UNIVERSITY OF PORTSMOUTH

Python for Data Analysis

Data Aggregation and Group Operations (Week 9)

Atefeh Khazaei

atefeh.khazaei@port.ac.uk

What we will learn this week?

- GroupBy Mechanics
- Data Aggregation

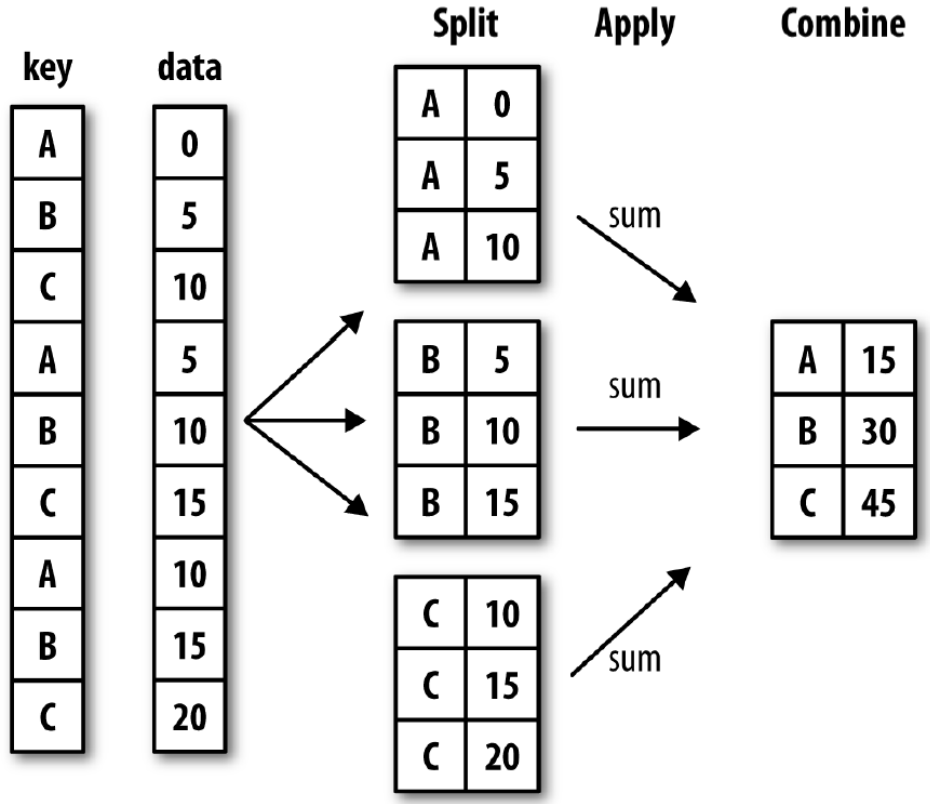
Data Aggregation and Group Operations

- 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.

GroupBy Mechanics

- Group operations = **split-apply-combine**

1. Data contained in a pandas object is split into groups based on one or more keys.
2. A function is applied to each group, producing a new value.
3. The results of all those function applications are combined into a result object.



GroupBy Mechanics (cont.)
Simple Examples

- Suppose you wanted to compute the mean of the data1 column using the labels from key1.

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

	key1	key2	data1	data2
0	a	one	0.493033	0.373159
1	a	two	-0.649038	0.955901
2	b	one	0.042338	1.685276
3	b	two	-1.706215	-0.172254
4	a	one	-1.022028	-0.845129

GroupBy Mechanics (cont.)
Simple Examples

- This grouped variable is now a GroupBy object.
- The idea is that this object has all of the information needed to then apply some operation to each of the groups (e.g. mean).

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

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x000001B462D3B8E0>
```

```
grouped.mean()
```

```
key1
a   -0.392678
b   -0.831938
Name: data1, dtype: float64
```

- The data in data1 series **aggregated** according to the **group key** produced a new **series** that is now **indexed** by the unique values in key1 column.

GroupBy Mechanics (cont.)
Simple Examples

- Pass multiple arrays as a list

	key1	key2	data1	data2
0	a	one	0.493033	0.373159
1	a	two	-0.649038	0.955901
2	b	one	0.042338	1.685276
3	b	two	-1.706215	-0.172254
4	a	one	-1.022028	-0.845129

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

```
key1  key2
a     one    -0.264498
      two    -0.649038
b     one     0.042338
      two    -1.706215
Name: data1, dtype: float64
```

GroupBy Mechanics (cont.)
Simple Examples

- 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:

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

```
key1  key2
a     one    -0.264498
      two    -0.649038
b     one     0.042338
      two    -1.706215
Name: data1, dtype: float64
```

```
means.unstack()
```

key2	one	two
key1		
a	-0.264498	-0.649038
b	0.042338	-1.706215

GroupBy Mechanics (cont.)
Simple Examples

	key1	key2	data1	data2
0	a	one	0.493033	0.373159
1	a	two	-0.649038	0.955901
2	b	one	0.042338	1.685276
3	b	two	-1.706215	-0.172254
4	a	one	-1.022028	-0.845129

- 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.

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

key1	data1	data2
a	-0.392678	0.161310
b	-0.831938	0.756511

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

key1	key2	data1	data2
a	one	-0.264498	-0.235985
	two	-0.649038	0.955901
b	one	0.042338	1.685276
	two	-1.706215	-0.172254

GroupBy Mechanics (cont.)
Simple Examples

- A generally useful GroupBy method is **size**, which returns a Series containing group sizes.
- Any **missing values** in a group key will be excluded from the result.

	key1	key2	data1	data2
0	a	one	0.493033	0.373159
1	a	two	-0.649038	0.955901
2	b	one	0.042338	1.685276
3	b	two	-1.706215	-0.172254
4	a	one	-1.022028	-0.845129

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

```
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

GroupBy Mechanics (cont.)
Iterating Over Groups

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

	key1	key2	data1	data2
0	a	one	0.493033	0.373159
1	a	two	-0.649038	0.955901
2	b	one	0.042338	1.685276
3	b	two	-1.706215	-0.172254
4	a	one	-1.022028	-0.845129

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

```
a
  key1 key2     data1     data2
0    a  one  0.493033  0.373159
1    a  two -0.649038  0.955901
4    a  one -1.022028 -0.845129
b
  key1 key2     data1     data2
2    b  one  0.042338  1.685276
3    b  two -1.706215 -0.172254
```

GroupBy Mechanics (cont.)
Iterating Over Groups

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

	key1	key2	data1	data2
0	a	one	0.493033	0.373159
1	a	two	-0.649038	0.955901
2	b	one	0.042338	1.685276
3	b	two	-1.706215	-0.172254
4	a	one	-1.022028	-0.845129

```
for (k1, k2), group in df.groupby(['key1', 'key2']):
    print((k1, k2))
    print(group)
```

```
('a', 'one')
  key1 key2     data1     data2
0    a  one  0.493033  0.373159
4    a  one -1.022028 -0.845129
('a', 'two')
  key1 key2     data1     data2
1    a  two -0.649038  0.955901
('b', 'one')
  key1 key2     data1     data2
2    b  one  0.042338  1.685276
('b', 'two')
  key1 key2     data1     data2
3    b  two -1.706215 -0.172254
```

GroupBy Mechanics (cont.)
Selecting a Column or Subset of Columns

- We can apply subsetting or a column selection to a groupby object.
- Especially for large datasets, it may be desirable to aggregate only a few columns.

```
df.groupby('key1')['data1']
df.groupby('key1')[['data2']]
```

=

```
df['data1'].groupby(df['key1'])
df[['data2']].groupby(df['key1'])
```

GroupBy Mechanics (cont.)
Selecting a Column or Subset of Columns

- In our current *df* dataframe if we want to compute means for just *data2* column and get **the results as dataframe**, we can code:

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

		data2
key1	**key2**	
a	**one**	-0.235985
	two	0.955901
b	**one**	1.685276
	two	-0.172254

Data Aggregation

- Aggregations refer to any data transformation that produces scalar values from arrays.
- This table aggregation methods have optimized implementations.
- However, you are not limited to only this set of methods.
  - e.g. An aggregations of your own using your defined function.

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

Data Aggregation (cont.)

	key1	key2	data1	data2
0	a	one	0.493033	0.373159
1	a	two	-0.649038	0.955901
2	b	one	0.042338	1.685276
3	b	two	-1.706215	-0.172254
4	a	one	-1.022028	-0.845129

```
# Group df based on key1
grouped = df.groupby('key1')
# Retrive only data1 column
data1 = grouped['data1']
# Apply agg groupby functions
data1.agg(['mean','count','sum'])
```

key1	mean	count	sum
a	-0.392678	3	-1.178033
b	-0.831938	2	-1.663877

Data Aggregation (cont.)

- We can write our own aggregation function.
  - E.g. value of different of max and min values of group elements.

	key1	key2	data1	data2
0	a	one	0.493033	0.373159
1	a	two	-0.649038	0.955901
2	b	one	0.042338	1.685276
3	b	two	-1.706215	-0.172254
4	a	one	-1.022028	-0.845129

```
def min_max_difference(arr):
    return arr.max()-arr.min()
```

```
grouped.agg({'data1':['mean','count','sum',min_max_difference]})
```

	data1			
	mean	count	sum	min_max_difference
key1				
a	-0.198070	3	-0.594209	3.805309
b	0.891071	2	1.782143	0.354192

Data Aggregation (cont.)

- We can apply different functions to one or more columns.
- So we need to map column name and which function will be applied in a dict.

	key1	key2	data1	data2
0	a	one	0.493033	0.373159
1	a	two	-0.649038	0.955901
2	b	one	0.042338	1.685276
3	b	two	-1.706215	-0.172254
4	a	one	-1.022028	-0.845129

```
# Group df based on key1
grouped = df.groupby('key1')
# Apply agg groupby functions
grouped.agg({'data1':'sum','data2':'max'})
```

key1	data1	data2
a	-1.178033	0.955901
b	-1.663877	1.685276

Data Aggregation (cont.)

- Some methods like **describe** also work, even though they are not aggregations:

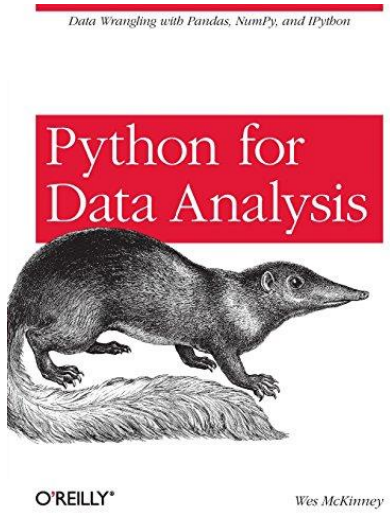
	key1	key2	data1	data2
0	a	one	0.493033	0.373159
1	a	two	-0.649038	0.955901
2	b	one	0.042338	1.685276
3	b	two	-1.706215	-0.172254
4	a	one	-1.022028	-0.845129

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

	data1							
	count	mean	std	min	25%	50%	75%	max
key1								
a	3.0	-0.392678	0.789394	-1.022028	-0.835533	-0.649038	-0.078002	0.493033
b	2.0	-0.831938	1.236414	-1.706215	-1.269077	-0.831938	-0.394800	0.042338

	data2							
	count	mean	std	min	25%	50%	75%	max
key1								
a	3.0	0.161310	0.919014	-0.845129	-0.235985	0.373159	0.664530	0.955901
b	2.0	0.756511	1.313471	-0.172254	0.292129	0.756511	1.220893	1.685276

References & More Resources



- References:
  - McKinney, Wes. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc., 2012.

- More Resources:
  - Machine Learning with Python: Foundations: https://www.linkedin.com/learning/machine-learning-with-python-foundations
  - Python Data Analysis on Linkedin Learning: https://www.linkedin.com/learning/python-data-analysis-2
    - To use Linkedin Learning, you can log in with your university account: https://myport.port.ac.uk/study-skills/linkedin-learning

Practical Session

- Please download Week09_Aggregation_and_Grouping.ipynb file, and run it to learn new points.
- Please read the practical sheet (Week09_Practicals.pdf) and do the exercise.