

Module 2: Exploratory Data Analysis

Part 2: Two Dimensional Data

The Data Frame

```
df = pd.DataFrame( <data>, <index>, <column_names> )
```

CSV

```
df = pd.read_csv( <file> )
```

```
df = pd.read_csv( <url> )
```

Excel

```
df = pd.read_excel( <file>, sheetname=<sheetname> )
```

```
[  
  {  
    "changed": "2015-04-03 00:00:00",  
    "created": "2014-04-10 00:00:00",  
    "dnssec": "False",  
    "expires": "2016-04-10 00:00:00",  
    "isMalicious": 0,  
    "url": "nuteczki.com"  
  },  
  ...  
]
```

JSON

```
df = pd.read_json( <file> )  
      or  
df = pd.read_json( <url> )
```

From a Database

```
df = pd.read_sql( <query>, <connection> )
```




Parquet

```
df = pd.read_parquet( <file> )
```

HTML

```
df = pd.read_html( <source> )
```

XML

```
import requests

user_agent_url = 'http://www.user-agents.org/allagents.xml'
xml_data = requests.get(user_agent_url).content
```

<http://www.austintaylor.io/lxml/python/pandas/xml/dataframe/2016/07/08/convert-xml-to-pandas-dataframe/>

XML

```
import xml.etree.ElementTree as ET

class XML2DataFrame:

    def __init__(self, xml_data):
        self.root = ET.XML(xml_data)

    def parse_root(self, root):
        return [self.parse_element(child) for child in iter(root)]

    def parse_element(self, element, parsed=None):
        if parsed is None:
            parsed = dict()
        for key in element.keys():
            parsed[key] = element.attrib.get(key)
        if element.text:
            parsed[element.tag] = element.text
        for child in list(element):
            self.parse_element(child, parsed)
        return parsed

    def process_data(self):
        structure_data = self.parse_root(self.root)
        return pd.DataFrame(structure_data)

xml2df = XML2DataFrame(xml_data)
xml_dataframe = xml2df.process_data()
```

Log Files...

070823	21:00:32	1	Connect	root@localhost on test1
070823	21:00:48	1	Query	show tables
070823	21:00:56	1	Query	select * from category
070917	16:29:01	21	Query	select * from location
070917	16:29:12	21	Query	select * from location where id = 1 LIMIT 1

```

070823 21:00:32      1 Connect      root@localhost on test1
070823 21:00:48      1 Query       show tables
070823 21:00:56      1 Query       select * from category
070917 16:29:01     21 Query       select * from location
070917 16:29:12     21 Query       select * from location where id = 1 LIMIT 1

```

```
logdf = pd.read_table('../data/mysql.log', names=[ 'raw' ])
```

	raw
0	070823 21:00:32 1 Connect root@local...
1	070823 21:00:48 1 Query show tables
2	070823 21:00:56 1 Query select * f...
3	070917 16:29:01 21 Query select * f...
4	070917 16:29:12 21 Query select * f...

```
logdf = pd.read_table( '../data/mysql.log', names=[ 'raw' ] )
logdf[ 'raw' ].str.extract( ' (\\d{6}\\s\\d{2}:\\d{2}:\\d{2})\\s+(\\d+)\\s(\\S+)\\s(\\S+)' , expand=False)
```

	0	1	2	3
0	070823 21:00:32	1	Connect	root@localhost on test1
1	070823 21:00:48	1	Query	show tables
2	070823 21:00:56	1	Query	select * from category
3	070917 16:29:01	21	Query	select * from location
4	070917 16:29:12	21	Query	select * from location where id = 1 LIMIT 1


```
logdf = pd.read_table('../data/mysql.log', names=[ 'raw' ])
logdf[ 'raw' ].str.extract( '(?P<date>\d{6}\s\d{2}:\d{2}:\d{2})\s+(?P<PID>\d+)\s(?P<Action>\S+)\s(?P<Query>.+)', expand=False)
```

	Date	PID	Action	Query
0	070823 21:00:32	1	Connect	root@localhost on test1
1	070823 21:00:48	1	Query	show tables
2	070823 21:00:56	1	Query	select * from category
3	070917 16:29:01	21	Query	select * from location
4	070917 16:29:12	21	Query	select * from location where id = 1 LIMIT 1

Web Server Logs



Web Server Logs

```
195.154.46.135 - - [25/Oct/2015:04:11:25 +0100] "GET /linux/  
doing-pxe-without-dhcp-control HTTP/1.1" 200 24323 "http://  
howto.basjes.nl/" "Mozilla/5.0 (Windows NT 5.1; rv:35.0)  
Gecko/20100101 Firefox/35.0"
```

Web Server Logs

```
195.154.46.135 - - [25/Oct/2015:04:11:25 +0100] "GET /linux/  
doing-pxe-without-dhcp-control HTTP/1.1" 200 24323 "http://  
howto.basjes.nl/" "Mozilla/5.0 (Windows NT 5.1; rv:35.0)  
Gecko/20100101 Firefox/35.0"
```

```
import apache_log_parser  
line_parser = apache_log_parser.make_parser("%h %l %u %t  
\"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"")
```

Web Server Logs

```
import apache_log_parser
line_parser = apache_log_parser.make_parser("%h %l %u %t
\"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"")

server_log = open("../data/hackers-access.httpd", "r")
parsed_server_data = []
for line in server_log:
    data = {}
    data = line_parser(line)
    parsed_server_data.append( data )

server_df = pd.DataFrame( parsed_server_data )
```


Web Server Logs

request_first_line	request_header_referer	request_header_user_agent	request_http_ver	request_method	request_url
GET /linux/doing-pxe-without-dhcp-control HTTP/1.1	http://howto.basjes.nl/	Mozilla/5.0 (Windows NT 5.1; rv:35.0) Gecko/20...	1.1	GET	/linux/doing-pxe-v
GET /join_form HTTP/1.0	http://howto.basjes.nl/	Mozilla/5.0 (Windows NT 5.1; rv:35.0) Gecko/20...	1.0	GET	/join_form
POST /join_form HTTP/1.1	http://howto.basjes.nl/join_form	Mozilla/5.0 (Windows NT 5.1; rv:35.0) Gecko/20...	1.1	POST	/join_form
GET /join_form HTTP/1.0	http://howto.basjes.nl/	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:34.0) G...	1.0	GET	/join_form
POST /join_form HTTP/1.1	http://howto.basjes.nl/join_form	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:34.0) G...	1.1	POST	/join_form
GET /acl_users/credentials_cookie_auth/require...	http://howto.basjes.nl/join_form	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:34.0) G...	1.1	GET	/acl_users/creden

Windows Event Logs

```
pip install python-evtx
```

```
import Evtx.Evtx as evtx
```

```
xml = "<Events>"
```

```
with evtx.Evtx("window_event_log.evtx") as log:
```

```
    for record in log.records():
```

```
        xml += record.xml()
```

```
xml += "</Events>"
```


Nested Data?



Nested Data?

```
{ "time": 1084443427.311224,  
  "timestamp": "2004-05-13T10:17:07.311224",  
  "IP": {  
    "version": 4,  
    "ttl": 128,  
    "proto": 6,  
    "options": [],  
    "len": 48,  
    "dst": "65.208.228.223",  
    "frag": 0,  
    "flags": 2, "src": "145.254.160.237",  
    "chksum": 37355  
  },  
  "Ethernet": { "src": "00:00:01:00:00:00", "type": 2048, "dst": "fe:ff:20:00:01:00" },  
  ...  
}
```

Nested Data

```
pd.read_json( 'nested_data.json' )
```

```
pd.read_json( 'nested_data.json' )
```

	DNS	Ethernet	IP	TCP	UDP	time	timestamp
0	NaN	{'type': 2048, 'dst': 'fe:ff:20:00:01:00', 'sr...	{'dst': '65.208.228.223', 'len': 48, 'version'...	{'window': 8760, 'chksum': 49932, 'sport': 337...	NaN	1.084443e+09	2004-05-13 10:17:07.311224
1	NaN	{'type': 2048, 'dst': '00:00:01:00:00:00', 'sr...	{'dst': '145.254.160.237', 'len': 48, 'version'...	{'window': 5840, 'chksum': 23516, 'sport': 80,...	NaN	1.084443e+09	2004-05-13 10:17:08.222534
2	NaN	{'type': 2048, 'dst': 'fe:ff:20:00:01:00', 'sr...	{'dst': '65.208.228.223', 'len': 40, 'version'...	{'window': 9660, 'chksum': 31076, 'sport': 337...	NaN	1.084443e+09	2004-05-13 10:17:08.222534
3	NaN	{'type': 2048, 'dst': 'fe:ff:20:00:01:00', 'sr...	{'dst': '65.208.228.223', 'len': 519, 'version'...	{'window': 9660, 'chksum': 43352, 'sport': 337...	NaN	1.084443e+09	2004-05-13 10:17:08.222534
4	NaN	{'type': 2048, 'dst': '00:00:01:00:00:00', 'sr...	{'dst': '145.254.160.237', 'len': 40, 'version'...	{'window': 6432, 'chksum': 33825, 'sport': 80,...	NaN	1.084443e+09	2004-05-13 10:17:08.783340

Nested Data

```
from pandas.io.json import json_normalize
import json
import pandas as pd

with open('nested.json') as data_file:
    pcap_data = json.load(data_file)

df = pd.DataFrame( json_normalize(pcap_data) )
```

```
df = pd.DataFrame( json_normalize(pcap_data) )
```

...	TCP.seq	TCP.sport	TCP.urgptr	TCP.window	L
...	951057939.0	3372.0	0.0	8760.0	
...	290218379.0	80.0	0.0	5840.0	
...	951057940.0	3372.0	0.0	9660.0	
...	951057940.0	3372.0	0.0	9660.0	
...	290218380.0	80.0	0.0	6432.0	

ElasticSearch & Splunk



splunk[®]>

ElasticSearch & Splunk

```
pip install huntlib
```

```
e = ElasticDF(  
    url="https://localhost:9200",  
    ssl=True,  
    username="myuser",  
    password="mypass"  
)  
df = e.search_df(  
    lucene="proto:tcp AND port:80",  
    index="myindex-*",  
    days=1,  
    normalize=False  
)
```

```
s = SplunkDF(  
    host "<splunk_ip>",  
    username="myuser",  
    password="mypass"  
)  
df = s.search_df(  
    spl='search index=win_events src="10.9.*.*"',  
    start_time=datetime.now()-timedelta(days=2),  
    end_time=datetime.now()  
)
```

Manipulating a DataFrame


```
series = df['column1']
```

Returns a **series**

```
df['ip'].value_counts().head()
```

Two Ways of Accessing Columns



```
series = df['column1']
```

```
series = df.column1
```

Don't use the dots!

```
df = df[ ['column1', 'column2', 'column3'] ]
```

Returns a DataFrame

Filtering a DataFrame

```
df[<boolean condition>]
```

```
df[['col1', 'col2']][df['col3'] > 5]
```

↑
Columns

↑
Filter

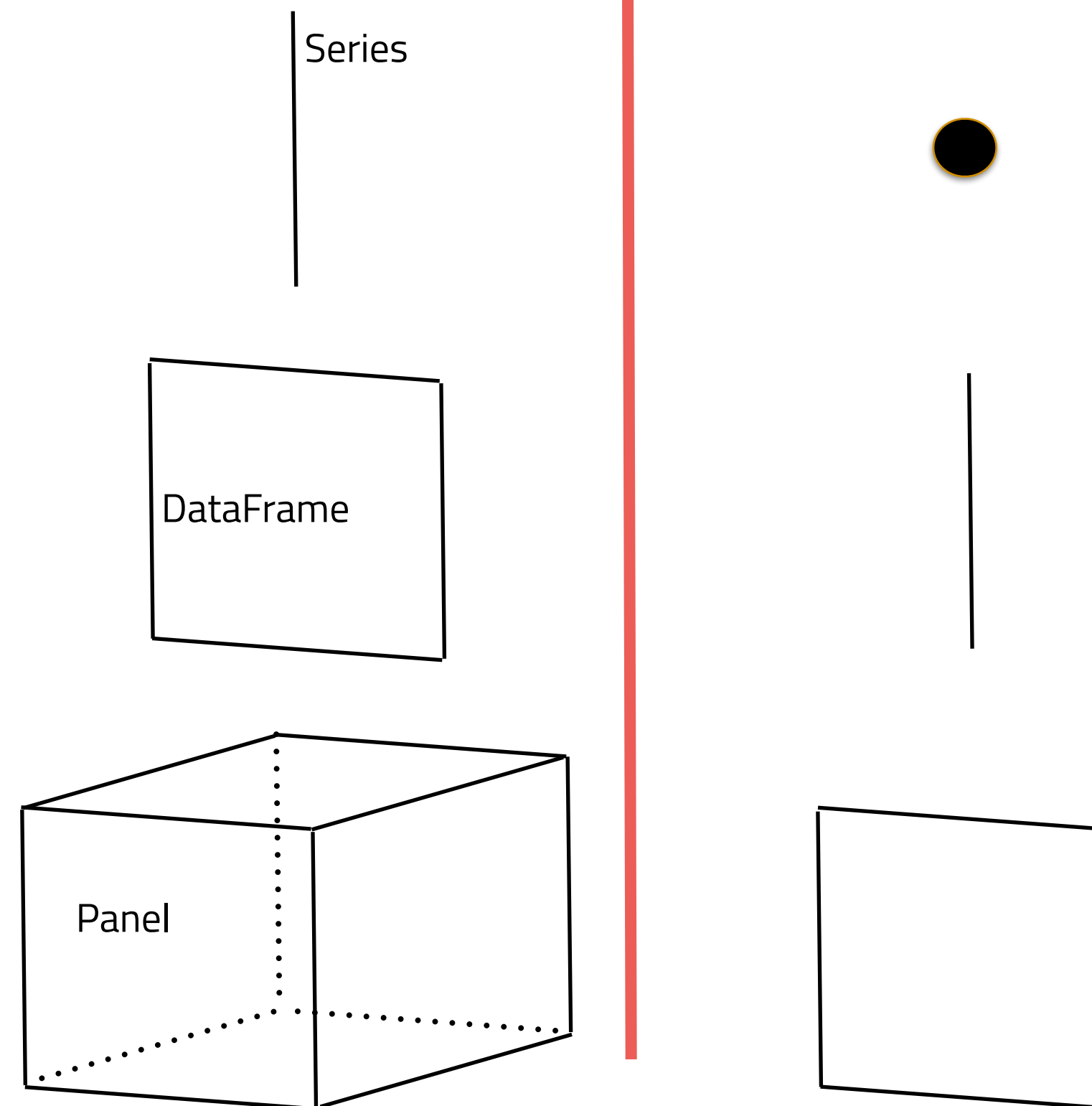
```
data.loc[<index>]  
data.loc[<list of indexes>]  
data.sample(<n>)
```

•

`data.apply(<function>)`

IF
Call Apply On...

THEN
Function
Receives



•

`data.apply(<function>)`

IF

Call Apply On...

Series

DataFrame

Panel

THEN

Function
Receives



IF

Function
Returns



THEN

Apply Returns...



Dataframe Demo

Exercise 1

```
#Data1
```

```
{ "first_name": { "0": "Robert", "1": "Steve", "2": "Anne", "3": "Alice" },  
  "last_name": { "0": "Hernandez", "1": "Smith", "2": "Raps", "3": "Muller" },  
  "birthday": { "0": "5\\/3\\/67", "1": "8\\/4\\/84", "2": "9\\/13\\/91", "3": "4\\/15\\/75" }  
}
```

```
df1 = pd.read_json( '.. /data/data1.json' )
```

Exercise 1

```
#Data2
{"0":{"first_name":"Robert","last_name":"Hernandez","birthday":"5\\/3\\
\\/67"},
"1":{"first_name":"Steve","last_name":"Smith","birthday":"8\\/4\\
84"},
"2":{"first_name":"Anne","last_name":"Raps","birthday":"9\\/13\\/91"},
"3":{"first_name":"Alice","last_name":"Muller","birthday":"4\\/15\\
75"}}

df2 = pd.read_json( '../data/data2.json', orient='index' )
```

Exercise 1

```
#Data3
[
{"first_name": "Robert", "last_name": "Hernandez", "birthday": "5\\/3\\/67"},
{"first_name": "Steve", "last_name": "Smith", "birthday": "8\\/4\\/84"},
{"first_name": "Anne", "last_name": "Raps", "birthday": "9\\/13\\/91"},
{"first_name": "Alice", "last_name": "Muller", "birthday": "4\\/15\\/75"}]

df3 = pd.read_json( '.. /data/data3.json',
orient='columns' )
```

Exercise 1

```
#Data4
{"columns":["first_name","last_name","birthday"],
"index":[0,1,2,3],
"data":[["Robert","Hernandez","5\\/3\\/67"],["Steve","Smith","8\\/4\\/84"],
["Anne","Raps","9\\/13\\/91"],["Alice","Muller","4\\/15\\/75"]]}

df4 = pd.read_json(' ../data/data4.json', orient='split')
```

Exercise 2

#Write the functions

```
def get_os(x):  
    user_agent = parse(x)  
    return user_agent.os.family
```

```
def get_browser(x):  
    user_agent = parse(x)  
    return user_agent.browser.family
```

Exercise 2

```
#Apply the functions to the dataframe
server_df['os'] =
server_df['request_header_user_agent'].apply( get_os )
server_df['browser'] =
server_df['request_header_user_agent'].apply( get_browser )
```

Exercise 2

```
#Apply the functions to the dataframe
```

```
server_df['os'] =
```

```
server_df['request_header_user_agent'].apply( get_os )
```

```
server_df['browser'] =
```

```
server_df['request_header_user_agent'].apply( get_browser )
```

```
#Get the top 10 values
```

```
server_df['os'].value_counts().head(10)
```

Exercise 2

```
#Get the top 10 values  
server_df['os'].value_counts().head(10)
```

Windows 7	2041
Windows Vista	500
Windows XP	423
Windows 8.1	221
Linux	125
Mac OS X	80
Chrome OS	60
Ubuntu	6

Exercise 3

```
bots = pd.read_csv('../data/dailybots.csv')
gov_bots = bots[['botfam', 'hosts']][bots['industry'] ==
"Government/Politics"]

gov_bots.groupby('botfam', as_index=False).sum()
```

Questions?

1	2	3
4	5	6
7	8	9

`data.T`

1	4	7
2	5	8
3	6	9

#Gets you the sum of columns
`data.sum(axis=0)`

#Gets you the sum of the rows
`data.sum(axis=1)`

```
DataFrame.drop(labels,  
axis=0,  
level=None,  
inplace=False,  
errors='raise')¶
```

Merging Data Sets

Union

Union

Series 1

Index	Value
0	6
1	4
2	2
3	3

Union

Series 1

Index	Value
0	6
1	4
2	2
3	3

Series 2

Index	Value
0	7
1	5
2	3
3	4

Union

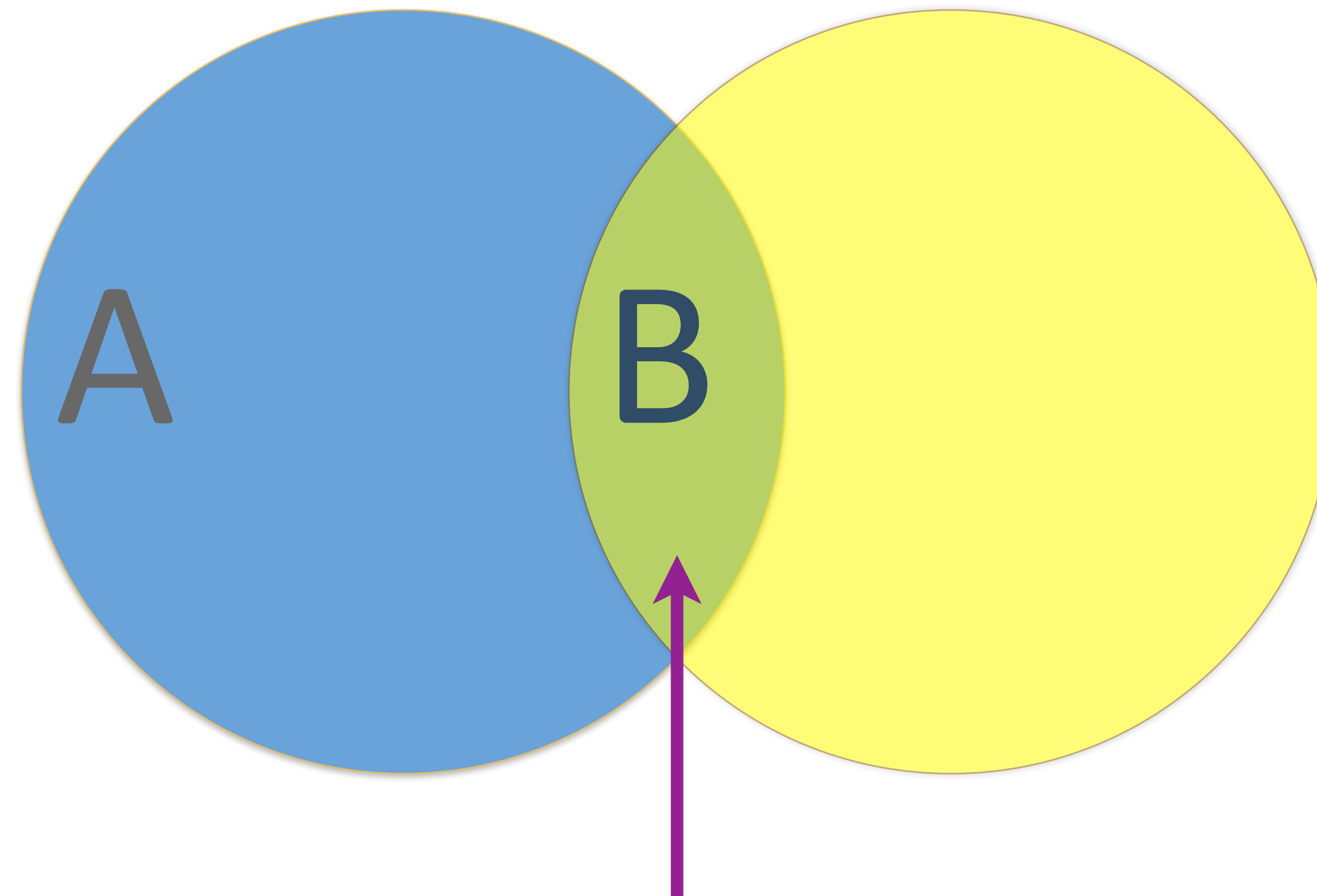
combinedSeries

Index	Value
0	6
1	4
2	2
3	3
4	7
5	5
6	3
7	4

```
combinedSeries = pd.concat(  
    [series1, series2],  
    ignore_index=True )
```

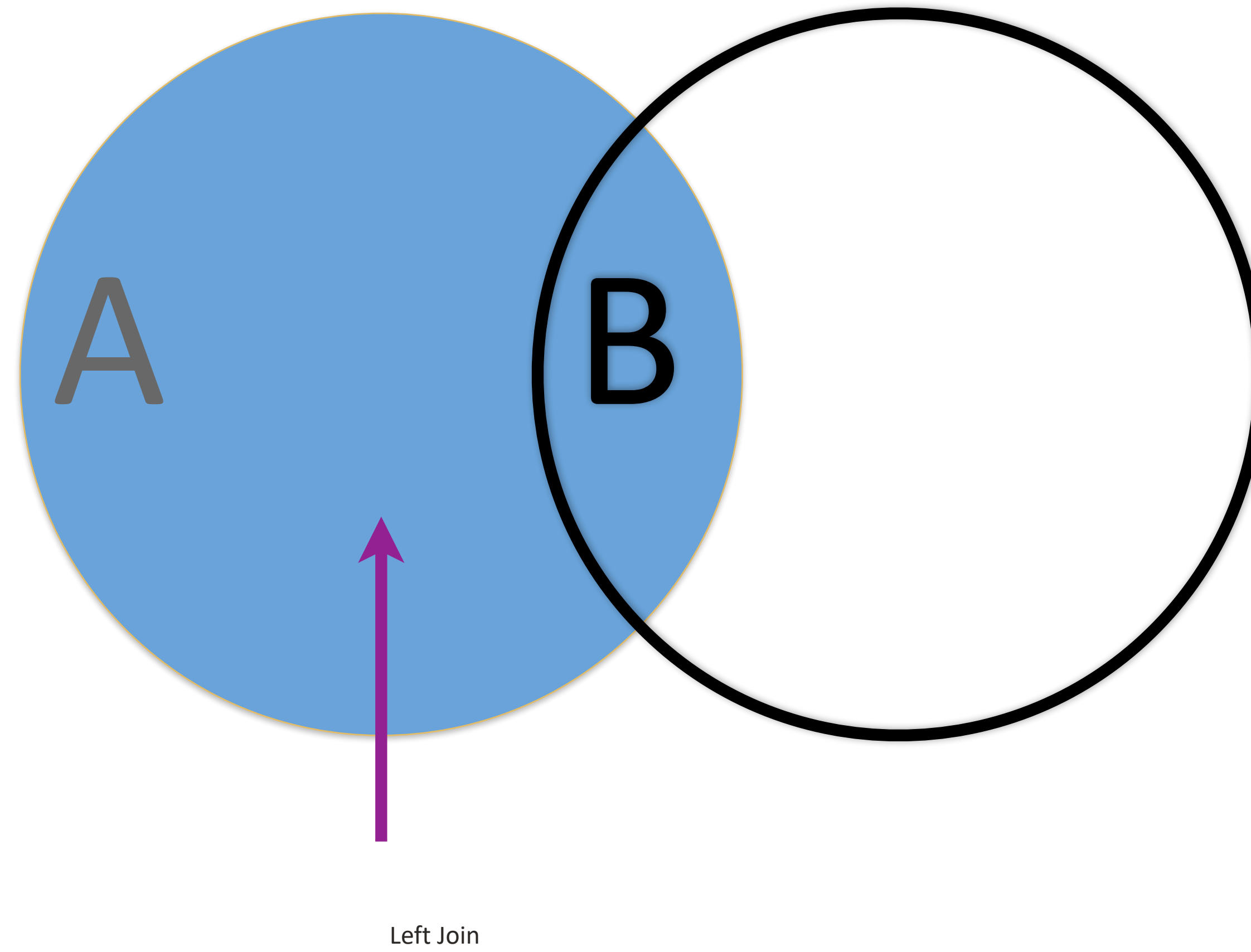
Joins

Inner Join (Intersection)

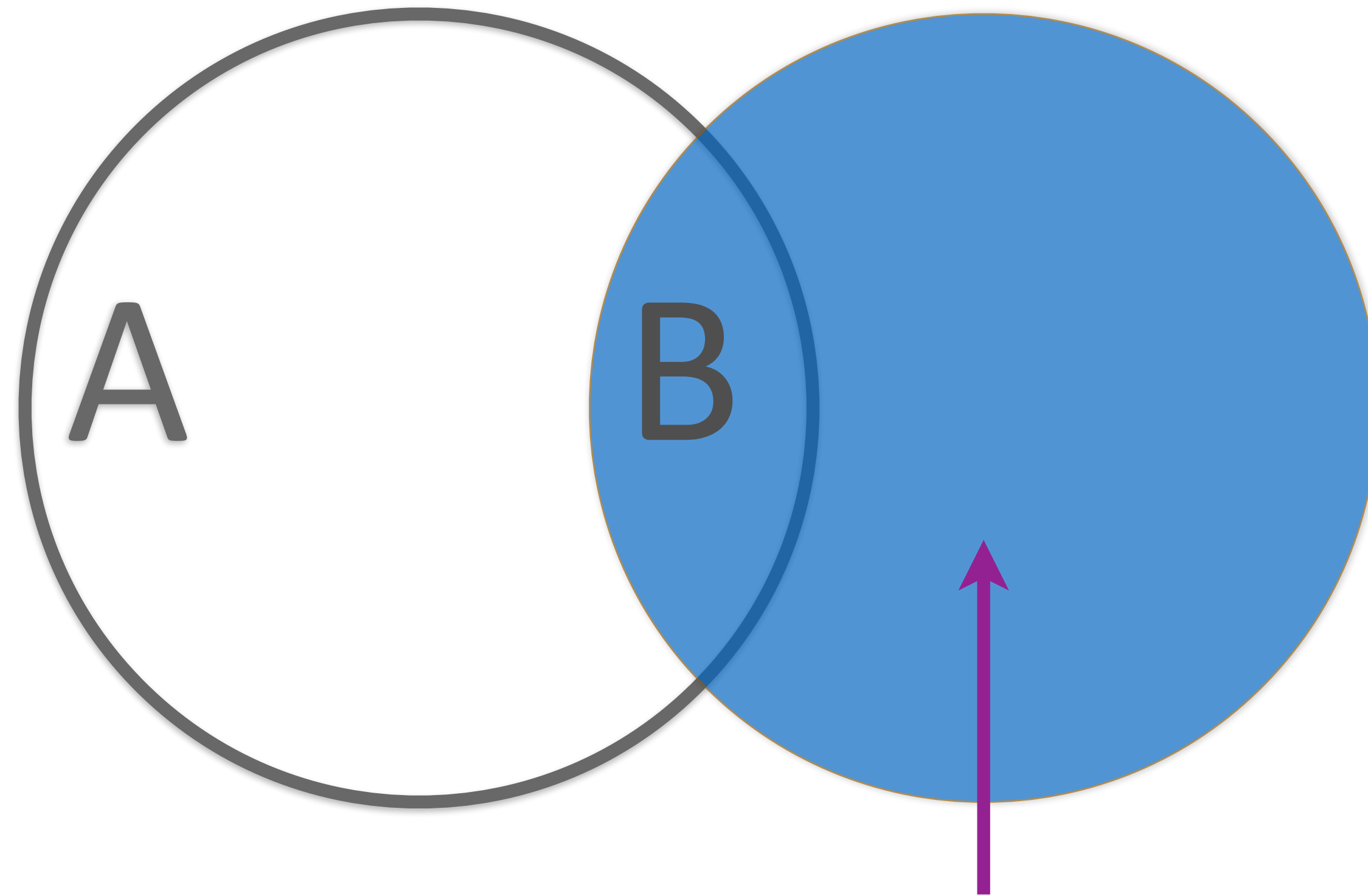


Inner Join

Left Join



Right Join



Right Join

```
pd.merge( leftData, rightData )
```



```
pd.merge( leftData,  
          rightData,  
          how="<join type>" )
```

```
pd.merge( leftData, rightData,  
how="<join type>" )
```

Option	SQL Equivalent	Description
inner	INNER JOIN	Intersection
left	LEFT OUTER JOIN	Returns items in Set A, but not in Set B
right	RIGHT OUTER JOIN	Returns items in Set B, but not in Set A
outer	FULL OUTER JOIN	Returns the union of both sets

```
pd.merge( leftData, rightData,  
          how="<join type>",  
          on="<field list> )
```

Grouping and Aggregating Data

Grouping and Aggregating Data

date	src_ip	dst_ip	port
2018-06-21	192.168.20.2	10.10.4.1	80
2018-06-21	192.168.20.1	10.10.4.2	443
2018-06-21	192.168.20.2	10.10.5.1	80
2018-06-22	192.168.20.2	10.10.4.1	80

`df.groupby(field or list of fields)`

Grouping and Aggregating Data

date	src_ip	dst_ip	port
2018-06-21	192.168.20.2	10.10.4.1	80
2018-06-21	192.168.20.1	10.10.4.2	443
2018-06-21	192.168.20.2	10.10.5.1	80
2018-06-22	192.168.20.2	10.10.4.1	80

```
df.groupby( 'src_ip' )[ 'port' ].count( )
```

src_ip	count
192.168.20.1	1
192.168.20.2	3

Grouping and Aggregating Data

date	src_ip	dst_ip	port
2018-06-21	192.168.20.2	10.10.4.1	80
2018-06-21	192.168.20.1	10.10.4.2	443
2018-06-21	192.168.20.2	10.10.5.1	80
2018-06-22	192.168.20.2	10.10.4.1	80

```
df.groupby(['date', 'src_ip'])['port'].count()
```

Multi-Index

date	src_ip	
2018-06-21	192.168.20.1	1
	192.168.20.2	2
2018-06-22	192.168.20.2	1

Grouping and Aggregating Data

date	src_ip	dst_ip	port
2018-06-21	192.168.20.2	10.10.4.1	80
2018-06-21	192.168.20.1	10.10.4.2	443
2018-06-21	192.168.20.2	10.10.5.1	80
2018-06-22	192.168.20.2	10.10.4.1	80

```
df.groupby([ 'date' , 'src_ip' ], as_index=False)  
[ 'port' ].count()
```

	date	src_ip	port
0	2018-06-21	192.168.20.1	1
1	2018-06-21	192.168.20.2	2
2	2018-06-22	192.168.20.2	1

Grouping Functions

Pivot

df

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t



```
df.pivot(index='foo',  
          columns='bar',  
          values='baz')
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

Grouping Functions

Melt

df3

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

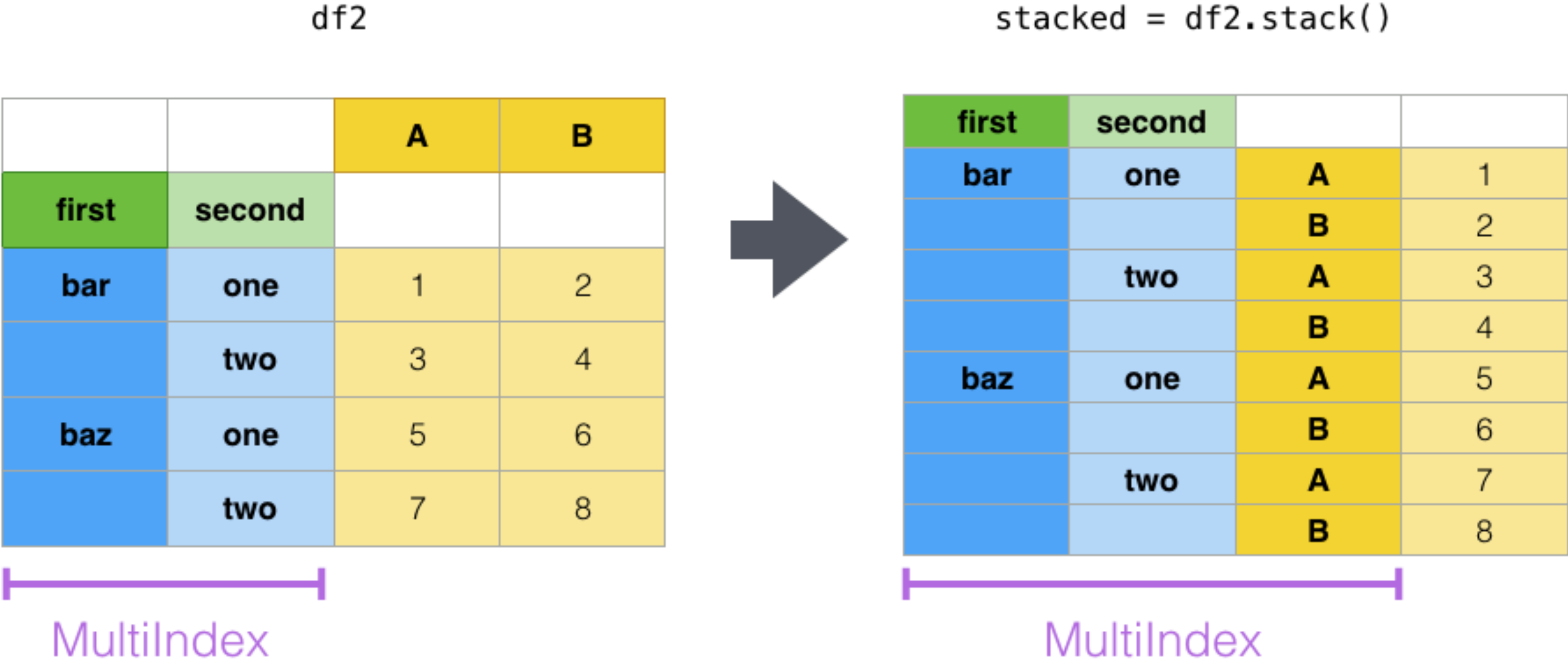


df3.melt(id_vars=['first', 'last'])

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

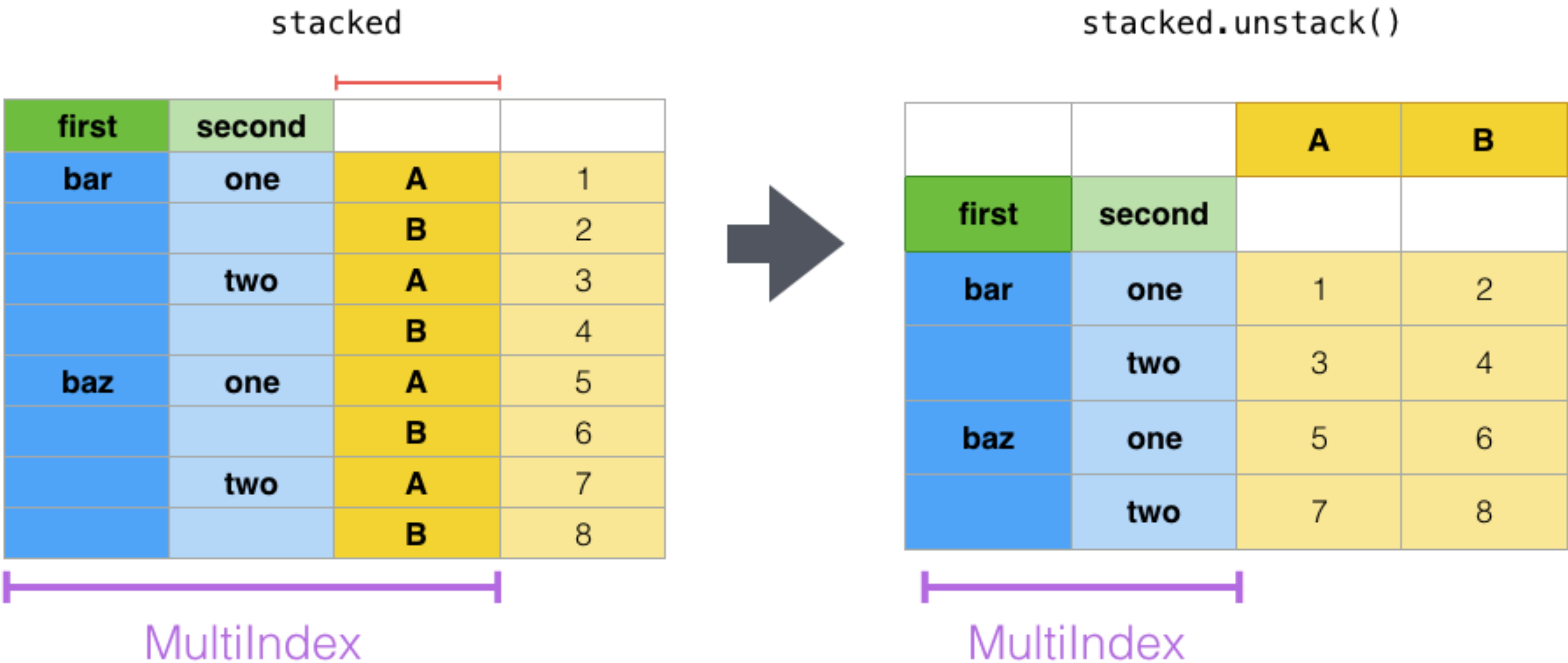
Grouping Functions

Stack



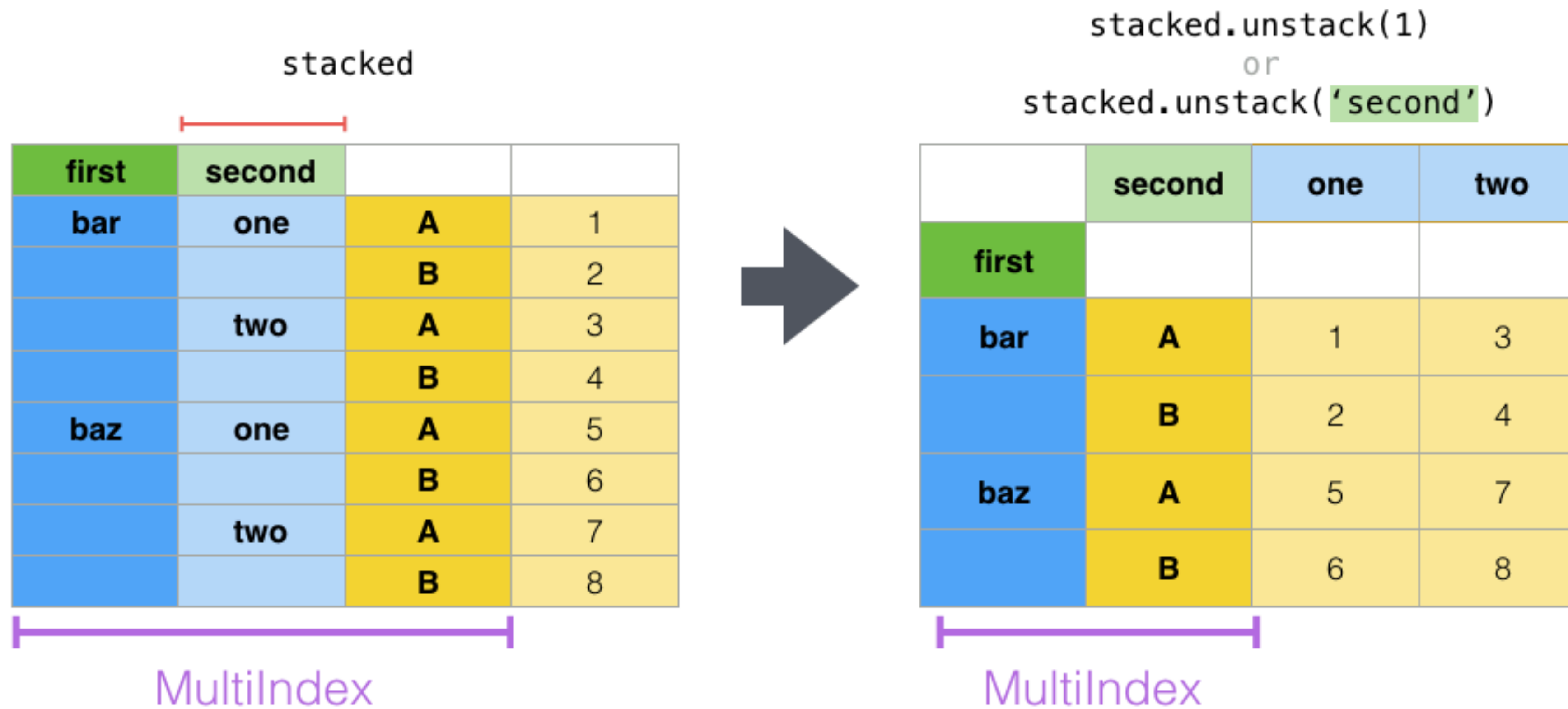
Grouping Functions

Unstack



Grouping Functions

Unstack(1)



```
Series.dropna( )  
Series.fillna(value="<something>" )
```

Group Exercise

In your groups, for your use case, discuss:

1. What data you will need?
2. How will you gather it?
3. What form is it in?
4. What will you have to do in order to transform it?