Creative Data Mining

Using ML algorithms in python

Artem Chirkin
Dr. Daniel Zünd
Danielle Griego

Lecture 7

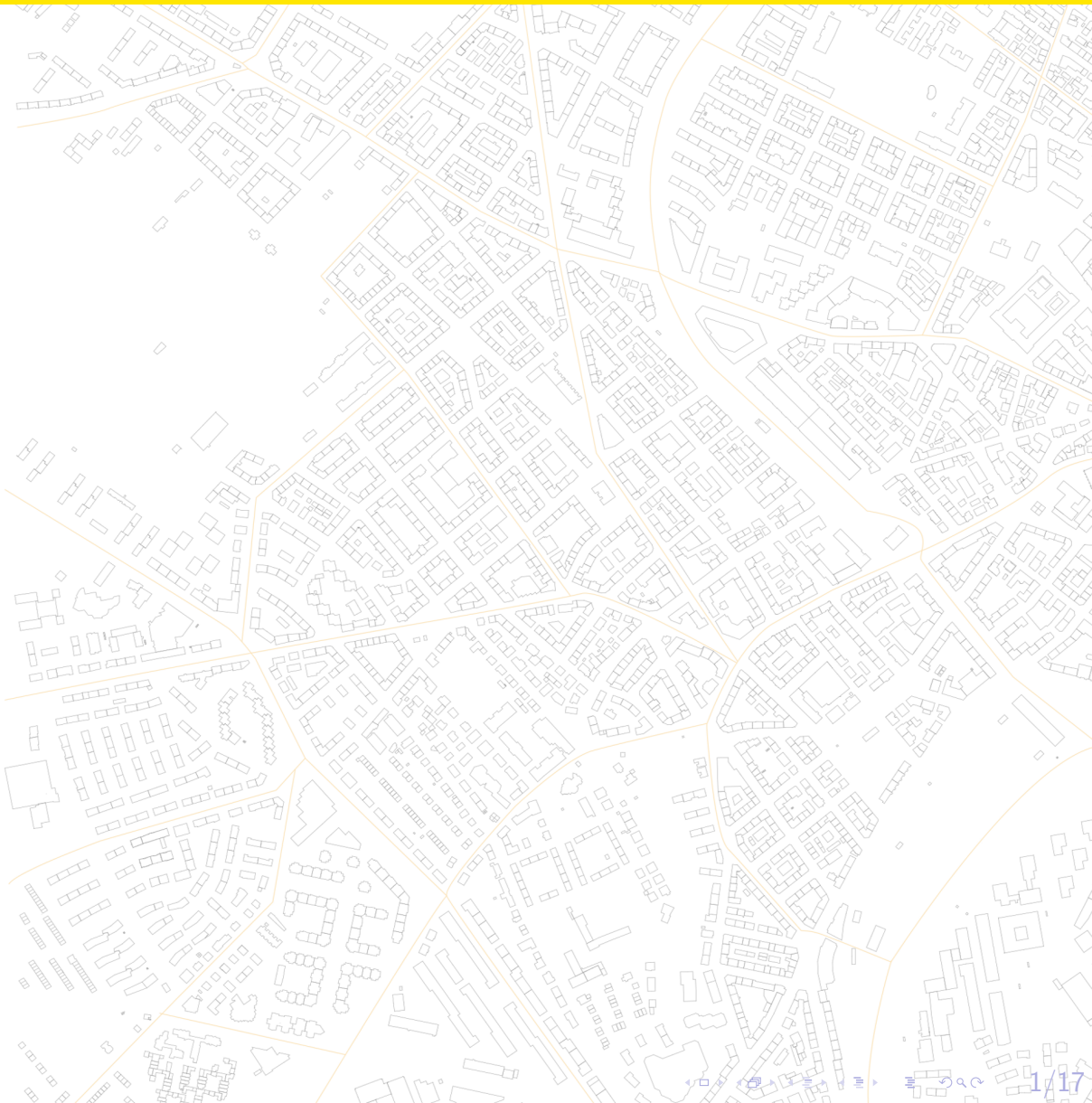
10.04.2017

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

DARCH iA Chair of Information Architecture

What we will cover today

Outline



Getting started

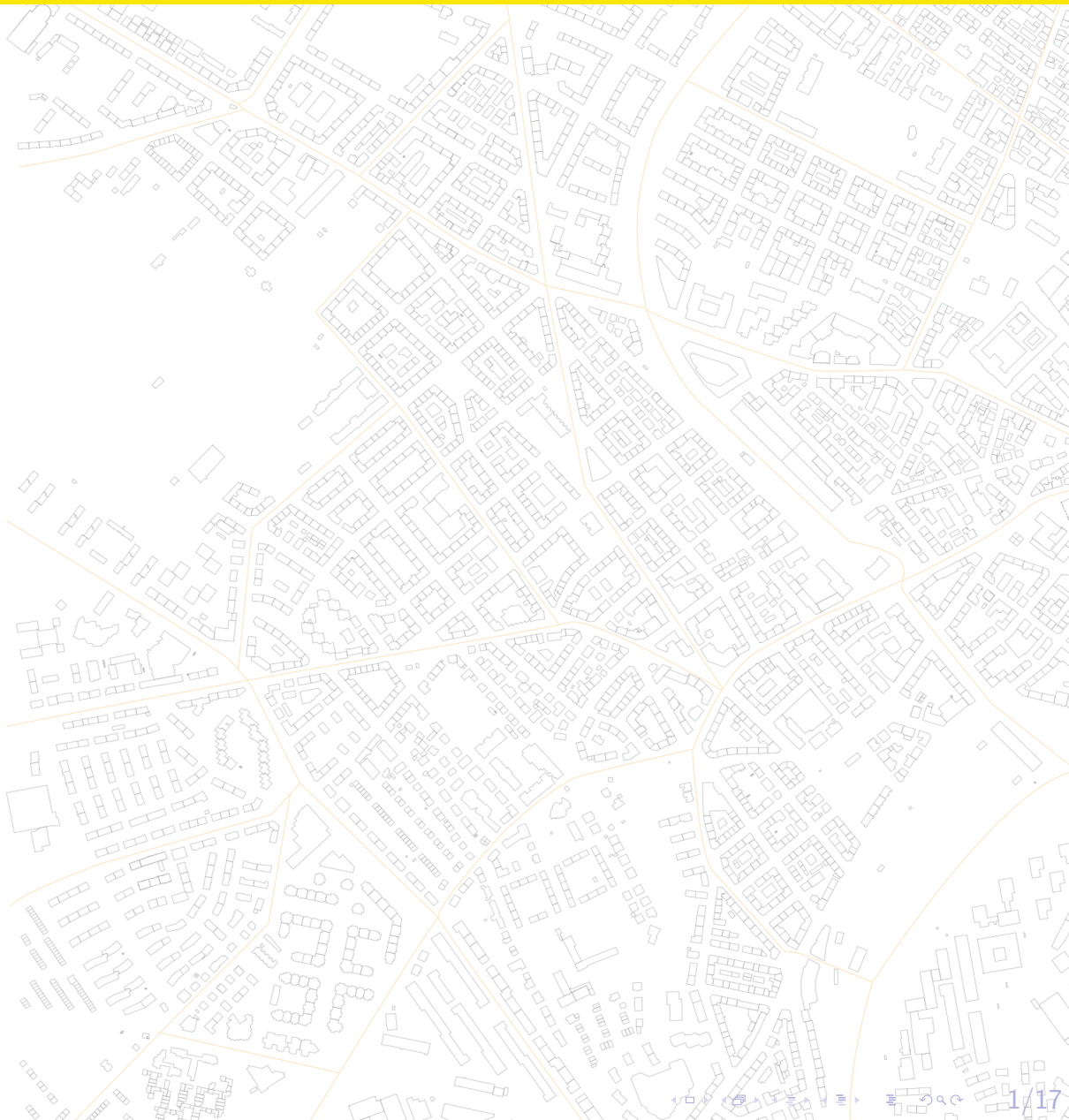
Explore dataset content

Inspect visually

Run clustering

Assess clustering quality

Outline



Getting started

Setting up python environment

At first, we need to set up our environment.

We'll use several package, most of which you have already installed

```
pip install numpy pandas matplotlib sklearn
```

```
# We need pandas to import and use .csv files
import pandas as pd

# numpy is used for various numeric operations
import numpy as np

# matplotlib lets us draw nice plots
import matplotlib.pyplot as plt

# We will use several modules of sklearn package
# that is a "swiss knife" ML toolset for python
from sklearn import cluster, metrics, decomposition
```

Get the Iris dataset

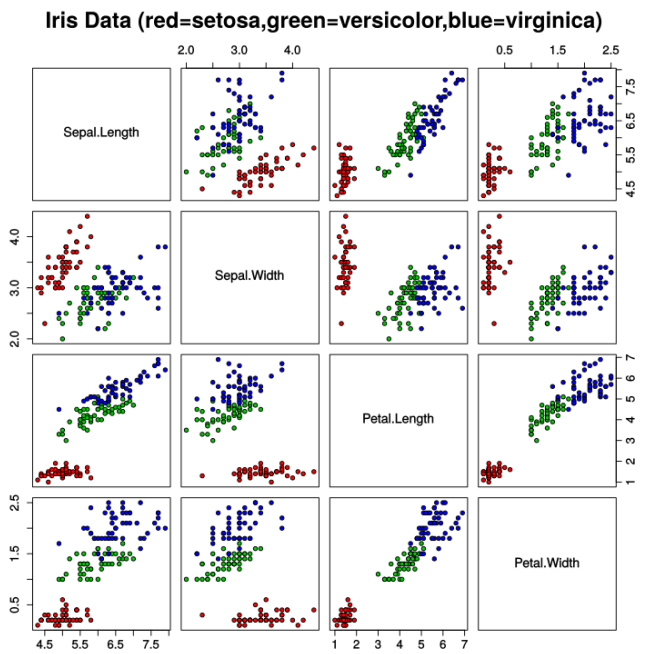
Example data for clustering

The Iris flower data set is a multivariate data set introduced by the British statistician and biologist Ronal Fisher in 1936.

You can read more about it on Wikipedia's page `https://en.wikipedia.org/wiki/Iris_flower_data_set`

You can find the `.csv` file easily; I got it from `https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/datasets/iris.csv`

Let's download it and use in our python script



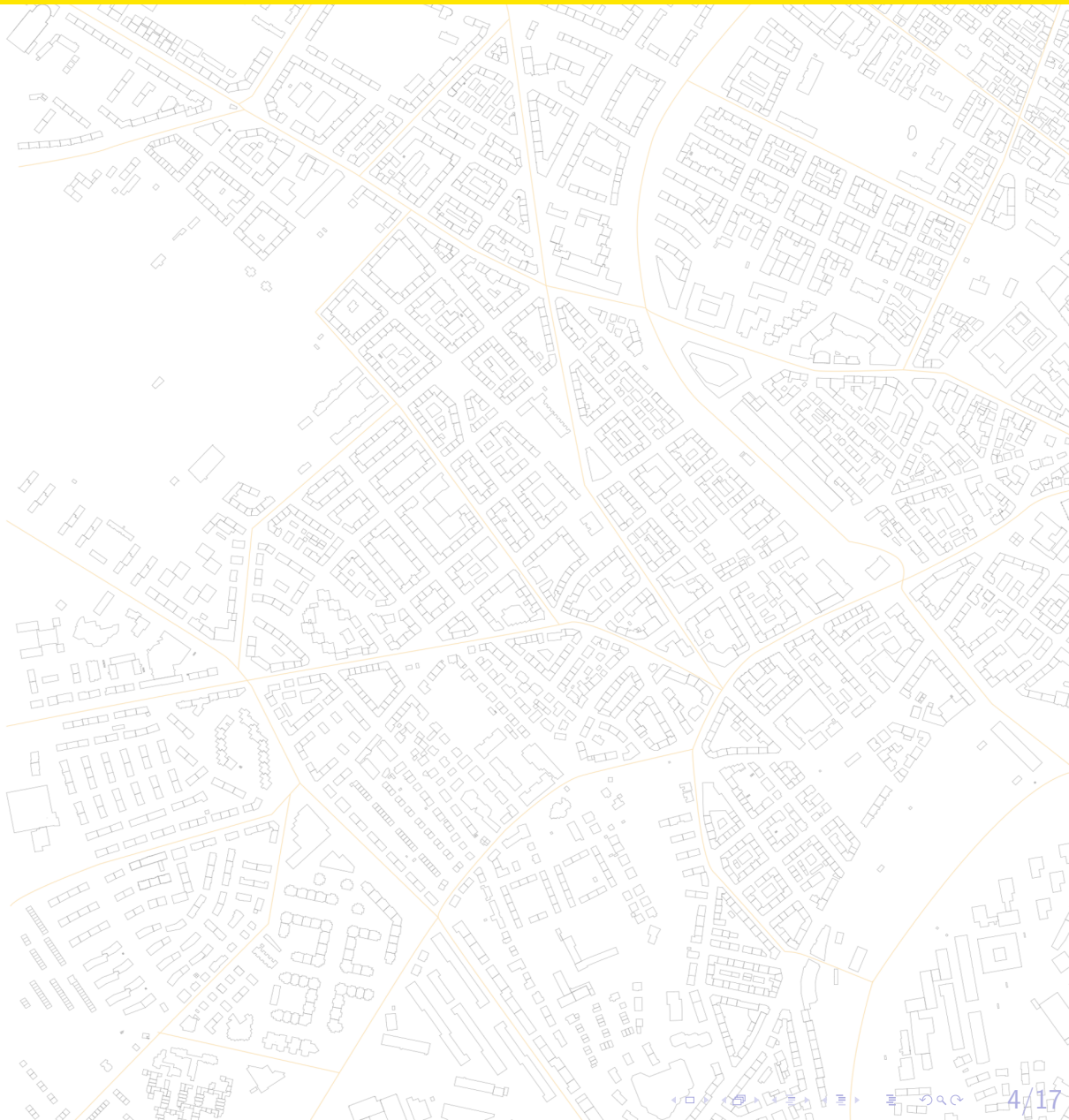
Get the Iris dataset

Example data for clustering

Now, we are ready to load the `.csv` file into the interpreter using `pandas` package.

```
# Using pandas library to load csv file
# into a DataFrame object
ourData = pd.read_csv("iris.csv")

# Created object 'ourData' contains
# a funny and famous data set of flowers.
# At first, we need to explore,
# what is in this data set.
# For this, pandas package provides
# several very useful functions
```

Outline



Explore the Iris dataset

Trying useful functions from pandas

The first useful command is `head()`
It returns first several lines of a dataframe. Very useful to get an idea of what kind of data you have!

```
1 ourData.head()
```

```
#    Unnamed: 0  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
# 0           1           5.1          3.5           1.4          0.2  setosa
# 1           2           4.9          3.0           1.4          0.2  setosa
# 2           3           4.7          3.2           1.3          0.2  setosa
# 3           4           4.6          3.1           1.5          0.2  setosa
# 4           5           5.0          3.6           1.4          0.2  setosa
```

Explore the Iris dataset

Trying useful functions from pandas

Second command, `info()` , can give a more precise information on what data types we operate on.

- The dataset has 150 entries
- Sepal and petal parameters are floating-point numbers
- Species is recognized as object type – in fact, this is a text

```
1 ourData.info()
```

```
# <class 'pandas.core.frame.DataFrame'>
# Int64Index: 150 entries, 0 to 149
# Data columns (total 6 columns):
# Unnamed: 0      150 non-null int64
# Sepal.Length    150 non-null float64
# Sepal.Width     150 non-null float64
# Petal.Length    150 non-null float64
# Petal.Width     150 non-null float64
# Species         150 non-null object
# dtypes: float64(4), int64(1), object(1)
# memory usage: 8.2+ KB
```

Explore the Iris dataset

Trying useful functions from pandas

The last command is more advanced: `ourData.describe()`.

This command gives some statistical information about numeric values in your dataset.

It is useful to understand what is the range of your values.

```
1 ourData.describe()
```

```
#        Unnamed: 0  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
# count  150.000000    150.000000   150.000000    150.000000   150.000000
# mean    75.500000      5.843333     3.057333      3.758000     1.199333
# std     43.445368      0.828066     0.435866      1.765298     0.762238
# min      1.000000      4.300000     2.000000      1.000000     0.100000
# 25%     38.250000      5.100000     2.800000      1.600000     0.300000
# 50%     75.500000      5.800000     3.000000      4.350000     1.300000
# 75%    112.750000      6.400000     3.300000      5.100000     1.800000
# max    150.000000      7.900000     4.400000      6.900000     2.500000
```

Explore the Iris dataset

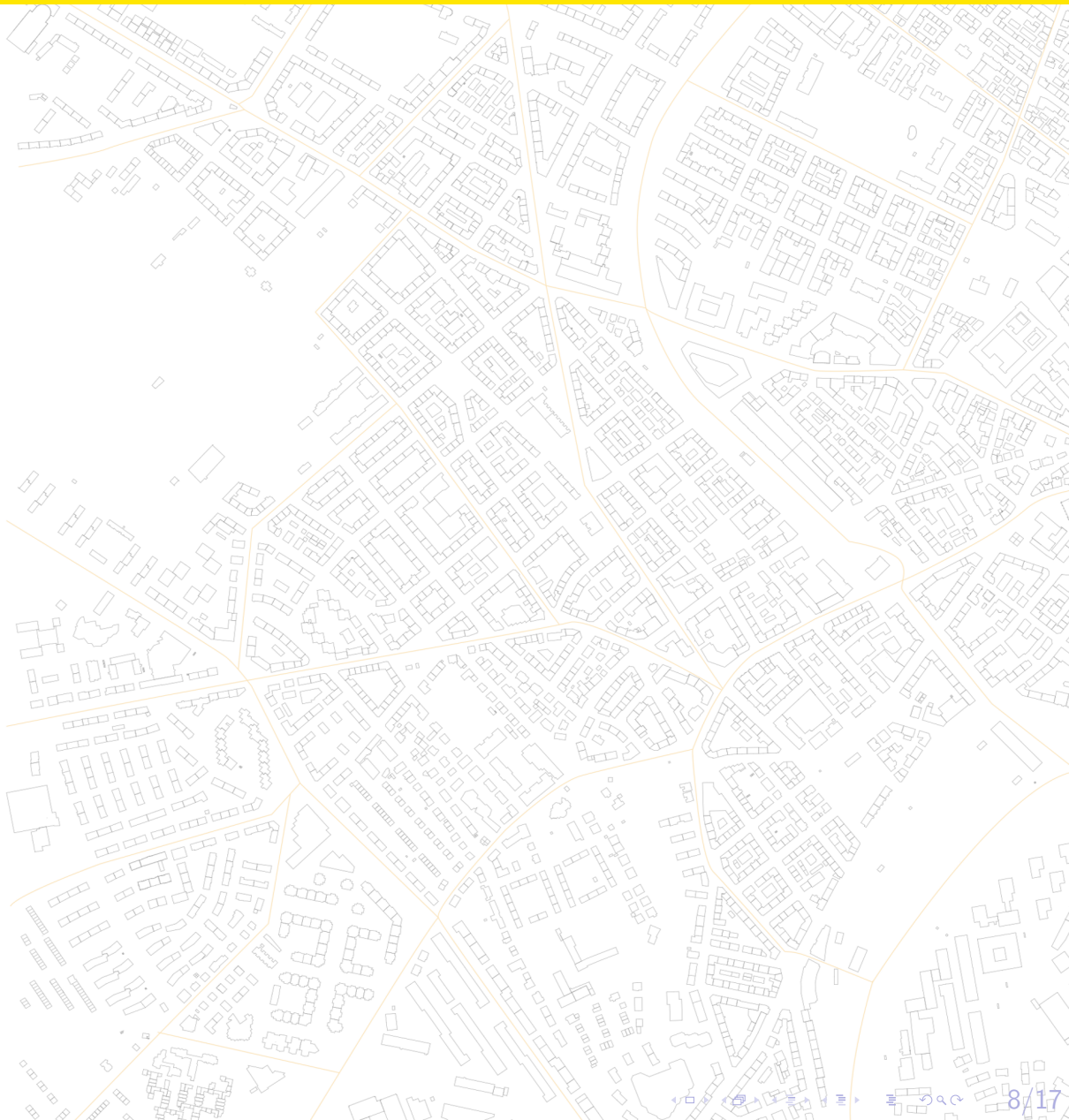
Trying useful functions from pandas

But what about the last column Species?

We've seen these are textual values. Now, let us see what kinds of values does this column have?

```
# Get unique values in the column
species = ourData.Species.unique()

print(species)


# ['setosa' 'versicolor' 'virginica']
```

Outline



Draw some plots

Using `matlplotlib`

Let's try to plot something!

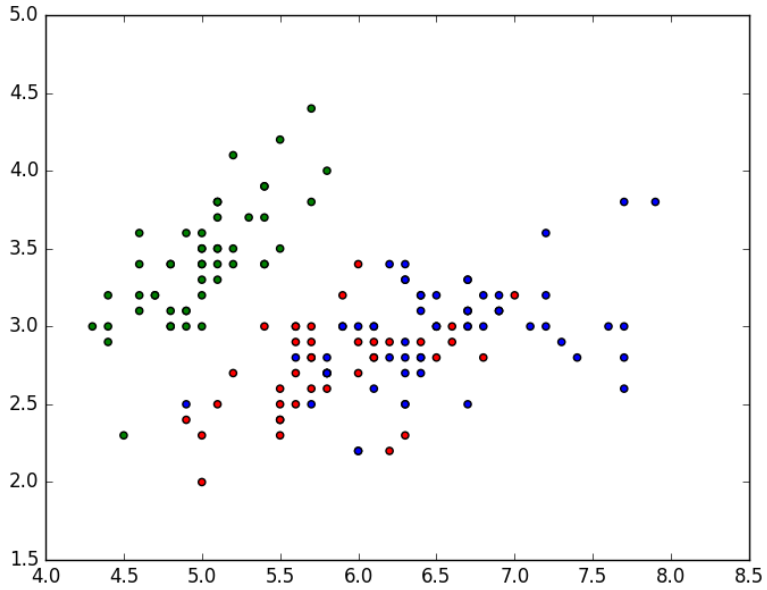
In this example I map names of the species onto colors.
Play with this code a little bit changing the columns to plot. This gives you an insight how the data looks like.

```
# plot different species in different colours
colors = ['g', 'r', 'b', 'c', 'm', 'k']
species_dict = dict(zip(species, colors))
plt.scatter( ourData['Sepal.Length']
           , ourData['Sepal.Width']
           , c=ourData['Species']
                      .apply(lambda x: species_dict[x])
           )
# use either
#   plt.show() # to draw on-screen
# or
#   plt.savefig("filename") # to save picture
plt.savefig("real-labels.png")
```

Draw some plots

Using `matlplotlib`

...And here is how the result looks like



Prepare data

Split input and labels

Copy the data to a new variable `ourDataNoLabels`.
We will play like we don't know the labels and want to estimate them.

```
# Here is our data, but without species information
ourDataNoLabels
  = pd.DataFrame( ourData
                , columns=[ 'Sepal.Length'
                          , 'Sepal.Width'
                          , 'Petal.Length'
                          , 'Petal.Width'
                          ]
                )
```

Principal Component analysis

Using `sklearn.decomposition`

This code performs PCA (lines 3-7).
(Most of the code is just to draw three plots).

Principal component transform is a procedure that rotates point space in such a way that points variance is the biggest along X-axis, the second is along Y-axis, and so on.

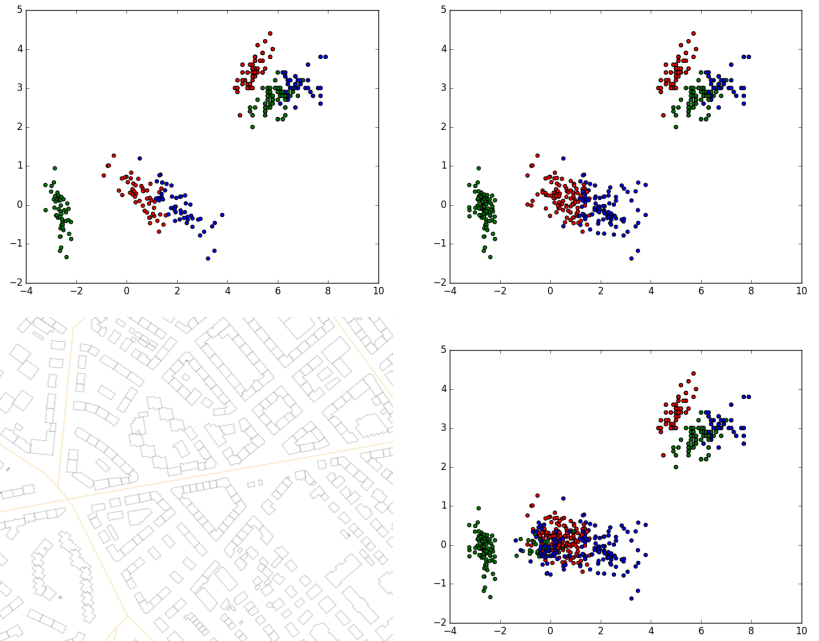
The method is simple but very powerful for data exploration. If data have many dimensions (i.e. 50) it is very convenient to look only at 2-5 most significant dimensions. Otherwise visual inspection of the data would be too difficult.

```
1 # Run Principal Component Analysis
2 # to get more understanding how our data looks like
3 ourDataReduced
4   = decomposition
5     .PCA(n_components=3)
6     .fit_transform(ourDataNoLabels)
7 plt.scatter( ourDataReduced[:,0]
8             , ourDataReduced[:,1]
9             , c=ourData['Species']
10              .apply(lambda x: species_dict[x]))
11 plt.savefig("real-pca-1-2.png")
12 plt.scatter( ourDataReduced[:,0]
13            , ourDataReduced[:,2]
14            , c=ourData['Species']
15              .apply(lambda x: species_dict[x]))
16 plt.savefig("real-pca-1-3.png")
17 plt.scatter( ourDataReduced[:,1]
18            , ourDataReduced[:,2]
19            , c=ourData['Species']
20              .apply(lambda x: species_dict[x]))
21 plt.savefig("real-pca-2-3.png")
```

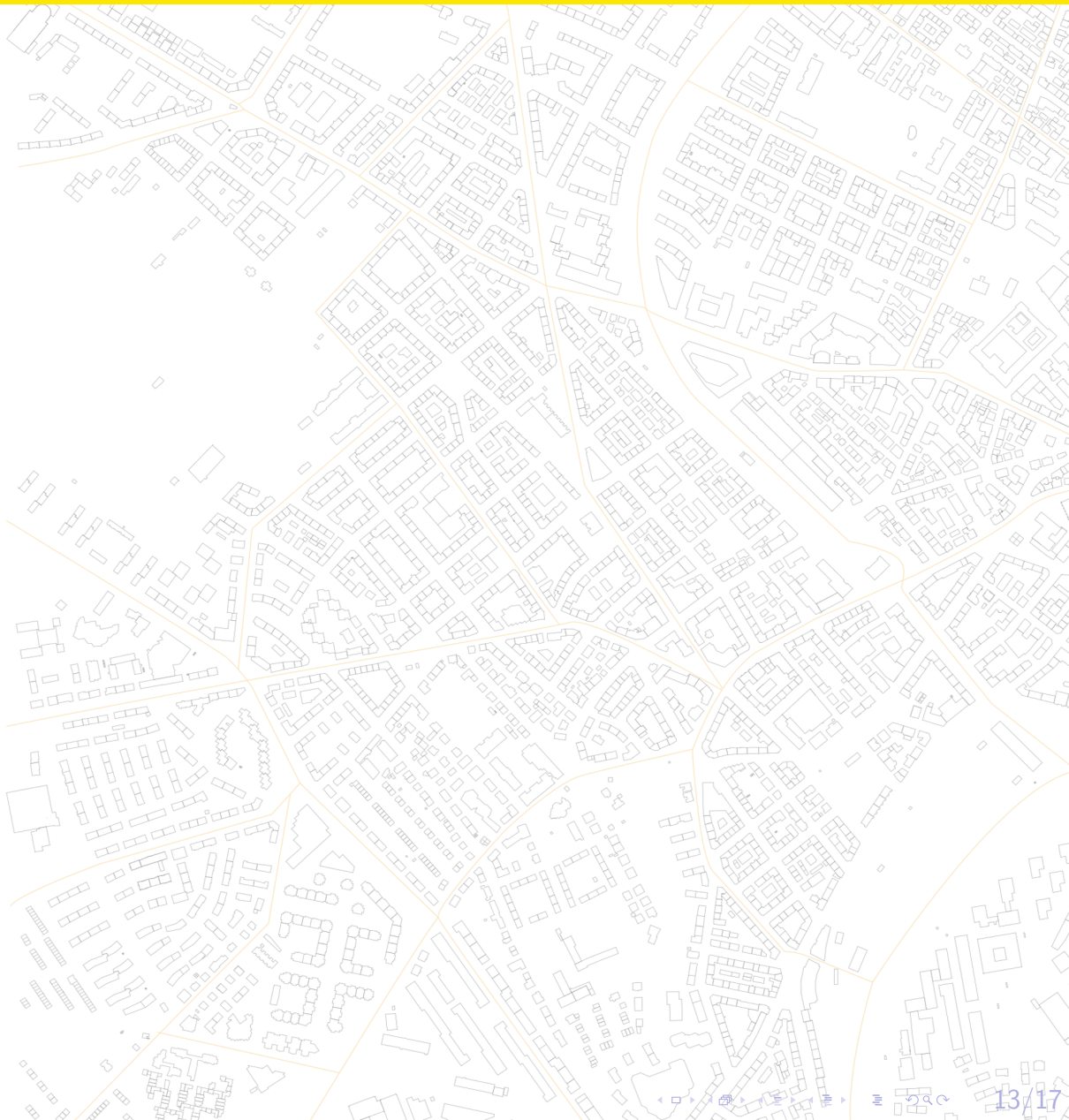
Draw some plots

Using `matlplotlib`

...And here are the plots



Outline



K-means clustering

Using `sklearn.cluster`

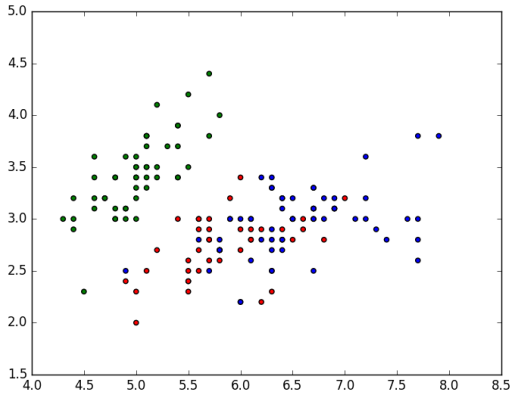
Finally, we are ready for some clustering!

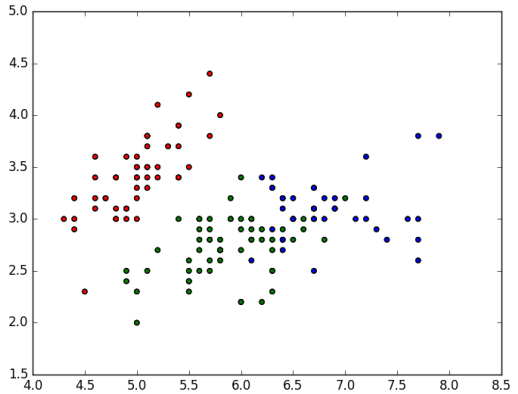
```
# We know that there must be 3 different clusters
#                                 - species of flowers.
# So let's give this hint to the algorithm
kmeans = cluster.KMeans(3)
                    .fit(np.array(ourDataNoLabels))
foundLabels
  = pd.DataFrame( kmeans.labels_
                , columns=['K-means clusters'])


plt.scatter( ourData['Sepal.Length']
           , ourData['Sepal.Width']
           , c=foundLabels['K-means clusters']
                .apply(lambda x: colors[x])
           )
plt.savefig("predicted-labels.png")
```

K-means clustering

Using `sklearn.cluster`

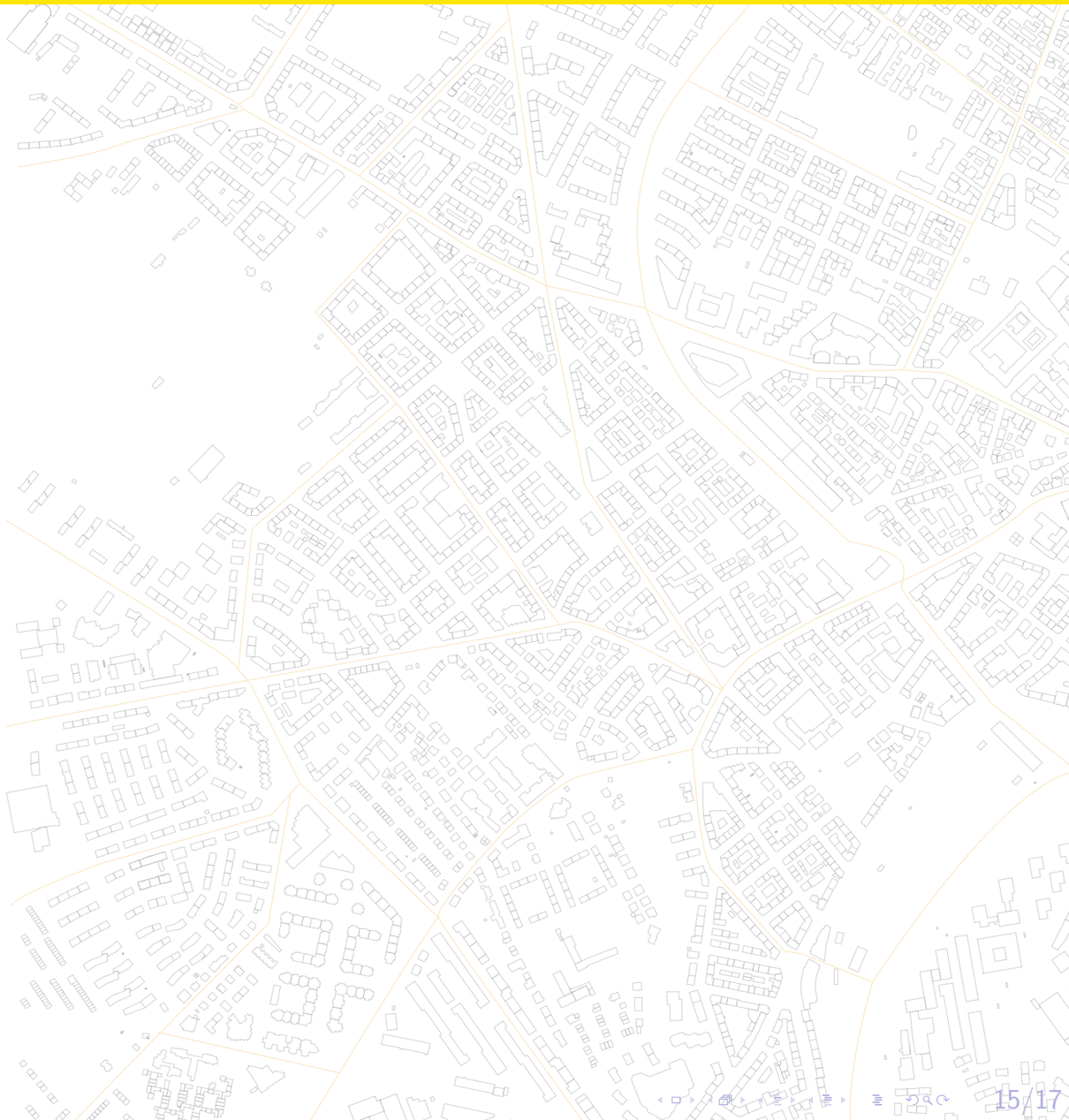
Let's compare plots...





...would you say this is a good result?

Outline



Assess clustering quality

Using pandas

A very good way to assess performance of an unsupervised clustering algorithm is to look at co-occurrence tables.

Package `pandas` provides a special function `crosstab()` that calculates how many times a value from one column occurs together with a value from another column.

So, what would be a conclusion now?

```
1 mat = pd.crosstab( ourData['Species']
2                    , foundLabels['K-means clusters']
3                    )
4 print(mat)
```

```
# K-means clusters   0   1   2
# Species
# setosa             0  50   0
# versicolor        48   0   2
# virginica         14   0  36
```

Bonus: DBSCAN

Using `sklearn.cluster`

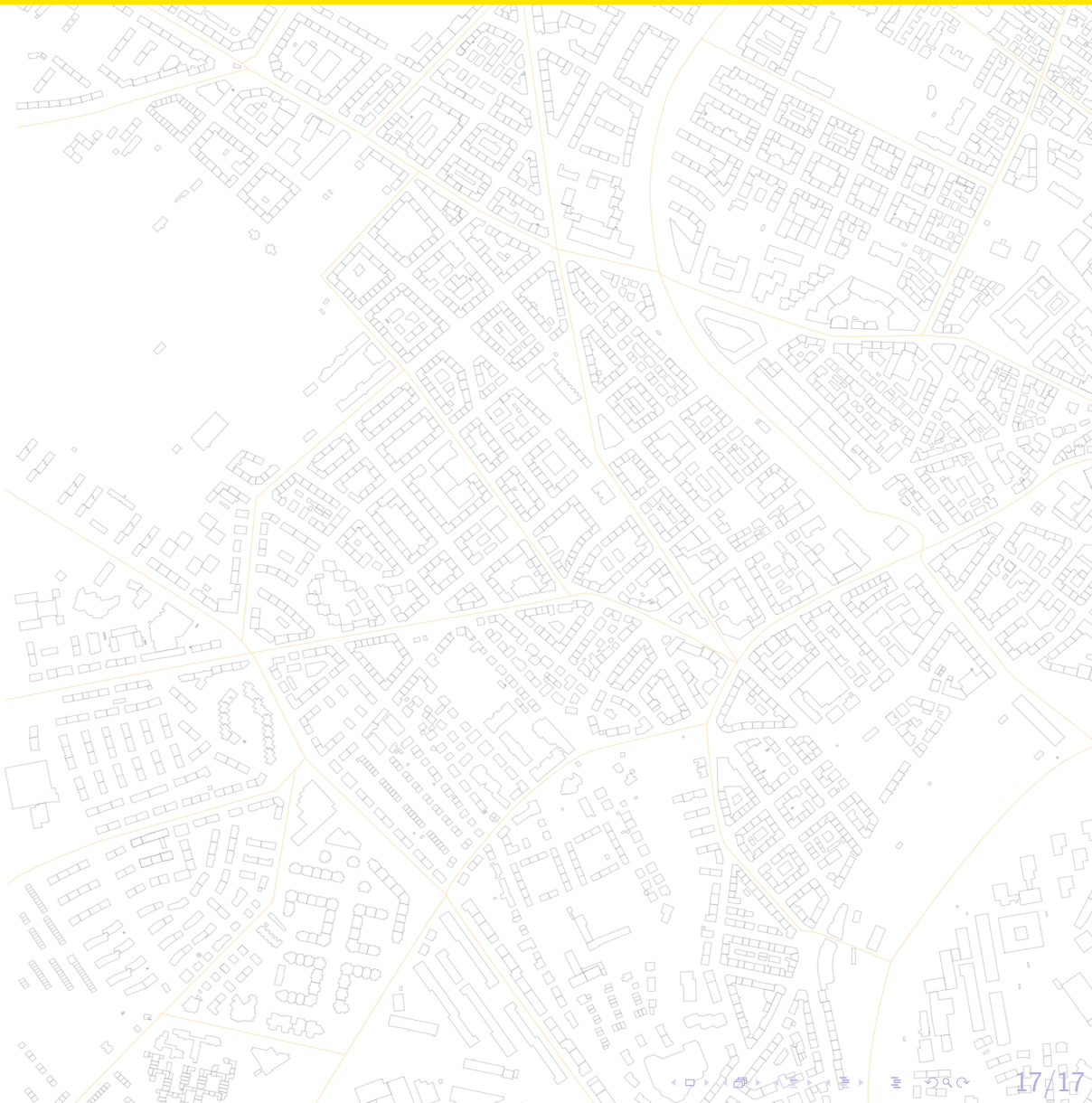
This code does everything the same way as KMeans clustering example.

Try it! And compare the results.
Which algorithm performs better?

```
dbscan = cluster.DBSCAN()
                .fit(np.array(ourDataNoLabels))
foundLabels['DBSCAN clusters']
    = pd.DataFrame( dbscan.labels_
                  , columns=['DBSCAN clusters'])
plt.scatter( ourData['Sepal.Length']
           , ourData['Sepal.Width']
           , c=foundLabels['DBSCAN clusters']
                .apply(lambda x: colors[x]))
plt.savefig("predicted-labels2.png")
mat = pd.crosstab( ourData['Species']
                 , foundLabels['DBSCAN clusters'])
print(mat)
```

What we have covered today

Thanks for your attention!



Getting started

Explore dataset content

Inspect visually

Run clustering

Assess clustering quality

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

DARCH

iA Chair of Information Architecture