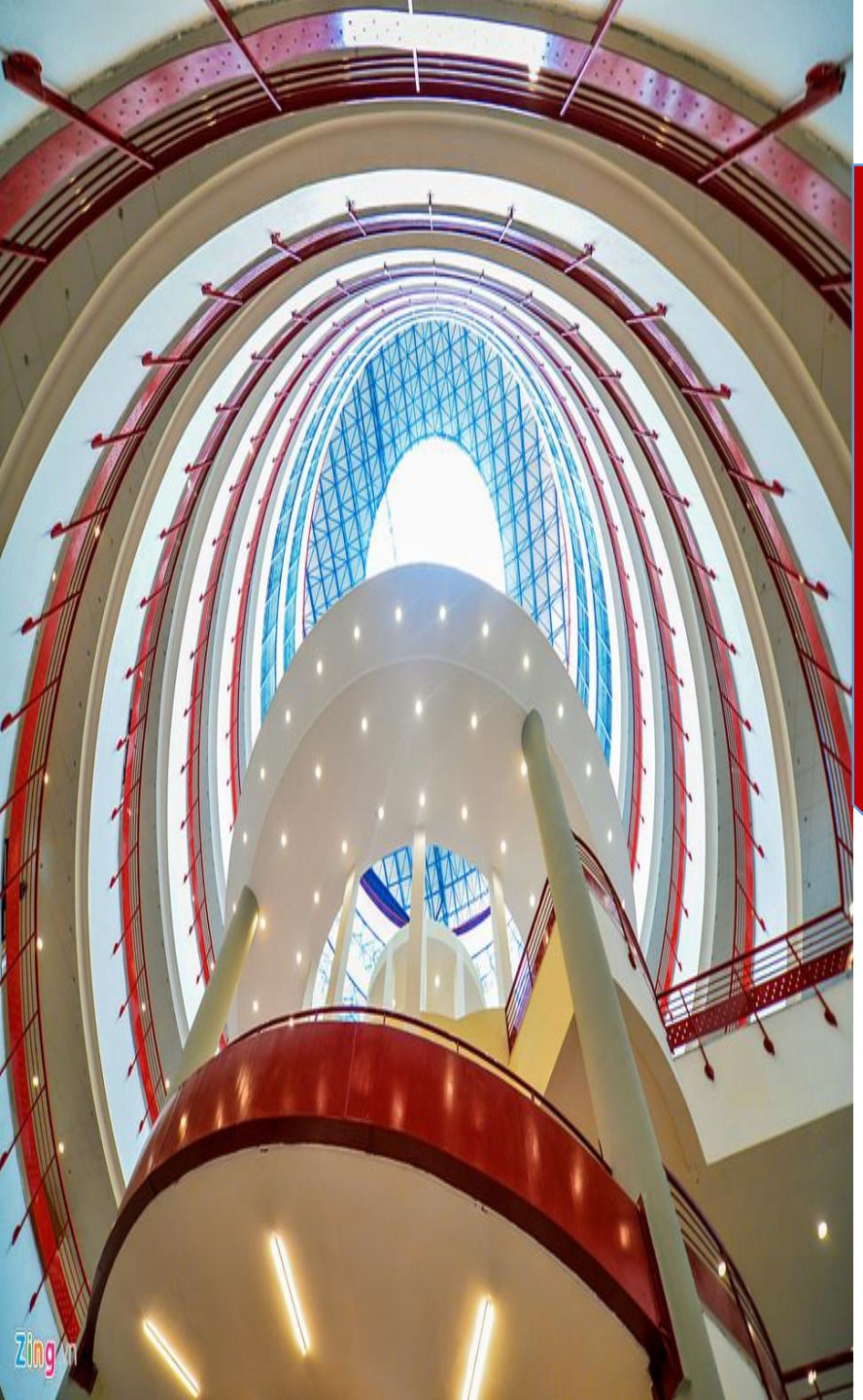
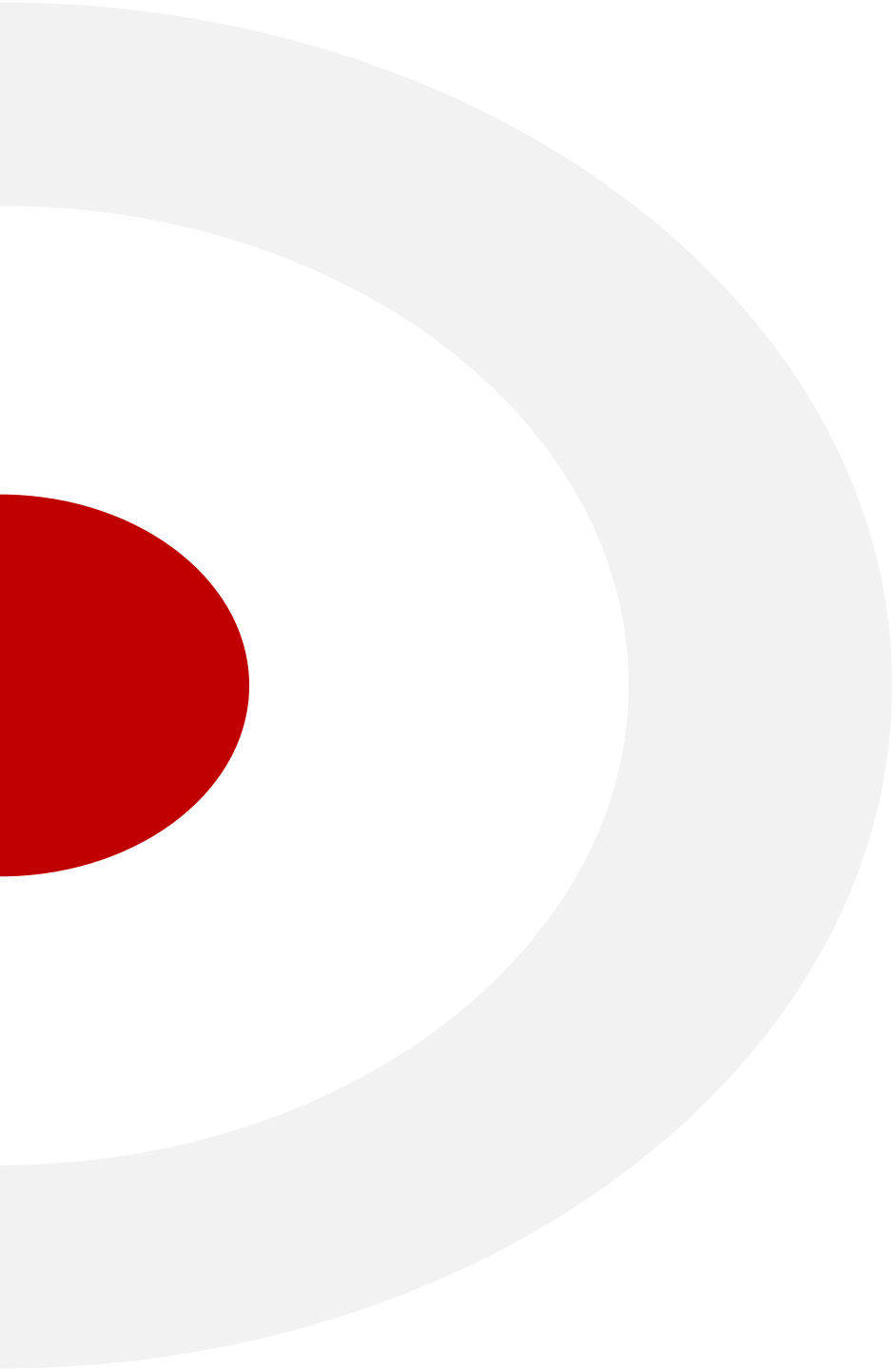


DATA PREPARATION AND VISUALIZATION

Mathematical Economics Faculty

National Economics University
https://www.neu.edu.vn/



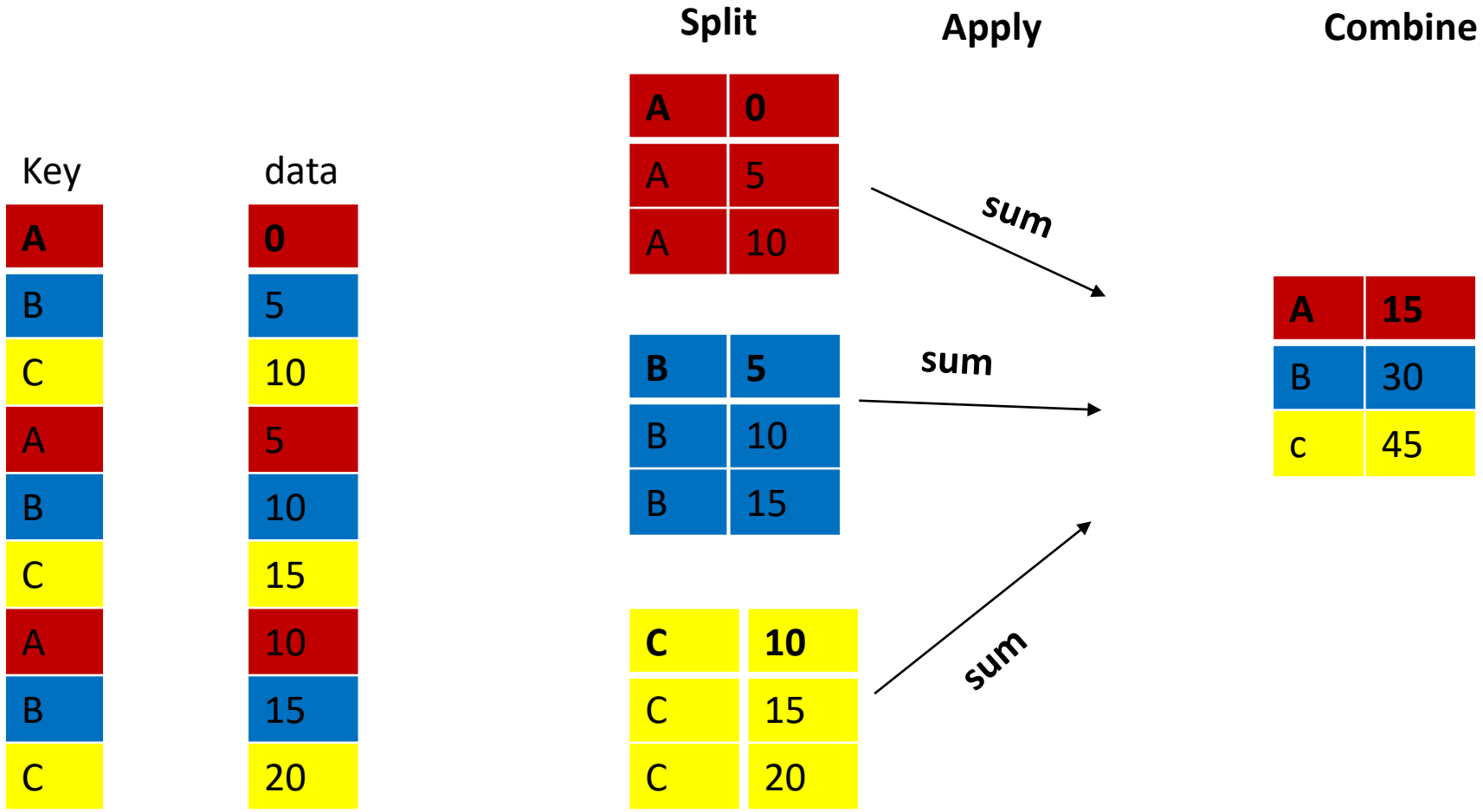
Data Aggregation and Group Operations

Objectives

- Groupby Mechanics
  - Iterating Over Groups
  - Selecting a Column or Subset of Columns
- Data Aggregation
  - Column-Wise and Multiple Function Application
  - Returning Aggregated Data Without Row Indexes
- Apply: General split-apply-combine
- Pivot-Table

GroupBy Mechanics

- The groupby operation performs the ‘split-apply-combine’ process on a DataFrame
  1. Split the DataFrame into groups
  2. Apply a function to each group
  3. Combine the results into one data structure



GroupBy Mechanics

- The groupby operation condenses into two steps:
  - Create a GroupBy object
  - Call a function on the GroupBy object
- The GroupBy object allows us to split the DataFrame into groups, but only in an abstract sense. Nothing actually gets computed until a function is called on the GroupBy object
- Example:

```
df = pd.DataFrame({'key':['A','B','C']*3,'data':[0,5,10,5,10,15,10,15,20]})
grouped = df.groupby('key')
```

```
grouped
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002122AABF070>
```

- To compute group means we call the GroupBy's mean method
  - grouped.mean()

GroupBy Mechanics

- We can group and aggregate by more columns in pandas
  - Create a new column named ‘key2’. Df[‘key2’] = [‘one’,’two’]*4 + [‘one’]
  - df.groupby([‘key’,’key2’]).mean()
  - Result:

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

		data
key	key2	
A	one	5.0
	two	5.0
B	one	10.0
	two	10.0
C	one	15.0
	two	15.0

GroupBy Mechanics

- We can use GroupBy.get_group() method to select data for a certain group
  - ex: grouped.get_group('A',df)

	key	data
0	A	0
3	A	5
6	A	10

- We can use GroupBy.groups attribute to get more information about the GroupBy object
  - Ex: grouped.groups

```
{'A': [0, 3, 6], 'B': [1, 4, 7], 'C': [2, 5, 8]}
```

More attributes: GroupBy — pandas 1.4.3 documentation (pydata.org)

GroupBy Mechanics
Iterating Over Groups

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

```
for name, group in grouped:
    print(name)
    print(group)
```

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

GroupBy Mechanics
Selecting a Column or Subset of Columns

- Select one column from a GroupBy object:

```
df.groupby('col_to_groupby')['col_selected']
```

- Select multiple columns from a GroupBy object:

```
df.groupby('col_to_groupby')[['col1','col2',…'coln']]
```

GroupBy Mechanics
Selecting a Column or Subset of Columns

- We can aggregate only a few columns, for example, to compute means for just ‘data2’ column
  - df.groupby([‘key’, ‘key2’][[‘data2’]].mean() (equivalent to df[‘data2’].groupby([‘key’,’key2’]).mean())
  - Try df.groupby([‘key’,’key2’])[‘data2’].mean()
  - What is the difference between the two results?
- The object returned by this indexing operation is a grouped DataFrame if a list or array is passed or a grouped

  Series if only a single column name is pass as scalar

Data Aggregation

- Aggregations refer to any data transformation that produces scalar values from arrays
  - Ex: df.groupby(['key','key2'])['data2'].mean()

Table 10-1. Optimized groupby 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

Data Aggregation

Some questions:

- Can I apply multi-functions at once
- Can I apply different functions to one or more of the columns

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

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

```
tips_df.groupby('smoker')['tip_pct'].agg(peak_to_peak)
```

```
smoker
No     0.235193
Yes    0.674707
Name: tip_pct, dtype: float64
```

Data Aggregation
Column-Wise and Multiple Function Application

- The GroupBy.agg() method can perform both aggregations at once. We can use the following syntax

GroupBy.agg([**func_name1,func_name2,func_name3**])

Example: tips dataset

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.059447
1	10.34	1.66	No	Sun	Dinner	3	0.160542
2	21.01	3.50	No	Sun	Dinner	3	0.166587
3	23.68	3.31	No	Sun	Dinner	2	0.139780
4	24.59	3.61	No	Sun	Dinner	4	0.146808
5	25.29	4.71	No	Sun	Dinner	4	0.186240

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

grouped_pct = grouped['tip_pct']
```

Data Aggregation
Column-Wise and Multiple Function Application

- The GroupBy.agg() method can perform both aggregations at once. We can use the following syntax

  GroupBy.agg([**func_name1,func_name2,func_name3**])

- Pass a list of functions or function names instead, you get back a DataFrame with column names taken from the functions:
  - Ex: grouped_pct.agg([‘mean’, ’std’, **peak_to_peak**])

		mean	std	peak_to_peak
day	smoker			
Fri	No	0.151650	0.028123	0.067349
	Yes	0.174783	0.051293	0.159925
Sat	No	0.158048	0.039767	0.235193
	Yes	0.147906	0.061375	0.290095
Sun	No	0.160113	0.042347	0.193226
	Yes	0.187250	0.154134	0.644685
Thur	No	0.160298	0.038774	0.193350
	Yes	0.163863	0.039389	0.151240

Data Aggregation
Column-Wise and Multiple Function Application

- We can pass a list of (name, function) tuples, the first element of each tuple will be used as the DataFrame column names
  - Ex: grouped_pct.agg([‘foo’,’mean’),(‘bar’,np.std)])

		foo	bar
day	smoker		
Fri	No	0.151650	0.028123
	Yes	0.174783	0.051293
Sat	No	0.158048	0.039767
	Yes	0.147906	0.061375
Sun	No	0.160113	0.042347
	Yes	0.187250	0.154134
Thur	No	0.160298	0.038774
	Yes	0.163863	0.039389

Data Aggregation
Column-Wise and Multiple Function Application

- We can specify a list of functions to apply to all of the columns
  - Ex: functions = [‘count’, ‘mean’, ‘max’]
  - Result = grouped[‘tip_pct’, ‘total_bill’].agg(functions)

		tip_pct			total_bill		
		count	mean	max	count	mean	max
day	smoker						
Fri	No	4	0.151650	0.187735	4	18.420000	22.75
	Yes	15	0.174783	0.263480	15	16.813333	40.17
Sat	No	45	0.158048	0.291990	45	19.661778	48.33
	Yes	42	0.147906	0.325733	42	21.276667	50.81
Sun	No	57	0.160113	0.252672	57	20.506667	48.17
	Yes	19	0.187250	0.710345	19	24.120000	45.35
Thur	No	45	0.160298	0.266312	45	17.113111	41.19
	Yes	17	0.163863	0.241255	17	19.190588	43.11

Data Aggregation
Column-Wise and Multiple Function Application

- We can apply potentially different functions to one or more of the columns. To do this, pass a dict to agg that contains a mapping of column names to any of the function specifications listed so far:
  - Ex: grouped.agg({'tip':['min','max','mean','std'], 'size':'sum']

		tip_pct				size
		min	max	mean	std	sum
day	smoker					
Fri	No	0.120385	0.187735	0.151650	0.028123	9
	Yes	0.103555	0.263480	0.174783	0.051293	31
Sat	No	0.056797	0.291990	0.158048	0.039767	115
	Yes	0.035638	0.325733	0.147906	0.061375	104
Sun	No	0.059447	0.252672	0.160113	0.042347	167
	Yes	0.065660	0.710345	0.187250	0.154134	49
Thur	No	0.072961	0.266312	0.160298	0.038774	112
	Yes	0.090014	0.241255	0.163863	0.039389	40

Data Aggregation
Returning Aggregated Data Without Row Indexes

- Use *as_index* parameter in groupby method
    - Ex: tips.groupby([‘day’, ‘smoker’], as_index = False).mean()
- We can use *the reset_index()* method:
    - Ex: tips.groupby([‘day’, ‘smoker’]).reset_index()

Data Aggregation
Apply: General split-apply-combine

- Apply splits the object being manipulated into pieces, invokes the passed function on each piece, and then attempts to concatenate the pieces together
- Syntax:

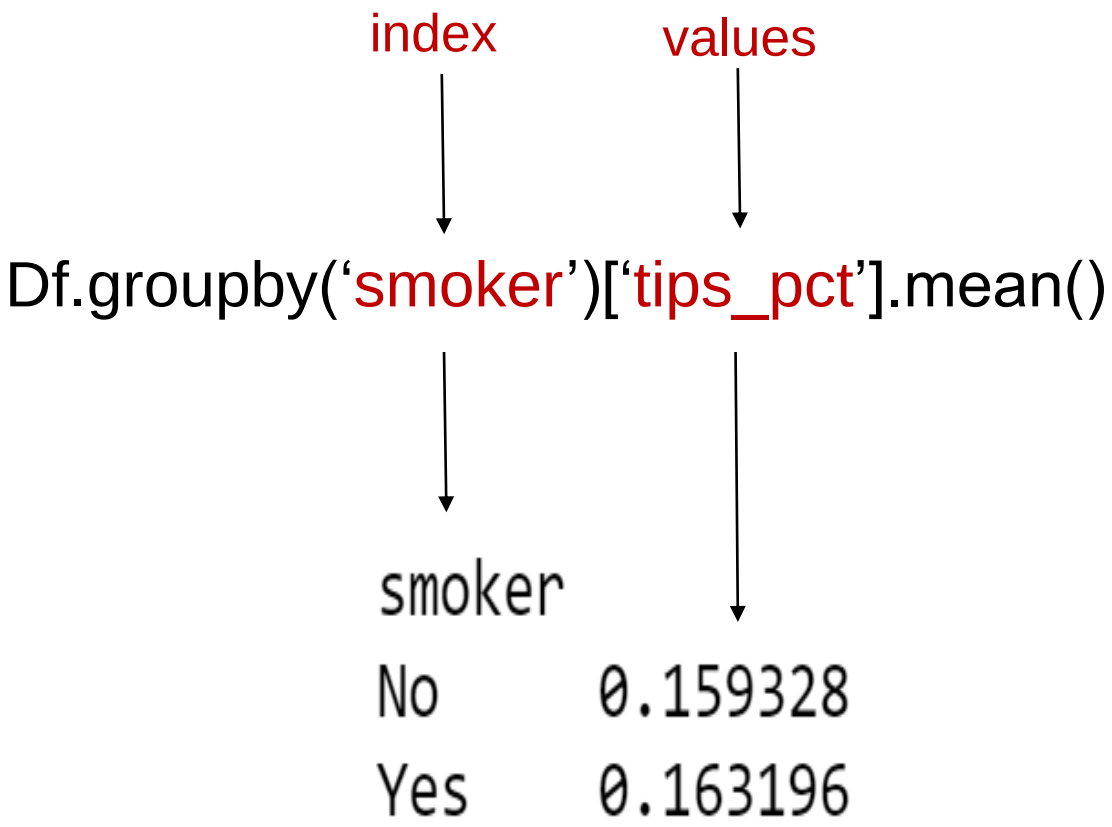
```
df.groupby([col1,col2,…,coln].apply(function)
```

- Ex:
  - Write a function that selects the rows with the largest values in a particular column
  - Group by smoker, apply this function

Data Aggregation
Apply: General split-apply-combine

- Groupby.agg, is very good for applying cython optimized functions (i.e ‘sum’,’mean’,’std’ etc. very fast)
- It also allows calculating multiple functions on different columns
- .agg() gives the flexiblility of applying multiple functions at once, or pass a list of function to each column
- Also, applying different functions at once to different columns of dataframe

Pivot Tables



- There is another method used to aggregate data – the DataFrame.pivot_table() method
- Index and values are arguments used in this method

Pivot Tables

```
Df.groupby('smoker')['tips_pct'].mean()
```

Equivalent to

```
Df.pivot_table(values = 'tips_pct', index = 'smoker', aggfunc = 'mean')
```

Pivot Tables

Table 10-2. pivot_table options

Function name	Description
`values`	Column name or names to aggregate; by default aggregates all numeric columns
`index`	Column names or other group keys to group on the rows of the resulting pivot table
`columns`	Column names or other group keys to group on the columns of the resulting pivot table
`aggfunc`	Aggregation function or list of functions (`'mean'` by default); can be any function valid in a groupby context
`fill_value`	Replace missing values in result table
`dropna`	If `True`, do not include columns whose entries are all NA
`margins`	Add row/column subtotals and grand total (`False` by default)

Pivot Tables

- The pivot_table method also allows us to aggregate multiple columns and apply multiple functions at once
- Ex: tips.pivot_table(values = ['total_bill','tip_pct'],index = ['smoker'])

smoker	tip_pct	total_bill
No	0.159328	19.188278
Yes	0.163196	20.756344

- To apply multiple functions, we can pass a list of the functions into the aggfunc parameter
  - Ex: tips_df.pivot_table('total_bill','smoker',aggfunc = ['mean', 'min','max'])

	mean	min	max
	total_bill	total_bill	total_bill
smoker			
No	19.188278	7.25	48.33
Yes	20.756344	3.07	50.81

- Try setting the margins parameter equal to True

Pivot Tables

- We want to aggregate only tip_pct and size. We want to put smoker in the table columns and day in the rows
- How to do it with pd.pivot_table()?

	size		tip_pct	
smoker	No	Yes	No	Yes
day				
Fri	2.250000	2.066667	0.151650	0.174783
Sat	2.555556	2.476190	0.158048	0.147906
Sun	2.929825	2.578947	0.160113	0.187250
Thur	2.488889	2.352941	0.160298	0.163863

Pivot Tables

- If some combinations are empty, we can replace the missing value using the fill_value parameter
- Ex: tips.pivot_table('tip_pct',index = ['time','size','smoker'], columns = 'day', aggfunc = 'mean')
- Try setting the fill_value parameter equal to 0

Guide Project

- Read the World_Happiness_2015.csv file into a DataFrame
- Print the first five rows of the DataFrame to become familiar with the data
- Use the DataFrame.info() method to print information about the DataFrame
- Use the df.groupby() method to group happiness2015 by the Region column. Assign the result to grouped
- Select only the Happiness Score column from grouped. Assign the result to happy_grouped
- Use the GroupBy.mean() method to compute the mean of happy_grouped. Assign the result to happy_mean

Guide Project

- Data info

Columns	Description
Country	Name of the country
Region	Region the country belongs to
Happiness Rank	Rank of the country based on the Happiness Score
std	The standard error of the happiness score.
Health	The extent to which Life expectancy contributed to the calculation of the Happiness Score
Freedom	The extent to which Freedom contributed to the calculation of the Happiness Score.
Governant	The extent to which Perception of Corruption contributes to Happiness Score.

Guide Project

- Apply the GroupBy.agg() method to happy_grouped. Pass a list containing np.mean and np.max into the method. Assign the result to happy_mean_max
- Create a custom function named dif to calculate the difference between the mean and maximum values. Pass dif into the GroupBy.agg() method. Assign the result to mean_max_dif
- Use the df.groupby() method to calculate the minimum, maximum, and mean family and happiness scores for each region in happiness2015
    - Group happiness2015 by the Region column
    - Select the Happiness Score and Family columns. Assign the result to grouped
    - Apply the GroupBy.agg() method to grouped. Pass a list containing min, max and mean into the method
    - Assign the result to happy_family_stats

Guide Project

- Use the pivot_table method to return the same information, but also calculate the minimum, maximum, and mean for the entire Family and Happiness Score columns
  - The aggregation columns should be Happiness Score and Family
  - The column to group by is Region
  - The aggregation functions are np.min, np.max, and np.mean
  - Set the margins parameter equal to True
  - Assign the result to pv_happy_family_stats