NaN
UP    0.211319
MH    0.925109
RJ   -0.204536
HP    0.293047
UK    0.497221
PB         NaN
KR         NaN
dtype: float64
```

```
# filling predefined vlaues
fill_values = {'Centre': 0.5,
               'North': -1}

def fill_func(group):
    return group.fillna(fill_values[group.name])

data4.groupby(group_key).apply(fill_func)
```

```
Centre  MP    0.500000
        UP    0.211319
        MH    0.925109
        RJ   -0.204536
North   HP    0.293047
        UK    0.497221
        PB   -1.000000
        KR   -1.000000
dtype: float64
```

Ex: Random Sampling and Permutation

```
suits = ['H', 'S', 'C', 'D'] # hearts, club, diammonds, spades
card_val = (list(range(1, 11)) + [10] * 3) * 4
```

```
base_names = ['A'] + list(range(2, 11)) + ['J', 'K', 'Q']
cards = []
for suit in suits:
```

```
    cards.extend(str(num) + suit for num in base_names)

deck= pd.Series(card_val, index = cards)
```

```
deck.head(13)
```

```
AH      1
2H      2
3H      3
4H      4
5H      5
6H      6
7H      7
8H      8
9H      9
10H    10
JH     10
KH     10
QH     10
dtype: int64
```

```
# drawing first five cards from deck
def draw(deck, n=5):
    return deck.sample(n)

draw(deck)
```

```
6H      6
6S      6
7S      7
10H    10
9D      9
dtype: int64
```

```
# two random cards from each suit
def get_suit(card):
    # last letter is suit
    return card[-1]
```

```
deck.groupby(get_suit).apply(draw, n = 2)
```

```
C  KC    10
   AC     1
D  JD    10
   6D     6
H  AH     1
   2H     2
S  9S     9
   AS     1
dtype: int64
```

```
# alternatively, group_keys = False
deck.groupby(get_suit, group_keys= False).apply(draw, n=2)
```

```
4C     4
KC    10
6D     6
JD    10
8H     8
AH     1
JS    10
8S     8
dtype: int64
```

Ex: Group weighted Average and Correlation

```
df5 = pd.DataFrame({"category": ['a', 'a', 'a', 'a'],
                   "data": np.random.standard_normal(4),
                    'weights': np.random.uniform(size = 4)})
```

```
df5
```

	category	data	weights
0	a	-0.272834	0.453331
1	a	1.032855	0.573681
2	a	0.123029	0.239600

	category	data	weights
3	a	-0.715634	0.474301

```
# average weight by category
grouped = df5.groupby('category')

def get_wavg(group):
    return np.average(group['data'],
                      weights=group['weights'])

grouped.apply(get_wavg)
```

```
category
a    0.091272
dtype: float64
```

Ex: Group-Wise linear Regression

- using regress function of statsmodel econometrics library

```
import statsmodels.api as sm
```

```
! pip install statsmodels
```

```
def regress(data, yvar= None, xvars = None):
    Y = data[yvar]
    X = data [xvar]
    X['intercept'] = 1
    result = sm.OLS(Y, X).fit()
    return result.params
```

```
iris2.columns
```

```
Index(['Id', 'Sepal Length (cm)', 'Sepal Width (cm)', 'Petal Length (cm)',
       'Petal Width (cm)'],
      dtype='object')
```

```
iris2.rename(columns={'Sepal Length (cm)' : 'sepal_length'}, inplace = True)
```

```
iris2.columns
```

```
Index(['Id', 'sepal_length', 'Sepal Width (cm)', 'Petal Length (cm)',
       'Petal Width (cm)'],
      dtype='object')
```

```
sepal_length.apply(regress, yvar= "AAPl",
             xvars = ["SPX"])
```

Group transforms and 'Unwrapped' GroupBys

```
df6 = pd.DataFrame({'key': ['a', 'b', 'c'] * 4,
                   'value' : np.arange(12.)
                   })
```

```
df6
```

	key	value
0	a	0.0
1	b	1.0
2	c	2.0
3	a	3.0
4	b	4.0
5	c	5.0
6	a	6.0
7	b	7.0
8	c	8.0
9	a	9.0
10	b	10.0
11	c	11.0

```
# group means by key

g = df6.groupby('key')['value']
```

```python
g.mean()
```

```
key
a    4.5
b    5.5
c    6.5
Name: value, dtype: float64
```

```python
def get_mean(group):
    return group.mean()

g.transform(get_mean)
```

```
0     4.5
1     5.5
2     6.5
3     4.5
4     5.5
5     6.5
6     4.5
7     5.5
8     6.5
9     4.5
10    5.5
11    6.5
Name: value, dtype: float64
```

```python
g.transform('mean')
```

```
0     4.5
1     5.5
2     6.5
3     4.5
4     5.5
5     6.5
6     4.5
7     5.5
8     6.5
```

```
9     4.5
10    5.5
11    6.5
Name: value, dtype: float64
```

```
def times_two(group):
    return group* 2

g.transform(times_two)
```

```
0      0.0
1      2.0
2      4.0
3      6.0
4      8.0
5     10.0
6     12.0
7     14.0
8     16.0
9     18.0
10    20.0
11    22.0
Name: value, dtype: float64
```

```
# ranks in descending order
def get_ranks(group):
    return group.rank(ascending=False)

g.transform(get_ranks)
```

```
0     4.0
1     4.0
2     4.0
3     3.0
4     3.0
5     3.0
6     2.0
7     2.0
8     2.0
9     1.0
```

```
10    1.0
11    1.0
Name: value, dtype: float64
```

```python
# transformation function composed of aggregations

def normalize(x):
    return (x - x.mean())/x.std()
```

```python
g.transform(normalize)
```

```
0    -1.161895
1    -1.161895
2    -1.161895
3    -0.387298
4    -0.387298
5    -0.387298
6     0.387298
7     0.387298
8     0.387298
9     1.161895
10    1.161895
11    1.161895
Name: value, dtype: float64
```

```python
g.apply(normalize)
```

```
key
a    0    -1.161895
     3    -0.387298
     6     0.387298
     9     1.161895
b    1    -1.161895
     4    -0.387298
     7     0.387298
     10    1.161895
c    2    -1.161895
     5    -0.387298
     8     0.387298
```

```
    11    1.161895
Name: value, dtype: float64
```

```python
# using built-in 'mean' and 'sum' functions

g.transform('mean')
```

```
0     4.5
1     5.5
2     6.5
3     4.5
4     5.5
5     6.5
6     4.5
7     5.5
8     6.5
9     4.5
10    5.5
11    6.5
Name: value, dtype: float64
```

```python
normalized2 = (df6['value'] - g.transform('mean')) / g.transform('std')
```

```python
normalized2
```

```
0    -1.161895
1    -1.161895
2    -1.161895
3    -0.387298
4    -0.387298
5    -0.387298
6     0.387298
7     0.387298
8     0.387298
9     1.161895
10    1.161895
11    1.161895
Name: value, dtype: float64
```

Pivot tables

- summarization tool
- uses groupby facility
- pandas.pivot_table fucntion can add partial tools known as margins
- summary - pivot table

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as mlt
```

```
df_iris = pd.read_csv('E:\pythonfordatanalysis\semainedu26fevrier\iris.csv')
```

```
df_iris.head()
```

	Id	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
df_iris.pivot_table(index = ['Species'])
```

	Id	Petal Length (cm)	Petal Width (cm)	Sepal Length (cm)	Sepal Width (cm)
Species					
Iris-setosa	25.5	1.464	0.244	5.006	3.418
Iris-versicolor	75.5	4.260	1.326	5.936	2.770
Iris-virginica	125.5	5.552	2.026	6.588	2.974

```
df_iris.columns
```

```
Index(['Id', 'Sepal Length (cm)', 'Sepal Width (cm)', 'Petal Length (cm)',
       'Petal Width (cm)', 'Species'],
      dtype='object')
```

```
df_iris.pivot_table(index = ['Species', 'Petal Length (cm)'])
```

		Id	Petal Width (cm)	Sepal Length (cm)	Sepal Width (cm)
Species	Petal Length (cm)				
	1.0	23.000000	0.200000	4.600000	3.600000
	1.1	14.000000	0.100000	4.300000	3.000000
	1.2	25.500000	0.200000	5.400000	3.600000
	1.3	31.714286	0.257143	4.842857	3.228571
Iris-setosa	1.4	21.833333	0.216667	4.916667	3.333333
	1.5	24.714286	0.221429	5.128571	3.535714
	1.6	31.000000	0.285714	4.914286	3.342857
	1.7	17.500000	0.350000	5.400000	3.600000
	1.9	35.000000	0.300000	4.950000	3.600000
	3.0	99.000000	1.100000	5.100000	2.500000
	3.3	76.000000	1.000000	4.950000	2.350000
	3.5	70.500000	1.000000	5.350000	2.300000
	3.6	65.000000	1.300000	5.600000	2.900000
	3.7	82.000000	1.000000	5.500000	2.400000
	3.8	81.000000	1.100000	5.500000	2.400000
	3.9	71.000000	1.233333	5.533333	2.633333
	4.0	74.400000	1.220000	5.780000	2.480000
	4.1	85.666667	1.200000	5.700000	2.833333
Iris-versicolor	4.2	87.500000	1.325000	5.725000	2.900000
	4.3	86.500000	1.300000	6.300000	2.900000
	4.4	80.250000	1.325000	6.275000	2.750000
	4.5	70.571429	1.485714	5.900000	2.928571
	4.6	68.666667	1.400000	6.400000	2.900000
	4.7	66.600000	1.420000	6.440000	3.060000
	4.8	74.000000	1.600000	6.350000	3.000000
	4.9	63.000000	1.500000	6.600000	2.800000
	5.0	78.000000	1.700000	6.700000	3.000000
	5.1	84.000000	1.600000	6.000000	2.700000
	4.5	107.000000	1.700000	4.900000	2.500000
	4.8	133.000000	1.800000	6.100000	2.900000
	4.9	124.666667	1.866667	6.000000	2.833333
	5.0	127.000000	1.800000	6.000000	2.400000
	5.1	128.142857	1.971429	6.142857	2.900000
	5.2	147.000000	2.150000	6.600000	3.000000
	5.3	114.000000	2.100000	6.400000	2.950000
	5.4	144.500000	2.200000	6.550000	3.250000
	5.5	122.666667	1.900000	6.566667	3.033333

Iris-virginica

Species	Petal Length (cm)	Id	Petal Width (cm)	Sepal Length (cm)	Sepal Width (cm)
	5.6	129.833333	2.050000	6.366667	2.933333
	5.7	130.333333	2.300000	6.766667	3.266667
	5.8	114.666667	1.866667	6.800000	2.833333
	5.9	123.500000	2.200000	6.950000	3.100000
	6.0	113.500000	2.150000	6.750000	3.250000
	6.1	125.666667	2.233333	7.433333	3.133333
	6.3	108.000000	1.800000	7.300000	2.900000
	6.4	132.000000	2.000000	7.900000	3.800000
	6.6	106.000000	2.100000	7.600000	3.000000
	6.7	120.500000	2.100000	7.700000	3.300000
	6.9	119.000000	2.300000	7.700000	2.600000

```
df_iris.pivot_table(index = ['Species', 'Petal Length (cm)'],
                    columns = 'Sepal Width (cm)', )
```

		Id									
	Sepal Width (cm)	2.0	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0
Species	Petal Length (cm)										
	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	1.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	14.000
	1.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	1.3	NaN	NaN	42.0	NaN	NaN	NaN	NaN	NaN	NaN	39.000
Iris-setosa	1.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	9.0	20.333
	1.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	1.6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	26.000
	1.7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	1.9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	3.0	NaN	NaN	NaN	NaN	99.0	NaN	NaN	NaN	NaN	NaN
	3.3	NaN	NaN	94.0	58.0	NaN	NaN	NaN	NaN	NaN	NaN
	3.5	61.0	NaN	NaN	NaN	NaN	80.0	NaN	NaN	NaN	NaN
	3.6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	65.0	NaN
	3.7	NaN	NaN	NaN	82.0	NaN	NaN	NaN	NaN	NaN	NaN
	3.8	NaN	NaN	NaN	81.0	NaN	NaN	NaN	NaN	NaN	NaN
	3.9	NaN	NaN	NaN	NaN	70.0	NaN	71.5	NaN	NaN	NaN
	4.0	NaN	63.0	54.0	NaN	90.0	93.0	NaN	72.0	NaN	NaN
	4.1	NaN	NaN	NaN	NaN	NaN	NaN	68.0	100.0	NaN	89.000
Iris-versicolor	4.2	NaN	NaN	NaN	NaN	NaN	NaN	95.0	NaN	97.0	79.000
	4.3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	86.5	NaN

Species	Sepal Width (cm) Petal Length (cm)	Id 2.0	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0
	4.4	NaN	NaN	88.0	NaN	NaN	91.0	NaN	NaN	NaN	76.000
	4.5	NaN	69.0	NaN	NaN	NaN	NaN	NaN	56.0	79.0	76.000
	4.6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	55.0	59.0	92.000
	4.7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	74.0	64.0	NaN
	4.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	77.0	NaN	NaN
	4.9	NaN	NaN	NaN	NaN	73.0	NaN	NaN	NaN	NaN	NaN
	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	78.000
	5.1	NaN	NaN	NaN	NaN	NaN	NaN	84.0	NaN	NaN	NaN
Iris-virginica	4.5	NaN	NaN	NaN	NaN	107.0	NaN	NaN	NaN	NaN	NaN
	4.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	127.0	NaN	139.00
	4.9	NaN	NaN	NaN	NaN	NaN	NaN	124.0	122.0	NaN	128.00
	5.0	NaN	120.0	NaN	NaN	130.5	NaN	NaN	NaN	NaN	NaN
	5.1	NaN	NaN	NaN	NaN	NaN	NaN	122.5	124.5	NaN	150.00
	5.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	147.00
	5.3	NaN	NaN	NaN	NaN	NaN	NaN	112.0	NaN	NaN	NaN
	5.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	5.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	115.00
	5.6	NaN	NaN	NaN	NaN	NaN	135.0	NaN	131.0	104.0	NaN
	5.7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	5.8	NaN	NaN	NaN	NaN	109.0	NaN	NaN	NaN	NaN	117.50
	5.9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	103.00
	6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	6.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	131.0	NaN	136.00
	6.3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	108.0	NaN
	6.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	6.6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	106.00
	6.7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	123.0	NaN	NaN
	6.9	NaN	NaN	NaN	NaN	NaN	119.0	NaN	NaN	NaN	NaN

```
# passing fill_value for empty combinations
df_iris.pivot_table(index = ['Species', 'Petal Length (cm)'],
                    columns = 'Sepal Width (cm)', fill_value=5)
```

Species	Sepal Width (cm) Petal Length (cm)	Id 2.0	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0	.. ..
Iris-setosa	1.0	5	5	5	5	5.0	5	5.0	5.0	5.0	5.000000	..

		Id										..
	Sepal Width (cm)	2.0	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0	..
Species	Petal Length (cm)											
	1.1	5	5	5	5	5.0	5	5.0	5.0	5.0	14.000000	..
	1.2	5	5	5	5	5.0	5	5.0	5.0	5.0	5.000000	..
	1.3	5	5	42	5	5.0	5	5.0	5.0	5.0	39.000000	..
	1.4	5	5	5	5	5.0	5	5.0	5.0	9.0	20.333333	..
	1.5	5	5	5	5	5.0	5	5.0	5.0	5.0	5.000000	..
	1.6	5	5	5	5	5.0	5	5.0	5.0	5.0	26.000000	..
	1.7	5	5	5	5	5.0	5	5.0	5.0	5.0	5.000000	..
	1.9	5	5	5	5	5.0	5	5.0	5.0	5.0	5.000000	..
	3.0	5	5	5	5	99.0	5	5.0	5.0	5.0	5.000000	..
	3.3	5	5	94	58	5.0	5	5.0	5.0	5.0	5.000000	..
	3.5	61	5	5	5	5.0	80	5.0	5.0	5.0	5.000000	..
	3.6	5	5	5	5	5.0	5	5.0	5.0	65.0	5.000000	..
	3.7	5	5	5	82	5.0	5	5.0	5.0	5.0	5.000000	..
	3.8	5	5	5	81	5.0	5	5.0	5.0	5.0	5.000000	..
	3.9	5	5	5	5	70.0	5	71.5	5.0	5.0	5.000000	..
	4.0	5	63	54	5	90.0	93	5.0	72.0	5.0	5.000000	..
	4.1	5	5	5	5	5.0	5	68.0	100.0	5.0	89.000000	..
Iris-versicolor	4.2	5	5	5	5	5.0	5	95.0	5.0	97.0	79.000000	..
	4.3	5	5	5	5	5.0	5	5.0	5.0	86.5	5.000000	..
	4.4	5	5	88	5	5.0	91	5.0	5.0	5.0	76.000000	..
	4.5	5	69	5	5	5.0	5	5.0	56.0	79.0	76.000000	..
	4.6	5	5	5	5	5.0	5	5.0	55.0	59.0	92.000000	..
	4.7	5	5	5	5	5.0	5	5.0	74.0	64.0	5.000000	..
	4.8	5	5	5	5	5.0	5	5.0	77.0	5.0	5.000000	..
	4.9	5	5	5	5	73.0	5	5.0	5.0	5.0	5.000000	..
	5.0	5	5	5	5	5.0	5	5.0	5.0	5.0	78.000000	..
	5.1	5	5	5	5	5.0	5	84.0	5.0	5.0	5.000000	..
	4.5	5	5	5	5	107.0	5	5.0	5.0	5.0	5.000000	..
	4.8	5	5	5	5	5.0	5	5.0	127.0	5.0	139.000000	..
	4.9	5	5	5	5	5.0	5	124.0	122.0	5.0	128.000000	..
	5.0	5	120	5	5	130.5	5	5.0	5.0	5.0	5.000000	..
	5.1	5	5	5	5	5.0	5	122.5	124.5	5.0	150.000000	..
	5.2	5	5	5	5	5.0	5	5.0	5.0	5.0	147.000000	..
	5.3	5	5	5	5	5.0	5	112.0	5.0	5.0	5.000000	..
	5.4	5	5	5	5	5.0	5	5.0	5.0	5.0	5.000000	..
	5.5	5	5	5	5	5.0	5	5.0	5.0	5.0	115.000000	..
Iris-virginica	5.6	5	5	5	5	5.0	135	5.0	131.0	104.0	5.000000	..
	5.7	5	5	5	5	5.0	5	5.0	5.0	5.0	5.000000	..
	5.8	5	5	5	5	109.0	5	5.0	5.0	5.0	117.500000	..

Species	Sepal Width (cm) / Petal Length (cm)	Id 2.0	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0	..
	5.9	5	5	5	5	5.0	5	5.0	5.0	5.0	103.000000	..
	6.0	5	5	5	5	5.0	5	5.0	5.0	5.0	5.000000	..
	6.1	5	5	5	5	5.0	5	5.0	131.0	5.0	136.000000	..
	6.3	5	5	5	5	5.0	5	5.0	5.0	108.0	5.000000	..
	6.4	5	5	5	5	5.0	5	5.0	5.0	5.0	5.000000	..
	6.6	5	5	5	5	5.0	5	5.0	5.0	5.0	106.000000	..
	6.7	5	5	5	5	5.0	5	5.0	123.0	5.0	5.000000	..
	6.9	5	5	5	5	5.0	119	5.0	5.0	5.0	5.000000	..

Cross- tabulation

```
from io import StringIO
```

```
data = """ Sample Court_paragraph
1. Il était une fois, dans un village paisible au bord de la mer,
2. un jeune garçon nommé Luca.
3. Luca aimait explorer les forêts environnantes,
4. où il découvrait des trésors cachés et des mystères oubliés.
5. Un jour, en suivant un papillon multicolore,
6. Luca découvrit une grotte secrète.
7. À l'intérieur, il trouva une carte ancienne indiquant
8. l'emplacement d'un trésor légendaire.
9. Déterminé à le trouver,
10. Luca partit en voyage, bravant des tempêtes et affrontant
11. des créatures magiques.
12. Après de nombreuses aventures, il trouva enfin le trésor,
13. qui se révéla être l'amitié des habitants de ce village enchanté.
14. Luca apprit alors que les véritables trésors
15. sont souvent cachés à l'intérieur de nous-mêmes.
"""
```

```
data = pd.read_table(StringIO(data), sep = '\s+')
```

```
data
```

1.0	Il	était	une	fois,	dans	un	village	pai
2.0	un	jeune	garçon	nommé	Luca.	NaN	NaN	Na
3.0	Luca	aimait	explorer	les	forêts	environnantes,	NaN	Na
4.0	où	il	découvrait	des	trésors	cachés	et	des
5.0	Un	jour,	en	suivant	un	papillon	multicolore,	Na
6.0	Luca	découvrit	une	grotte	secrète.	NaN	NaN	Na
7.0	À	l'intérieur,	il	trouva	une	carte	ancienne	ind
8.0	l'emplacement	d'un	trésor	légendaire.	NaN	NaN	NaN	Na
9.0	Déterminé	à	le	trouver,	NaN	NaN	NaN	Na
10.0	Luca	partit	en	voyage,	bravant	des	tempêtes	et
11.0	des	créatures	magiques.	NaN	NaN	NaN	NaN	Na
12.0	Après	de	nombreuses	aventures,	il	trouva	enfin	le
13.0	qui	se	révéla	être	l'amitié	des	habitants	de
14.0	Luca	apprit	alors	que	les	véritables	trésors	Na
15.0	sont	souvent	cachés	à	l'intérieur	de	nous-mêmes.	Na

```
pd.crosstab(data['Sample'], data['Court_paragraph'],
            margins = True)
```

Court_paragraph	mer,	All
Sample		
la	1	1
All	1	1