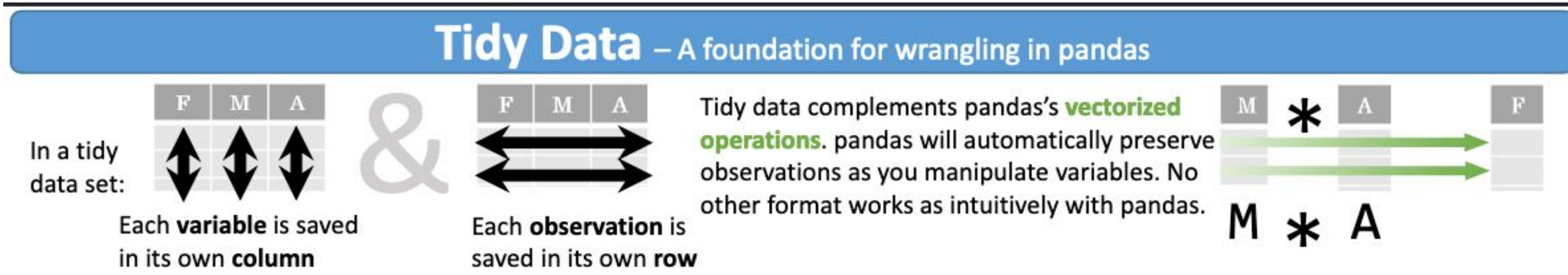
Data journalism
Week 3

Today

- tidy data
- subsetting and slicing
- merging and aggregating
- joining datasets

Tidy data



Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

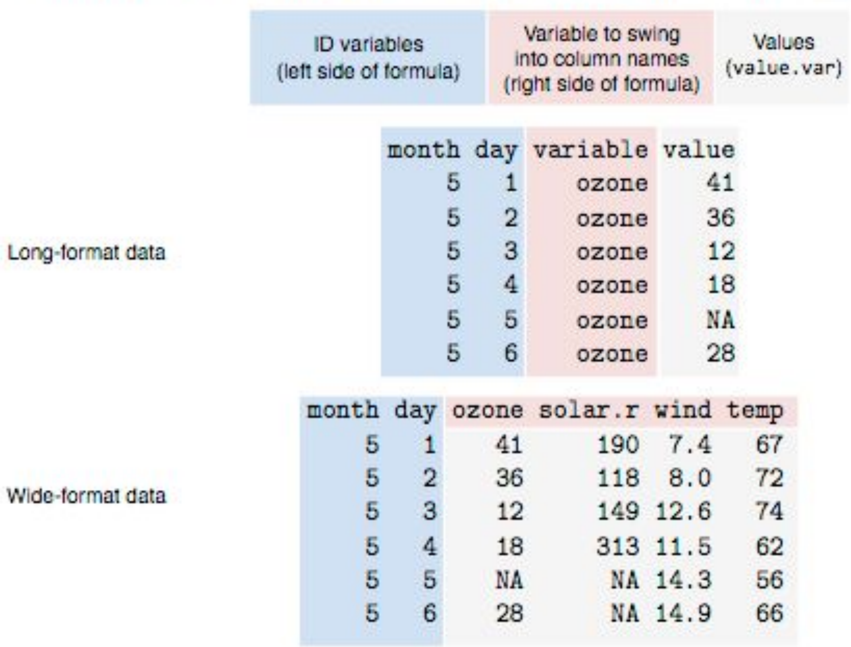
Each **variable** is saved in its own **column**

&

Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

See the article by Wickham:
https://www.jstatsoft.org/article/view/v059i10



ID variables (left side of formula)	Variable to swing into column names (right side of formula)	Values (value.var)

Long-format data

month	day	variable	value
5	1	ozone	41
5	2	ozone	36
5	3	ozone	12
5	4	ozone	18
5	5	ozone	NA
5	6	ozone	28

Wide-format data

month	day	ozone	solar.r	wind	temp
5	1	41	190	7.4	67
5	2	36	118	8.0	72
5	3	12	149	12.6	74
5	4	18	313	11.5	62
5	5	NA	NA	14.3	56
5	6	28	NA	14.9	66

Subsetting (slicing)

Why do we need this?

- We are seldom interested in a table as a whole.
- Rather, we may want to investigate specific columns (“variables”)
- Or, we may want to zoom into a specific interesting row
- Or we need to clean up messy data

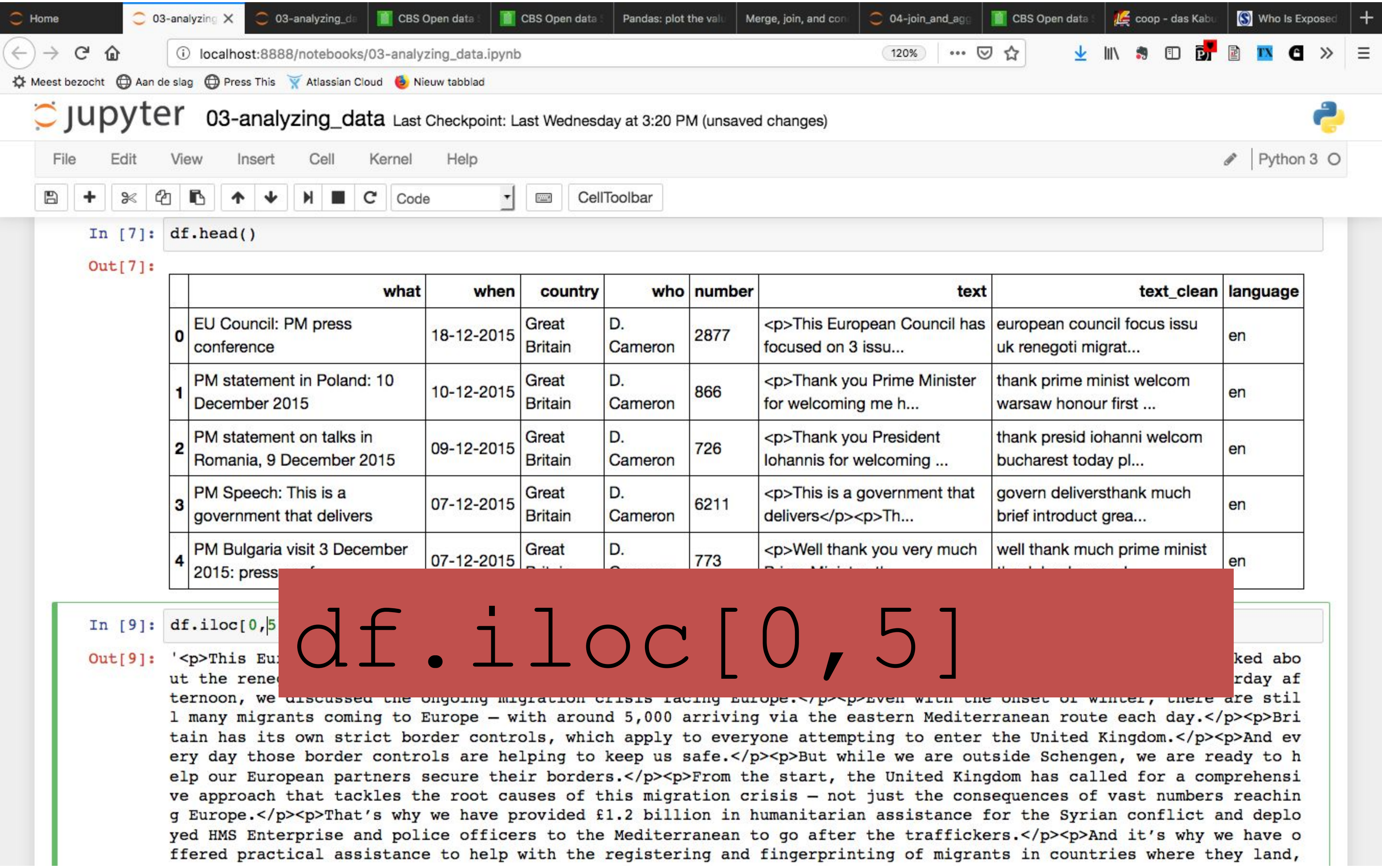
Some general subsetting (slicing)

- [0:5] to get elements 0, 1, 2, 3, 4 (works with lists, dataframes …)
- mydict[‘keyicareabout’] to get value (content) associated with the key
- df[[‘col1’, ‘col2’]] to get only these two *columns* of a dataset
- df[df[‘col1’]==‘whatever’] to get only the *rows* in which col1 is identical to the string ‘whatever’
- df[df[‘col2’]>0] to get only the *rows* in which col2 is a number bigger than 0

More subsetting and slicing

- To get a specific row and/or column, you can use .iloc[] and loc[]
  - iloc[] takes an int (the row/column *numbers*, loc[] the names)
  - df.iloc[0,5] to get row 0, column 5
  - df.loc[0,'what'] to get row 0, column 'what'

Check out the pandas cheat sheet

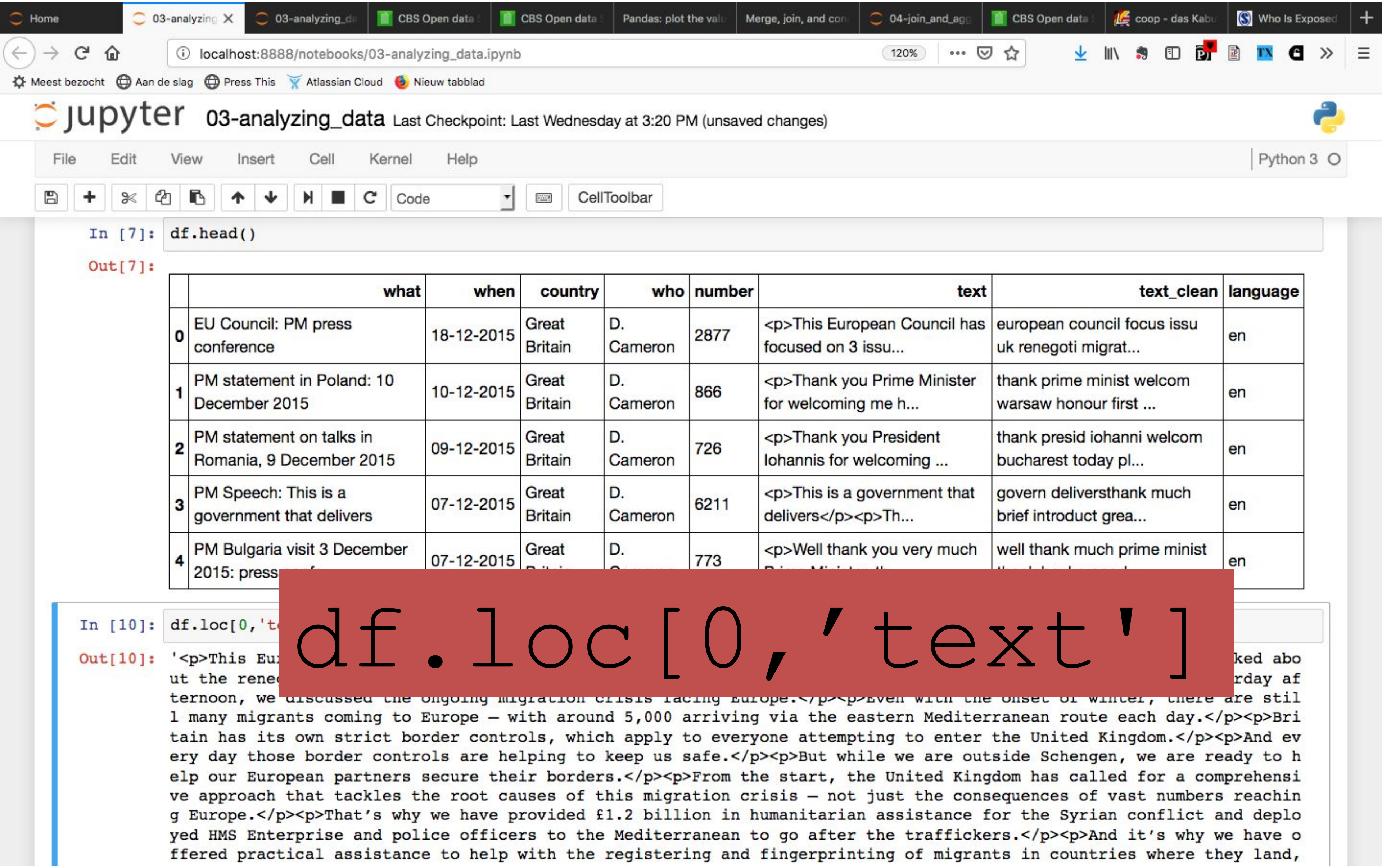


```
In [7]: df.head()
Out[7]:
```

	what	when	country	who	number	text	text_clean	language
0	EU Council: PM press conference	18-12-2015	Great Britain	D. Cameron	2877	<p>This European Council has focused on 3 issu...	european council focus issu uk renegoti migrat...	en
1	PM statement in Poland: 10 December 2015	10-12-2015	Great Britain	D. Cameron	866	<p>Thank you Prime Minister for welcoming me h...	thank prime minist welcom warsaw honour first ...	en
2	PM statement on talks in Romania, 9 December 2015	09-12-2015	Great Britain	D. Cameron	726	<p>Thank you President Iohannis for welcoming ...	thank presid iohanni welcom bucharest today pl...	en
3	PM Speech: This is a government that delivers	07-12-2015	Great Britain	D. Cameron	6211	<p>This is a government that delivers</p><p>Th...	govern deliversthank much brief introduct grea...	en
4	PM Bulgaria visit 3 December 2015: press	07-12-2015	Great	D.	773	<p>Well thank you very much	well thank much prime minist	en

```
In [9]: df.iloc[0,5
Out[9]: '<p>This Eu                                                                  ked abo
ut the rene                                                                  rday af
ternoon, we discussed the ongoing migration crisis facing Europe.</p><p>Even with the onset of winter, there are stil
l many migrants coming to Europe – with around 5,000 arriving via the eastern Mediterranean route each day.</p><p>Bri
tain has its own strict border controls, which apply to everyone attempting to enter the United Kingdom.</p><p>And ev
ery day those border controls are helping to keep us safe.</p><p>But while we are outside Schengen, we are ready to h
elp our European partners secure their borders.</p><p>From the start, the United Kingdom has called for a comprehensi
ve approach that tackles the root causes of this migration crisis – not just the consequences of vast numbers reachin
g Europe.</p><p>That's why we have provided £1.2 billion in humanitarian assistance for the Syrian conflict and deplo
yed HMS Enterprise and police officers to the Mediterranean to go after the traffickers.</p><p>And it's why we have o
ffered practical assistance to help with the registering and fingerprinting of migrants in countries where they land,
```

```
df.iloc[0,5]
```



```
In [7]: df.head()
```

Out[7]:

	what	when	country	who	number	text	text_clean	language
0	EU Council: PM press conference	18-12-2015	Great Britain	D. Cameron	2877	<p>This European Council has focused on 3 issu...	european council focus issu uk renegoti migrat...	en
1	PM statement in Poland: 10 December 2015	10-12-2015	Great Britain	D. Cameron	866	<p>Thank you Prime Minister for welcoming me h...	thank prime minist welcom warsaw honour first ...	en
2	PM statement on talks in Romania, 9 December 2015	09-12-2015	Great Britain	D. Cameron	726	<p>Thank you President Iohannis for welcoming ...	thank presid iohanni welcom bucharest today pl...	en
3	PM Speech: This is a government that delivers	07-12-2015	Great Britain	D. Cameron	6211	<p>This is a government that delivers</p><p>Th...	govern deliversthank much brief introduct grea...	en
4	PM Bulgaria visit 3 December 2015: press	07-12-2015	Great	D.	773	<p>Well thank you very much	well thank much prime minist	en

```
In [10]: df.loc[0,'t
```

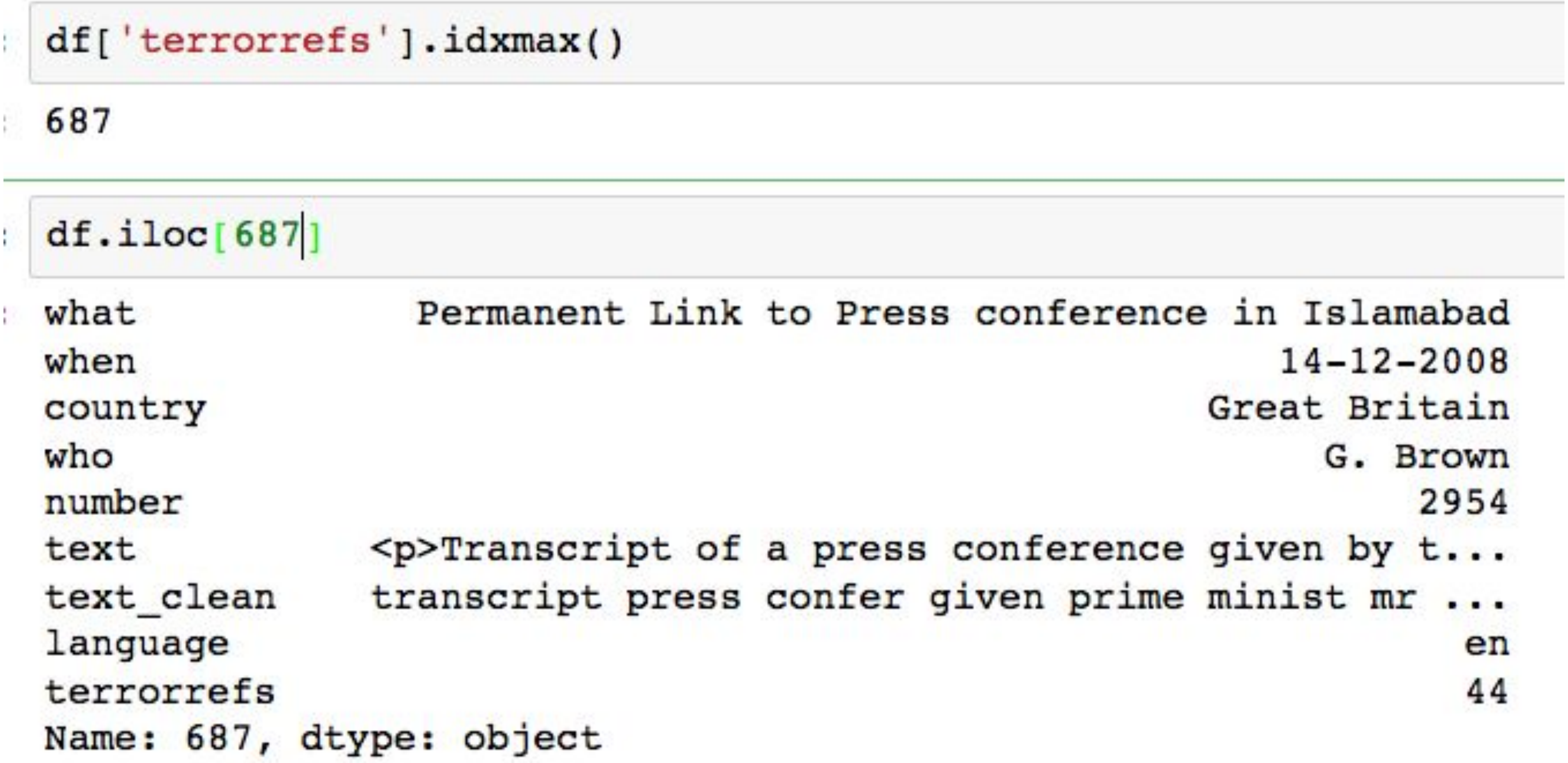
```
Out[10]: '<p>This Eu                                                                        ked abo
         ut the rene                                                                        rday af
         ternoon, we discussed the ongoing migration crisis facing Europe.</p><p>Even with the onset of winter, there are stil
         l many migrants coming to Europe – with around 5,000 arriving via the eastern Mediterranean route each day.</p><p>Bri
         tain has its own strict border controls, which apply to everyone attempting to enter the United Kingdom.</p><p>And ev
         ery day those border controls are helping to keep us safe.</p><p>But while we are outside Schengen, we are ready to h
         elp our European partners secure their borders.</p><p>From the start, the United Kingdom has called for a comprehensi
         ve approach that tackles the root causes of this migration crisis – not just the consequences of vast numbers reachin
         g Europe.</p><p>That’s why we have provided £1.2 billion in humanitarian assistance for the Syrian conflict and deplo
         yed HMS Enterprise and police officers to the Mediterranean to go after the traffickers.</p><p>And it’s why we have o
         ffered practical assistance to help with the registering and fingerprinting of migrants in countries where they land,
```

df.loc[0,’text']

More subsetting and slicing

- Advanced example: Get the whole row where the column ‘terrorrefs’ has the highest value in the whole dataset:
  - df.iloc[df[‘terrorrefs’].idxmax()]
- That works because df.iloc[] expects an integer to identify the row number, and df[‘terrorrefs’].idxmax() returns an integer (687 in our case)
- We could also do it in two steps:

df.iloc[df['terrorrefs'].idxmax()] is the same as:



```
df['terrorrefs'].idxmax()
```

```
687
```

```
df.iloc[687]
```

```
what            Permanent Link to Press conference in Islamabad
when                                                 14-12-2008
country                                           Great Britain
who                                                    G. Brown
number                                                     2954
text          <p>Transcript of a press conference given by t...
text_clean    transcript press confer given prime minist mr ...
language                                                     en
terrorrefs                                                   44
Name: 687, dtype: object
```

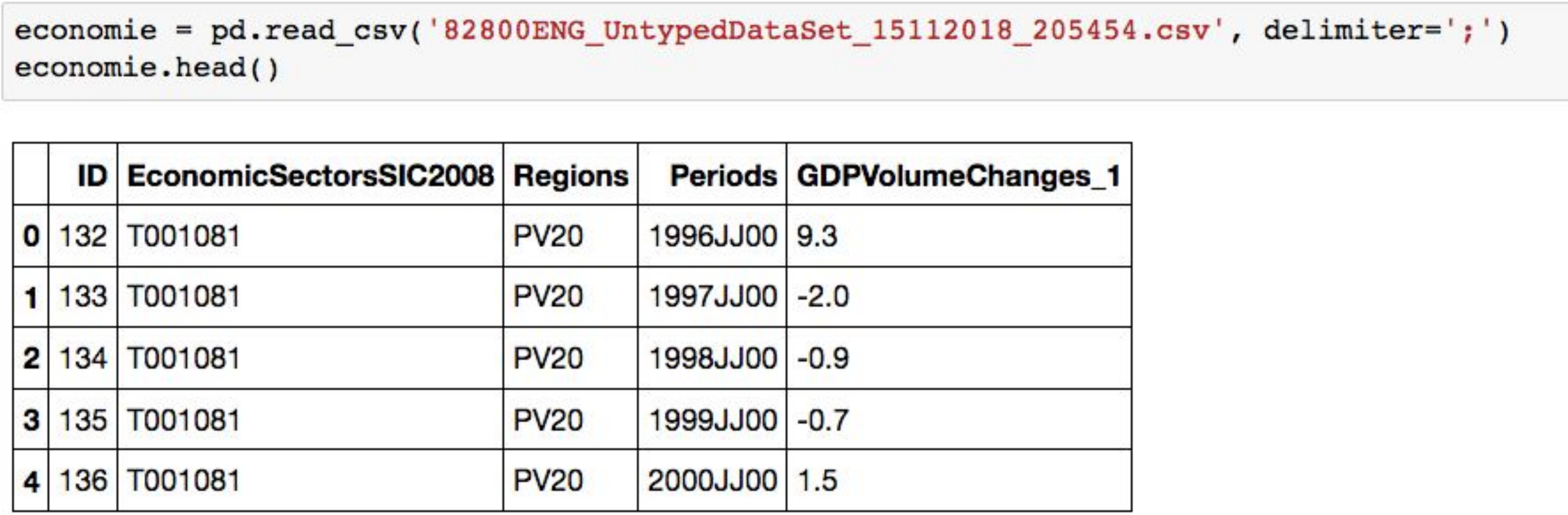
DO try this at home!

- Try to get specific columns and rows from *your* dataset.
- Try to select parts of your data that match some condition!

Merging two datasets

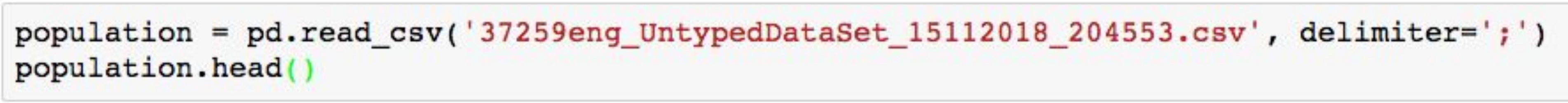
Why do we need this?

- 1+1=3
  - A dataset on poverty in different neighbourhoods may be modestly interesting, as well as a dataset on schools in different neighbourhoods. But combining them may make an interesting story.
- But: Consider the possibility of an ecological fallacy!
  - correlation ≠ causation

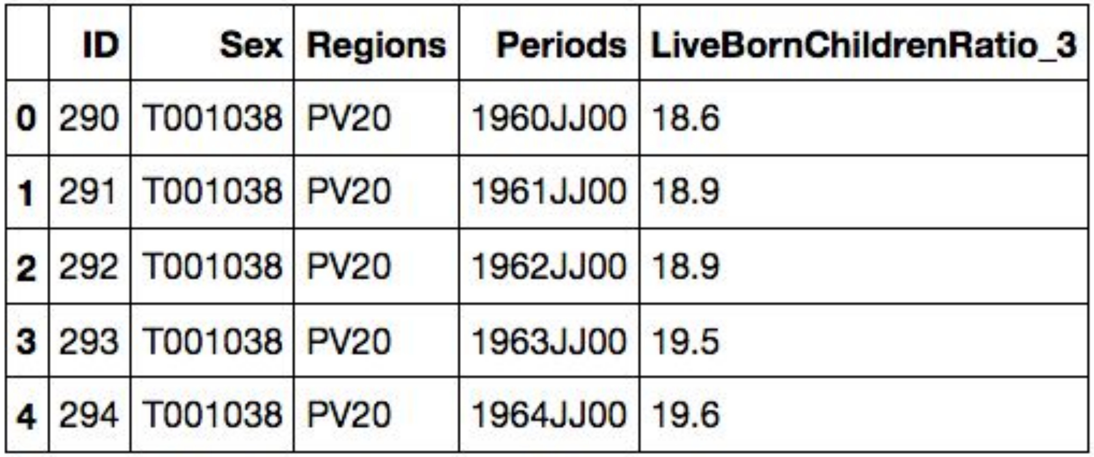


```
economie = pd.read_csv('82800ENG_UntypedDataSet_15112018_205454.csv', delimiter=';')
economie.head()
```

	ID	EconomicSectorsSIC2008	Regions	Periods	GDPVolumeChanges_1
0	132	T001081	PV20	1996JJ00	9.3
1	133	T001081	PV20	1997JJ00	-2.0
2	134	T001081	PV20	1998JJ00	-0.9
3	135	T001081	PV20	1999JJ00	-0.7
4	136	T001081	PV20	2000JJ00	1.5



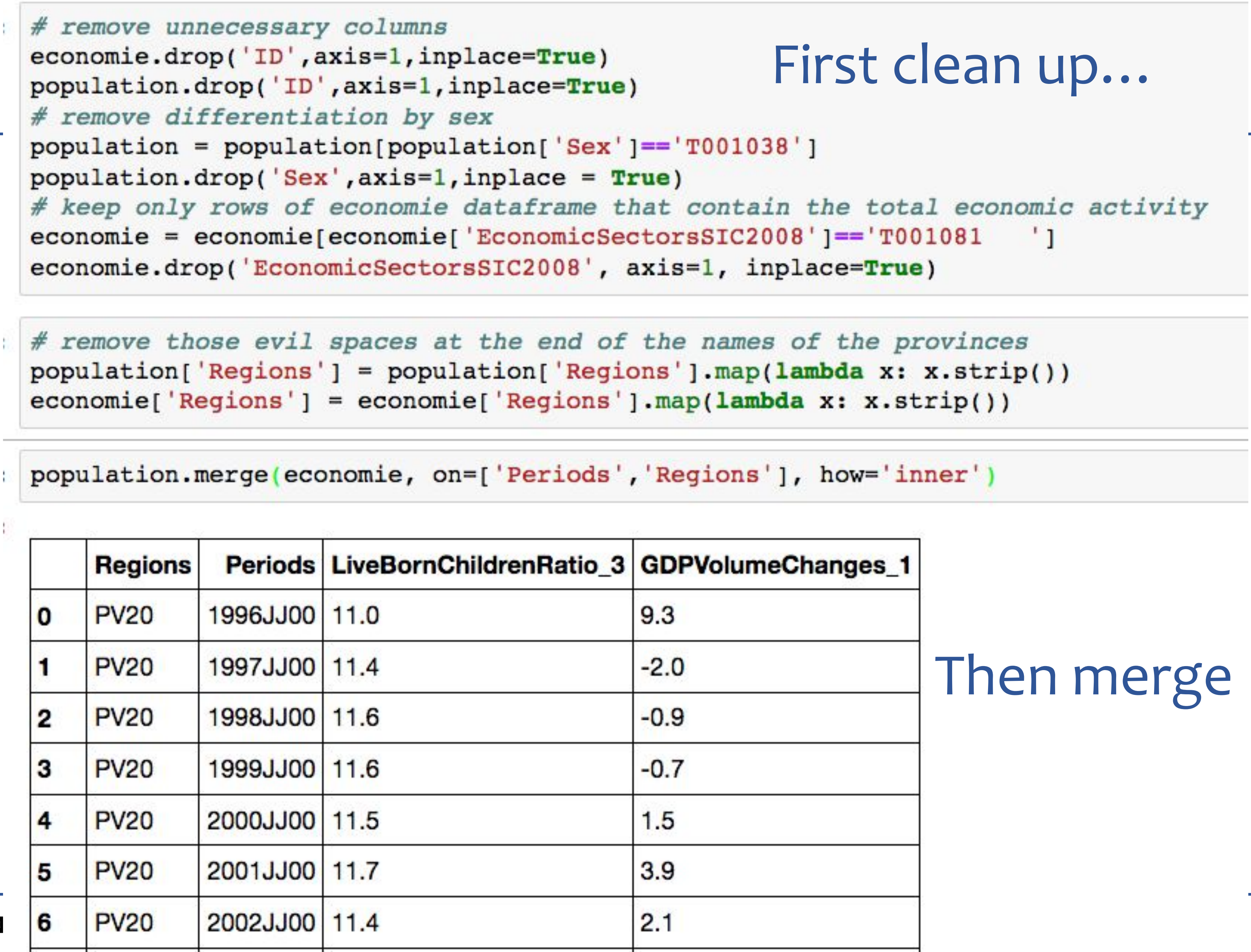
```
population = pd.read_csv('37259eng_UntypedDataSet_15112018_204553.csv', delimiter=';')
population.head()
```



	ID	Sex	Regions	Periods	LiveBornChildrenRatio_3
0	290	T001038	PV20	1960JJ00	18.6
1	291	T001038	PV20	1961JJ00	18.9
2	292	T001038	PV20	1962JJ00	18.9
3	293	T001038	PV20	1963JJ00	19.5
4	294	T001038	PV20	1964JJ00	19.6

What do you think: How could/should a joined table look like?

First clean up…



```
# remove unnecessary columns
economie.drop('ID',axis=1,inplace=True)
population.drop('ID',axis=1,inplace=True)
# remove differentiation by sex
population = population[population['Sex']=='T001038']
population.drop('Sex',axis=1,inplace = True)
# keep only rows of economie dataframe that contain the total economic activity
economie = economie[economie['EconomicSectorsSIC2008']=='T001081    ']
economie.drop('EconomicSectorsSIC2008', axis=1, inplace=True)
```

```
# remove those evil spaces at the end of the names of the provinces
population['Regions'] = population['Regions'].map(lambda x: x.strip())
economie['Regions'] = economie['Regions'].map(lambda x: x.strip())
```

```
population.merge(economie, on=['Periods','Regions'], how='inner')
```

	Regions	Periods	LiveBornChildrenRatio_3	GDPVolumeChanges_1
0	PV20	1996JJ00	11.0	9.3
1	PV20	1997JJ00	11.4	-2.0
2	PV20	1998JJ00	11.6	-0.9
3	PV20	1999JJ00	11.6	-0.7
4	PV20	2000JJ00	11.5	1.5
5	PV20	2001JJ00	11.7	3.9
6	PV20	2002JJ00	11.4	2.1

Then merge

On what do you want to merge/join?

- Standard behavior of.join(): on the row *index* (i.e., the row number, unless you changed it to sth else like a date)

- df3 = df1.join(df2)

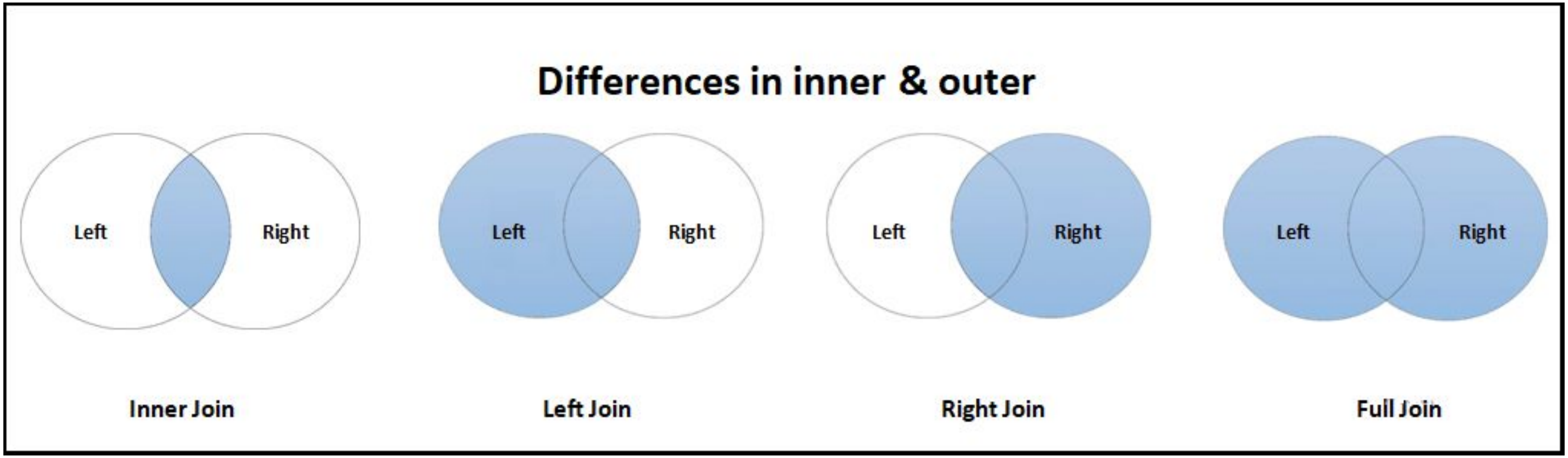
- But that’s only meaningful if the indices of df1 and df2 *mean* the same. Therefore you can also join on a column if both dfs have it:

- df3 = df1.merge(df2, on=‘Regions’)

- *.merge() is the more powerful tool, .join() is a bit easier when joining on indices.*

Inner, Outer, Left, and Right

- Main question: What do you want to do with keys (columns) that exist only in one of the dataframes?
- df3 = df1.join(df2, how='xxx')



INNER JOIN: Returns only matched rows

LEFT JOIN: Return all rows from the left table, and the **matched** rows from the right table

RIGHT JOIN: Return all rows from the right table, and the **matched** rows from the left table

FULL JOIN: Return all rows from both the tables

Depending on the join, you will end up with dataframes of *different* lengths.

Why is that?

Think back of your assignment!

- Is there a shared key/column on which you can merge? Which one?
    - Important: the column needs to be *exactly* the same, if there is some different formatting, use preprocessing.
- Does it need to be a left, right, inner, or outer join?

Aggregating a dataset

Why do we need this?

- Another way of describing/summarizing your dataset
- But more flexible: Instead of getting the overall mean of media consumption, get the mean *by gender* instead
- More in general: every time when you want to combine multiple rows into one (e.g., by summing, averaging, …), you *aggregate*

An example

- Suppose you have two dataframes, both containing information on something per region per year.
- You want to merge (join) the two, however, in one of them, the information is also split up by age groups. You don't want that.
- How do you bring these rows back to one row? With .agg()!

.agg()

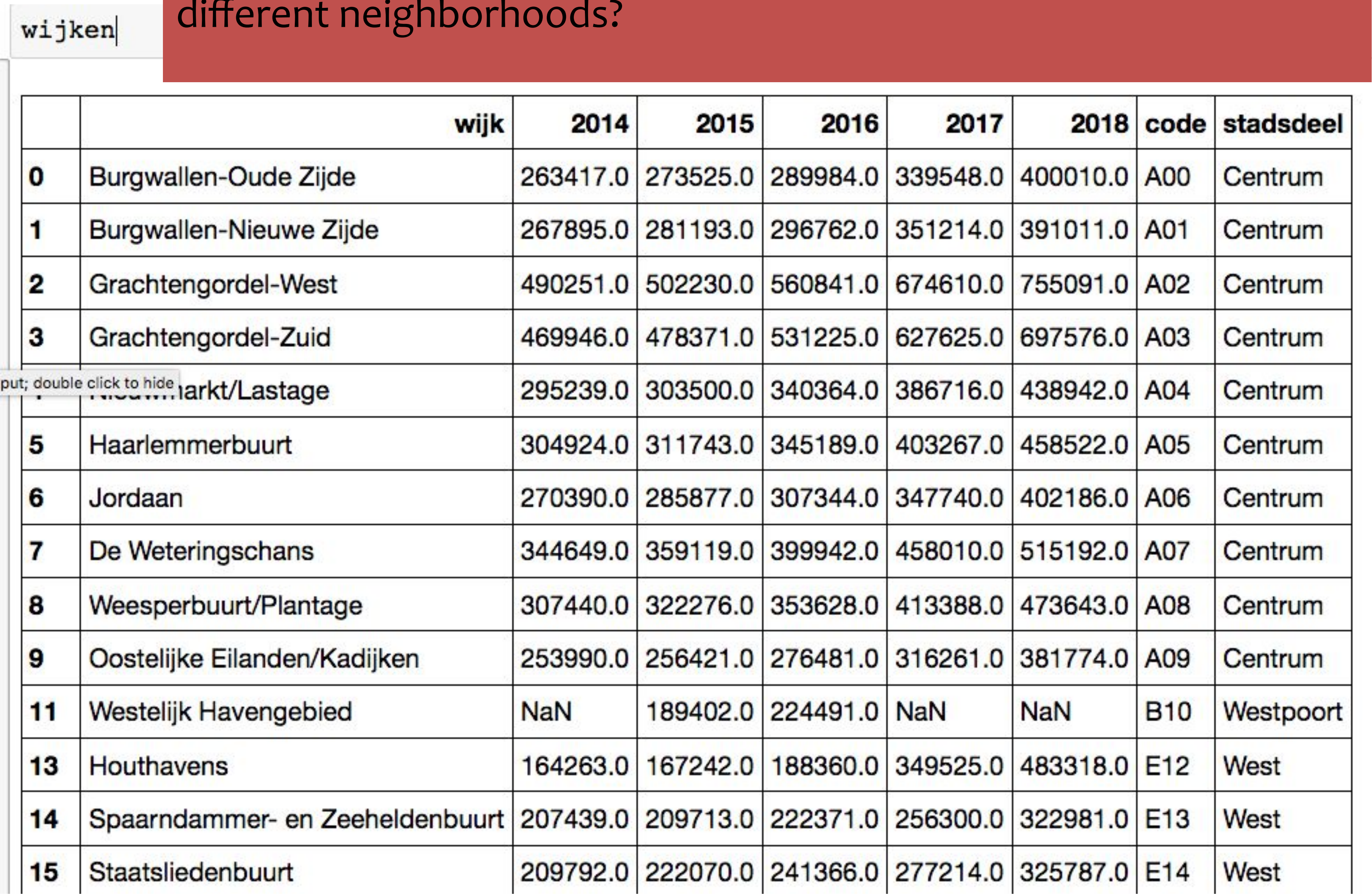
- Very useful after a .groupby()
- Takes a function as argument
- df2 = df.groupby(‘region’).agg(sum)
- Or multiple functions:
- df2 = df.groupby(‘region’).agg([sum, np.mean])
  - → yes, you could do .describe(), but .agg() is more flexible

Reshaping a dataset

Why do we need this?

- We already turned columns into rows by *transposing* a dataframe using .T
- But there may be more complex transformations we need to do.
- Many analyses and visualisations require that each variable equals a column (think of your SPSS classes….)
    - Unfortunately, many datasets you encounter in the wild aren't as nicely formatted.
    - Look at the example on the next slide. There is no column called "year" – hence, we cannot .groupby("year"), for instance

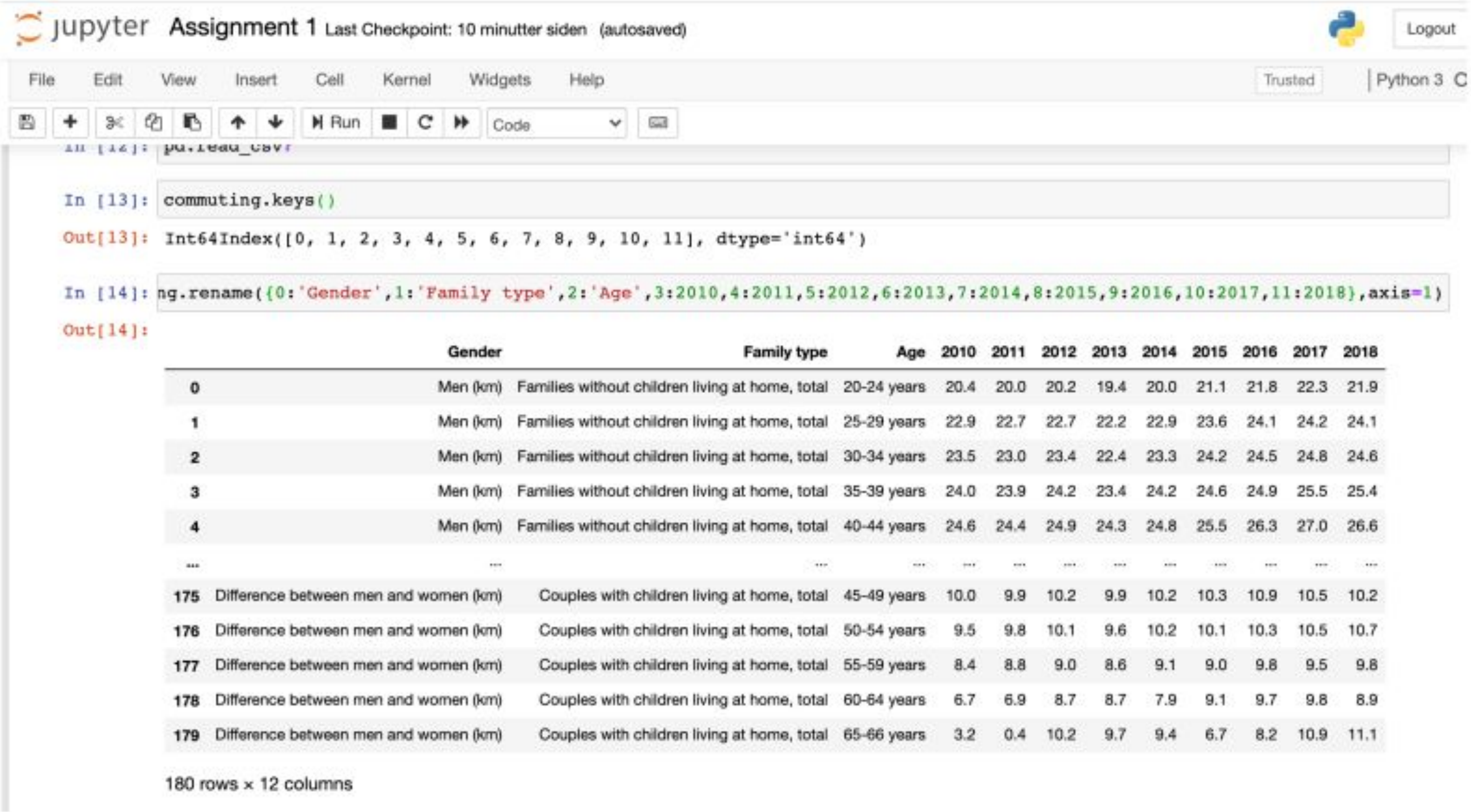
How do housing prices (WOZ-waarde) develop over time in different neighborhoods?



```
wijken
```

	wijk	2014	2015	2016	2017	2018	code	stadsdeel
0	Burgwallen-Oude Zijde	263417.0	273525.0	289984.0	339548.0	400010.0	A00	Centrum
1	Burgwallen-Nieuwe Zijde	267895.0	281193.0	296762.0	351214.0	391011.0	A01	Centrum
2	Grachtengordel-West	490251.0	502230.0	560841.0	674610.0	755091.0	A02	Centrum
3	Grachtengordel-Zuid	469946.0	478371.0	531225.0	627625.0	697576.0	A03	Centrum
4	Nieuwmarkt/Lastage	295239.0	303500.0	340364.0	386716.0	438942.0	A04	Centrum
5	Haarlemmerbuurt	304924.0	311743.0	345189.0	403267.0	458522.0	A05	Centrum
6	Jordaan	270390.0	285877.0	307344.0	347740.0	402186.0	A06	Centrum
7	De Weteringschans	344649.0	359119.0	399942.0	458010.0	515192.0	A07	Centrum
8	Weesperbuurt/Plantage	307440.0	322276.0	353628.0	413388.0	473643.0	A08	Centrum
9	Oostelijke Eilanden/Kadijken	253990.0	256421.0	276481.0	316261.0	381774.0	A09	Centrum
11	Westelijk Havengebied	NaN	189402.0	224491.0	NaN	NaN	B10	Westpoort
13	Houthavens	164263.0	167242.0	188360.0	349525.0	483318.0	E12	West
14	Spaarndammer- en Zeeheldenbuurt	207439.0	209713.0	222371.0	256300.0	322981.0	E13	West
15	Staatsliedenbuurt	209792.0	222070.0	241366.0	277214.0	325787.0	E14	West

At least one person’s assignment contains data that look **exactly** like the WOZ-example!



```
In [12]: pd.read_csv?
```

```
In [13]: commuting.keys()
```

```
Out[13]: Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], dtype='int64')
```

```
In [14]: ng.rename({0:'Gender',1:'Family type',2:'Age',3:2010,4:2011,5:2012,6:2013,7:2014,8:2015,9:2016,10:2017,11:2018},axis=1)
```

Out[14]:

	Gender	Family type	Age	2010	2011	2012	2013	2014	2015	2016	2017	2018
0	Men (km)	Families without children living at home, total	20-24 years	20.4	20.0	20.2	19.4	20.0	21.1	21.8	22.3	21.9
1	Men (km)	Families without children living at home, total	25-29 years	22.9	22.7	22.7	22.2	22.9	23.6	24.1	24.2	24.1
2	Men (km)	Families without children living at home, total	30-34 years	23.5	23.0	23.4	22.4	23.3	24.2	24.5	24.8	24.6
3	Men (km)	Families without children living at home, total	35-39 years	24.0	23.9	24.2	23.4	24.2	24.6	24.9	25.5	25.4
4	Men (km)	Families without children living at home, total	40-44 years	24.6	24.4	24.9	24.3	24.8	25.5	26.3	27.0	26.6
...	...	...	...	...	...	...	...	...	...	...	...	...
175	Difference between men and women (km)	Couples with children living at home, total	45-49 years	10.0	9.9	10.2	9.9	10.2	10.3	10.9	10.5	10.2
176	Difference between men and women (km)	Couples with children living at home, total	50-54 years	9.5	9.8	10.1	9.6	10.2	10.1	10.3	10.5	10.7
177	Difference between men and women (km)	Couples with children living at home, total	55-59 years	8.4	8.8	9.0	8.6	9.1	9.0	9.8	9.5	9.8
178	Difference between men and women (km)	Couples with children living at home, total	60-64 years	6.7	6.9	8.7	8.7	7.9	9.1	9.7	9.8	8.9
179	Difference between men and women (km)	Couples with children living at home, total	65-66 years	3.2	0.4	10.2	9.7	9.4	6.7	8.2	10.9	11.1

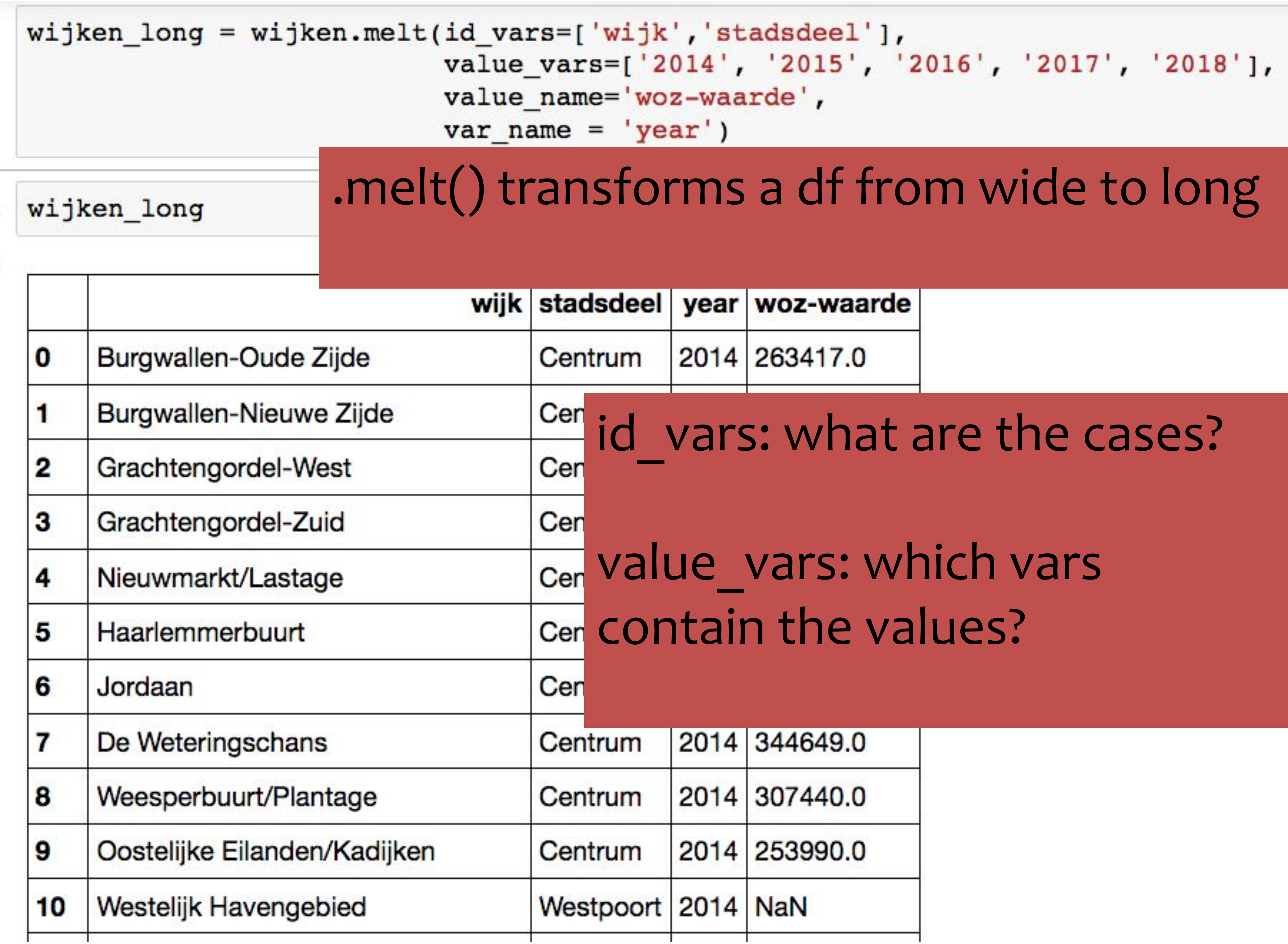
180 rows × 12 columns

Steps

- Get it into a tidy format (1 row = 1 observation) (“long” format)

- Optionally, but more neat (also for automatically get correct plot labels): *index rows by year*

- use .groupby() and .agg() to aggregate the data



```
wijken_long = wijken.melt(id_vars=['wijk','stadsdeel'],
                          value_vars=['2014', '2015', '2016', '2017', '2018'],
                          value_name='woz-waarde',
                          var_name = 'year')
```

```
wijken_long
```

.melt() transforms a df from wide to long

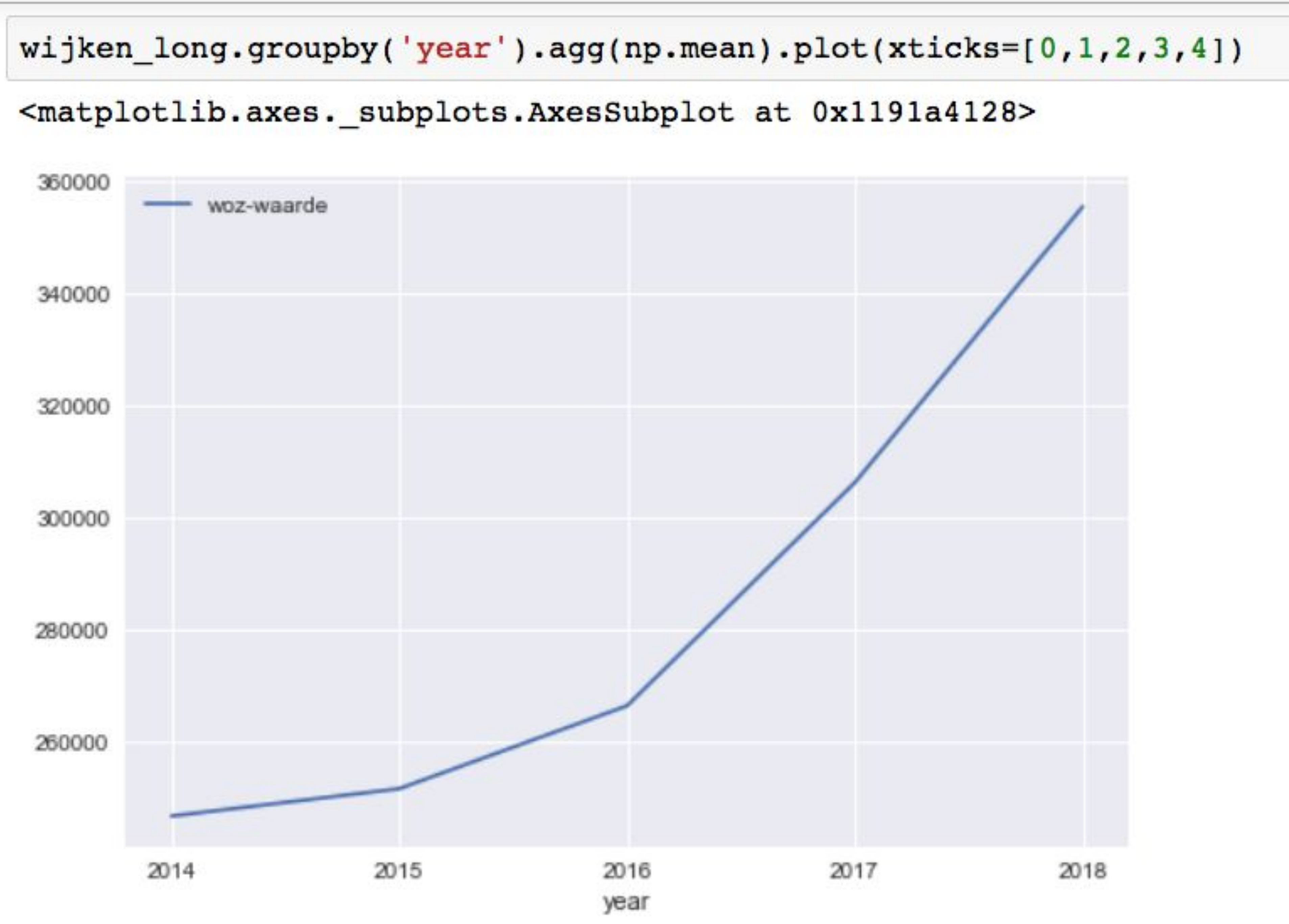
	wijk	stadsdeel	year	woz-waarde
0	Burgwallen-Oude Zijde	Centrum	2014	263417.0
1	Burgwallen-Nieuwe Zijde	Cen		
2	Grachtengordel-West	Cen		
3	Grachtengordel-Zuid	Cen		
4	Nieuwmarkt/Lastage	Cen		
5	Haarlemmerbuurt	Cen		
6	Jordaan	Cen		
7	De Weteringschans	Centrum	2014	344649.0
8	Weesperbuurt/Plantage	Centrum	2014	307440.0
9	Oostelijke Eilanden/Kadijken	Centrum	2014	253990.0
10	Westelijk Havengebied	Westpoort	2014	NaN

id_vars: what are the cases?

value_vars: which vars contain the values?

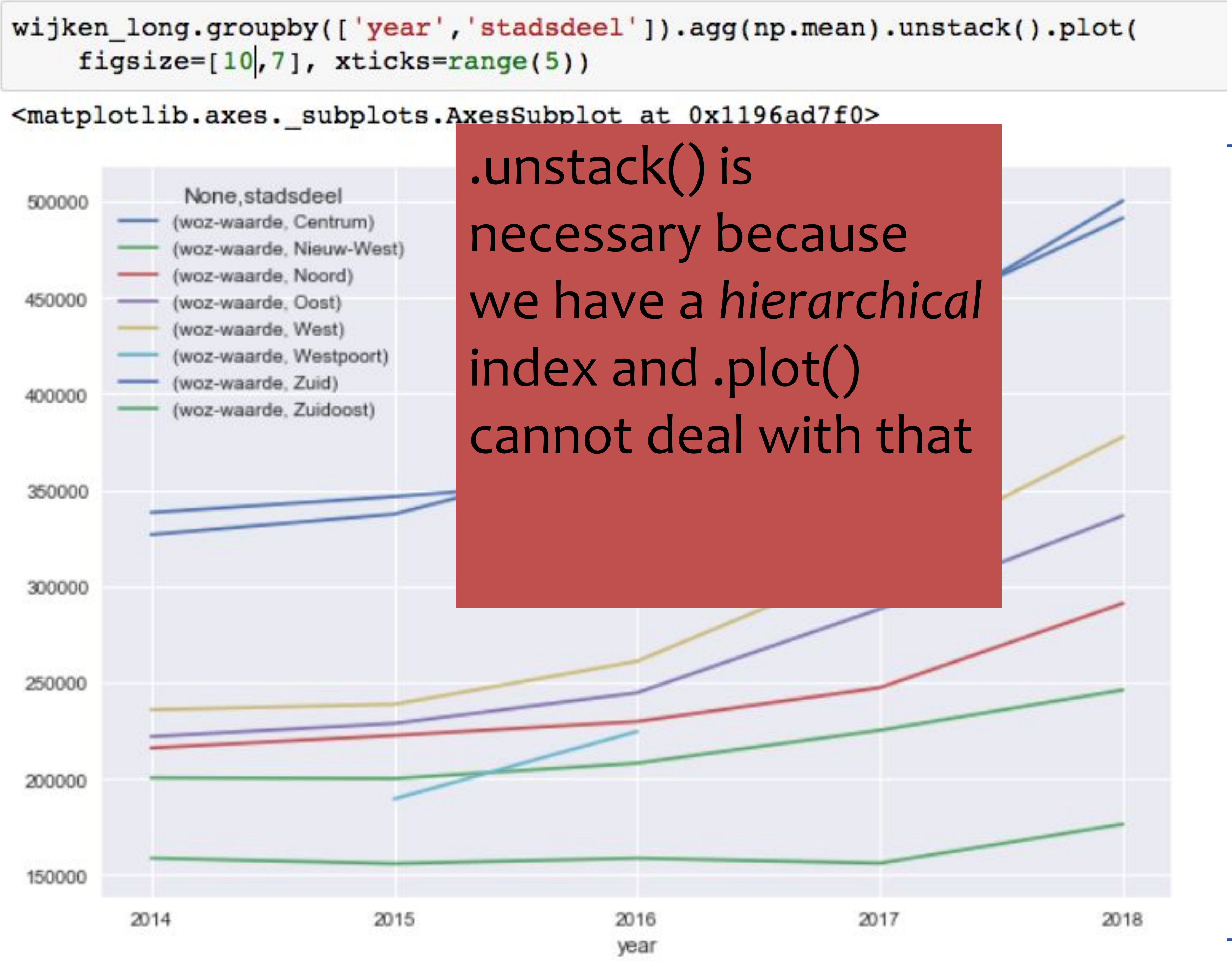
And now?

- Let’s think about a strategy for .groupby().agg(): What should we group by and how do we need to aggregate?
- Group by:
  - (1) Group only by year
  - (2) Group by year *and* ‘stadsdeel’
- Aggregation function
  - mean
  - Possibly also min, max, or even lambda x: max(x)-min(x)



```
wijken_long.groupby('year').agg(np.mean).plot(xticks=[0,1,2,3,4])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1191a4128>
```

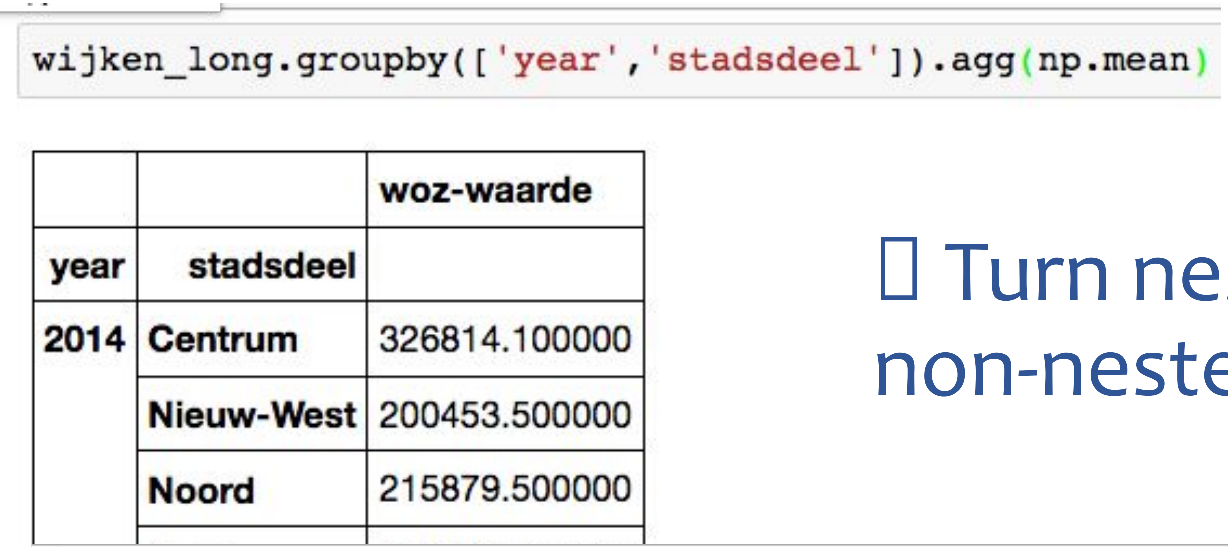


```
wijken_long.groupby(['year','stadsdeel']).agg(np.mean).unstack().plot(
    figsize=[10,7], xticks=range(5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1196ad7f0>
```

.unstack() is necessary because we have a *hierarchical* index and .plot() cannot deal with that

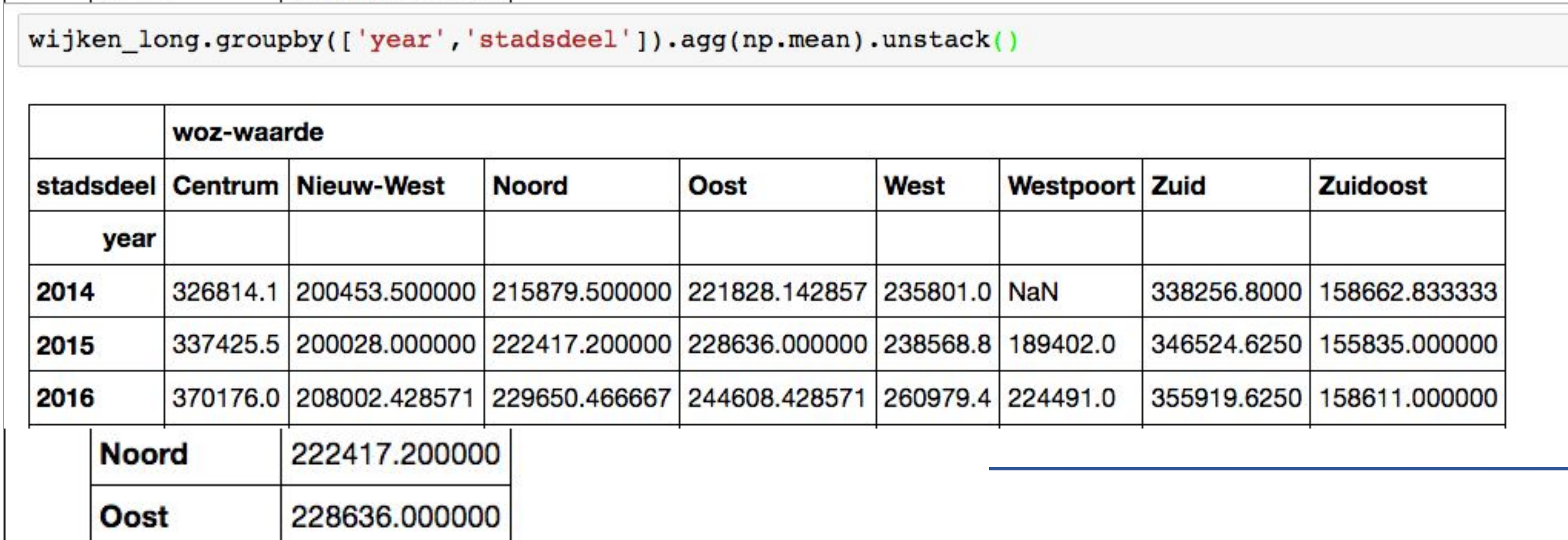
What’s unstacking?



```
wijken_long.groupby(['year','stadsdeel']).agg(np.mean)
```

		woz-waarde
year	stadsdeel	
2014	Centrum	326814.100000
	Nieuw-West	200453.500000
	Noord	215879.500000

- Turn nested indices into non-nested structure



```
wijken_long.groupby(['year','stadsdeel']).agg(np.mean).unstack()
```

	woz-waarde							
stadsdeel	Centrum	Nieuw-West	Noord	Oost	West	Westpoort	Zuid	Zuidoost
year								
2014	326814.1	200453.500000	215879.500000	221828.142857	235801.0	NaN	338256.8000	158662.833333
2015	337425.5	200028.000000	222417.200000	228636.000000	238568.8	189402.0	346524.6250	155835.000000
2016	370176.0	208002.428571	229650.466667	244608.428571	260979.4	224491.0	355919.6250	158611.000000

	Noord	222417.200000
	Oost	228636.000000

Let’s summarize: Tools you can use for data wrangling

- .loc() and .iloc()

- .join() and .merge()

- .melt() and .unstack()

- .groupby() and .agg()

Thursday

- On Thursday, will will walk through the notebooks
  - Python Data Wrangling I
  - Data Aggregation
- All datafiles are in the online book.
- There is Python Data Wrangling II that explains how we made the dataset behind the first notebook
- Have a first look at the notebooks already and try to understand what's going on!
- **Make sure you have the notebooks open and the datafiles downloaded before the session!**