Python

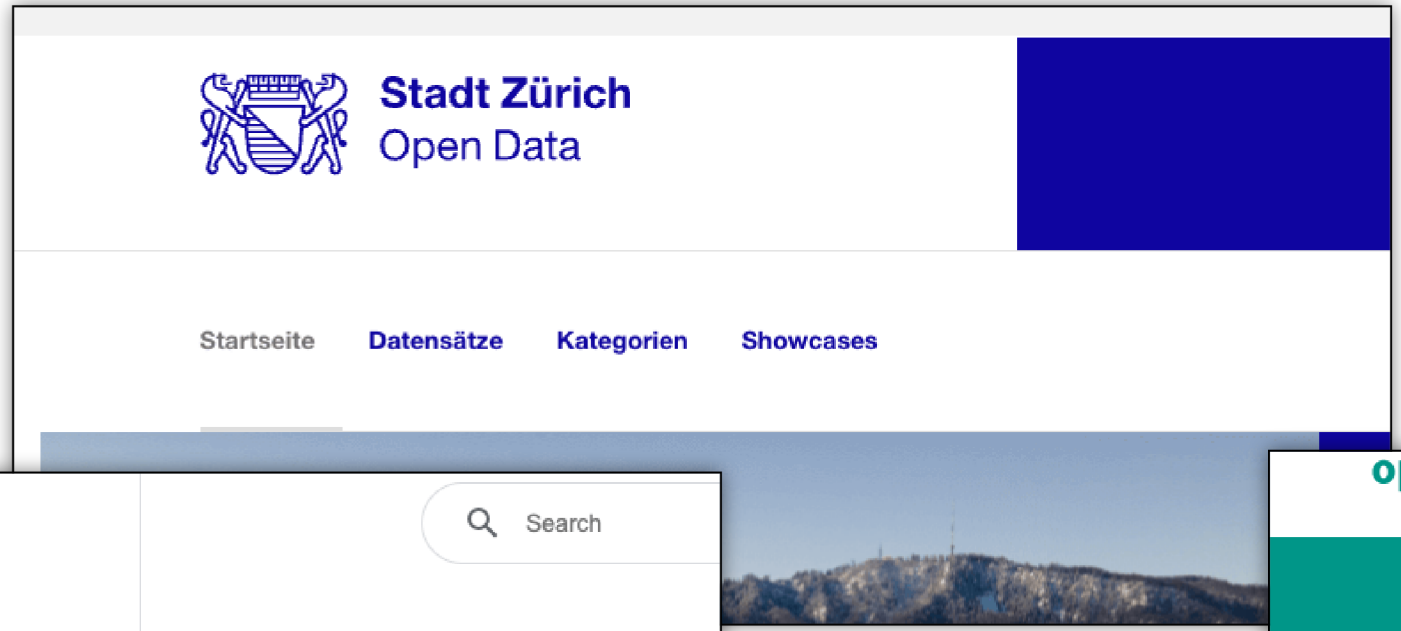
Data & Automation

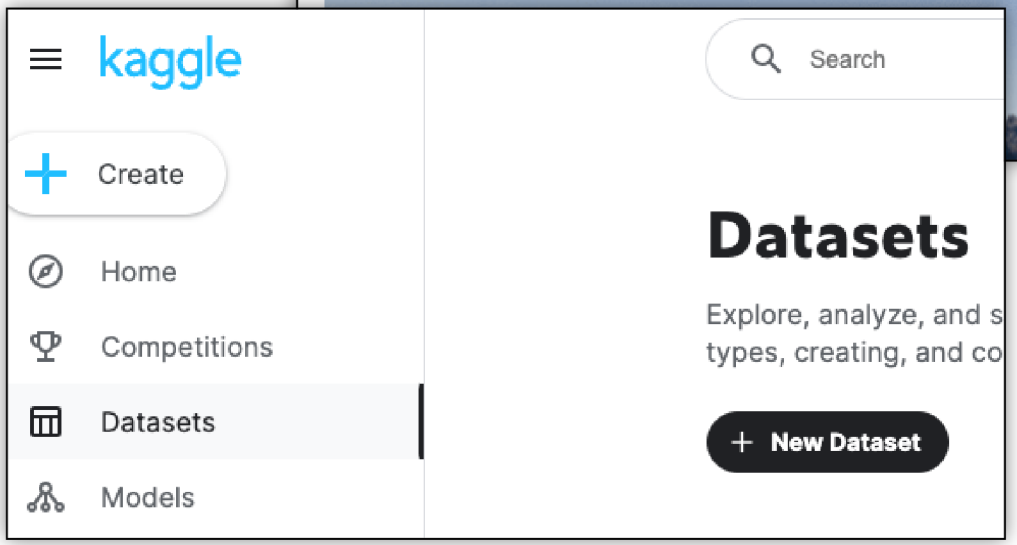
Data

- We encounter data in many forms, some **`well-formed and prepared`** and some still **`very raw or even faulty`**.

- Now that we know how to obtain data from websites, for example, we want to read in data and play it back **`fully automatically`**.

Scenario: In search of data





Scenario: In search of data

- CSV: Comma-Separated Values.
- DXF: Drawing Interchange File Format.
- GPKG: GeoPackage.
- JSON: GeoJSON / JSON.
- SHP: Shapefile.
- WFS: Web Feature Service.
- WMS: Web Map Service.
- WMTS: Web Map Tile Service.

Scenario: In search of data

- CSV: Comma-Separated Values.
- DXF: Drawing Interchange File Format.
- GPKG: GeoPackage.
- JSON: GeoJSON / JSON.
- SHP: Shapefile.
- WFS: Web Feature Service.
- WMS: Web Map Service.
- WMTS: Web Map Tile Service.

Unfortunately, time is too short for GEO data

Scenario: In search of data

We receive the following link from GeoJSON:

https://www.ogd.stadt-zuerich.ch/wfs/geoportal/Standorte_ZueriVelo?service=WFS&version=1.1.0&request=GetFeature&outputFormat=GeoJSON&typename=view_zuerivelo_publibike

Scenario: In search of data

Pretty complicated... Let's look for metadata:

https://www.geocat.ch/geonetwork/srv/ger/catalog.search#/metadata/a7e477f1-75ce-46d2-a32a-c547e37ccfaf/formatters/xsl-view?root=div&view=advanced

Scenario: In search of data


```
      "id": "view_zuerivelo_publibike.9",
      "properties": {
        "adresse": "Bahnhof Stettbach",
        "datum": "2024-01-17T00:00:03.000",
        "id1": 353643.0,
        "id_publibike": 514.0,
        "lat": 47.397885,
        "lon": 8.596202,
        "name": "Bahnhof Stettbach",
        "objectid": 9.0,
        "plz": 8051,
        "stadt": "Zürich",
        "status": "Active"
      },
      "type": "Feature"
    },
    {
      "geometry": {
        "coordinates": [
          8.549831,
          47.421914
        ],
        "type": "Point"
      },
```


Questions:

- Where is the bike located?
- When is the data from?
- What ID does the object have?

Scenario: In search of data

The bottom line is that finding data is not particularly difficult.

Now it's time to **`import and edit`**.

Data formats

We have already learned how to query web resources.
In fact, we have used Soup & request to request XML files (HTML).

But now we want to deal with csv and json.

CSV

CSV stands for comma-separated-values and is a file format. The extension of the file name is .csv.

Roughly speaking, each "data record" in the table corresponds to a text line in the text file.

Each field is separated from the next by a comma.

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

The code contains a lot to discuss, please take a look at it for 1-2 min.

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 0:

Why do we not need to install csv?

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 0:

Why do we not need to install csv?

Answer:

Because it is part of the Python standard library.

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 1:

Why is "open, edit and close" not used here instead of "with"?

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 1:

Why is "open, edit and close" not used here instead of "with"?

Response:

The spelling is simpler and it is safely closed.

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 2:

What does the delimiter mean?

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 2:

What does the delimiter mean?

Answer:

CSV files do not necessarily have to be separated with , - other separators are also possible.

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 3:

What does f' and \t mean?

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 3:

What does f' and \t mean?

Answer:

f' is the formatting function of Python \t is a tab or indent.

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 4:

How exactly is the file processed?

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 4:

How exactly is the file processed?

Response:

The file is processed line by line, with special treatment for the first line.

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 5:

Aren't row[0] and row[2] a bit error-prone?

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 00_employee_birthday.py

import csv

with open('00_employee_birthday.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, '
                  f'and was born in {row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Question 5:

Aren't row[0] and row[2] a bit error-prone?

Answer:

Yes, for this we need to know **`exactly`** where our information is located.

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 01_employee_birthday_dict.py

import csv

with open('00_employee_birthday.csv', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        print(f'\t{row["name"]} works in the {row["department"]} '
              f'department, and was born in {row["birthday month"]}.')
        line_count += 1
    print(f'Processed {line_count} lines.')
```

In response to the question from earlier, we now want to improve a few things, what has been adjusted?

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 01_employee_birthday_dict.py

import csv

with open('00_employee_birthday.csv', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        print(f'\t{row["name"]} works in the {row["department"]} '
              f'department, and was born in {row["birthday month"]}.')
        line_count += 1
    print(f'Processed {line_count} lines.')
```

In response to the question from earlier, we now want to improve a few things, what has been adjusted?

DictReader was used, which treats the csv file like a dictionary.

Accordingly, row[0] also became the speaking row[name].

CSV

Additional information:

- **`delimiter`** specifies the character used to separate the individual fields.
  - The default is the comma (',').
- **`quotechar`** specifies the character used to enclose fields containing the delimiter.
  - The default is a double quotation mark (' " ').
- **`escapechar`** specifies the character used to enclose the delimiter if no quotation marks are used.
  - The default is not an escape character.

CSV

Mode:

Specify File Mode

Here are five different modes you can use to open the file:

Character	Mode	Description
'r'	Read (default)	Open a file for read only
'w'	Write	Open a file for write only (overwrite)
'a'	Append	Open a file for write only (append)
'r+'	Read+Write	open a file for both reading and writing
'x'	Create	Create a new file

Python File Modes

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 02_employee_birthday_writing.py

import csv

with open('02_employee_birthday.csv', mode='w') as employee_file:
    employee_writer = csv.writer(employee_file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)

    employee_writer.writerow(['John Smith', 'Accounting', 'November'])
    employee_writer.writerow(['Erica Meyers', 'IT', 'March'])
```

Now we also want to write to the file. Here it is important to use a **`write mode`** - see last slide.

csv - Exercise

```
1 name,department,birthday month
2 John Smith,Accounting,November
3 Erica Meyers,IT,March
```

We will now deal with 03-exercises/00_own_birthday_writing.

Please copy the .csv and .py into your userfolder.

Questions:

- What happens when the file is executed (with the data)?
- Which **`mode`** could have been used to prevent this? (Simply copy the file again)
- Please add 3 more people

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

```python
# 03_employee_birthday_writing_dict.py

import csv

with open('employee_file2.csv', mode='w') as csv_file:
    fieldnames = ['emp_name', 'dept', 'birth_month']
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'emp_name': 'John Smith', 'dept': 'Accounting', 'birth_month': 'November'})
    writer.writerow({'emp_name': 'Erica Meyers', 'dept': 'IT', 'birth_month': 'March'})
```

Of course, this also works with DictWriter (the counterpart to DictReader) and the nice thing is that dictionaries can now be transferred.

CSV

```
with open('test.csv', 'w') as csvfile:
     fieldnames = test_dict.keys()
     fieldvalues = zip(*test_dict.values())

     writer = csv.writer(csvfile)
     writer.writerow(fieldnames)
     writer.writerows(fieldvalues)
```

Multiple lines are only possible here with detours; we will use pandas for this later.

JSON

Although JSON is called **`JavaScript Object Notation`**, it is a completely independent data format that is not tied to the JavaScript language.

At the same time, with its very simple structure and encoding in the Unicode character set, JSON is the ideal format for exchanging data between systems.

JSON

```
{
    "customers":[
        {
            "firstName": "Max",
            "middleInitial": "M",
            "lastName": "Mustermann",
            "gender": "M",
            "age": 24,
            "address": {
                "streetAddress": "Nordring 98",
                "city": "Nürnberg",
                "state": "Bayern",
                "postalCode": 90409
            },
            "phoneNumbers": [
                { "type": "home", "number": "0911/161803399" }
            ]
        }
        ,
        {
            "firstName": "Erika",
            "lastName": "Gabler",
            "gender": "F",
            "age": 57,
            "address": {
                "streetAddress": "Neuer Wall 72",
                "city": "Hamburg",
                "state": "Hamburg",
                "postalCode": 20354
            },
            "phoneNumbers": [
                { "type": "home", "number": "040/27182818" }
                ,{ "type": "mobil", "number": "0176/31415926" }
            ]
        }
    ]
}
```

https://cdn.bissantz.de/images/2020/02/Bildschirmfoto-2020-02-07-um-08.42.09.png

JSON

```json
{
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ["running", "sky diving", "singing"],
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

As we can see, JSON supports primitive types such as strings and numbers as well as nested lists and objects.

JSON

```json
{
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ["running", "sky diving", "singing"],
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

Wait, that looks like a Python dictionary! I know, right?

That's because it's a form of UON (Universal Object Notation), so they are very similar to each other.

JSON

Python	JSON
`dict`	`object`
`list, tuple`	`array`
`str`	`string`
`int, long, float`	`number`
`True`	`true`
`False`	`false`
`None`	`null`

JSON

If we now have a Python object:

```
# 04_json_write.py

data = {
    "president": {
        "name": "Zaphod Beeblebrox",
        "species": "Betelgeusian"
    }
}
```

We are able to simply serialize it:

```
with open("json_write.json", "w") as write_file:
    json.dump(data, write_file)
```

JSON

If we now have a Python object:

```
data = {
    "president": {
        "name": "Zaphod Beeblebrox",
        "species": "Betelgeusian"
    }
}
```

and this is to be used directly:

```
json_string = json.dumps(data)
```

JSON

Of course, we can also deserialize the JSON file again:

```python
# 05_json_read.py

import json

with open("04_json_write.json", "r") as read_file:
    data = json.load(read_file)

print(data)
print(data['president']['species'])
```

JSON - Exercise

Now it's time for a real example, please create a new file in the userfolder (name for example: json_example.py)

```
1 import json
2 import requests
```

```
1 response = requests.get("https://jsonplaceholder.typicode.com/todos")
2 todos = json.loads(response.text)
```

```
1 type(todos)
2 todos[:10]
```

JSON - Exercise Part 2

Now extend json_example.py so that the first 20 todos are written to a file (json_example.json).

Pandas

As soon as we have to deal with larger amounts of data, or have to switch very flexibly from csv to json to excel, the Pandas package comes into play.

Note: Pandas is very generously explained in APPD, I only use it here to read data easier.

Pandas

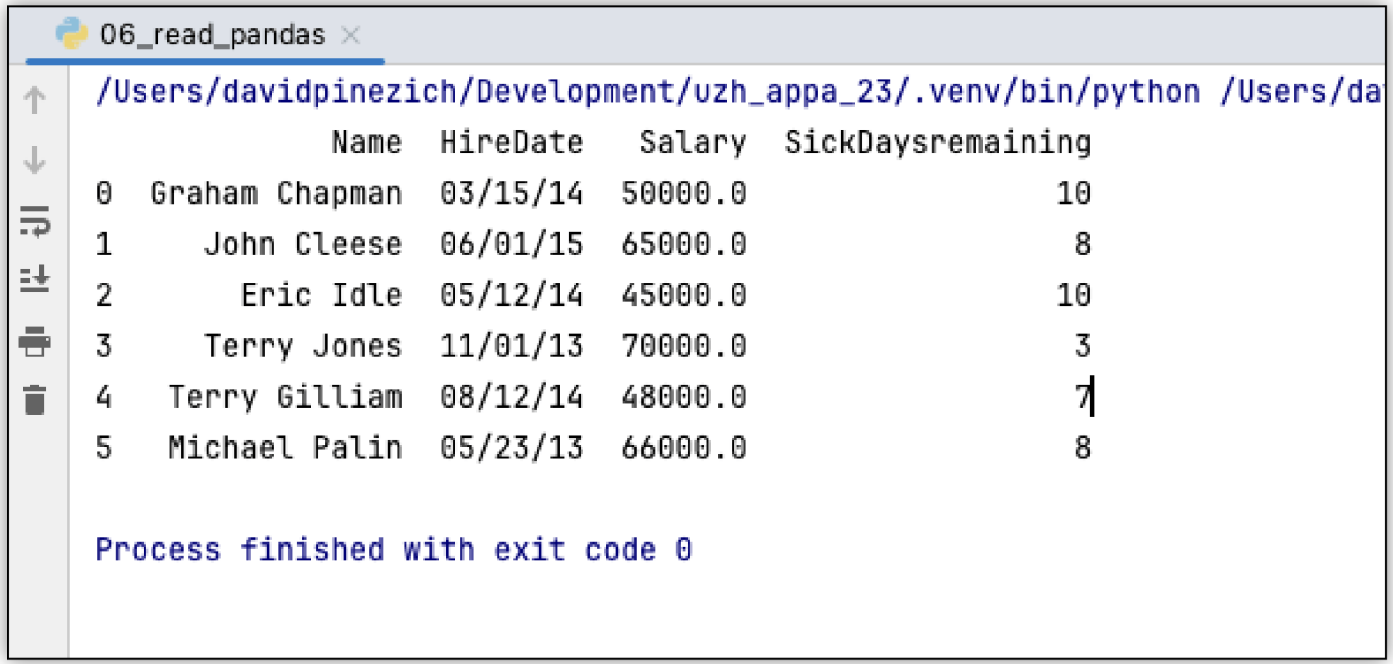
This time it's about an HR data set:

```
Name,HireDate,Salary,SickDaysremaining
Graham Chapman,03/15/14,50000.00,10
John Cleese,06/01/15,65000.00,8
Eric Idle,05/12/14,45000.00,10
Terry Jones,11/01/13,70000.00,3
Terry Gilliam,08/12/14,48000.00,7
Michael Palin,05/23/13,66000.00,8
```

```
# 06_read_pandas.py

import pandas


df = pandas.read_csv('06_hrdata.csv')
print(df)
```

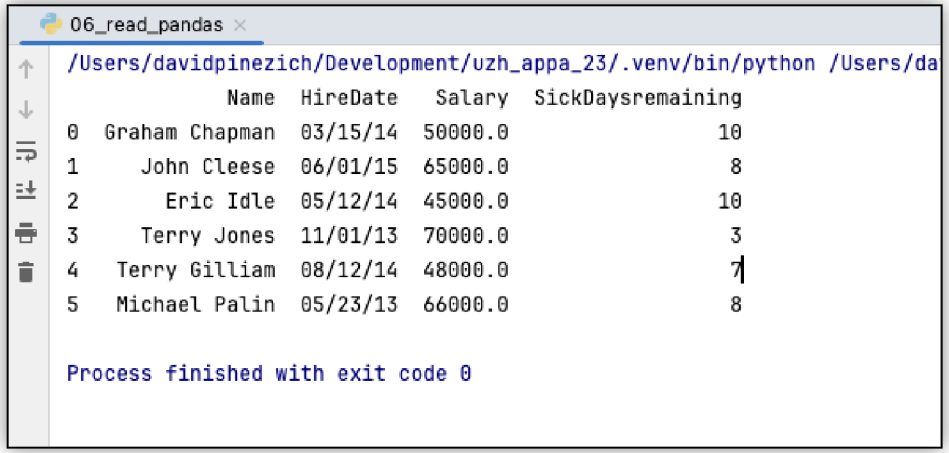
Pandas

Now we get a so-called DataFrame



This is both **highly efficient** and **easy to operate**.

Pandas



- Initially, Pandas recognized that the first line of the CSV file contains column names and used them automatically.
- However, Pandas also uses zero-based integer indices in the DataFrame. This is because we did not tell it **`what our index`** should **`be called`**.
- If you look at the data types of our columns, you will notice that Pandas has properly converted the Salary and Remaining Sick Days columns to numbers, but the Hiring Date column is still a string.

Pandas

Now we want to query something:

```
print(type(df['Hire Date'][0]))
```

Or structure it with a different index:

```
# 07_hrdata_name.csv

import pandas


df = pandas.read_csv('06_hrdata.csv', index_col='Name')
print(df)
```

Pandas

And also parse data:

```
# 08_hrdata_parsed.py

import pandas


df = pandas.read_csv('06_hrdata.csv',
             index_col='Employee',
             parse_dates=['Hired'],
             header=0,
             names=['Employee', 'Hired','Salary', 'Sick Days'])
print(df)
```

Pandas

And finally, we simply create a JSON from it:

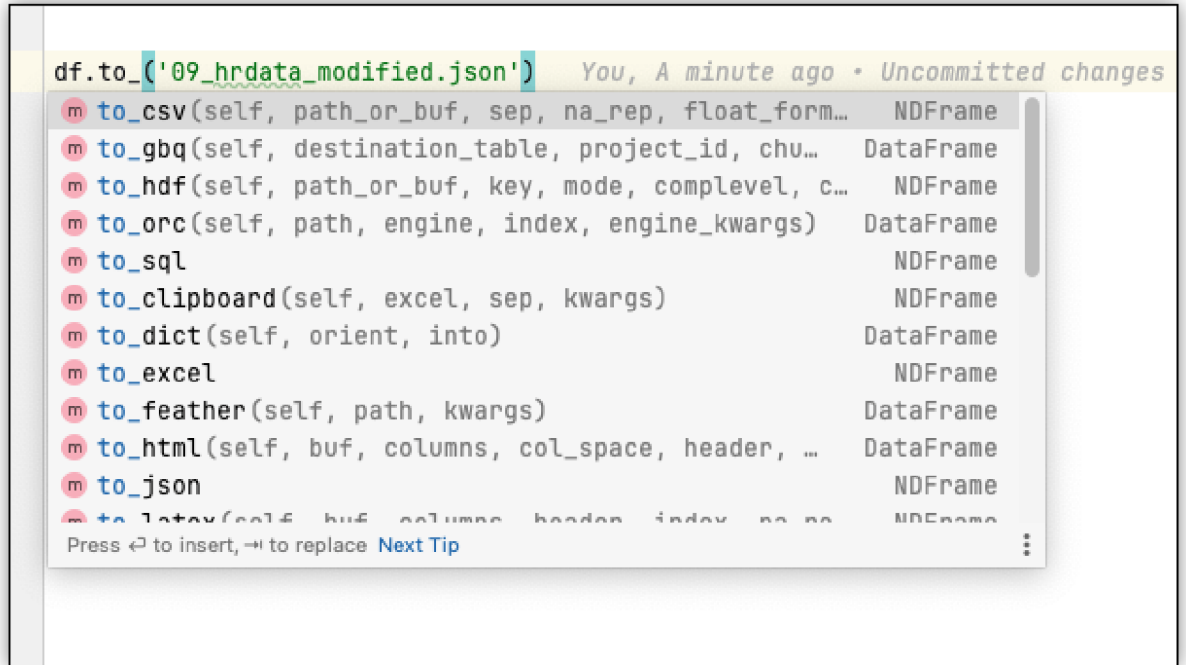
```
# 09_hrdata_to_json.py

import pandas

df = pandas.read_csv('06_hrdata.csv',
                     index_col='Employee',
                     parse_dates=['Hired'],
                     header=0,
                     names=['Employee', 'Hired', 'Salary', 'Sick Days'])

df.to_json('09_hrdata_modified.json')
```

Pandas

But that's not all, please copy the file 09_hrdata_to_json into your userfolder.

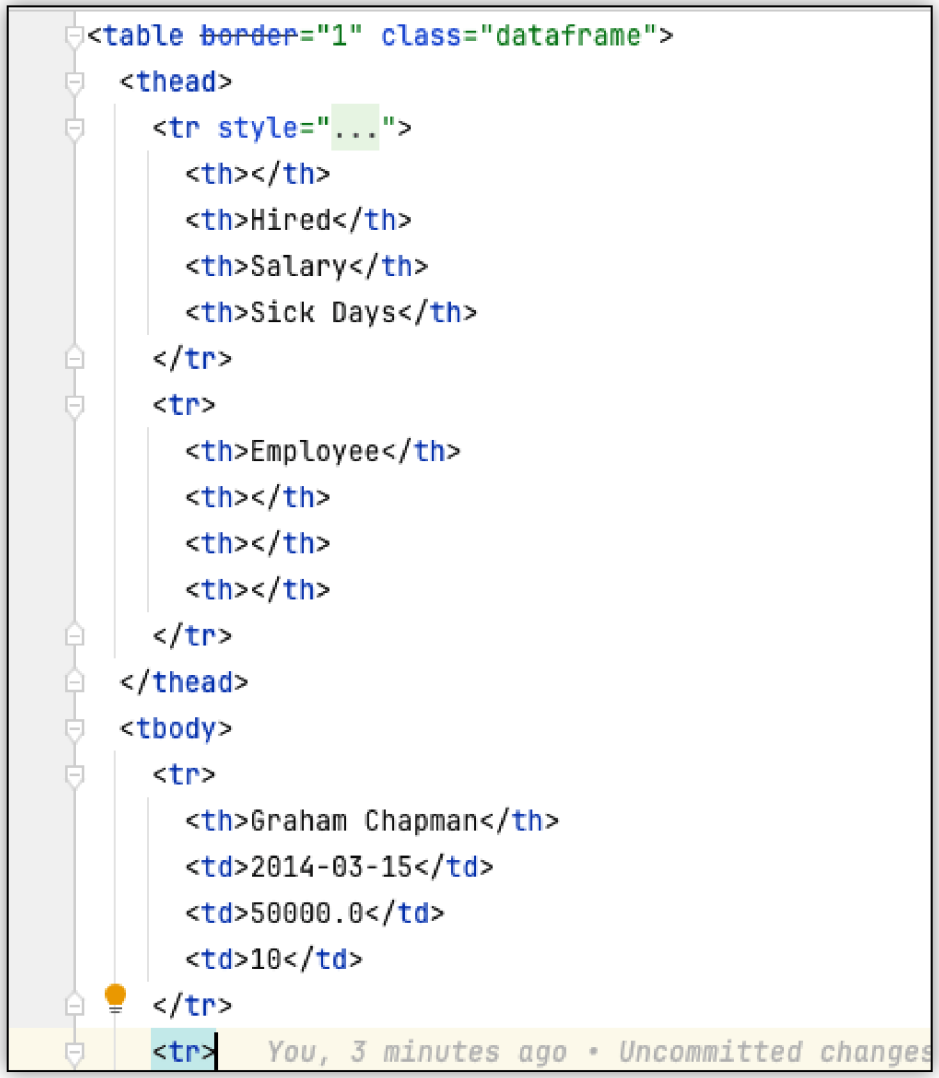
Try to create an Excel file and an HTML file file.

For Excel you need the package openpyxl (pip install openpyxl).



Flask

We received a wonderful HTML from the last task.



Flask

We received a wonderful HTML from the last task.

We can also simply view this in the browser:

	Hired	Salary	Sick Days
Employee			
Graham Chapman	2014-03-15	50000.0	10
John Cleese	2015-06-01	65000.0	8
Eric Idle	2014-05-12	45000.0	10
Terry Jones	2013-11-01	70000.0	3
Terry Gilliam	2014-08-12	48000.0	7
Michael Palin	2013-05-23	66000.0	8

```
<table border="1" class="dataframe">
  <thead>
    <tr style="...">
      <th></th>
      <th>Hired</th>
      <th>Salary</th>
      <th>Sick Days</th>
    </tr>
    <tr>
      <th>Employee</th>
      <th></th>
      <th></th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>Graham Chapman</th>
      <td>2014-03-15</td>
      <td>50000.0</td>
      <td>10</td>
    </tr>
    <tr>   You, 3 minutes ago • Uncommitted changes
```

Flask

We will use this example to familiarize ourselves with Flask.

Flask is a web framework. This means that Flask provides us with tools, libraries, and technologies that can be used to create a web application.

This web application can consist of a few web pages, a blog or a wiki, or even be as large as a web-based calendar application or a commercial website.

Flask

```python
# 10_flask_starter.py

import flask

# Create the application.
APP = flask.Flask(__name__)

@APP.route('/')
def index():
    """ Displays the index page accessible at '/' found in templates
    """
    return flask.render_template('10_hrdata_modified.html')


if __name__ == '__main__':
    APP.debug=True
    APP.run()
```

Flask

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 342-442-388
127.0.0.1 - - [18/Jan/2024 00:36:27] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2024 00:36:27] "GET /favicon.ico HTTP/1.1" 404 -
 * Detected change in '/Users/davidpinezich/Development/uzh_appa_23/data-automation/02-examples/10_flask_starter.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 342-442-388
 * Detected change in '/Users/davidpinezich/Development/uzh_appa_23/data-automation/02-examples/10_flask_starter.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 342-442-388
```

Trial and error is the order of the day on our local server.

Flask

This website could now easily be placed on the Internet using DigitalOcean or PythonAnywhere for around USD 5 per month.

https://www.pythonanywhere.com/

https://www.digitalocean.com/

Flask

This way we can easily post the results of our automation on the Internet if desired.

Excel

Next, we also want to look at Excel, because a lot of data is (and will continue to be) in Excel.

Excel - Exercise

With the openpyxl package, working with Excel is surprisingly easy.

```python
# 11_data_from_excel.py

import openpyxl

wb = openpyxl.load_workbook('ch-bevoelkerung.xlsx')
wb.sheetnames # The workbook's sheets' names.
['VZ RFP 1850', 'VZ RFP 1860', 'VZ RFP 1870']

sheet = wb['VZ RFP 1850'] # Get a sheet from the workbook.

type(sheet)
# <class 'openpyxl.worksheet.worksheet.Worksheet'>
print(sheet.title) # Get the sheet's title as a string.

print(sheet['C15'].value)
print(sheet['D15'].value)
```

Excel - Exercise

Try displaying all the municipality names:

Excel - Exercise

Try displaying all the municipality names:

```python
# 12_data_from_excel_iterate.py

import openpyxl

wb = openpyxl.load_workbook('ch-bevoelkerung.xlsx')
wb.sheetnames # The workbook's sheets' names.
['VZ RFP 1850', 'VZ RFP 1860', 'VZ RFP 1870']

sheet = wb['VZ RFP 1850'] # Get a sheet from the workbook.

type(sheet)
# <class 'openpyxl.worksheet.worksheet.Worksheet'>
print(sheet.title) # Get the sheet's title as a string.

print(sheet['C15'].value)
print(sheet['D15'].value)

for cell in sheet['C']:
    print(cell.value)
```

Excel - Exercise

Now try the same thing but with pandas.

```
1 import pandas
2
3 df = pandas.read_excel(open('ch-bevoelkerung.xlsx', 'rb'),
4                sheet_name='VZ RFP 1850', index_col=2, header=2)
```

Visualize data

Visualizing data is often something that follows from our previous work.

The topic is very broad, and the following is just an overview.

Visualization should be an iterative process

1. problem definition
2. data
3. evaluation (how do I define success)
4. features (what exactly do I need)
5. visualization
6. testing (experimenting)

A step back is possible at any time due to new findings (and not necessarily a bad thing)!

(Rough) procedure

This can also be used for a visualization:

1. problem definition
2. data
3. evaluation (how do I define success)
4. features (what exactly do I need)
5. visualization
6. testing (experimenting)

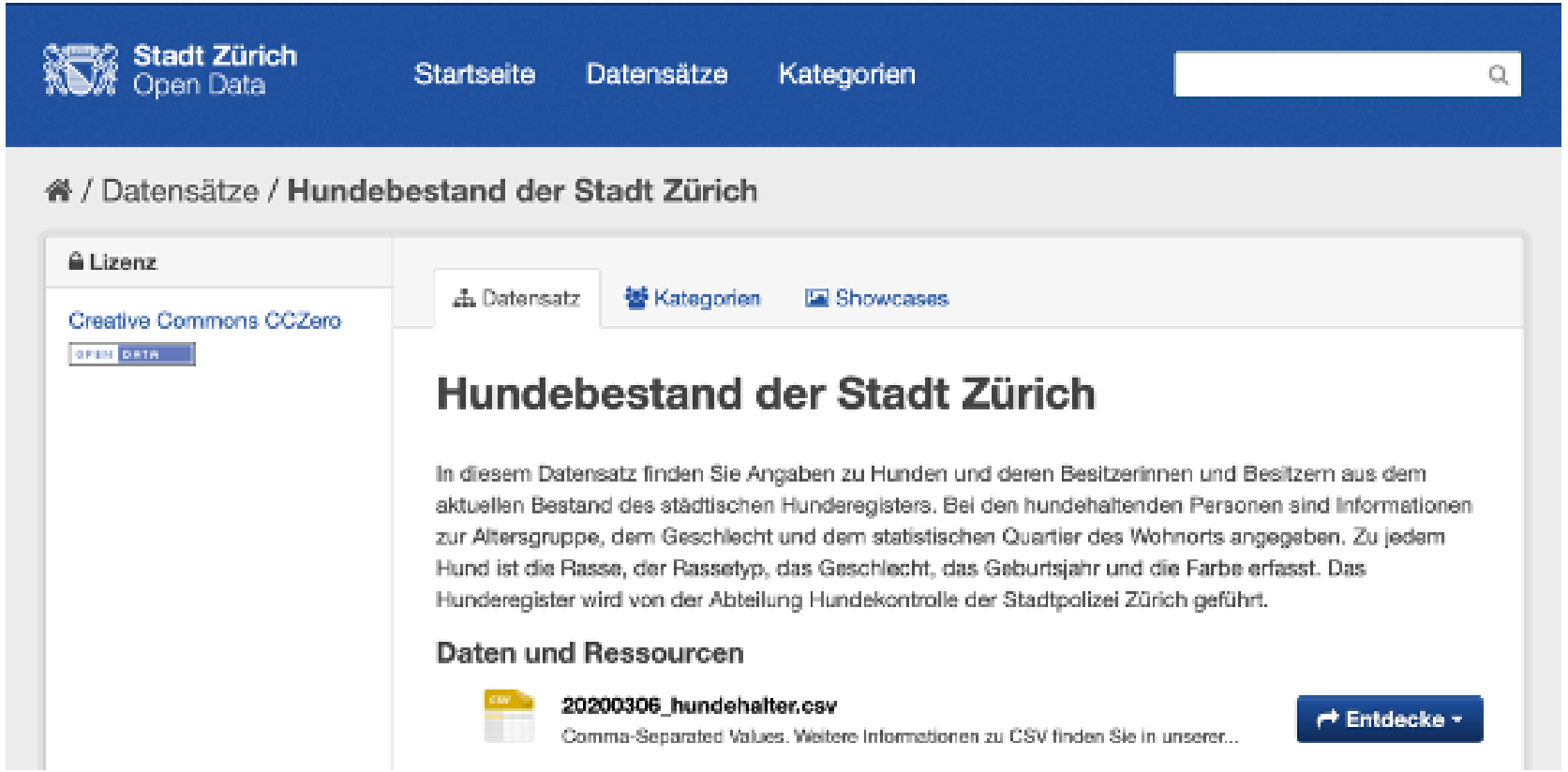
Dog-Science

The trendy brand for your dog

Problem definition (problem definition)

- Dog-Science is a trendy brand from Asia and would like to expand to Zurich.
- In Dog-Science's home country, it is well known where most dog owners live and which products are preferred. But Switzerland, and Zurich in particular, is unknown.
- Dog-Science's budget is limited, but the city of Zurich has (notionally) promised data and support for a pop-up store
- Can we improve Dog-Science's go-to-market position?

A first "look" at the data



A first "look" at the data

HALTER_ID	ALTER	GESCHLECHT	STADTKREIS	STADTQUARTIER	RASSE1	GEBURTSJAHR_HUND	GESCHLECHT_HUND	HUNDEFARBE
574	61-70	w	2	23	Mischling gross	2013	w	schwarz
695	41-50	m	6	63	Labrador Retriever	2012	w	braun
893	71-80	w	7	71	Mittelschnauzer	2010	w	schwarz
916	41-50	m	3	34	Mischling klein	2015	w	hellbraun
1177	51-60	m	10	102	Shih Tzu	2011	m	schwarz/weiss
4054	51-60	w	11	111	Lagotto Romagnolo	2016	w	weiss/beige
4135	41-50	w	9	91	Mischling klein	2016	w	schwarz
4206	71-80	w	8	82	Havaneser	2016	w	hellbraun/weiss
4281	61-70	w	9	91	Chihuahua	2011	w	hellbraun
4388	61-70	w	11	115	Mops	2006	m	beige
4726	51-60	m	5	52	Mischling gross	2007	m	schwarz
4726	51-60	m	5	52	Golden Retriever	2013	w	creme
4747	61-70	m	2	24	Chihuahua	2013	m	weiss/braun
4850	51-60	m	4	42	Chihuahua	2013	w	beige
4862	51-60	m	4	42	Mops	2006	m	braun
5040	61-70	m	10	102	Labrador Retriever	2016	w	gelb
5088	61-70	m	7	72	Labrador Retriever	2014	w	schwarz
5113	61-70	m	11	119	Beagle	2010	w	tricolor
5113	61-70	m	11	119	Chihuahua	2017	w	beige
5225	71-80	m	3	34	Lagotto Romagnolo	2007	m	braun
5227	71-80	m	10	101	Border Terrier	2011	w	tricolor

A first "look" at the data

- The data is of good, but not perfect quality

  The breeds were unfortunately defined very imprecisely
- Often defined as "mixed breed large" or "mixed breed small", but not further defined
- Often represented
  - Chihuahua 573
  - Labrador Retriever 426
- Rarely represented
  - Austrian Pinscher 1
  - Daisy-Dog 1

A first "look" at the data

- Significantly more female than male dogs (5402 to 2439)

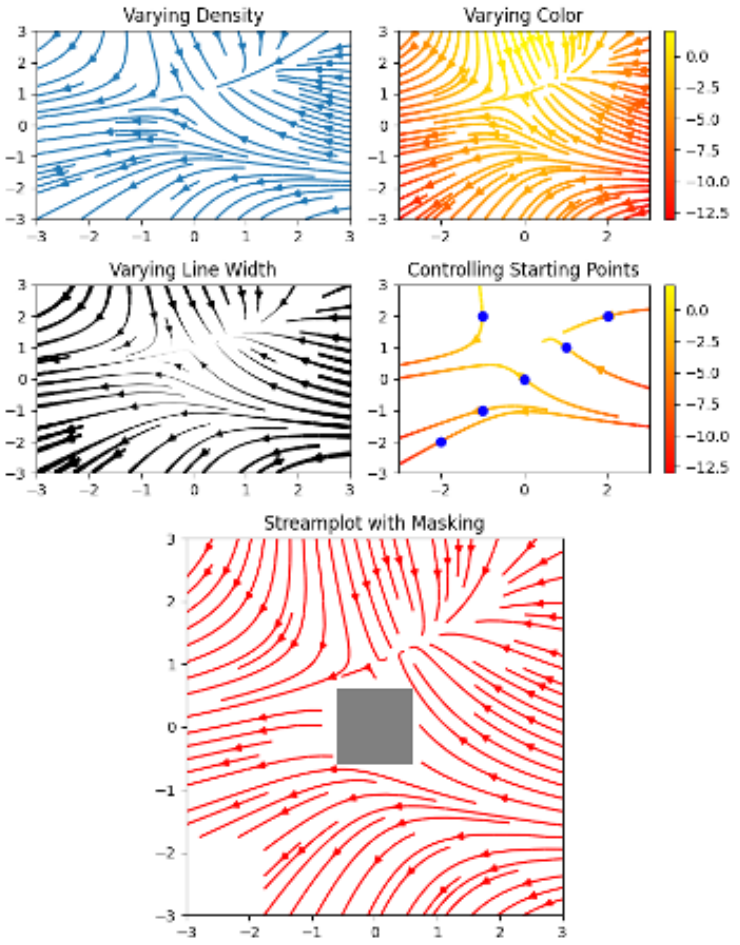
  Black is the most common coat color (800), followed by tricolor (725) and white (634), less often "black mottled (melliert)" or "yellow / black" once each.
- We take a look at the age of the dogs in the visualizations (please note the outliers)
- What is the dog owner / dog ratio?
  - 6562 dog owners have one registered dog
  - 581 dog owners have 2 or more registered dogs
  - The top dog owner has 14 registered dogs

Visualization

- Many libraries with different specialties
  - Matplotlib: For static, animated and interactive visualizations (one of the oldest libraries)
  - Pygal: Dynamic SVG charting library
  - Seaborn: Based on Matplotlib and offers a high-level interface for statistical graphics
  - Altair: Based on Vega/Vega Lite and is a declarative statistical visualization library
  - Ggplot2: System for the creation of declarative graphics
  - Plotly: Interactive and analyzable by the user
  - Bokeh: Library for interactive visualizations
  - Geoplotlib: Mainly for maps

Matplotlib

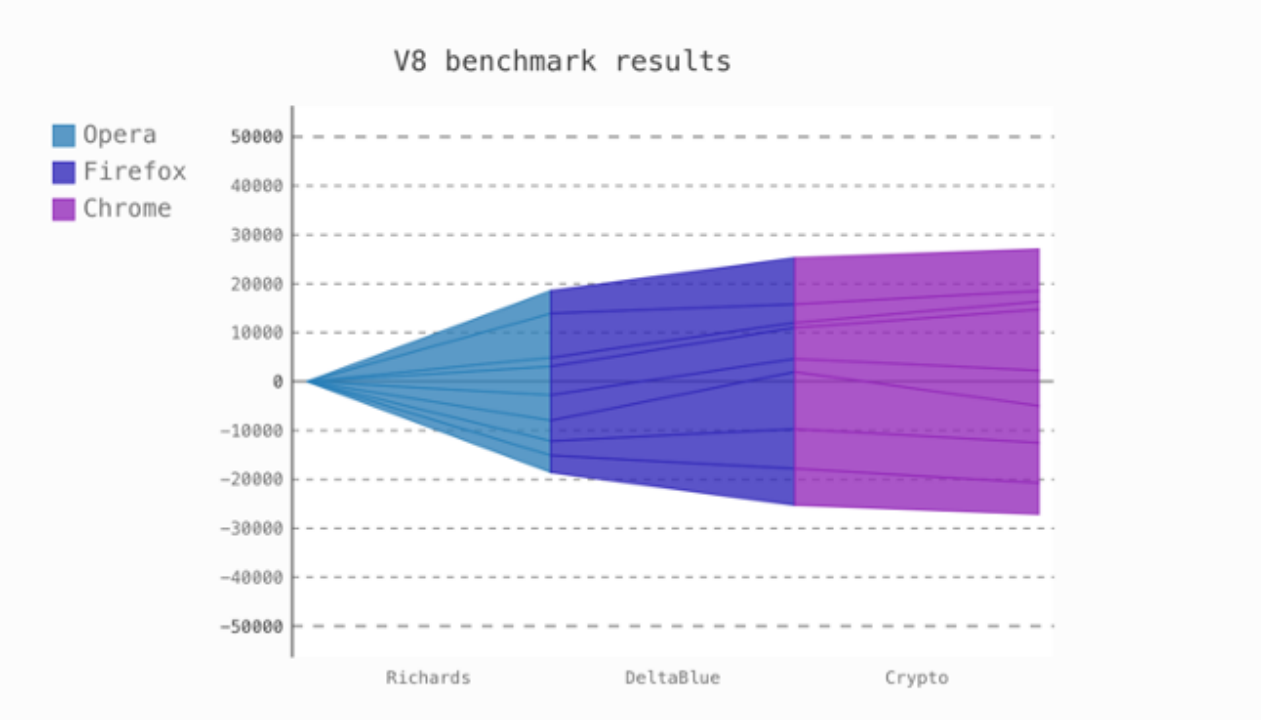
- Suitable for simple and complex representations
- Multiple representations possible via subplots
- Offers the largest and most general spectrum



Streamplot with various plotting options.

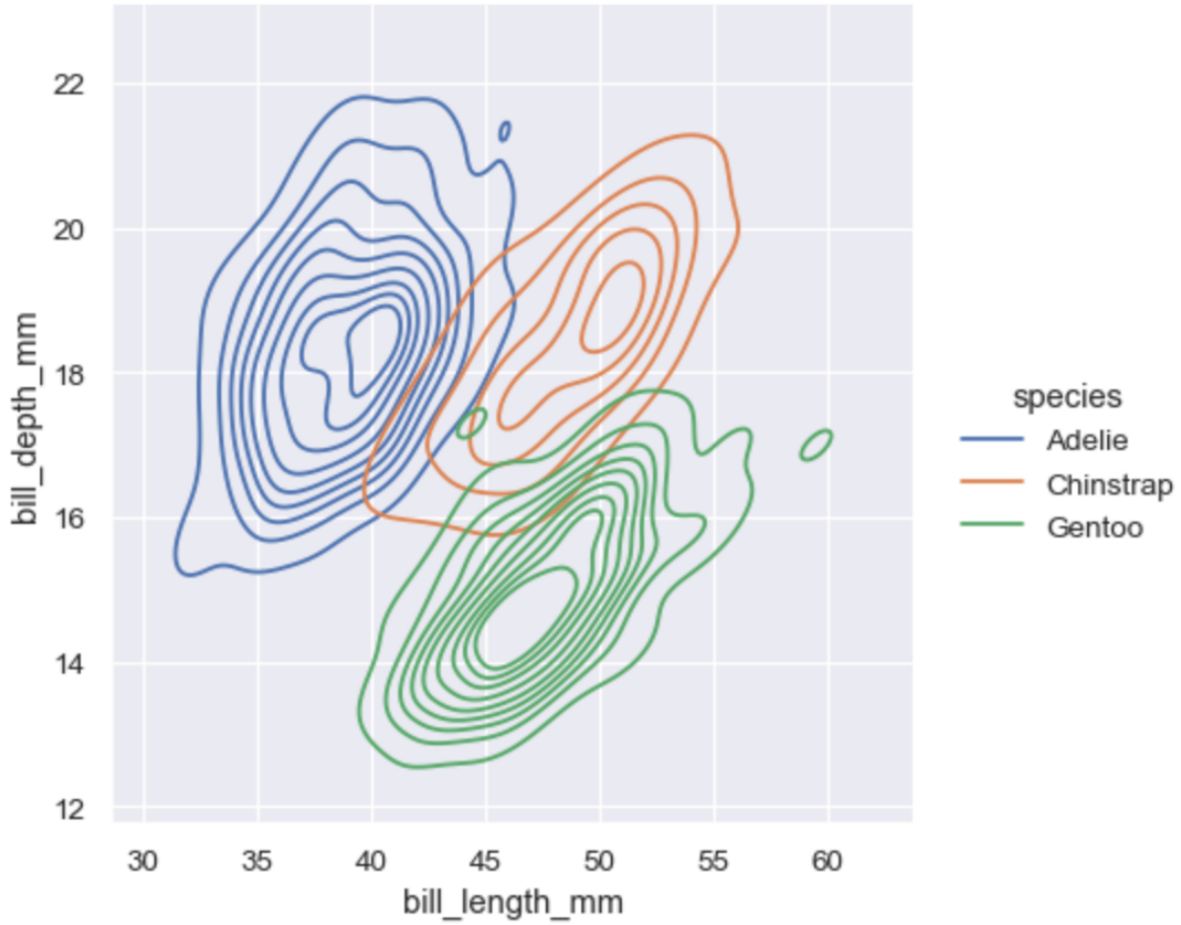
Pygal

- Unfortunately no longer (actively) developed
- Good range of functions and relatively easy to learn
- Easy to output as interactive HTML



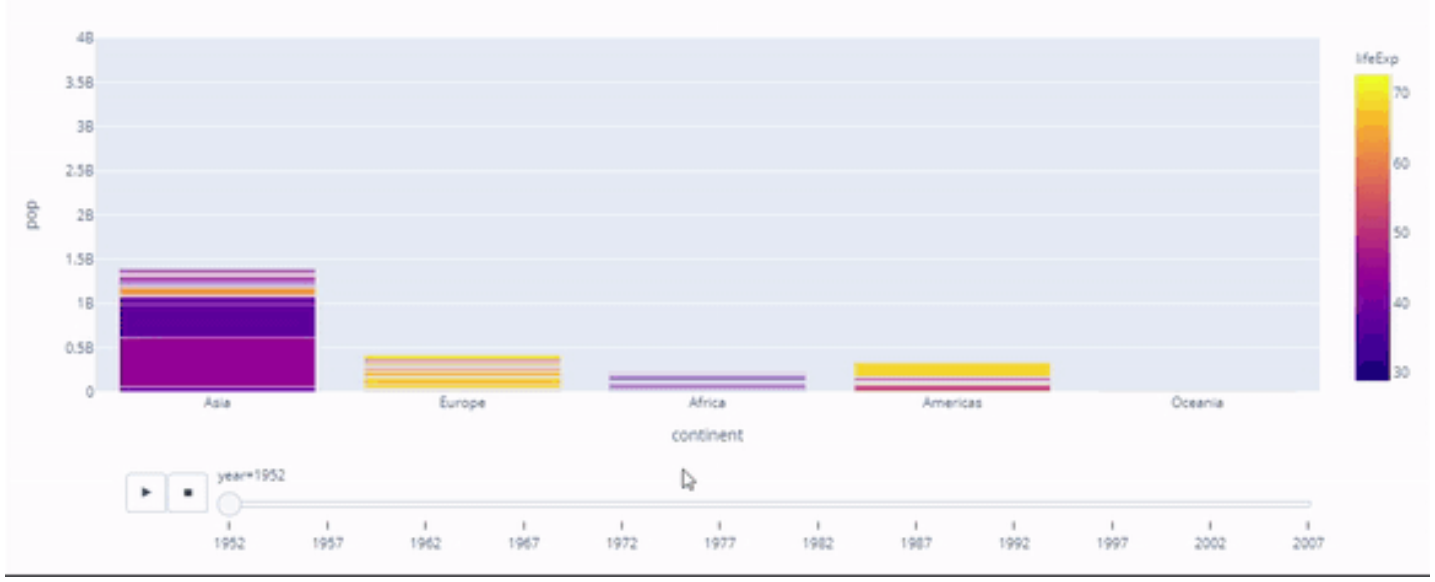
Seaborn

- Based on Matplotlib
- Especially suitable for statistical visualizations
- **`Will be our first example`**



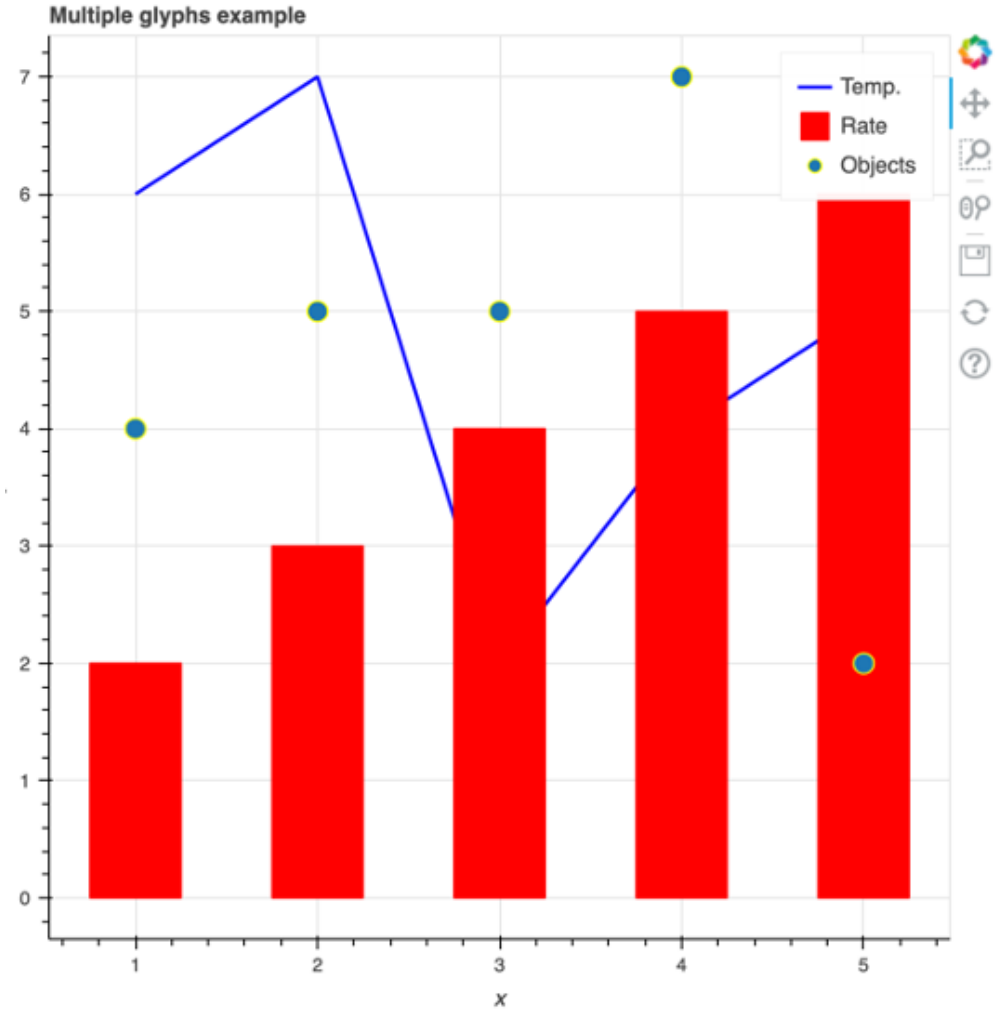
Plotly

- Offers a wide range of charts
- Can be animated
- Has "out of the box" manipulation tools

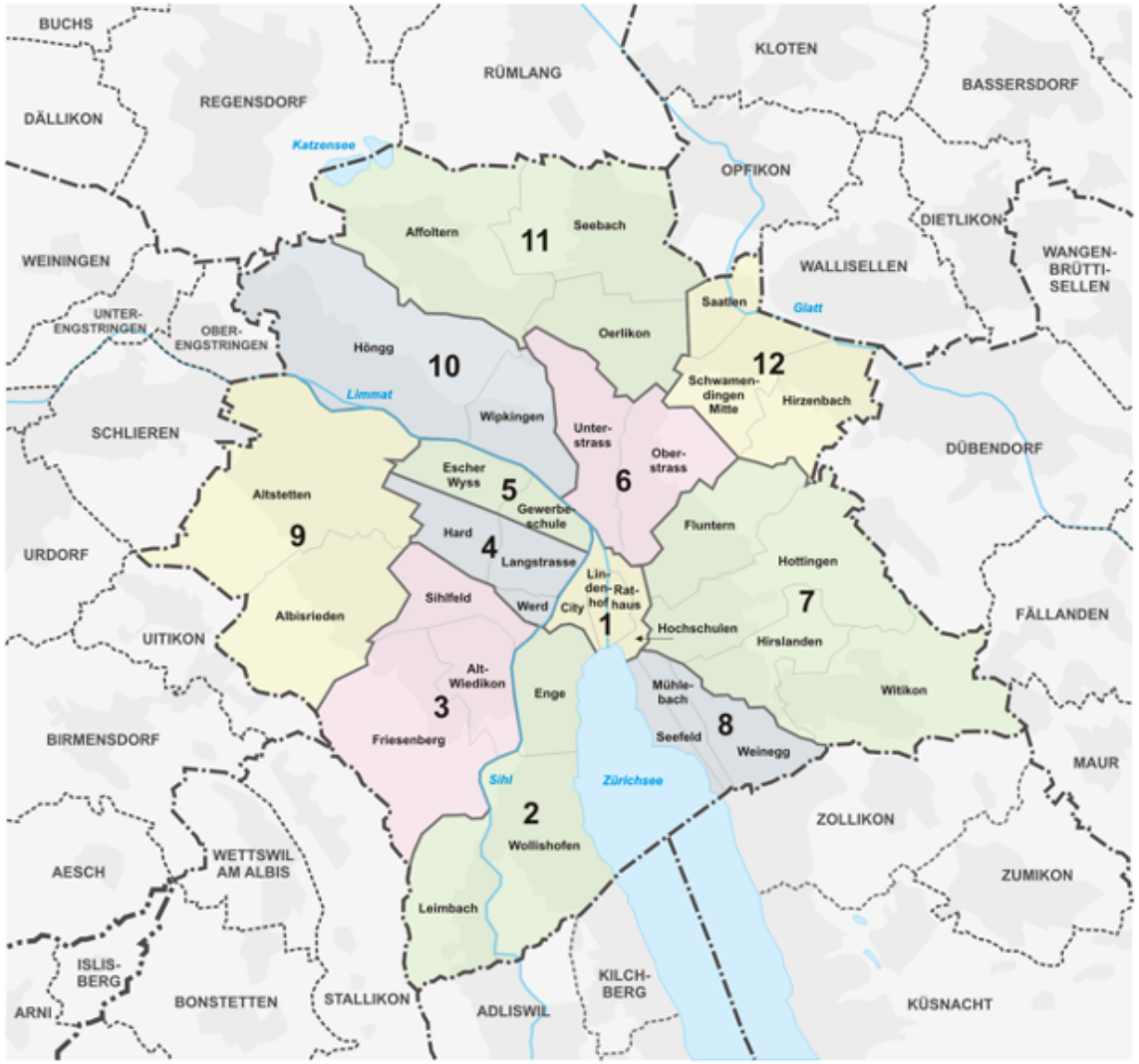


Bokeh

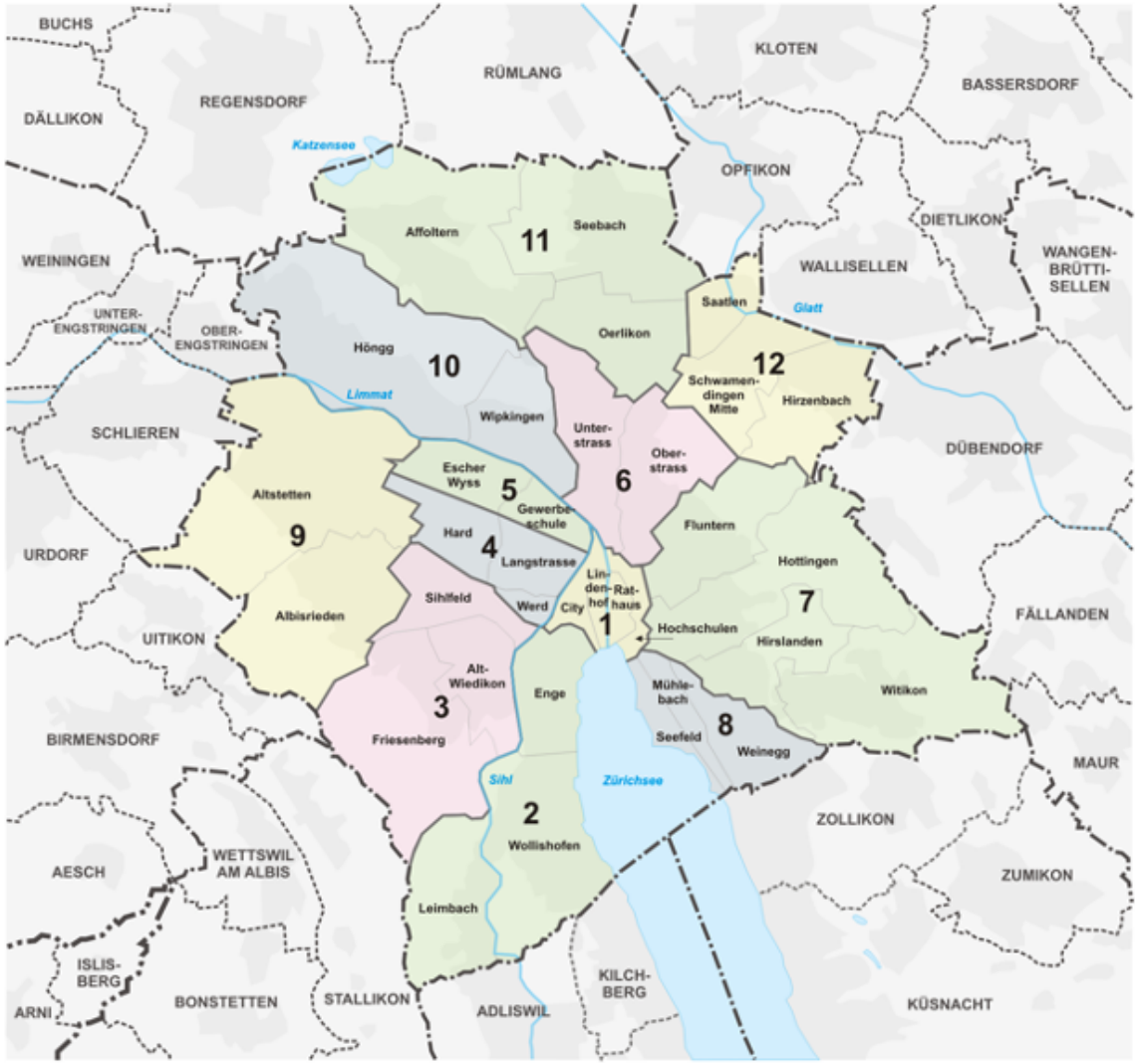
- Offers a wide range of interactive charts
- Has "out of the box" manipulation tools



The place is important...



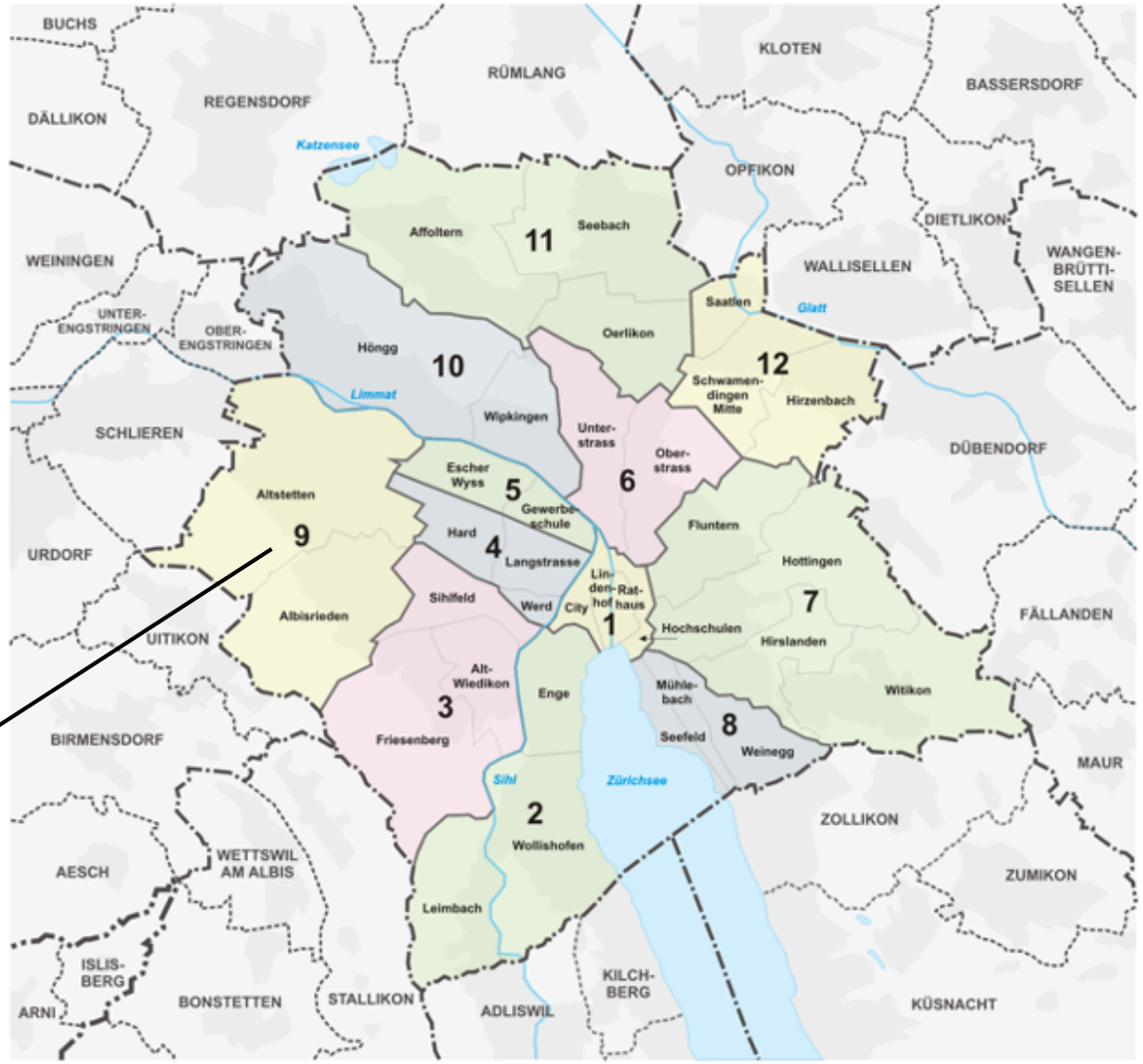
The place is important...



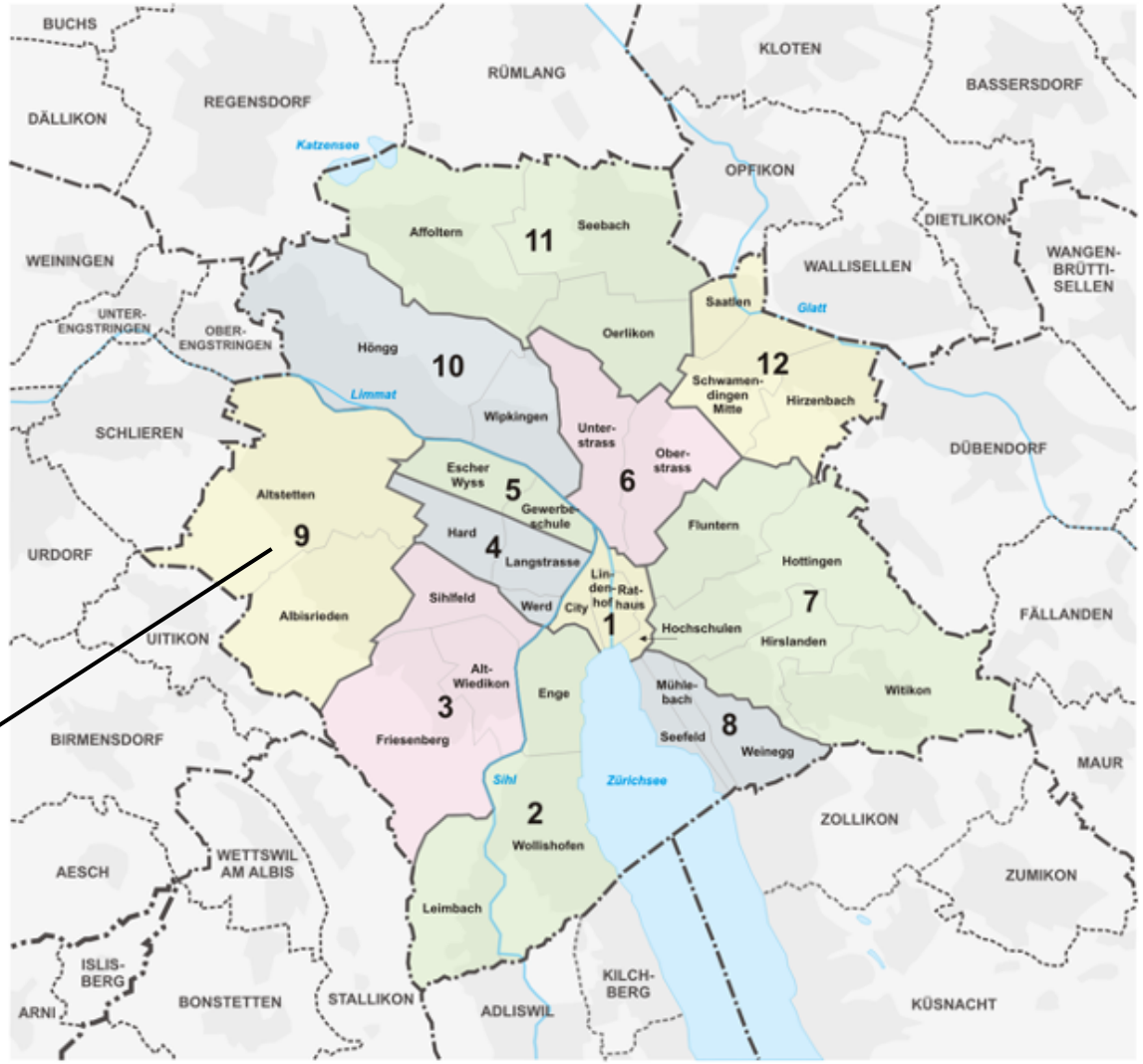
Place 3: 984

The place is important...

Place 3: 984



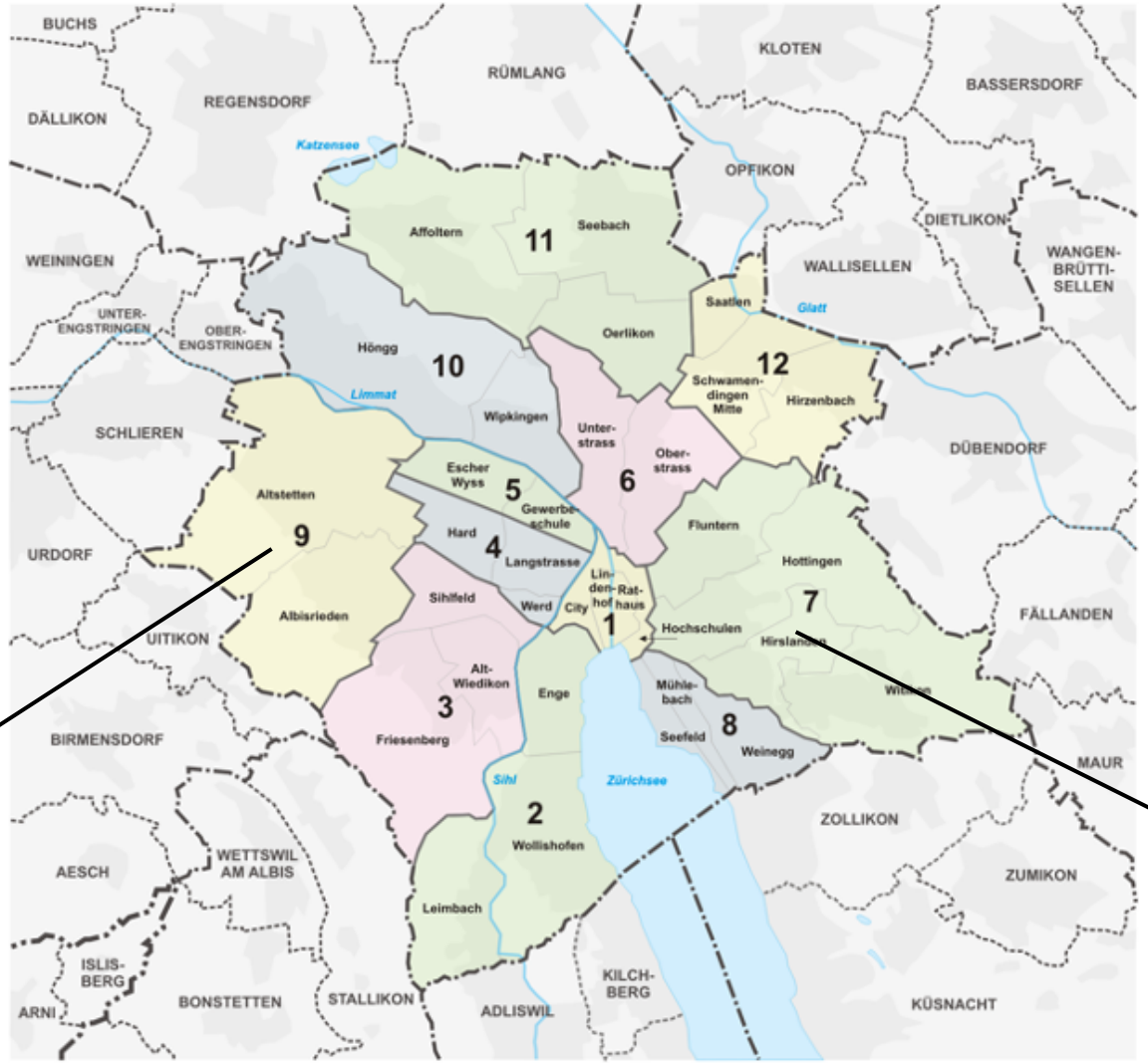
The place is important...



Place 3: 984

Place 2: 1087

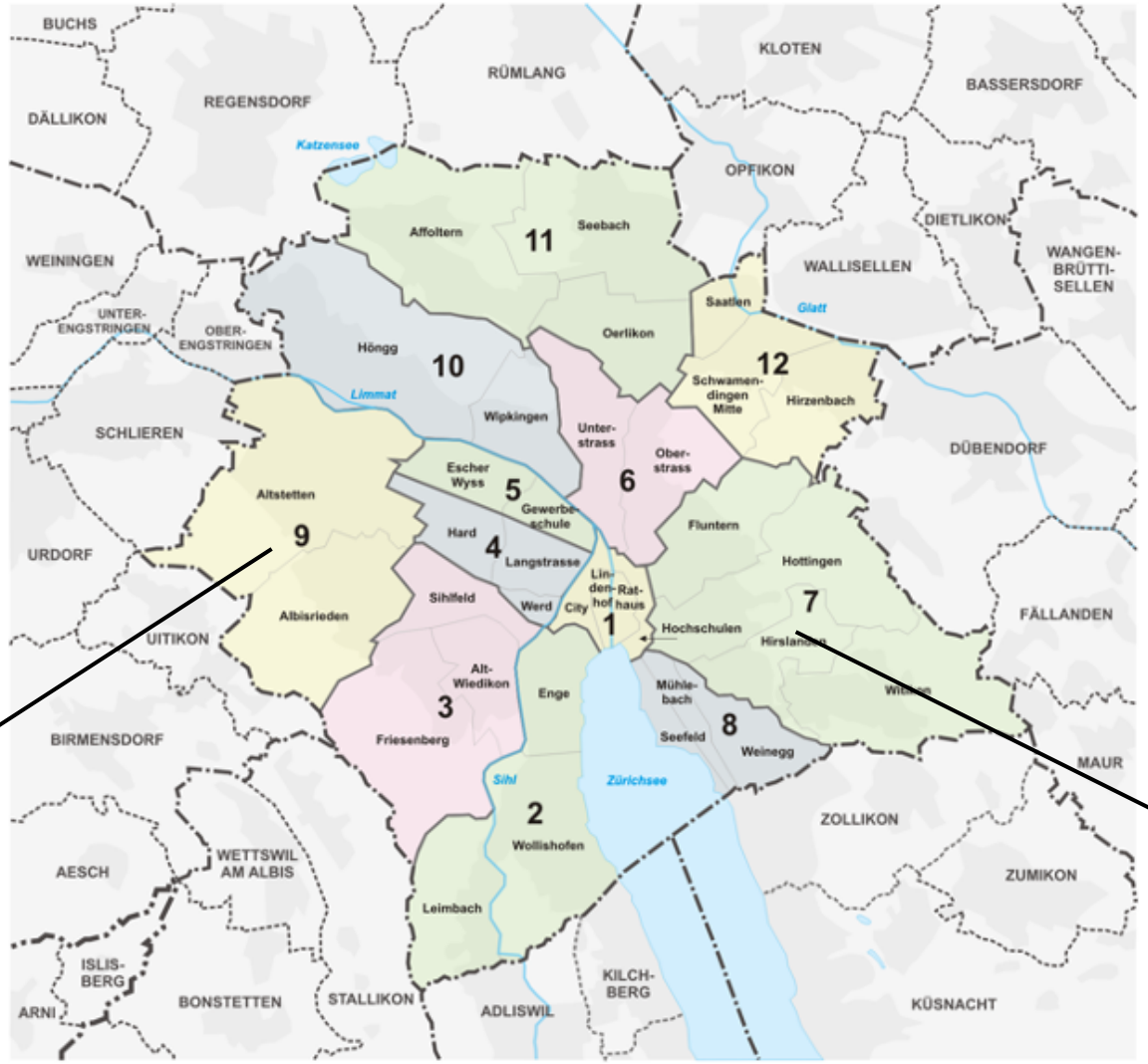
The place is important...



Place 3: 984

Place 2: 1087

The place is important...

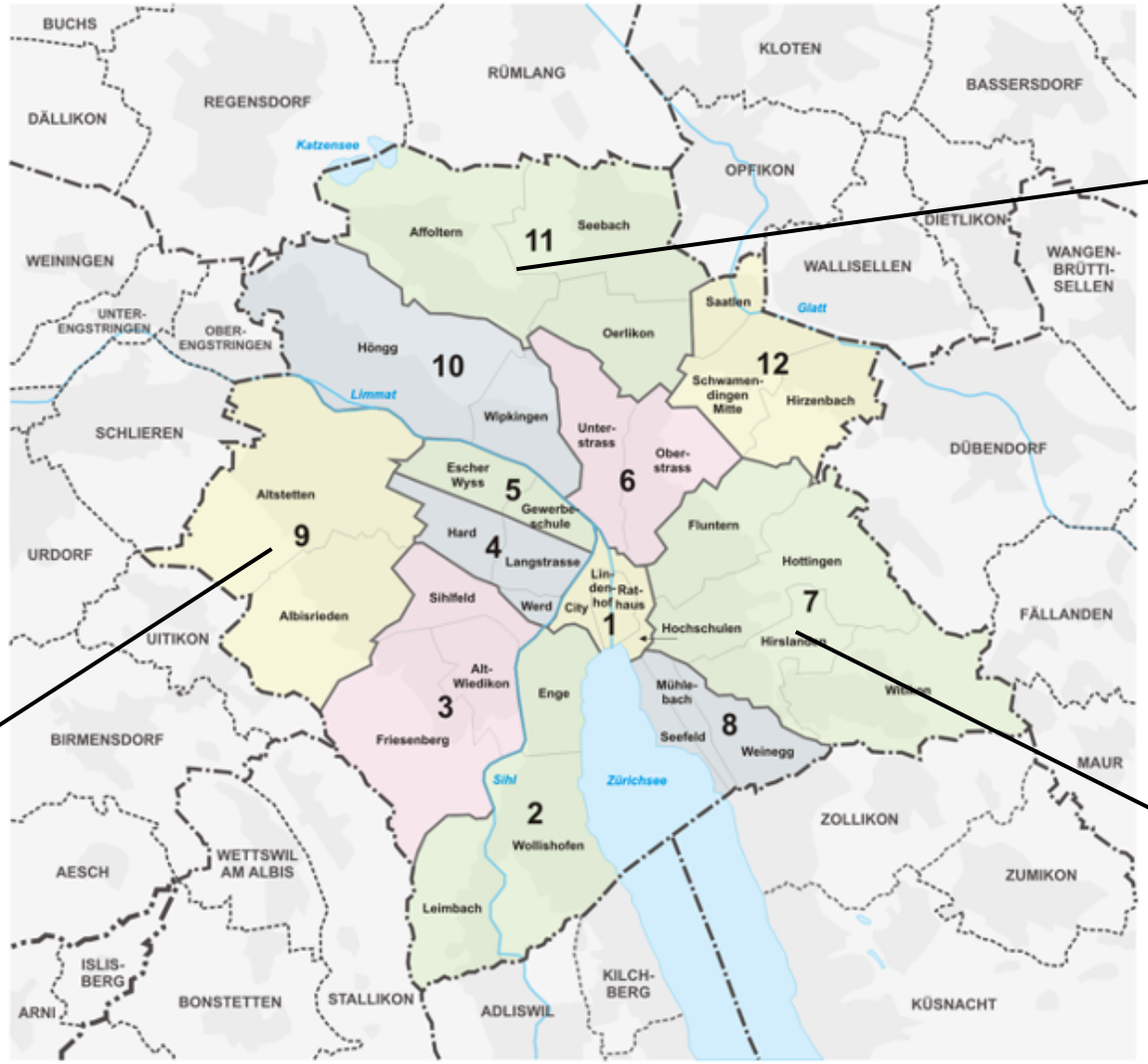


Place 1: 1352

Place 3: 984

Place 2: 1087

The place is important...

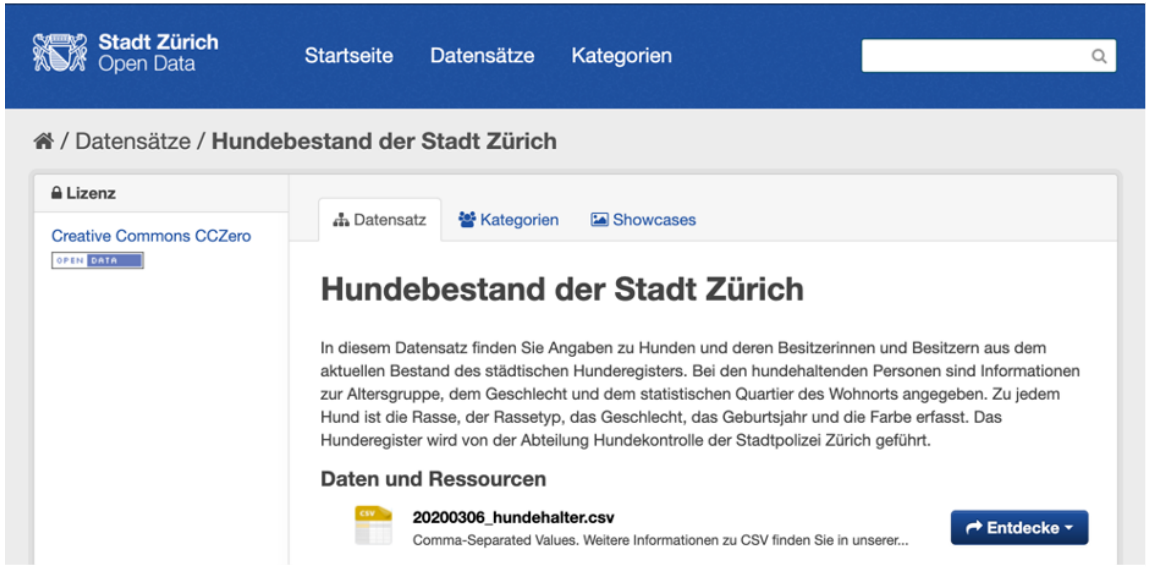


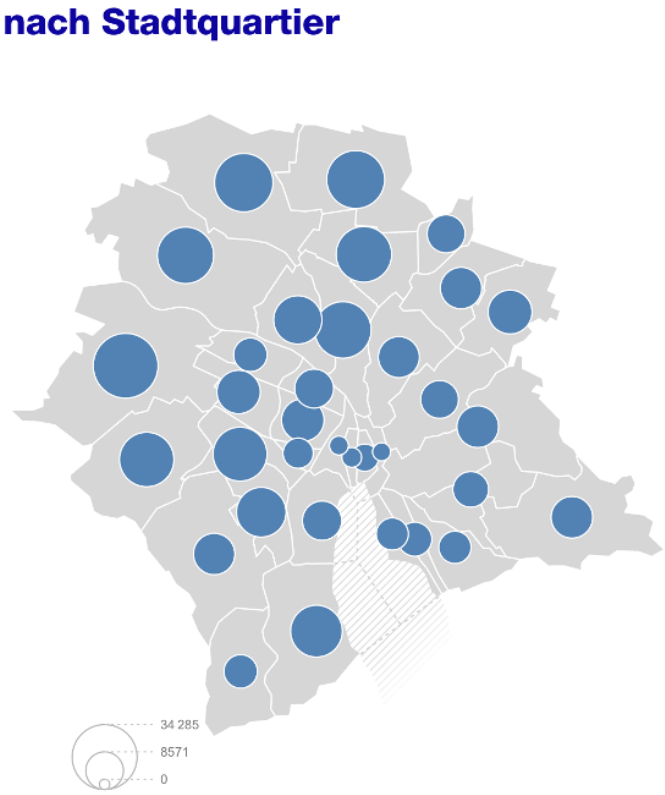
Place 1: 1352

Place 3: 984

Place 2: 1087

The place is important...



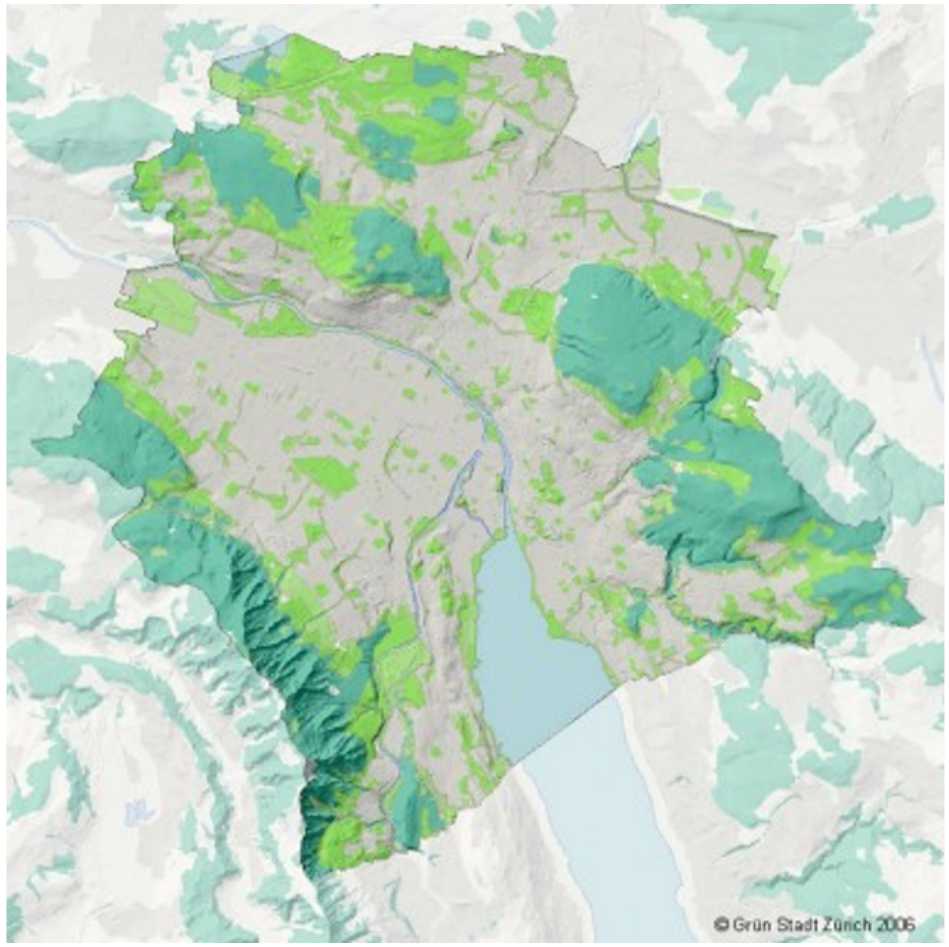


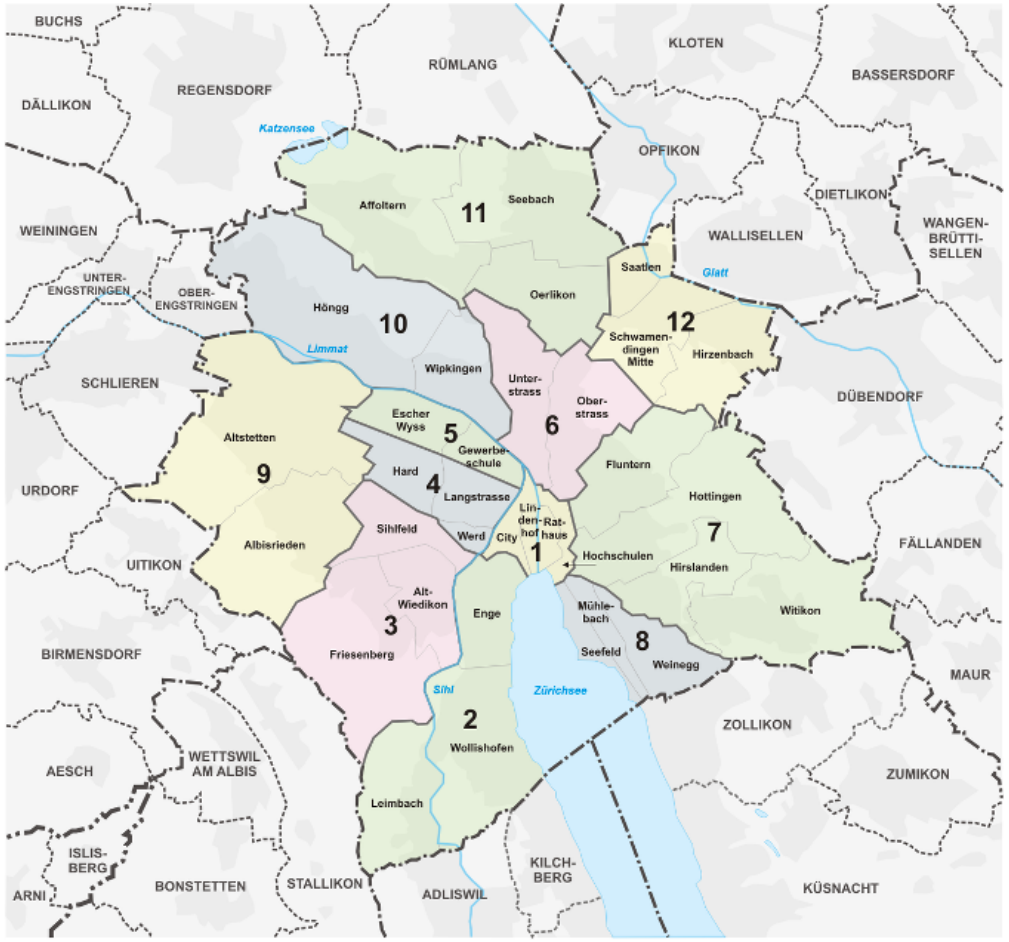
Pay particular attention to temporal differences:
Dog data: March 2020 / Population data: 1993-2020

Is the number of people important?

- Kreis 1; 5 831
- Kreis 2; 35 552
- Kreis 3; 50 756 => Many people, fewer dogs (697)
- Kreis 4; 29 034
- Kreis 5; 15 622
- Kreis 6; 35 317
- Kreis 7; 38 629
- Kreis 8; 17 456
- Kreis 9; 56 462
- Kreis 10; 41 044
- Kreis 11; 76 188
- Kreis 12; 32 845

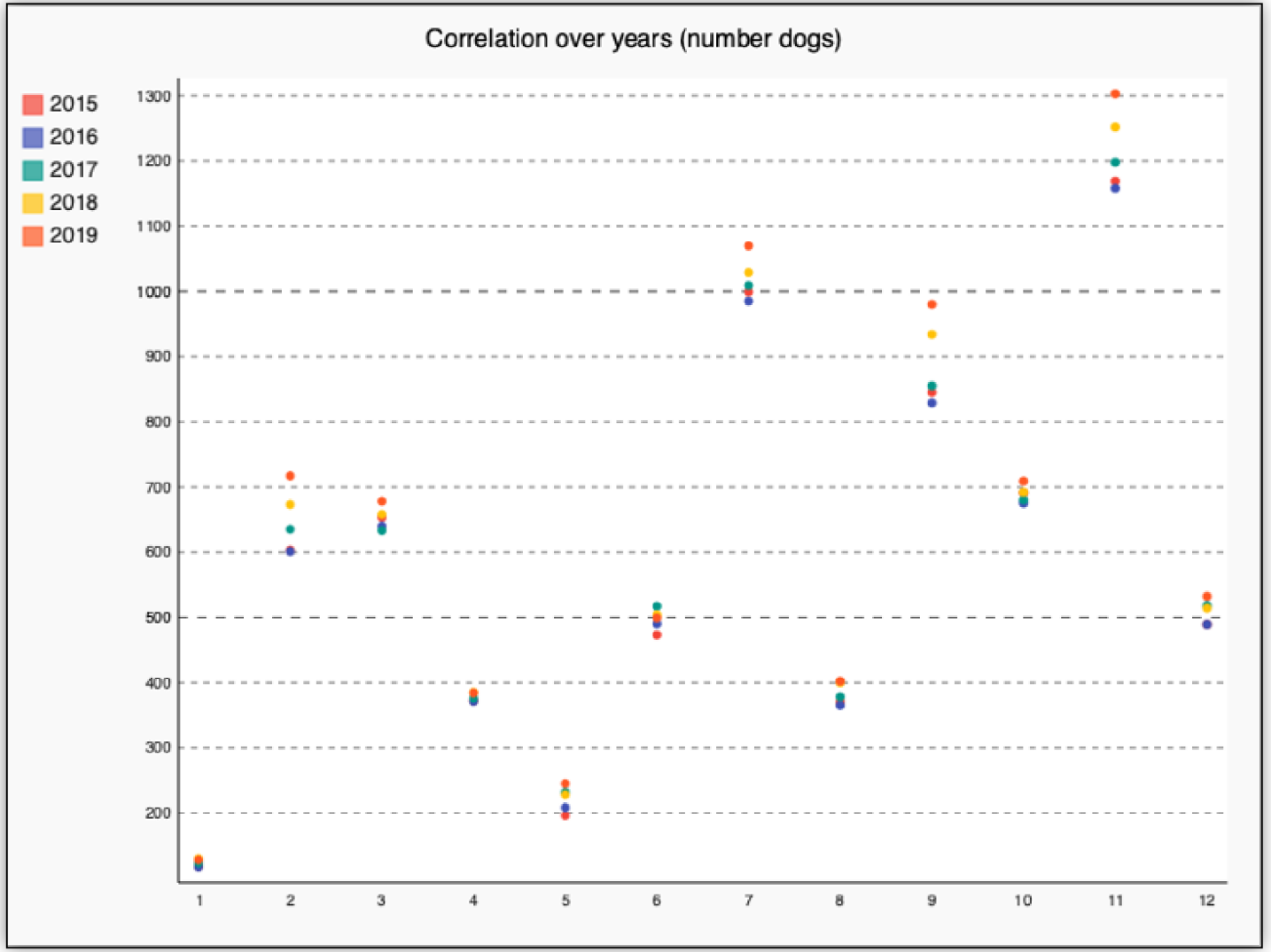
Are Green Spaces important?





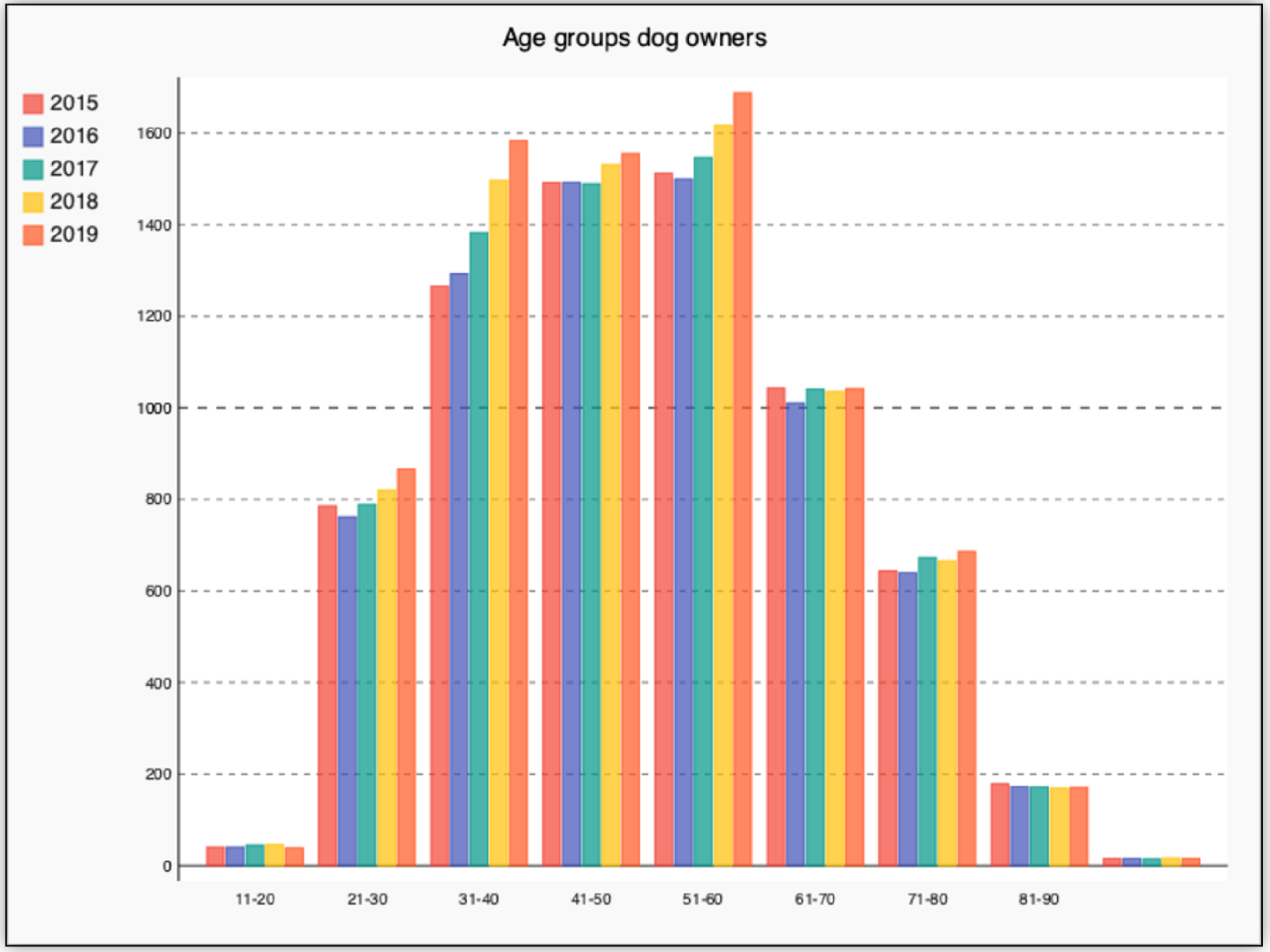
Calculate the correlation

See Step 2 in the documents



Age groups

The most pronounced group is 31-60



What next?

- Of course, many more methods could be used here:
  - K-Nearest Neighbor
  - Naive Bayes
  - Linear Regression
  - Multiple Regression
  - Logistic regression
  - Decision Trees
  - Neural Networks / Deep Learning
  - clustering
  - etc.

Conclusion: What do we recommend Dog-Science?

- We recommend Dog-Science to **apply for** a place **in `Kreis 11 near a green area`**.
- We recommend designing the product for 31-61 year olds.
- It may cost something "more" (see average salary in Zurich).
- It should not be too playful and should be easy to understand (interpretation of the age group).

Capstone Project

We build a dashboard together with Dash.

Capstone Project

Take a look at the following first:

https://tracking-dashboard-app.herokuapp.com/dashboard

Capstone Project

Our example is simpler, but combines dash (plotly) with our data skills and visualization ideas.

Capstone Project

Dash Hello World

Capstone Project

Dash Hello World

```python
# 13_hello_world_dash.py

import openpyxl
from dash import Dash, html

app = Dash(__name__)

app.layout = html.Div([
    html.Div(children='Hello World')
])

if __name__ == '__main__':
    app.run(debug=True)
```

Capstone Project

Connect with data

Capstone Project

Connect with data

```python
# 14_dash_with_data.py

# Import packages
from dash import Dash, html, dash_table
import pandas as pd

# Incorporate data
df = pd.read_csv('gapminder2007.csv')

# Initialize the app
app = Dash(__name__)

# App layout
app.layout = html.Div([
    html.Div(children='My First App with Data'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10)
])

# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```

Capstone Project

And a visualization

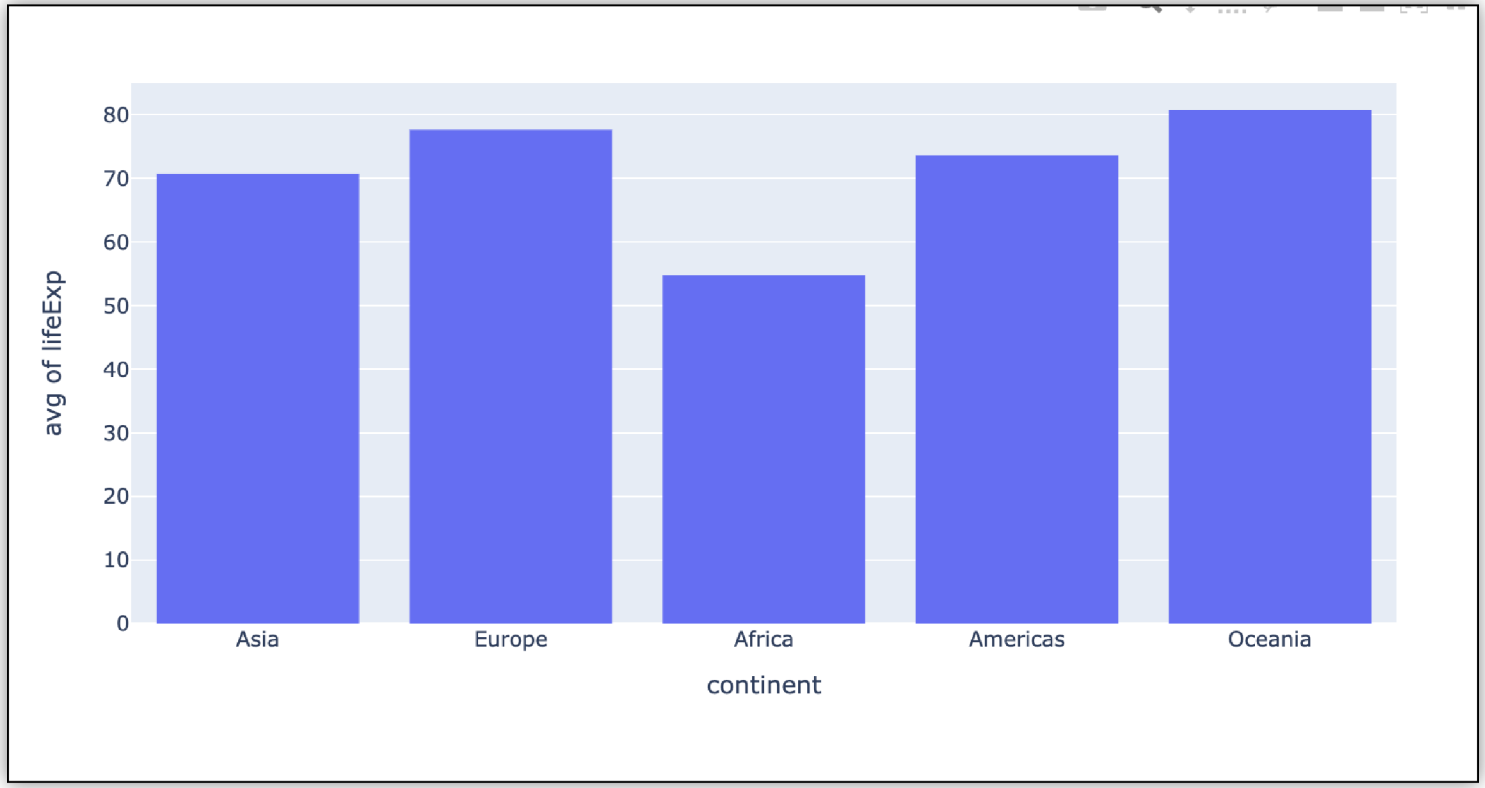
Capstone Project

And a visualization

```python
# 15_dash_with_data_and_vis.py

# Import packages
from dash import Dash, html, dash_table, dcc
import pandas as pd
import plotly.express as px

# Incorporate data
df = pd.read_csv('gapminder2007.csv')

# Initialize the app
app = Dash(__name__)

# App layout
app.layout = html.Div([
    html.Div(children='My First App with Data and a Graph'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10),
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp', histfunc='avg'))
])

# Run the app
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5005)
```

Capstone Project

And a visualization



Capstone Project

And Controls

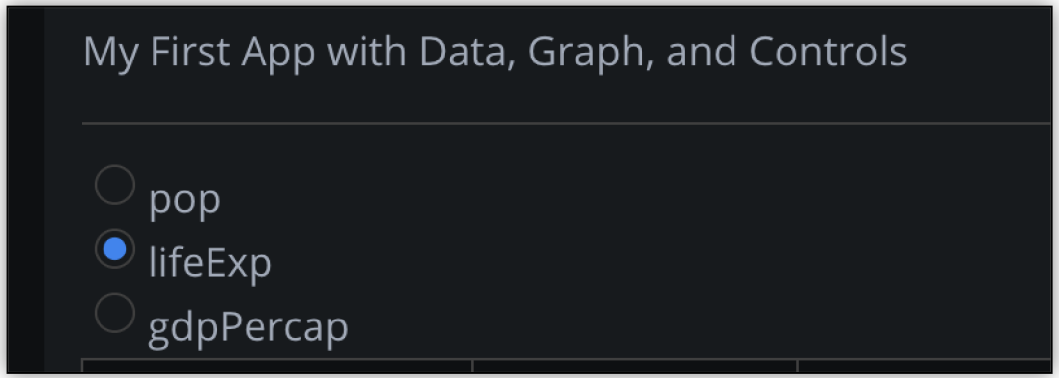
Capstone Project

And Controls

```
# 16_dash_with_data_and_vis_cont.py

# Add controls to build the interaction
@callback(
    Output(component_id='controls-and-graph', component_property='figure'),
    Input(component_id='controls-and-radio-item', component_property='value')
)
def update_graph(col_chosen):
    fig = px.histogram(df, x='continent', y=col_chosen, histfunc='avg')
    return fig
```

Capstone Project

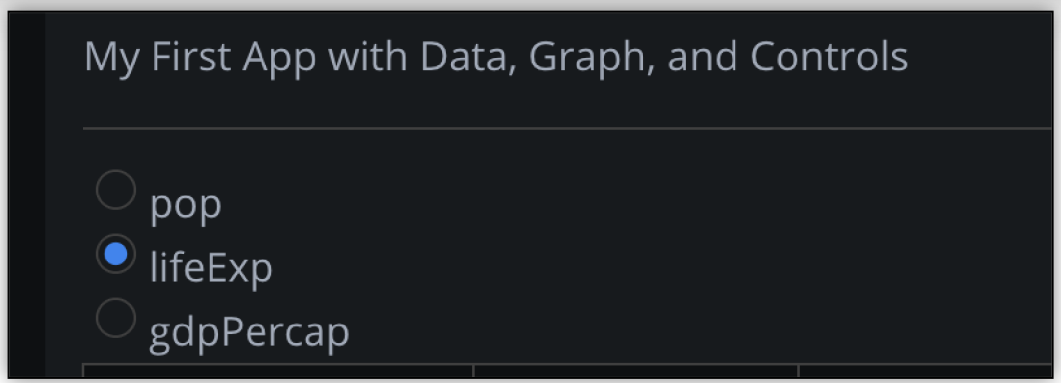
As well as the effective radio buttons



Capstone Project

As well as the effective radio buttons

```
# 16_dash_with_data_and_vis_cont.py

# App layout
app.layout = html.Div([
    html.Div(children='My First App with Data, Graph, and Controls'),
    html.Hr(),
    dcc.RadioItems(options=['pop', 'lifeExp', 'gdpPercap'], value='lifeExp', id='controls-and-radio-item'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=6),
    dcc.Graph(figure={}, id='controls-and-graph')
])
```



Capstone Project

Let's take a look at the final - externally styled variant in the code.

06-capstone.