Big Data & Automated Content Analysis
Week 4 – Wednesday: »Data Wrangling«

Damian Trilling

d.c.trilling@uva.nl
@damian0604
www.damiantrilling.net

24 February 2021

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

Today

- Statistics in Python
  - General considerations
  - Useful packages
- Pandas basics
  - Working with dataframes
  - Plotting and calculating with Pandas
- Pandas II: Data wrangling
  - Subsetting and slicing
  - Joining and Merging
  - Aggregation
- Next steps

Everything clear from last week?

Statistics in Python

Statistics in Python

General considerations

General considerations

So you retrieved some cool data from a JSON API, selected some interesting key-value pairs, and maybe also created a rectangular dataframe.

Of course, you can always export to .csv and use R or Stata or SPSS or whatever. . .

BUT:

General considerations

So you retrieved some cool data from a JSON API, selected some interesting key-value pairs, and maybe also created a rectangular dataframe.

Of course, you can always export to .csv and use R or Stata or SPSS or whatever...

BUT:

Reasons for not exporting and analyzing somewhere else

- the dataset might be too big
- it's cumbersome and wastes your time
- it may introduce errors and makes it harder to reproduce
- you want to learn Python ;-)

What statistics capabilities does Python have?

- Basically all standard stuff (bivariate and multivariate statistics) you know from SPSS
- Nowadays: also really advanced stuff (e.g., time series analysis via `statsmodels`; structural equation modelling via `semopy`; ...)
- Yet, for some really specific advanced statistical models, you may want to look somewhere else (else==R)

Statistics in Python

Useful packages

Useful packages

numpy (numerical python) Provides a lot of frequently used functions, like mean, standard deviation, correlation, ...

scipy (scientic python) More of that ;-)

statsmodels Statistical models (e.g., regression or time series)

matplotlib Plotting

seaborn Even nicer plotting

plot.ly Also nicer plotting (+ interactive)

Example 1: basic numpy

```
1  import numpy as np
2  x = [1,2,3,4,3,2]
3  y = [2,2,4,3,4,2]
4  z = [9.7, 10.2, 1.2, 3.3, 2.2, 55.6]
5  np.mean(x)
```

```
1  2.5
```

```
1  np.std(x)
```

```
1  0.9574271077563381
```

```
1  np.corrcoef([x,y,z])
```

```
1  array([[ 1.        ,  0.67883359, -0.37256219],
2         [ 0.67883359,  1.        , -0.56886529],
3         [-0.37256219, -0.56886529,  1.        ]])
```

Example 1: basic numpy

```
1 from scipy.stats import skew, kurtosis
2 for li in (x,y):
3     print(f"Skewness of {li}: {skew(li)}. Kurtosis: {kurtosis(li)}")
```

```
1 Skewness of [1, 2, 3, 4, 3, 2]: 0.0. Kurtosis: -0.942148760330578
2 Skewness of [2, 2, 4, 3, 4, 2]: 0.3329709512140237. Kurtosis:
      -1.676575505350732
```

```
1 from scipy.stats import kendalltau, spearmanr, pearsonr
2 print(kendalltau(x,y), spearmanr(x,y), pearsonr(x,y))
```

```
1 KendalltauResult(correlation=0.5853694070049636, pvalue
      =0.1373671546813069) SpearmanrResult(correlation
      =0.7627700713964739, pvalue=0.0777416409478997)
      (0.6788335930269976, 0.1381579775049088)
```

Characteristics

- Operates (also) on simple lists
- Returns output in standard datatypes (you can print it, store it, calculate with it, ...)
- it's fast! `np.mean(x)` is faster than `sum(x)/len(x)`
- it is more accurate (less rounding errors)

Example 2: basic plotting

```
1  import matplotlib.pyplot as plt
2  x = [1,2,3,4,3,2]
3  y = [2,2,4,3,4,2]
4  plt.hist(x)
5  plt.plot(x,y)
6  plt.scatter(x,y)
```

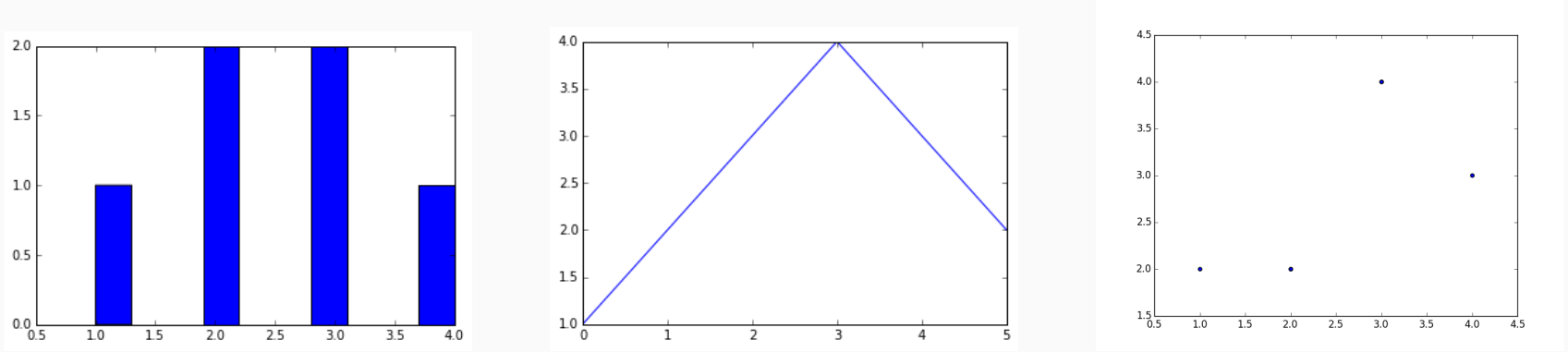


Figure 1: Examples of plots generated with `matplotlib`

Pandas basics

Pandas basics

Working with dataframes

When to use dataframes

Lists, dicts, generators

pro:

- flexible (especially dicts!)
- fast
- straightforward and easy to understand

con:

- if your data is a table, modeling this as, e.g., lists of lists feels unintuitive
- very low-level: you need to do much stuff 'by hand'

Pandas dataframes

pro:

- like an R dataframe or a STATA or SPSS dataset
- many built-in methods for statistics, plotting, grouping, subsetting, . . .)

con:

- 'overkill' if you just want correlate two lists or so
- unsuitable for REALLY large datasets

When to use dataframes

Lists, dicts, generators

pro:

- flexible (especially dicts!)
- fast
- straightforward and easy to understand

con:

- if your data is a table, modeling this as, e.g., lists of lists feels unintuitive
- very low-level: you need to do much stuff 'by hand'

Pandas dataframes

pro:

- like an R dataframe or a STATA or SPSS dataset
- many built-in methods for statistics, plotting, grouping, subsetting, . . .)

con:

- 'overkill' if you just want correlate two lists or so
- unsuitable for REALLY large datasets

Pandas basics

Plotting and calculating with Pandas

More examples here: https://github.com/damian0604/bdaca/blob/master/ipynb/basic_statistics.ipynb

OLS regression in pandas

```
1 import pandas as pd
2 import statsmodels.formula.api as smf
3
4 df = pd.DataFrame({'income': [10,20,30,40,50], 'age': [20, 30, 10, 40,
      50], 'facebooklikes': [32, 234, 23, 23, 42523]})
5
6 myfittedregression = smf.ols(formula='income ~ age + facebooklikes',
      data=df).fit()
7 print(myfittedregression.summary())
```

```
OLS Regression Results
==============================================================================
Dep. Variable:                 income   R-squared:                       0.579
Model:                            OLS   Adj. R-squared:                  0.158
Method:                 Least Squares   F-statistic:                     1.375
Date:                Mon, 05 Mar 2018   Prob (F-statistic):              0.421
Time:                        18:07:29   Log-Likelihood:                -18.178
No. Observations:                   5   AIC:                             42.36
Df Residuals:                       2   BIC:                             41.19
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept      14.9525     17.764      0.842      0.489      -61.481    91.386
age             0.4012      0.650      0.617      0.600       -2.394     3.197
facebooklikes   0.0004      0.001      0.650      0.583       -0.002     0.003
==============================================================================
Omnibus:                        nan   Durbin-Watson:                   1.061
Prob(Omnibus):                  nan   Jarque-Bera (JB):                0.498
Skew:                        -0.123   Prob(JB):                        0.780
Kurtosis:                     1.474   Cond. No.                     5.21e+04
==============================================================================
```

Other cool df operations

df['age'].plot() to plot a column

df['age'].describe() to get descriptive statistics

df['age'].value_counts() to get a frequency table

and MANY more...

Recoding and transforming

To transform your data, you can use (amongst others) `.apply()`, `.applymap()`, and `.map()` or the .str.XXX() methods:

```
df['is_center'] = df['hood'].str.contains('[cC]enter')
```

or define your own function:

```
def is_center(x):
    return int(x.lower().find('center') > -1)

df['is_center'] = df['hood'].map(is_center)
```

or use a throwaway-function:

```
df['is_center'] = df['hood'].map(lambda x: int(x.lower().find('center')
    > -1))
```

or use `.replace()` for simple recoding based on lists or a dict (not shown, see https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html)

Pandas II: Data wrangling

Pandas II: Data wrangling

Subsetting and slicing

Subsetting and slicing

Subsetting and slicing

Recap:

- `[0:5]` to get elements 0, 1, 2, 3, 4 (works with lists, dataframes . . .)
- `mydict['keyicareabout']` to get value (content) associated with the key

And therefore, also:

- `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

Subsetting and slicing

Recap:

- `[0:5]` to get elements 0, 1, 2, 3, 4 (works with lists, dataframes ...)
- `mydict['keyicareabout']` to get value (content) associated with the key

And therefore, also:

- `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

Subsetting and slicing

Recap:

- `[0:5]` to get elements 0, 1, 2, 3, 4 (works with lists, dataframes ...)
- `mydict['keyicareabout']` to get value (content) associated with the key

And therefore, also:

- `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

Subsetting and slicing

Recap:

- `[0:5]` to get elements 0, 1, 2, 3, 4 (works with lists, dataframes ...)
- `mydict['keyicareabout']` to get value (content) associated with the key

And therefore, also:

- `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

Subsetting and slicing

Recap:

- `[0:5]` to get elements 0, 1, 2, 3, 4 (works with lists, dataframes ...)
- `mydict['keyicareabout']` to get value (content) associated with the key

And therefore, also:

- `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

Subsetting and slicing

Recap:

- `[0:5]` to get elements 0, 1, 2, 3, 4 (works with lists, dataframes ...)
- `mydict['keyicareabout']` to get value (content) associated with the key

And therefore, also:

- `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

To get a apecific 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'

More subsetting

To get a apecific 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'

More subsetting

To get a apecific 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'

Home | 03-analyzing | 03-analyzing_da | CBS Open data | CBS Open data | Pandas: plot the val | Merge, join, and con | 04-join_and_agg | CBS Open data | coop - das Kabu | Who Is Exposed

localhost:8888/notebooks/03-analyzing_data.ipynb 120%

Meest bezocht | Aan de slag | Press This | Atlassian Cloud | Nieuw tabblad

jupyter 03-analyzing_data Last Checkpoint: Last Wednesday at 3:20 PM (unsaved changes)

File Edit View Insert Cell Kernel Help Python 3

Code CellToolbar

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

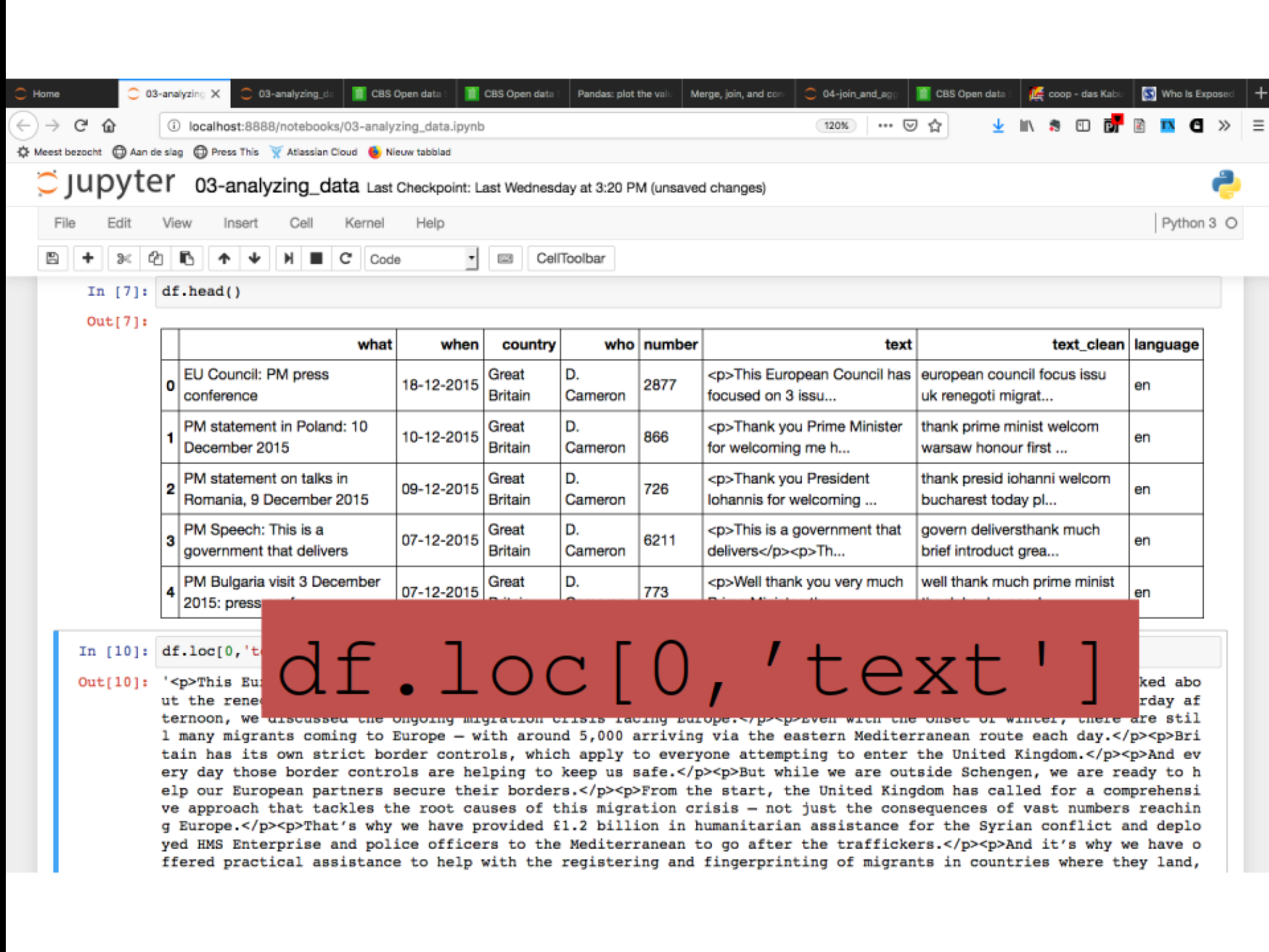
Out[7]:

	what	when	country	who	number	text	text_clean	language
0	EU Council: PM press conference	18-12-2015	Great Britain	D. Cameron	2877	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	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	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	This is a government that delivers Th...	govern deliversthank much brief introduct grea...	en
4	PM Bulgaria visit 3 December 2015: press	07-12-2015	Great	D.	773	Well thank you very much	well thank much prime minist	en

In [9]:
```
df.iloc[0,5
```

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,
```


Home
03-analyzing
03-analyzing_da
CBS Open data
CBS Open data
Pandas: plot the val
Merge, join, and con
04-join_and_agg
CBS Open data
coop - das Kabu
Who Is Exposed
localhost:8888/notebooks/03-analyzing_data.ipynb
120%
Meest bezocht
Aan de slag
Press This
Atlassian Cloud
Nieuw tabblad
jupyter
03-analyzing_data
Last Checkpoint: Last Wednesday at 3:20 PM (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Help
Python 3
Code
CellToolbar
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
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
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
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
This is a government that delivers Th...
govern deliversthank much brief introduct grea...
en
4
PM Bulgaria visit 3 December 2015: press
07-12-2015
Great
D.
773
Well thank you very much
well thank much prime minist
en
In [10]: df.loc[0,'t
df.loc[0,'text']
Out[10]: ' This Eu
ked abo
ut the rene
rday af
ternoon, we discussed the ongoing migration crisis facing Europe. 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. Bri
tain has its own strict border controls, which apply to everyone attempting to enter the United Kingdom. And ev
ery day those border controls are helping to keep us safe. But while we are outside Schengen, we are ready to h
elp our European partners secure their borders. 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. 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. And it's why we have o
ffered practical assistance to help with the registering and fingerprinting of migrants in countries where they land,

Advanced Example

Out of a dataset with 1,000 speeches, get the one that talks most about [Tt]error

1. We create a new column to count how many a word is mentioned:
   `df['terror'] = df['speech'].str.count('[Tt]error')`
2. We do
   `df.iloc[df['terror'].idxmax()]`
3. That works because df.iloc[] expects an integer to identify the row number, and `df['terror'].idxmax()` returns an integer (687 in our case)

Advanced Example

Out of a dataset with 1,000 speeches, get the one that talks most about [Tt]error

1. We create a new column to count how many a word is mentioned:
   `df['terror'] = df['speech'].str.count('[Tt]error')`
2. We do
   `df.iloc[df['terror'].idxmax()]`
3. That works because df.iloc[] expects an integer to identify the row number, and `df['terror'].idxmax()` returns an integer (687 in our case)

Advanced Example

Out of a dataset with 1,000 speeches, get the one that talks most about [Tt]error

1. We create a new column to count how many a word is mentioned:
   `df['terror'] = df['speech'].str.count('[Tt]error')`
2. We do
   `df.iloc[df['terror'].idxmax()]`
3. That works because df.iloc[] expects an integer to identify the row number, and `df['terror'].idxmax()` returns an integer (687 in our case)

```
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
```

A note on hard-coded "magic numbers"

Hard-coding "magic numbers" like `687` or `(0, 5)` in the examples above is really bad style and should be avoided. Always *calculate* them from your data.

If you *really* cannot do this, define them as a constant at the beginning of your script: `ROW_WITH_CORRUPT_DATA=438`.)

Pandas II: Data wrangling

Joining and Merging

Joining and Merging

Typical scenario

- You have two datasets that share one column
- For instance, data from www.cbs.nl: one with economic indicators, one with social indicators
- You want to make one dataframe

```
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)

```
1 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:

```
1 df3 = df1.merge(df2, on='Regions')
```

`.merge()` is the more powerful tool, `.join()` is a bit easier when joining ion indices.

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)

```
1 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:

```
1 df3 = df1.merge(df2, on='Regions')
```

`.merge()` is the more powerful tool, `.join()` is a bit easier when joining ion indices.

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)

```
1 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:

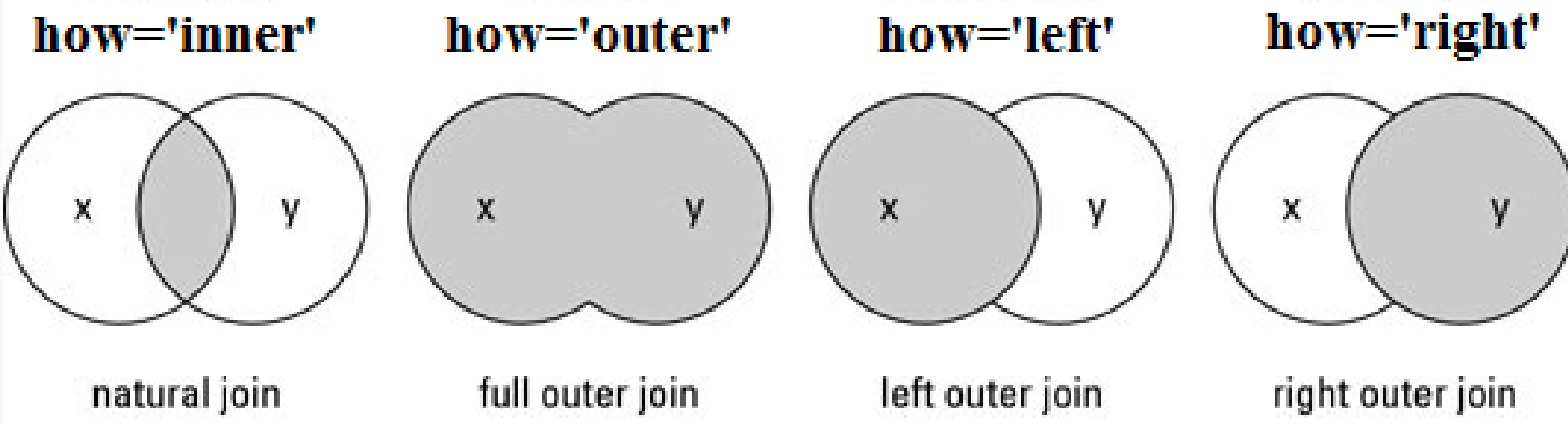
```
1 df3 = df1.merge(df2, on='Regions')
```

`.merge()` is the more powerful tool, `.join()` is a bit easier when joining ion indices.

Inner, Outer, Left, and Right

Main question: What do you want to do with keys that exist only in one of the dataframes?

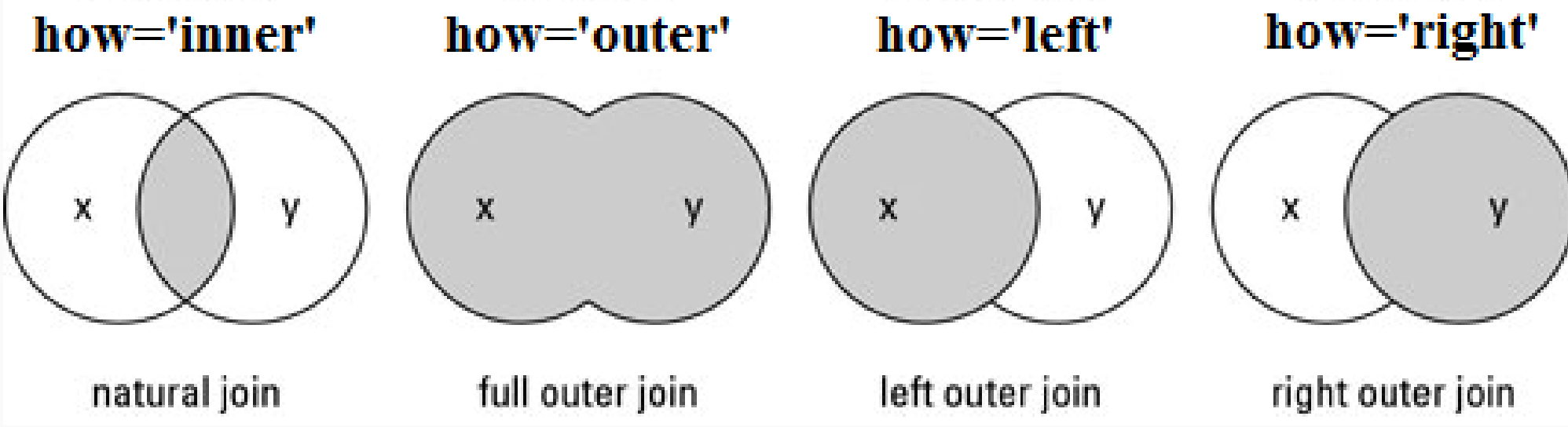
```
df3 = df1.join(df2, how='xxx')
```



Inner, Outer, Left, and Right

Main question: What do you want to do with keys that exist only in one of the dataframes?

```
df3 = df1.join(df2, how='xxx')
```



Pandas II: Data wrangling

Aggregation

Aggregation

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

.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

.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

.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])
```

- $\rightarrow$ yes, you could do `.describe()`, but `.agg()` is more flexible

An example

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

```
wijken
```

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

put; double click to hide

Steps

1. Get it into a tidy format (1 row = 1 observation) ("long" format)
2. Optionally, but more neat (also for automatically get correct plot labels):
   index rows by year
3. 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 `'studfeed'`
- Aggregation function
  1. `'mean'`
  2. Possibly also `'min'`, `'max'`, or even `lambda x: x.max(x) - x.min(x)`

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
  1. `mean`
  2. Possibly also `min`, `max`, or even `lambda x: x.max(x) - x.min(x)`

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
  1. mean
  2. Possibly also min, max, or even `lambda x: x.max(x) - x.min(x)`

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
  1. mean
  2. Possibly also min, max, ... (even lambda's or functions of your own)

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
  1. `mean`
  2. Possibly also `min`, `max`, or even `lambda x: max(x)-min(x)`

And now?

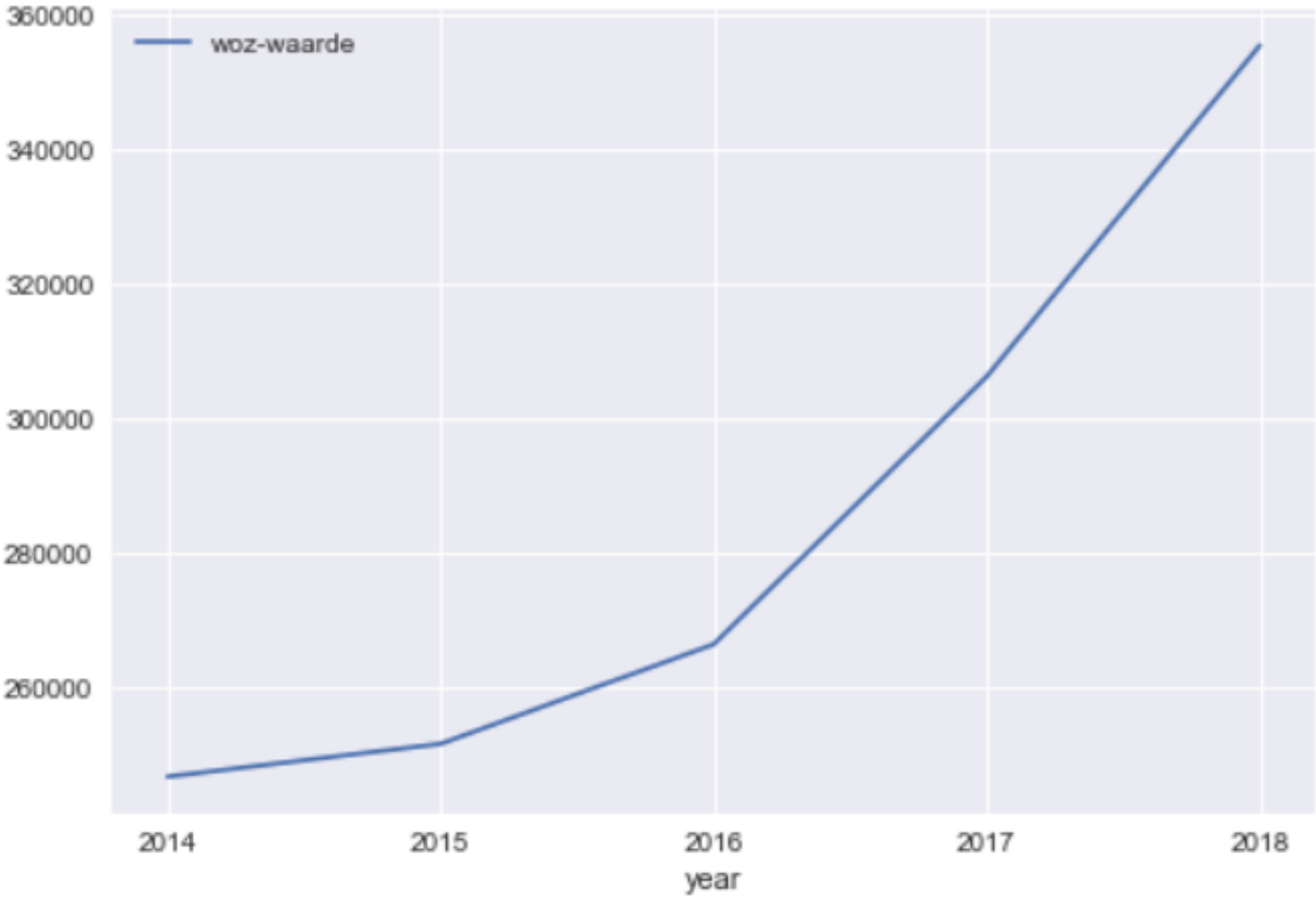
- Let's think about a strategy for `.groupby().agg()`: What should we group by and how do we need to aggregate?
- Group by:
  1. Group only by year
  2. Group by year and 'stadsdeel'
- Aggregation function
  1. `mean`
  2. Possibly also `min`, `max`, or even `lambda x: max(x)-min(x)`

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
  1. `mean`
  2. Possibly also `min`, `max`, or even `lambda x: max(x)-min(x)`

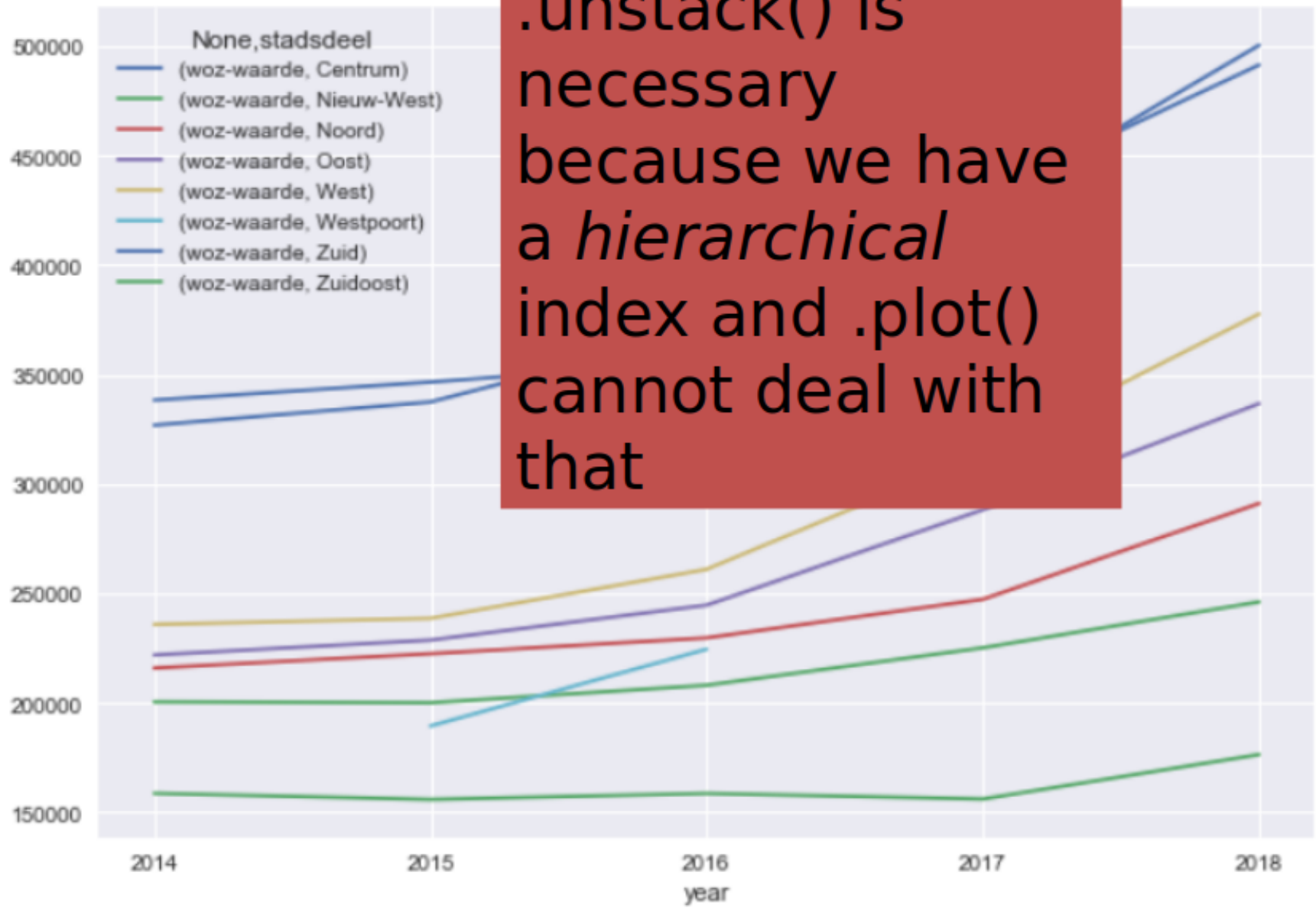
```
wijken_long.groupby('year').agg(np.mean).plot(xticks=[0,1,2,3,4])
```

```
<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
	Oost	221828.142857
	West	235801.0
	Westpoort	NaN
	Zuid	338225.0
	Zuidoost	158600.0
2015	Centrum	337425.5
	Nieuw-West	200028.000000
	Noord	222417.200000
	Oost	228636.000000

-> Turn hierarchical indices into non-hierarchical structure

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

	woz-waarde						
stadsdeel	Centrum	Nieuw-West	Noord	Oost	West	Westpoort	Zuid
year							
2014	326814.1	200453.500000	215879.500000	221828.142857	235801.0	NaN	3382
2015	337425.5	200028.000000	222417.200000	228636.000000	238568.8	189402.0	3465
2016	370176.0	208002.428571	229650.466667	244608.428571	260979.4	224491.0	3559

Exercise

Can you understand the code (join_and_aggregate.ipynb) and explain to a classmate?

https://github.com/damian0604/bdaca/tree/master/12ec/week04/exercises

(tip: click on "raw" to and then "File/Save as" to download the notebook and/or the datasets directly)

Next steps

Friday: Visualization in Python

- First part: I'll walk you through different libraries
- Second part: Take a dataset of your choice and try to apply the techniques discussed.

Take home exam: 5 to 9 March

Three parts:

1. Essay-like literature question
2. Programming task ("analyze this dataset", "write a program that does X")
3. Methods question ("You do not have to implement this right now, but how would you. . . ")