

Data Science

Lecture 2-1: Data Science Fundamentals (Preprocessing)



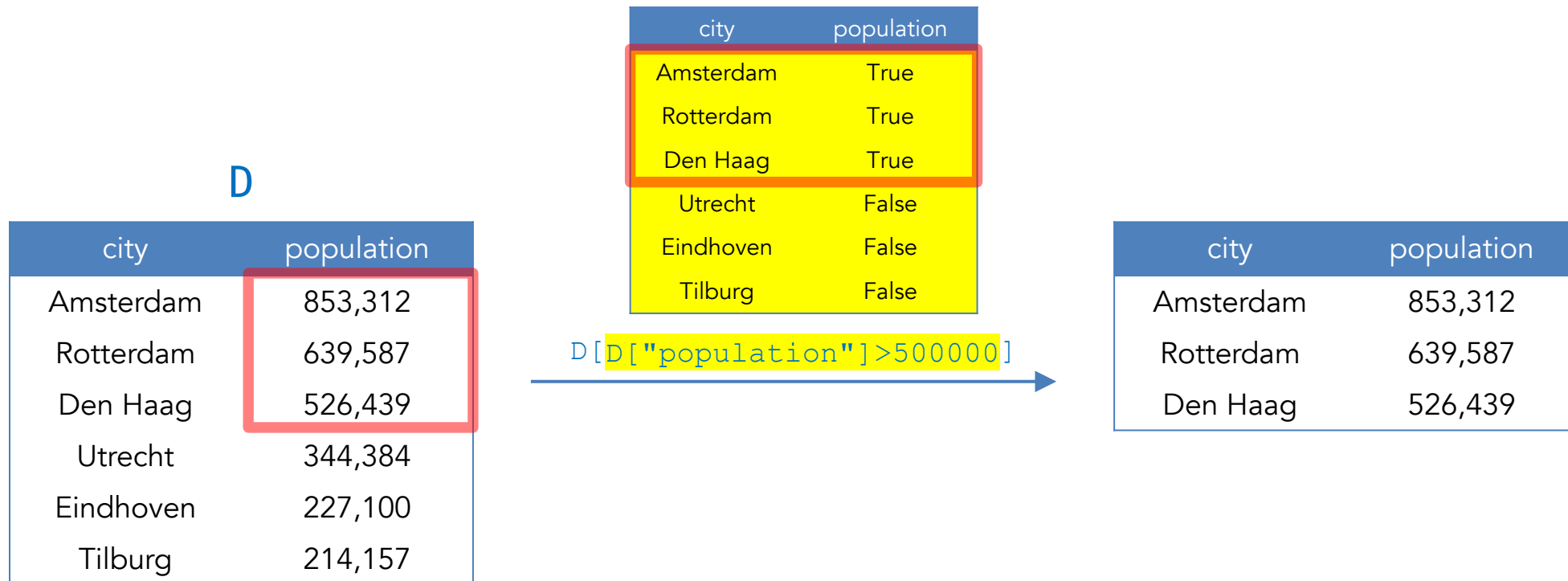
Lecturer: Yen-Chia Hsu

Date: Feb 2026

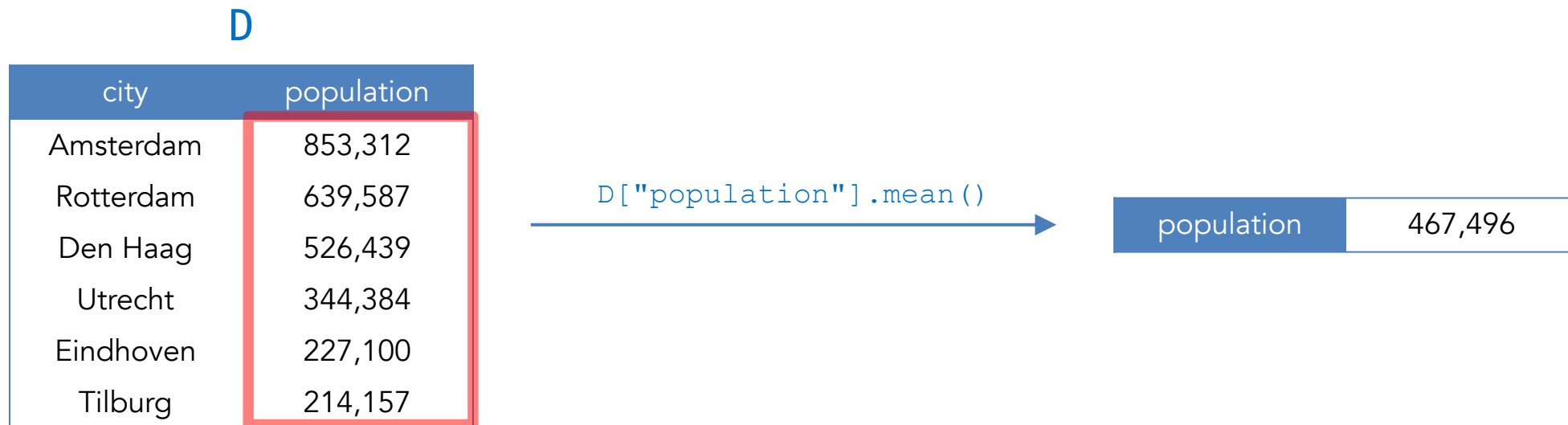
We will use [pandas](#) for this course, which is a very handy Python library for preprocessing structured data. We will cover the following techniques:

- | | |
|--|---|
| <ul style="list-style-type: none">• Filter unwanted data• Aggregate data (e.g., sum)• Group data based on a column• Sort rows based on a column• Concatenate data frames• Merge and join data frames• Quantize continuous values into bins | <ul style="list-style-type: none">• Scale column values• Resample time series data• Roll time series data in a window• Apply a transformation function• Use regular expressions• Drop rows or columns• Treat missing values |
|--|---|

Filtering can reduce a set of data based on specific criteria. For example, the left table can be reduced to the right table using a population threshold.



Aggregation reduces a set of data to a descriptive statistic. For example, the left table is reduced to a single number by computing the mean value.



Grouping divides a table into groups by column values, which can be chained with data aggregation to produce descriptive statistics for each group.

D

city	province	population
Amsterdam	Noord-Holland	853,312
Rotterdam	Zuid-Holland	639,587
Utrecht	Utrecht	344,384
Eindhoven	Noord-Brabant	227,100
Den Haag	Zuid-Holland	526,439
Tilburg	Noord-Brabant	214,157

province	population	city
Noord-Holland	853,312	Amsterdam
Zuid-Holland	639,587	Rotterdam
	526,439	Den Haag
Utrecht	344,384	Utrecht
Noord-Brabant	227,100	Eindhoven
	214,157	Tilburg

D.groupby("province").sum()

province	population
Noord-Holland	853,312
Zuid-Holland	1,166,026
Utrecht	344,384
Noord-Brabant	441,257

Sorting rearranges data based on values in a column, which can be useful for inspection. For example, the right table is sorted by population.

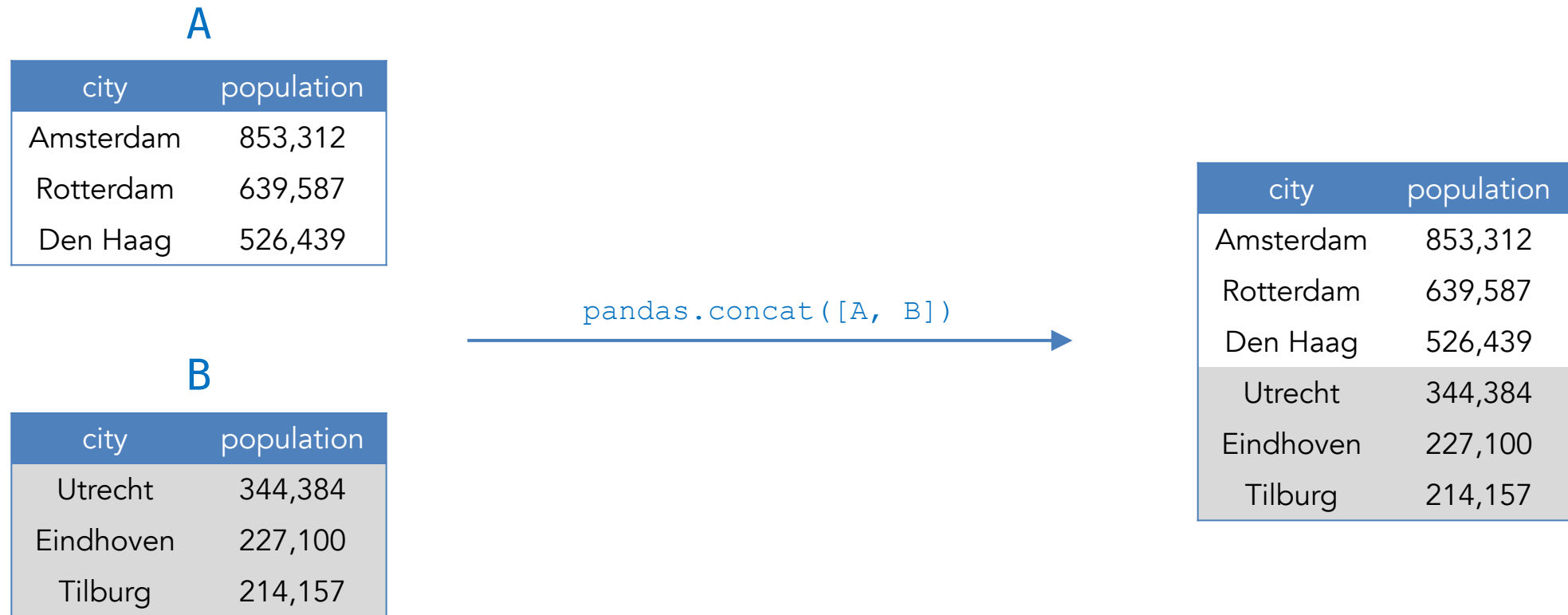
D

city	population
Eindhoven	227,100
Den Haag	526,439
Tilburg	214,157
Rotterdam	639,587
Amsterdam	853,312
Utrecht	344,384

`D.sort_values(by=["population"])`

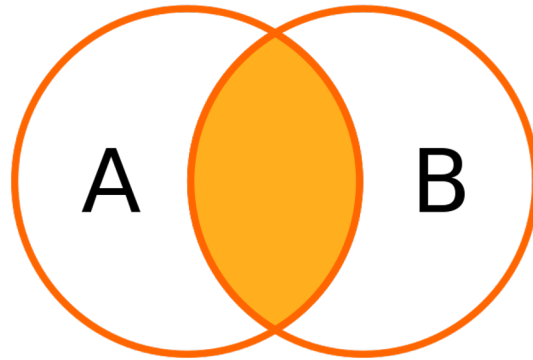
city	population
Tilburg	214,157
Eindhoven	227,100
Utrecht	344,384
Den Haag	526,439
Rotterdam	639,587
Amsterdam	853,312

Concatenation combines multiple datasets that have the same variables. For example, the two left tables can be concatenated into the right table.

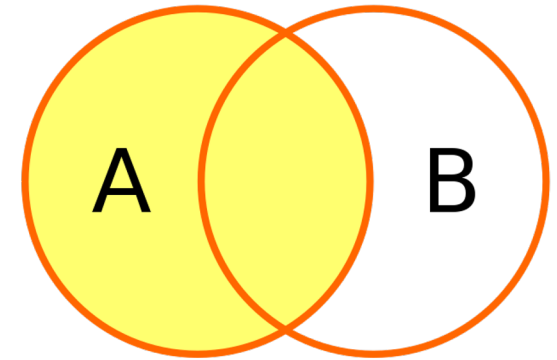


Merging and joining is a common method (in relational databases) to merge multiple data tables which have overlapping set of instances.

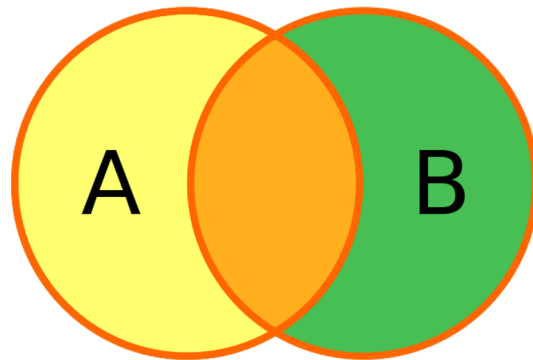
- Inner join



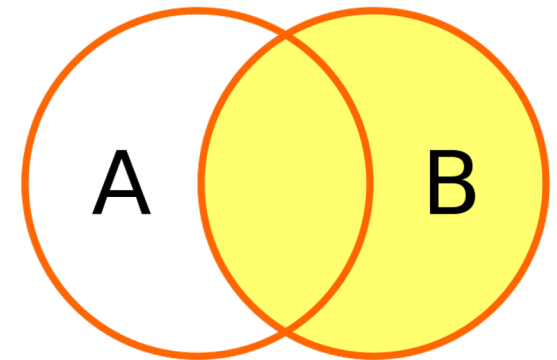
- Left (outer) join



- Outer join



- Right (outer) join



A

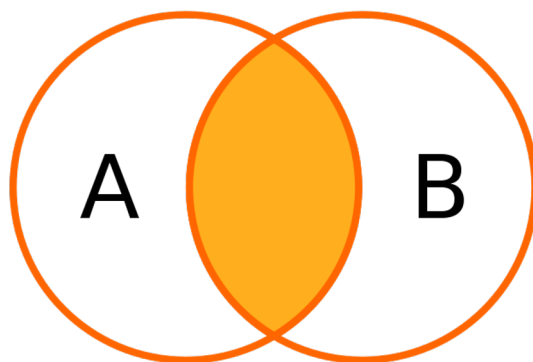
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

B

city	air_quality
Amsterdam	42.4
Rotterdam	40.9
Den Haag	41.1
Utrecht	41.4
Eindhoven	43.8
Zwolle	40.9

Use "city" as the key to merge A and B

`A.merge(B, how="inner", on="city")`



- Inner join

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8

A

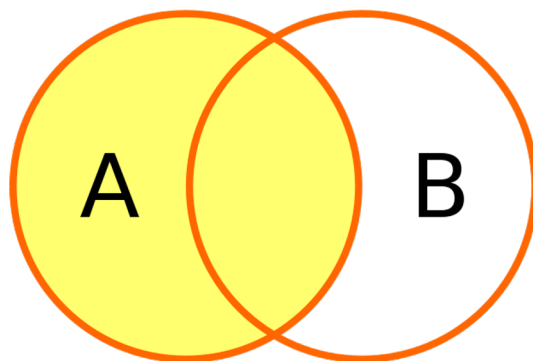
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

B

city	air_quality
Amsterdam	42.4
Rotterdam	40.9
Den Haag	41.1
Utrecht	41.4
Eindhoven	43.8
Zwolle	40.9

Use "city" as the key to merge A and B

`A.merge(B, how="left", on="city")`



- Left join

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	NaN

A

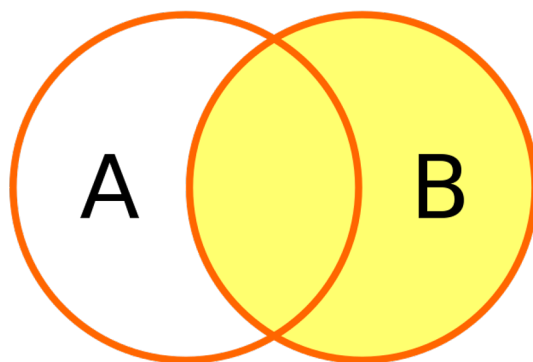
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

B

city	air_quality
Amsterdam	42.4
Rotterdam	40.9
Den Haag	41.1
Utrecht	41.4
Eindhoven	43.8
Zwolle	40.9

Use "city" as the key to merge A and B

`A.merge(B, how="right", on="city")`



- Right join

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Zwolle	NaN	40.9

A

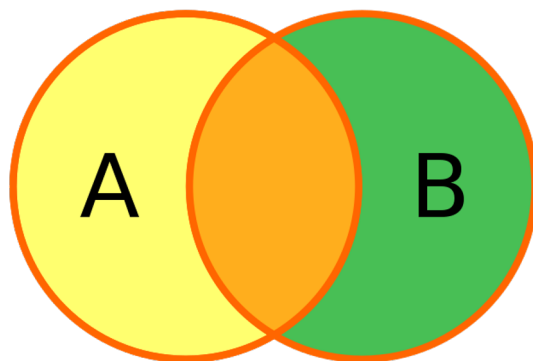
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439
Utrecht	344,384
Eindhoven	227,100
Tilburg	214,157

B

city	air_quality
Amsterdam	42.4
Rotterdam	40.9
Den Haag	41.1
Utrecht	41.4
Eindhoven	43.8
Zwolle	40.9

Use "city" as the key to merge A and B

`A.merge(B, how="outer", on="city")`



- Outer join

city	population	air_quality
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	NaN
Zwolle	NaN	40.9

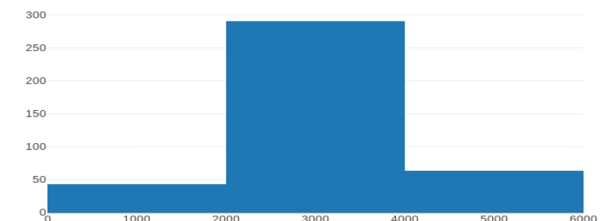
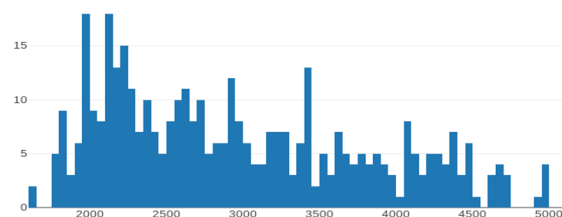
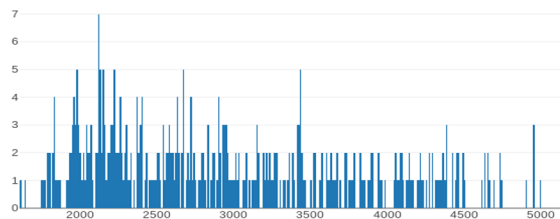
Quantization transforms a continuous set of values (e.g., integers) into a discrete set (e.g., categories). For example, age is quantized to age range.

D

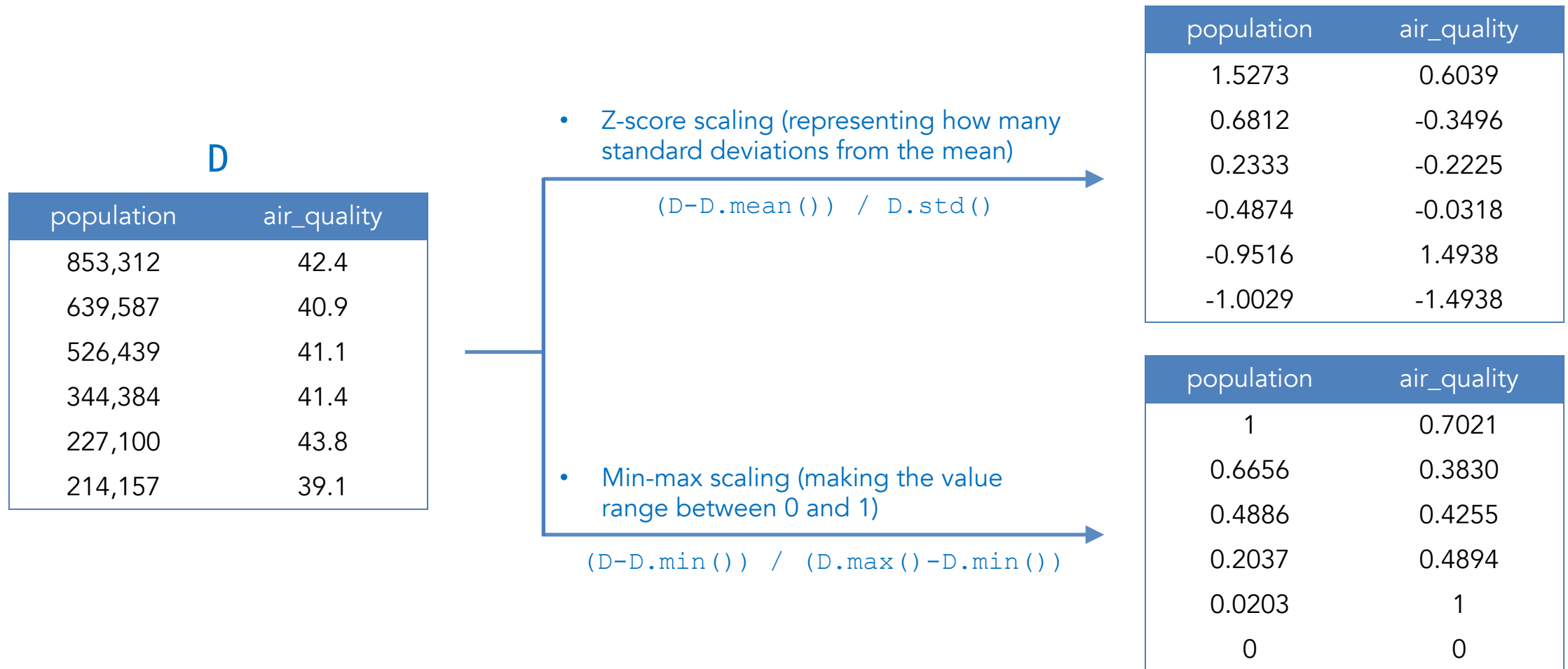
name	age
Jantje	8
Piet	16
Maria	22
Renske	34
Donald	65

```
bin = [0,20,50,200]
L = ["1-20","21-50","51+"]
pandas.cut(D["age"], bin, labels=L)
```

name	age
Jantje	1-20
Piet	1-20
Maria	21-50
Renske	21-50
Donald	51+



Scaling transforms variables to have another distribution, which puts variables at the same scale and makes the data work better on many models.



You can **resample** time series data (i.e., the data with time stamps) to a different frequency (e.g., hourly) using different aggregation methods (e.g., mean).

D

timestamp	v1
2016-10-31 07:30:00	52.60
2016-10-31 08:30:00	48.30
2016-10-31 08:53:20	44.20
2016-10-31 09:30:00	31.10

`D.resample("60Min", label="right").mean()`

timestamp	v1
2016-10-31 08:00:00	52.60
2016-10-31 09:00:00	46.25
2016-10-31 10:00:00	31.10

You can use the **rolling** window operation to transform time series data using different aggregation methods (e.g., sum).

D

timestamp	v1
2016-10-31 08:00:00	52.60
2016-10-31 09:00:00	46.25
2016-10-31 10:00:00	31.10
2016-10-31 11:00:00	12.21
2016-10-31 12:00:00	28.64

`D["v2"]=D["v1"].rolling(window=3).sum()`

timestamp	v2
2016-10-31 08:00:00	NaN
2016-10-31 09:00:00	NaN
2016-10-31 10:00:00	129.95
2016-10-31 11:00:00	89.56
2016-10-31 12:00:00	71.95

You can apply a **transformation** to rows or columns in the data frame.

D

wind_mph
3.6
NaN
5.1

```
def f(x):  
    if pd.isna(x): return None  
    else: return x<5  
D["is_calm"] = D["wind_mph"].apply(f)
```

wind_mph	is_calm
3.6	True
NaN	None
5.1	False

D

wind_deg
343
351
359
5
41
25
:



Very slow if you have a lot of rows!

```
def f(x):  
    return numpy.sin(numpy.deg2rad(x))  
D["wind_sine"] = D["wind_deg"].apply(f)
```



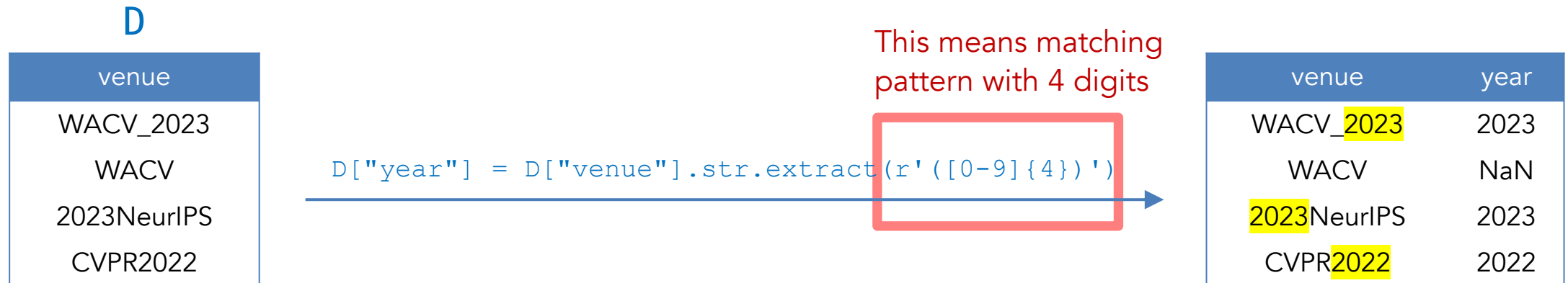
```
D["wind_sine"] = np.sin(np.deg2rad(D["wind_deg"]))
```



Better to transform the entire column directly!

wind_deg	wind_sine
343	-0.292372
351	-0.156434
359	-0.017452
5	0.087156
41	0.656059
25	0.422618
:	:

To extract data from text or match text patterns, you can use [regular expression](#), which is a language to specify search patterns.



We can **drop** data that we do not need, such as duplicate data records or those that are irrelevant to our research question.

city	population	year
Amsterdam	853,312	2018
Rotterdam	639,587	2018
Den Haag	526,439	2018

`pandas.drop(columns=["year"])`

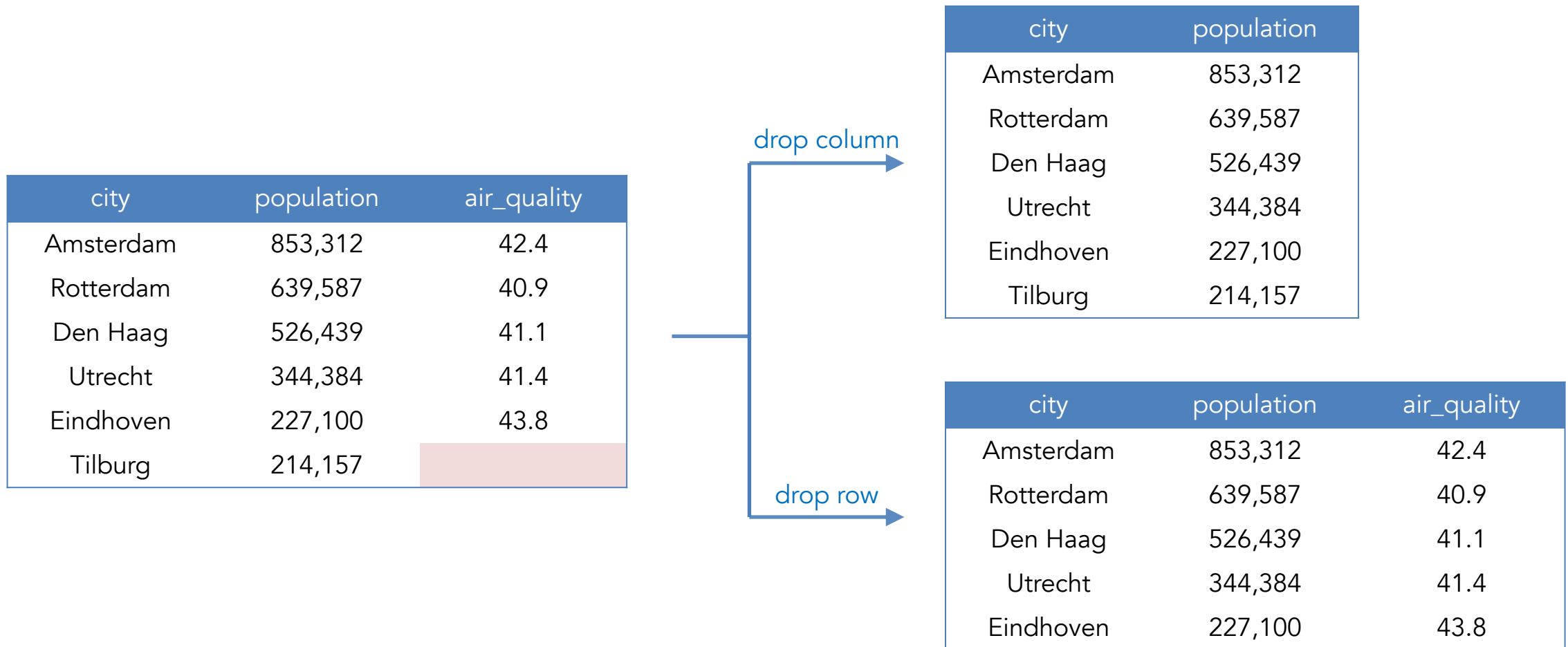
city	population
Amsterdam	853,312
Rotterdam	639,587
Den Haag	526,439

	city	population	year
0	Amsterdam	853,312	2018
1	Rotterdam	639,587	2018
2	Den Haag	526,439	2018
3	Utrecht	344,384	2018
4	Eindhoven	227,100	2018
5	Amsterdam	862,965	2019
6	Utrecht	344,384	2018

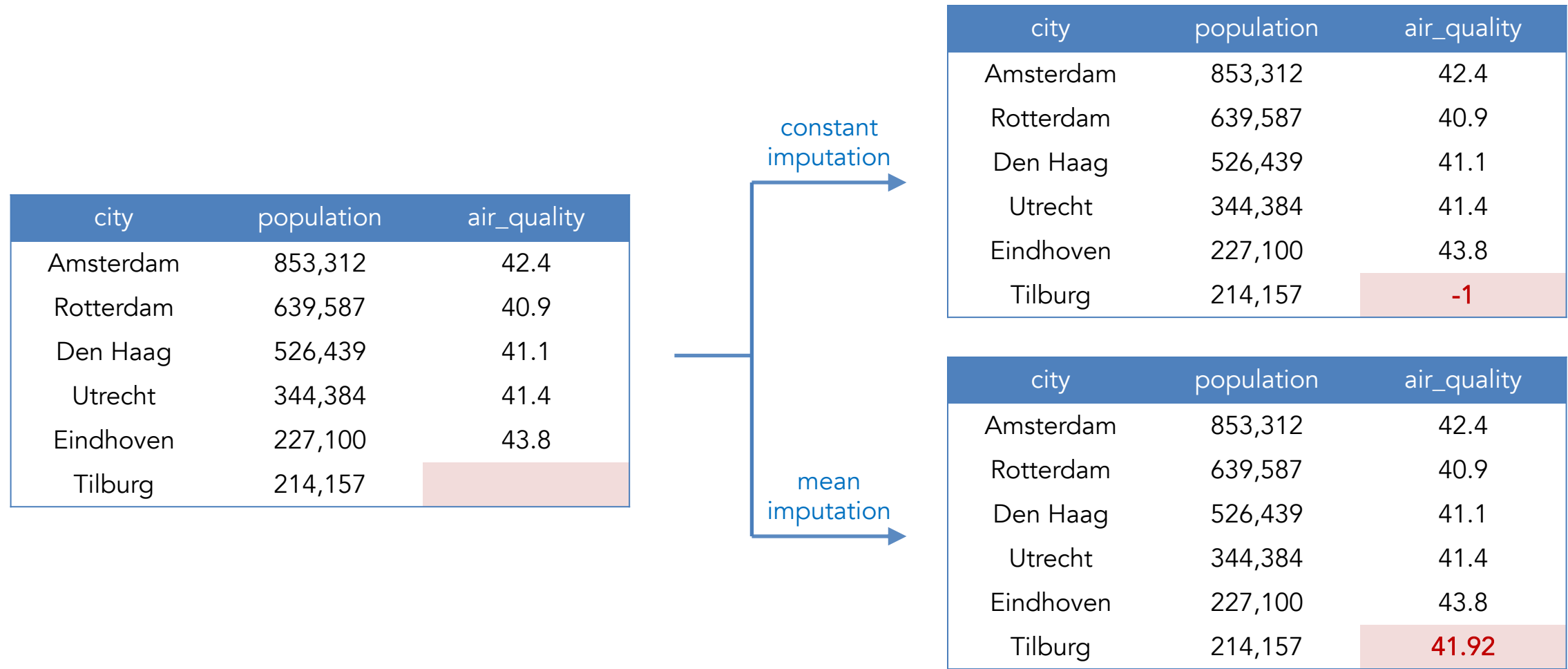
`pandas.drop([5, 6])`

	city	population	year
0	Amsterdam	853,312	2018
1	Rotterdam	639,587	2018
2	Den Haag	526,439	2018
3	Utrecht	344,384	2018
4	Eindhoven	227,100	2018

We can either **drop the rows** (i.e., the records/observations) or **the columns** (i.e., the variables/attributes) that contain the missing values.



We can **replace the missing values** (i.e., imputation) with a constant, mean, median, or the most frequent value along the same column.



We can **model missing values**, where y is the variable/column that has the missing values, X means other variables, and F is a regression function.

city	population (X)	air_quality (y)
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	

$$y = F(X)$$

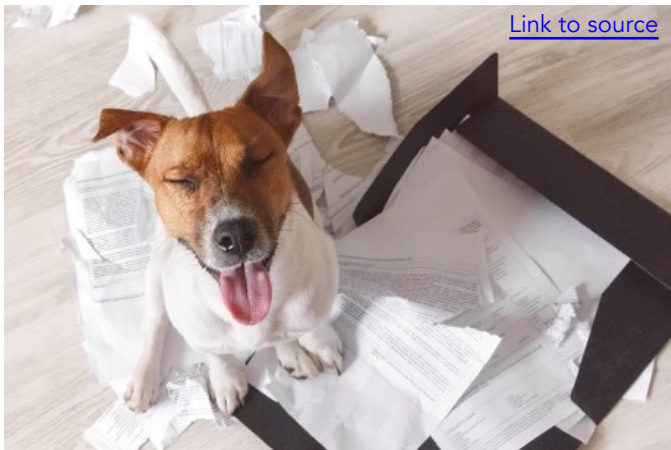
city	population (X)	air_quality (y)
Amsterdam	853,312	42.4
Rotterdam	639,587	40.9
Den Haag	526,439	41.1
Utrecht	344,384	41.4
Eindhoven	227,100	43.8
Tilburg	214,157	42.46

Different missing data may require different data cleaning methods. Missing Not At Random is a big problem and cannot be solved simply with imputation.

MCAR

Missing Completely At Random:

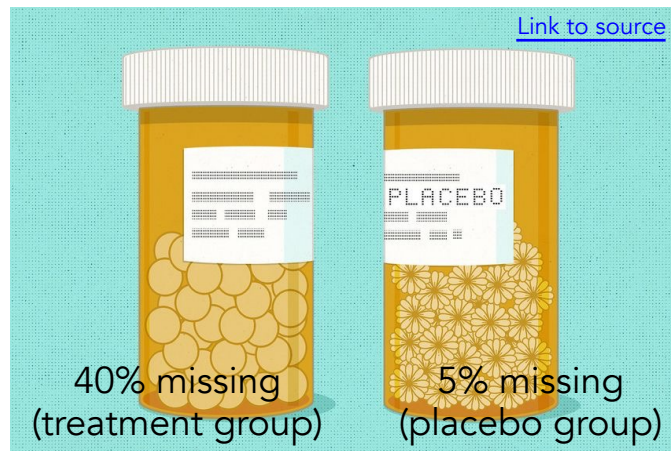
- Missing data is a completely random subset (no relations) of the entire dataset.



MAR

Missing at Random:

- Missing data is only related to variables other than the one having missing data.



MNAR

Missing Not At Random:

- Missing data is related to the variable that has the missing data. (e.g., sensitive questions)

A screenshot of a survey question. The text reads: "Do you have any history of mental illness in your family? If yes, who in your family?". Below the text are two radio button options: "No" and "Other: _____". The entire form is enclosed in a light purple border.



Questions?