Workshop “Introduction to Python”

organized by the
Cluster of Excellence “The Politics of Inequality” at the
University of Konstanz
in cooperation with the
Zeppelin University Friedrichshafen

July 8 and 12, 2021

Contents

1 Introduction, Warm Up, Set Up

- Python puzzles / recap
  - data types
  - control structures
  - classes and objects
  - modules
- Python runtime and development environments
  - Python interpreter
  - editors, IDEs
  - Jupyter notebooks, Anaconda
  - virtual environment, Docker

1.1 Python Puzzles / Recap

What will the Python3 interpreter return on the following statements…

1.1.1 Data Types

```
In [ ]: a = 3 # integer
        b = 2
        a * b
```

```
In [ ]: c = 2.0 # floating point number
        a * c
```

```
In [ ]: t = True # boolean value
        f = False
        t and f
```

```
In [ ]: t or f
```

```
In [ ]: s = 'foo' # string
        s + s
```

```
In [ ]: s[0]
```

```
In [ ]: l = [1, 2, 3] # list
        l[0]
```

```
In [ ]: l[3]
```

```
In [ ]: l[-1]
```

```
In [ ]: d = {'a': 1, 'b': 2, 'c': 3, 'b': 1.5} # dictionary
        d['b']
```

```
In [ ]: s = {'a', 'b', 'c', 'a'} # set
        s
```

```
In [ ]: t = (1, 2) # tuple
        t[0]
```

```
In [ ]: l[2] = 4
        l
```

```
In [ ]: t = (1, 2)
        t[1] = 3
```

Mutable and Immutable Data Types

- tuples are immutable, i.e. once created you cannot change the content
- lists, dictionaries, sets are mutable
- numbers and strings are also immutable
- immutable data types avoid programming errors and also allow for certain optimizations

```
In [ ]: s = 'foo'
        s[0] = 'F'
```

```
In [ ]: # but you can assign a new string to the variable `s`
        s = 'Foo'
        s
```

```
In [ ]: l = [1, 2, 3]
        l2 = l
        l2
```

```
In [ ]: l[2] = 4
        l2
```

1.1.2 Control Structures

Loops

```
In [ ]: l = [1, 2, 3]
        for i in l:
            print(i)
```

```
In [ ]: i = 1
        while i <= 3:
            print(i)
            i += 1
```

If-Else Conditions

```
In [ ]: for i in range(0, 5):
            if i % 2 == 0:
                print("Even:", i)
            else:
                print("Odd:", i)
```

```
In [ ]: i = 1
        while True:
            print(i)
            if i == 3:
                break
            i += 1
```

Functions

Functions are...

- code blocks only executed when called
- reusable (can be called repeatedly from various places in the code)
- the primary method to organize code and make it readable and understandable

```
In [ ]: def fun(n): # one required argument
            for i in range(0, n):
                print("You called me?")
        fun(2)
```

```
In [ ]: def fun(x='You'): # one optional argument
            """Ask whether X called me"""
            print(x, "called me?")

        fun()
        fun('Who')
        fun(x='They')
```

```
In [ ]: def fun(x='You'):
            return "%s called me?" % x

        question = fun('Who')
        question
```

1.1.3 Classes and Objects

The object-oriented programming paradigm combines data and code in "objects". Every "object" is an instance of a "class". The "class" defines

- the data types and possible values an object of the class holds
- "methods" - functions to read, write or interact with data values hold by the object

Object Methods

Variables of built-in data types are all objects of built-in classes and provide multiple methods…

```
In [ ]: s.capitalize() # call a method of a string object
```

Tip: many Python editors let you show a list of available methods for a given object variable.

In the Jupyter notebook editor: enter `s.` and press `<tab>` to get a list of methods of `str` objects.

```
In [ ]: #s.
```

```
In [ ]: type(s)
```

```
In [ ]: help(str)
```

```
In [ ]: help(str.endswith)
```

```
In [ ]: !pydoc str.endswith  # `!` runs another command (not the Python interpreter)
```

What could be the methods provided by the `list` built-in class? Think about it before calling `help(list)`!

Defining Classes

```
In [ ]: class Sentiment:

            values = {'sad', 'neutral', 'happy'}

            def __init__(self, value='neutral'):
                if value not in Sentiment.values:
                    raise ValueError("Only the following values are supported: %s"
                                     % Sentiment.values)
                self.value = value

            def get(self):
                return self.happy_or_not

            def __repr__(self):
                return self.value

            @staticmethod
            def guess(text):
                if 'happy' in text or 'excited' in text:
                    return Sentiment('happy')
                if 'sad' in text or 'angry' in text:
                    return Sentiment('sad')
                return Sentiment('neutral')


        im_feeling = Sentiment.guess("I'm really happy!")

        print(im_feeling)
```

```
In [ ]: im_feeling = Sentiment('sick')
```

1.1.4 Modules

Modules make Python code reusable.

Create a Python Module

Copy the definition of the class “Sentiment” into a file sentiment.py in the folder `scripts`. Now you can load the class by…

```
In [ ]: from scripts.sentiment import Sentiment

        Sentiment()
```

The Python Standard Library

The Python Standard Library includes many modules to handle file formats, process texts, use the internet, etc., etc. Just import one of the modules or functions or classes defined there:

```
In [ ]: import time

        time.asctime()
```

```
In [ ]: from time import asctime, sleep

        print(asctime())
        sleep(3)
        print(asctime())
```

Third-Party Modules

To install a package from the Python Package Index, run `pip install <package>`…

```
In [ ]: !pip install matplotlib
```

… but before run `pip list` or `pip show matplotlib` (or just try `import matplotlib`) to figure out whether it is already installed.

A good and common practice is to list all modules required by a project in a file `requirements.txt`. The entire list of requirements can then be installed by `pip install -r requirements.txt`.

1.2 Python Runtime and Development Environments

1.2.1 The Python Interpreter

- installed from python.org
- on Linux: already installed or installable as package of the Linux Distribution (Debina, Ubuntu, Red Hat, SuSE, etc.)
- otherwise: it's recommended to rely on a distribution which bundles the Python interpreter with common Python modules and tools - esp. Anaconda, a distribution of Python and R for scientific computing

1.2.2 Jupyter Notebooks

The Jupyter notebook is an enviroment to interactively create a "notebook", a JSON-encoded document containing a list of input/output pairs (code, text using Markdown markup, images/plots). Notebooks are served by the notebook server and viewed/edited in the browser or can be converted into various document formats.

1.2.3 Editor and IDE

A good editor or an integrated development environment (IDE) will speed up coding by providing autocompletion, syntax highlighting and syntax checking. If your code gets bigger, an IDE supports the development by automated builds and deployments of the code, a runtime for tests and a visual debugger to locate errors ("bugs") in your code.

Unfortunately, there are many good IDEs available for Python, to list just a few:

- PyDev
- Visual Studio Code
- PyCharm (commercial)

1.2.4 Virtual Environment and Docker

Why you need encapsulated environments to run applications or projects? The documentation of the Python virtual environements explains…

> Python applications will often use packages and modules that don't come as part of the standard library. Applications will sometimes need a specific version of a library, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library's interface.
>
> This means it may not be possible for one Python installation to meet the requirements of every application. If application A needs version 1.0 of a particular module but application B needs version 2.0, then the requirements are in conflict and installing either version 1.0 or 2.0 will leave one application unable to run.

1. create a virtual environment in current director in the subfolder `.venv/`

   ```
   virtualenv .venv
   ```

2. activate the environment

```
source .venv/bin/activate
```

3. install packages (placed below `./.venv/`)

   ```
   pip install ...
   ```

4. run Python…
5. deactivate the environment

   ```
   deactivate
   ```

If more than Python modules are project-specific: Docker allows to bundle a Python interpreter (eg. an older version), specific modules and additional software, pack it as runtime image and run it in a “container” without the need to install anything on the host system.

2 Working with Structured Data

- read data from local files
- read CSV and JSON
- first steps data analysis with data frames and the pandas library
- basic plotting of data

2.1 Example: "Tree Cadastre of the City of Konstanz"

First, get the tree cadastre data from the open data portal of the city of Konstanz. Save it on the file path shown below. The CSV file is then loaded into a pandas "DataFrame":

```
In [1]: import pandas as pd

        tree_cadastre_file = './data/KN_Baumkataster_2020.csv'
        df = pd.read_csv(tree_cadastre_file)
        df.shape  # table size (rows, columns)
```

```
Out[1]: (15711, 13)
```

Note: Pandas could read the CSV directly from the WWW if a URL is passed. With internet access and supposed the download URL is still valid, the data frame is also loaded by

```
df = pd.read_csv('https://opendata.arcgis.com/datasets/c160f0a79a584ddf80cc65477fe58f4e_0.csv')
```

Let's now have a first and quick look into the data using pandas methods:

```
In [2]: df.head() # first lines of the table
```

```
Out[2]:          X          Y  OBJECTID  baumId  baumNr  baumart  hoeheM  \
        0  9.159063  47.739307         1       2       1       52    12.0
        1  9.158918  47.739471         2       4       4      182    11.0
        2  9.159193  47.739428         3       5       3       52    11.0
        3  9.158987  47.739541         4       6       5       37    14.0
        4  9.159219  47.739676         5       9       8      284    22.0

           kronendurchmesserM  stammumfangCM                  location  \
        0                   6           72.0  Bubenbad Dingelsdorf (754)
        1                  12          169.0  Bubenbad Dingelsdorf (754)
        2                   7           74.0  Bubenbad Dingelsdorf (754)
        3                   7          135.0  Bubenbad Dingelsdorf (754)
        4                  20          380.0  Bubenbad Dingelsdorf (754)

                        Name_dt           Name_lat AGOL_Name
        0     Erle, Schwarz-Erle     Alnus glutinosa     Alnus
        1      Nussbaum, Walnuss       Juglans regia   Juglans
        2     Erle, Schwarz-Erle     Alnus glutinosa     Alnus
```

```
3       Ahorn, Berg-Ahorn  Acer pseudoplatanus     Acer
4  Pappel, Schwarz-Pappel        Populus nigra   Populus
```

```
In [3]: df.describe() # descriptive statistics (numerical columns)
```

```
Out[3]:                   X             Y      OBJECTID        baumId        baumNr  \
        count  15711.000000  15711.000000  15711.000000  15711.000000  15711.000000
        mean       9.169897     47.681721   7856.000000  13361.111832     57.941315
        std        0.022084      0.023527   4535.519375   9558.292963    109.965696
        min        9.106630     47.653444      1.000000      2.000000      0.000000
        25%        9.153555     47.666961   3928.500000   5844.500000      5.000000
        50%        9.170588     47.674747   7856.000000  12181.000000     20.000000
        75%        9.180610     47.683773  11783.500000  17923.500000     58.000000
        max        9.217534     47.748520  15711.000000  39080.000000    805.000000

                    baumart        hoeheM  kronendurchmesserM  stammumfangCM
        count  15711.000000  15706.000000        15711.000000   15704.000000
        mean     307.457959     10.688718            6.124944     113.009488
        std      206.677390      6.416883            3.883879      83.834009
        min        1.000000      1.000000            0.000000       0.000000
        25%       77.000000      5.000000            3.000000      50.000000
        50%      322.000000      9.000000            6.000000      93.000000
        75%      501.000000     15.000000            8.000000     157.000000
        max      637.000000     40.000000           30.000000     900.000000
```

```
In [4]: df.nunique() # number of unique values in each column
```

```
Out[4]: X                     15705
        Y                     15705
        OBJECTID              15711
        baumId                15711
        baumNr                  801
        baumart                 296
        hoeheM                   36
        kronendurchmesserM       26
        stammumfangCM           464
        location                775
        Name_dt                 294
        Name_lat                296
        AGOL_Name                35
        dtype: int64
```

… and we identify the following columns (cf. the provided tree cadastre metadata):

- the pandas row index
- "X" and "Y": geographic coordinates (longitude and latitude)
- "OBJECTID", "baumid", "baumNr": three different tree IDs
- "baumart": a nummeric species ID
- "hoeheM": the tree height (m)
- "kronendurchmesserM": treetop diameter (m)
- "stammumfangCM": trunk perimeter (cm)
- "location": coarse location of the tree (street name)
- "Name_dt": German tree name

- “Name_lat”: Latin tree name
- “AGOL_Name”: vendor-specific name (“AGOL” = “ArcGIS Online”)

We clean up the data a little bit: - translate the German column names - drop the columns not used later on - use the column “OBJECTID” as row index

```
In [5]: df.rename(columns={'hoeheM': 'height (m)',
                           'kronendurchmesserM': 'treetop diameter (m)',
                           'stammumfangCM': 'trunk perimeter (cm)'},
                  inplace=True)
        df.drop(columns=['baumId', 'baumNr', 'baumart', 'AGOL_Name'], inplace=True)
        df.set_index('OBJECTID', inplace=True)
        df.head()
```

```
Out[5]:                  X          Y  height (m)  treetop diameter (m)  \
        OBJECTID
        1         9.159063  47.739307        12.0                     6
        2         9.158918  47.739471        11.0                    12
        3         9.159193  47.739428        11.0                     7
        4         9.158987  47.739541        14.0                     7
        5         9.159219  47.739676        22.0                    20

                  trunk perimeter (cm)                    location  \
        OBJECTID
        1                         72.0  Bubenbad Dingelsdorf (754)
        2                        169.0  Bubenbad Dingelsdorf (754)
        3                         74.0  Bubenbad Dingelsdorf (754)
        4                        135.0  Bubenbad Dingelsdorf (754)
        5                        380.0  Bubenbad Dingelsdorf (754)

                                 Name_dt             Name_lat
        OBJECTID
        1             Erle, Schwarz-Erle      Alnus glutinosa
        2              Nussbaum, Walnuss         Juglans regia
        3             Erle, Schwarz-Erle      Alnus glutinosa
        4              Ahorn, Berg-Ahorn  Acer pseudoplatanus
        5         Pappel, Schwarz-Pappel         Populus nigra
```

2.2 Count Items

```
In [6]: # count tree names and show the N most frequent tree names
        N = 20
        top_trees = df['Name_lat'].value_counts().head(N).to_frame()
        top_trees
```

```
Out[6]:                                        Name_lat
        Platanus x acerifolia                       887
        Betula pendula                              809
        Quercus robur                               667
        Fraxinus excelsior                          614
        Tilia cordata                               605
```

```
Malus domestica                            539
Salix alba                                 536
Acer platanoides                           523
Acer pseudoplatanus                        517
Pyrus communis                             513
Carpinus betulus                           503
Acer campestre                             428
Juglans regia                              397
Aesculus hippocastanum                     372
Fagus sylvatica                            293
Fraxinus excelsior 'Westhof's Glorie'      261
Tilia platyphyllos                         252
Prunus avium                               250
Tilia cordata 'Greenspire'                 244
Gleditsia triacanthos 'Inermis'            234
```

```
In [7]: # also show the top N German names
        df['Name_dt'].value_counts().head(20).to_frame()
```

```
Out[7]:                                  Name_dt
        Platane                              952
        Birke, Sand-Birke                    809
        Eiche, Stiel-Eiche, Sommer-Eiche     667
        Esche, Esche gemeine                 614
        Linde, Winter-Linde                  605
        Kultur-Apfel                         539
        Weide, Silber-Weide                  536
        Ahorn, Spitz-Ahorn                   523
        Ahorn, Berg-Ahorn                    517
        Birne, Holz-Birne                    513
        Weißbuche, Hainbuche                 503
        Ahorn, Feld-Ahorn                    428
        Nussbaum, Walnuss                    397
        Rosskastanie                         372
        Buche, Rotbuche                      293
        Straßen-Esche                        261
        Linde, Sommer-Linde                  252
        Kirsche, Vogel-Kirsche               250
        Linde "Greespire"                    244
        Dornenlose Gleditschie               234
```

Obviously, German names are less specific (there are more items of “Platane” than “Platanus x acerifolia”). To avoid inconsistencies we’ll use the Latin names in the next steps. Because not everybody knows Latin well enough or studied botanology, let’s prepare a translation table to see the Latin and German names site by site. We will later look how we could get the tree names in other languages as well.

```
In [8]: tree_name_translation = df.loc[df['Name_lat'].isin(top_trees.index),
                                       ['Name_lat', 'Name_dt']]
        tree_name_translation['count'] = 1
        tree_name_translation.groupby(['Name_lat', 'Name_dt']).sum() \
            .sort_values('count', ascending=False)
```

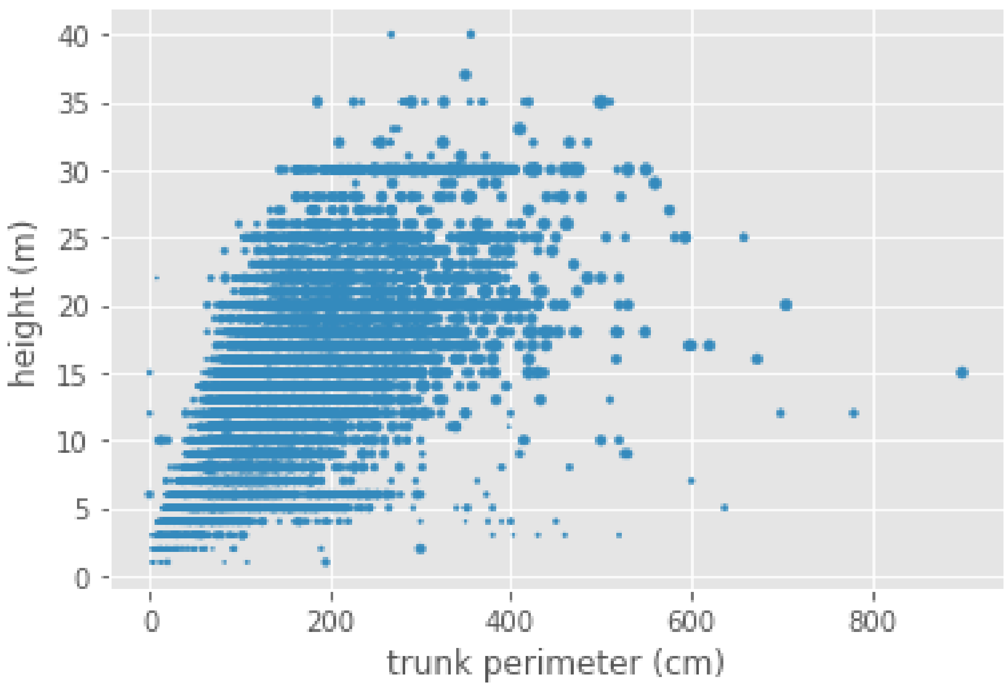
```
Out[8]:                                                                  count
        Name_lat                             Name_dt
        Platanus x acerifolia                Platane                       887
        Betula pendula                       Birke, Sand-Birke             809
        Quercus robur                        Eiche, Stiel-Eiche, Sommer-Eiche    667
        Fraxinus excelsior                   Esche, Esche gemeine          614
        Tilia cordata                        Linde, Winter-Linde           605
        Malus domestica                      Kultur-Apfel                  539
        Salix alba                           Weide, Silber-Weide           536
        Acer platanoides                     Ahorn, Spitz-Ahorn            523
        Acer pseudoplatanus                  Ahorn, Berg-Ahorn             517
        Pyrus communis                       Birne, Holz-Birne             513
        Carpinus betulus                     Weißbuche, Hainbuche          503
        Acer campestre                       Ahorn, Feld-Ahorn             428
        Juglans regia                        Nussbaum, Walnuss             397
        Aesculus hippocastanum               Rosskastanie                  372
        Fagus sylvatica                      Buche, Rotbuche               293
        Fraxinus excelsior 'Westhof's Glorie' Straßen-Esche                261
        Tilia platyphyllos                   Linde, Sommer-Linde           252
        Prunus avium                         Kirsche, Vogel-Kirsche        250
        Tilia cordata 'Greenspire'           Linde "Greespire"             244
        Gleditsia triacanthos 'Inermis'      Dornenlose Gleditschie        234
```

2.3 Plotting

We start with a first trivial scatter plot of the 3 metric values using the plot method of the DataFrame. We choose the matplotlib's style "ggplot" which mimics the look of the plots produced by a popular plotting package for R. There are many more styles available.

```
In [9]: import matplotlib
        import matplotlib.pyplot as plt
        plt.style.use('ggplot')

        df.plot(kind='scatter', x='trunk perimeter (cm)',
                y='height (m)', s='treetop diameter (m)')
```

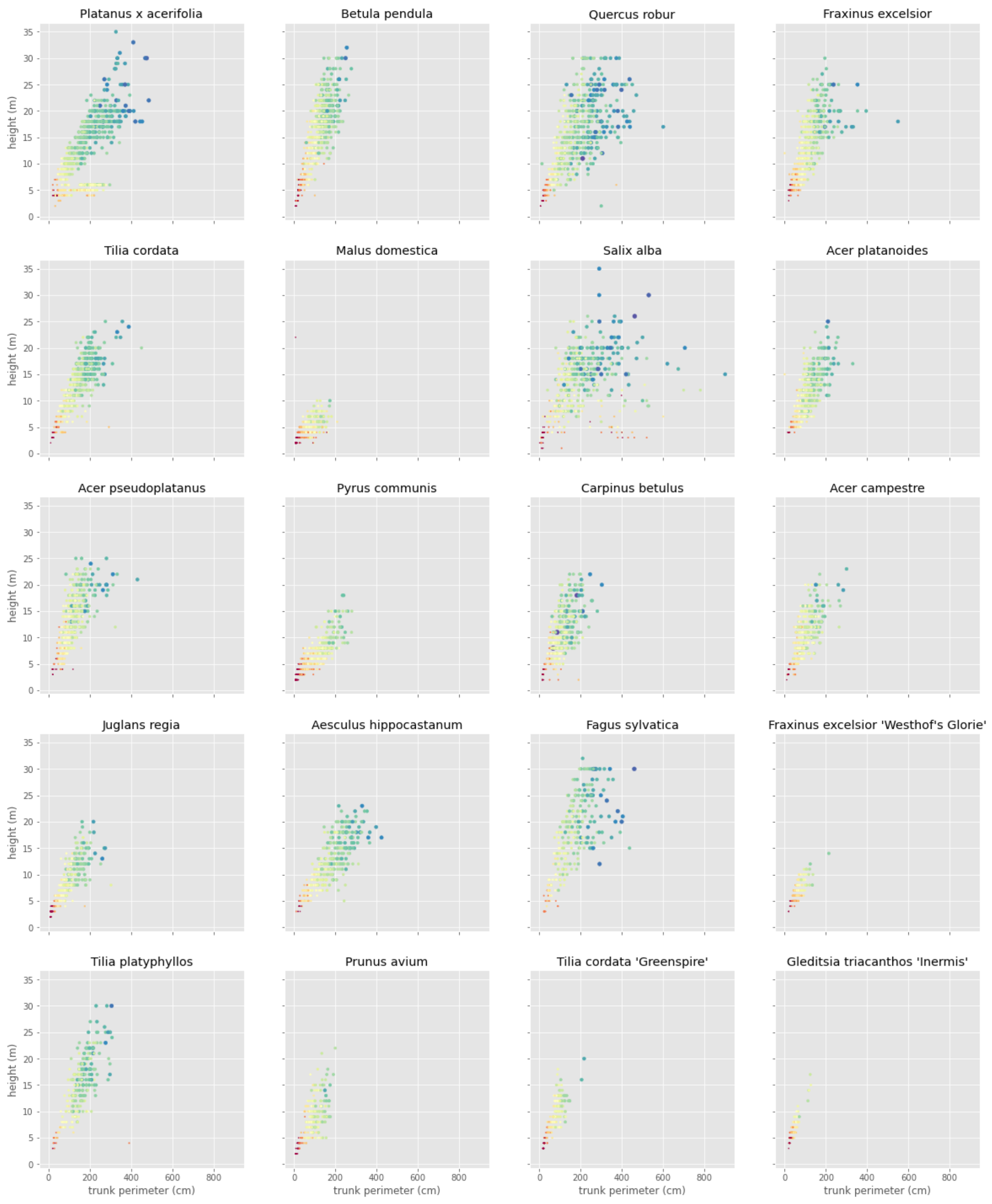
```
Out[9]: <AxesSubplot:xlabel='trunk perimeter (cm)', ylabel='height (m)'>
```



Insights from the first plot: - data gathering: heights above 25m are rather estimates - some noise, eg. hight trees with thin truncs - tree height and trunk perimeter correlate

To take into account the tree types, we'll focus on the top-20 most frequent names only and plot them on a 4x5 matrix:

```
In [10]: fig, axes = plt.subplots(nrows=5, ncols=4, sharex=True, sharey=True,
                                  squeeze=False, figsize=[20,25])


         n = 0
         for tree in top_trees.index.to_list():
             plot = df[df['Name_lat']==tree].plot(
                 kind='scatter',
                 ax=axes[int(n/4),n%4],
                 title=tree,
                 x='trunk perimeter (cm)',
                 y='height (m)',
                 s='treetop diameter (m)', # show by point size
                 c='treetop diameter (m)', # also indicated by color
                 colormap='Spectral',
                 norm=matplotlib.colors.LogNorm(vmin=1, vmax=25),
                 colorbar=None)
             n += 1
         plt.savefig('figures/trees_size_by_species.svg')
```



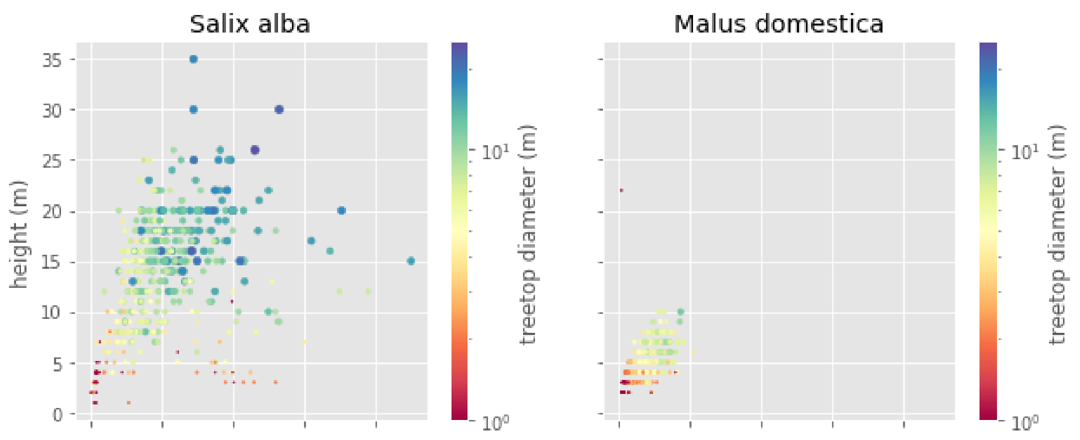
Notes about choosing the colormap for the treetop diameter: - the point size is hard to catch, while color is easier to discriminate (if not colorblind) - a spectral color map represents a continuous scale and allows for maximum discrimination - the range 1m - 25m (few trees reach 30m) is mapped on a logarithmic scale to make the smaller diameters (60% are 6m or smaller) look more different for small trees

See below the plot of willows and apple trees side by side. Try to change the color normalization!

```
In [11]: # distribution of treetop diameters
         df['treetop diameter (m)'].describe(percentiles=[i/20 for i in range(1, 20)])
```

```
Out[11]: count    15711.000000
         mean         6.124944
         std          3.883879
         min          0.000000
         5%           1.000000
         10%          1.000000
         15%          2.000000
         20%          2.000000
         25%          3.000000
         30%          4.000000
         35%          4.000000
         40%          5.000000
         45%          5.000000
         50%          6.000000
         55%          6.000000
         60%          6.000000
         65%          7.000000
         70%          8.000000
         75%          8.000000
         80%          9.000000
         85%         10.000000
         90%         12.000000
         95%         13.000000
         max         30.000000
         Name: treetop diameter (m), dtype: float64
```

```
In [12]: fig, axes = plt.subplots(nrows=1, ncols=2, sharex=True, sharey=True,
                                  squeeze=False, figsize=[10,4])


         n = 0
         for tree in ['Salix alba', 'Malus domestica']:
             df[df['Name_lat']==tree].plot(
                 kind='scatter',
                 ax=axes[0,n],
                 title=tree,
                 x='trunk perimeter (cm)',
                 y='height (m)',
                 s='treetop diameter (m)',
                 c='treetop diameter (m)',
                 colormap='Spectral',
                 norm=matplotlib.colors.LogNorm(vmin=1, vmax=25),
                 #norm=matplotlib.colors.Normalize(vmin=1, vmax=25),
                 colorbar=True)
             n += 1
```



2.4 Processing JSON

JSON is a standardized and common data format to store and interchange data independent from any programming language. JSON data types are numbers, Unicode strings, boolean values, the `null` value (`None`), arrays (Python lists) and objects (Python dictionaries). The JSON data types and the JSON syntax are similar to Python. But there are subtle differences and we use the json module of the Python standard libary to read or write JSON data:

```
In [13]: import json

         data = [{"key1": "value1", "key2": 2, 'key3': [1, 2, 3]}, True, False, None, 17, 1.123]
         json_data = json.dumps(data)
         json_data

Out[13]: '[{"key1": "value1", "key2": 2, "key3": [1, 2, 3]}, true, false, null, 17, 1.123]'

In [14]: json.loads(json_data)

Out[14]: [{'key1': 'value1', 'key2': 2, 'key3': [1, 2, 3]},
          True,
          False,
          None,
          17,
          1.123]

In [15]: # load translations of tree names from a JSON file
         tree_translations = json.load(open('data/trees-wikispecies.json'))

In [16]: list(tree_translations.keys())[:10]

Out[16]: ['Platanus x acerifolia',
          'Platanus × hispanica',
          'Betula pendula',
          'Quercus robur',
          'Fraxinus excelsior',
          'Tilia cordata',
          'Malus domestica',
          'Salix alba',
          'Acer platanoides',
          'Acer pseudoplatanus']
```

2.4.1 Remark: Get Translations from Wikispecies

The translations of the tree names were obtained from the Wikispecies project via the Mediawiki API. We will later learn how to use an API (Application Programming Interface) and how to send requests over the internet. But here very short

```
import json
import requests

query_params = {
    'action': 'query',
    'format': 'json',
    'prop': 'iwlinks|langlinks|description',
    'lllimit': 200,
    'llprop': 'url|langname'
}

trees_wikispecies = {}

for tree in top_trees.index.to_list():
    if tree in trees_wikispecies:
        continue
    query_params['titles'] = tree.replace(' ', '_')
    response = requests.get('https://species.wikimedia.org/w/api.php',
                            params=query_params)
    trees_wikispecies[tree] = json.loads(response.text)

with open('trees-wikispecies.json', 'w') as fp:
    json.dump(trees_wikispecies, fp)
```

The script trees_wikispecies.py was used to create the data file

Because the data was queried from Wikispecies, the values per tree represent response to a query and we need to navigate into the result object to get the translations.

```
In [17]: tree_translations['Gleditsia triacanthos']
```

```
Out[17]: {'batchcomplete': '',
          'query': {'normalized': [{'from': 'Gleditsia_triacanthos',
             'to': 'Gleditsia triacanthos'}],
           'pages': {'124231': {'pageid': 124231,
             'ns': 0,
             'title': 'Gleditsia triacanthos',
             'iwlinks': [{'prefix': 'commons', '*': ''},
              {'prefix': 'commons', '*': 'Category:Gleditsia_triacanthos'},
              {'prefix': 'en', '*': 'International_Plant_Names_Index'},
  {'prefix': 'en', '*': 'Royal_Botanic_Gardens,_Kew'}],
 'langlinks': [{'lang': 'ar',
   'url': 'https://ar.wikipedia.org/wiki/%D8%BA%D9%84%D8%A7%D8%AF%D9%8A%D8%B4%D9%8A%D8%A9_%D8%AB%D9%84%D8%A7
   'langname': 'Arabic',
   '*': □□□□□□□□□□' □□□□□□□ {'□□□□□□□□□,
  {'lang': 'az',
   'url': 'https://az.wikipedia.org/wiki/%C3%9C%C3%A7tikan_%C5%9Feytana%C4%9Fac%C4%B1',
   'langname': 'Azerbaijani',
   '*': 'Üçtikan şeytanağacı'},
  {'lang': 'ca',
   'url': 'https://ca.wikipedia.org/wiki/Ac%C3%A0cia_de_tres_punxes',
   'langname': 'Catalan',
   '*': 'Acàcia de tres punxes'},
  {'lang': 'ceb',
   'url': 'https://ceb.wikipedia.org/wiki/Gleditsia_triacanthos',
   'langname': 'Cebuano',
   '*': 'Gleditsia triacanthos'},
  {'lang': 'cs',
   'url': 'https://cs.wikipedia.org/wiki/D%C5%99ezovec_trojtrnn%C3%BD',
   'langname': 'Czech',
   '*': 'Dřezovec trojtrnný'},
  {'lang': 'da',
   'url': 'https://da.wikipedia.org/wiki/Almindelig_tretorn',
   'langname': 'Danish',
   '*': 'Almindelig tretorn'},
  {'lang': 'de',
   'url': 'https://de.wikipedia.org/wiki/Amerikanische_Gleditschie',
   'langname': 'German',
   '*': 'Amerikanische Gleditschie'},
  {'lang': 'en',
   'url': 'https://en.wikipedia.org/wiki/Honey_locust',
   'langname': 'English',
   '*': 'Honey locust'},
  {'lang': 'eo',
   'url': 'https://eo.wikipedia.org/wiki/Kristodorna_gledi%C4%89io',
   'langname': 'Esperanto',
   '*': 'Kristodorna gledîĉio'},
  {'lang': 'es',
   'url': 'https://es.wikipedia.org/wiki/Gleditsia_triacanthos',
   'langname': 'Spanish',
   '*': 'Gleditsia triacanthos'},
  {'lang': 'eu',
   'url': 'https://eu.wikipedia.org/wiki/Akazia_hiruarantza',
   'langname': 'Basque',
   '*': 'Akazia hiruarantza'},
  {'lang': 'fa',
   'url': 'https://fa.wikipedia.org/wiki/%D9%84%DB%8C%D9%84%DA%A9%DB%8C_%D8%A2%D9%85%D8%B1%DB%8C%DA%A9%D8%A7
   'langname': 'Persian',
   '*': □□□□□□' {'□□□□□□□□□□,
  {'lang': 'fi',
   'url': 'https://fi.wikipedia.org/wiki/Kolmioka',
  'langname': 'Finnish',
  '*': 'Kolmioka'},
 {'lang': 'fr',
  'url': 'https://fr.wikipedia.org/wiki/F%C3%A9vier_d%27Am%C3%A9rique',
  'langname': 'French',
  '*': "Févier d'Amérique"},
 {'lang': 'ga',
  'url': 'https://ga.wikipedia.org/wiki/Gleditsia_triacanthos',
  'langname': 'Irish',
  '*': 'Gleditsia triacanthos'},
 {'lang': 'hr',
  'url': 'https://hr.wikipedia.org/wiki/Ameri%C4%8Dka_gledi%C4%8Dija',
  'langname': 'Croatian',
  '*': 'Američka gledičija'},
 {'lang': 'hsb',
  'url': 'https://hsb.wikipedia.org/wiki/Ameriska_gledi%C4%8Dija',
  'langname': 'Upper Sorbian',
  '*': 'Ameriska gledičija'},
 {'lang': 'hu',
  'url': 'https://hu.wikipedia.org/wiki/T%C3%B6vises_lep%C3%A9nyfa',
  'langname': 'Hungarian',
  '*': 'Tövises lepényfa'},
 {'lang': 'hy',
  'url': 'https://hy.wikipedia.org/wiki/%D4%B3%D5%AC%D5%A5%D5%A4%D5%AB%D5%B9%D5%A1',
  'langname': 'Armenian',
  '*': '□□□□□□□'},
 {'lang': 'it',
  'url': 'https://it.wikipedia.org/wiki/Gleditsia_triacanthos',
  'langname': 'Italian',
  '*': 'Gleditsia triacanthos'},
 {'lang': 'kbd',
  'url': 'https://kbd.wikipedia.org/wiki/%D0%91%D0%B0%D0%BD%D1%8D%D0%B6%D1%8B%D0%B3',
  'langname': 'Kabardian',
  '*': 'Банэжыг'},
 {'lang': 'kk',
  'url': 'https://kk.wikipedia.org/wiki/%D2%AE%D1%88%D1%82%D1%96%D0%BA%D0%B5%D0%BD%D0%B4%D1%96_%D2%9B%D0%B0
  'langname': 'Kazakh',
  '*': 'Үштікенді қарамала'},
 {'lang': 'lt',
  'url': 'https://lt.wikipedia.org/wiki/Tridygl%C4%97_gledi%C4%8Dija',
  'langname': 'Lithuanian',
  '*': 'Tridyglė gledičija'},
 {'lang': 'nl',
  'url': 'https://nl.wikipedia.org/wiki/Valse_christusdoorn',
  'langname': 'Dutch',
  '*': 'Valse christusdoorn'},
 {'lang': 'no',
  'url': 'https://no.wikipedia.org/wiki/Korstorn',
  'langname': 'Norwegian',
  '*': 'Korstorn'},
 {'lang': 'nv',
      'url': 'https://nv.wikipedia.org/wiki/Naazt%C3%A1n%C3%AD',
      'langname': 'Navajo',
      '*': 'Naaztání'},
     {'lang': 'pl',
      'url': 'https://pl.wikipedia.org/wiki/Glediczja_tr%C3%B3jcierniowa',
      'langname': 'Polish',
      '*': 'Glediczja trójcierniowa'},
     {'lang': 'pms',
      'url': 'https://pms.wikipedia.org/wiki/Gleditsia_triacanthos',
      'langname': 'Piedmontese',
      '*': 'Gleditsia triacanthos'},
     {'lang': 'pt',
      'url': 'https://pt.wikipedia.org/wiki/Gleditsia_triacanthos',
      'langname': 'Portuguese',
      '*': 'Gleditsia triacanthos'},
     {'lang': 'ro',
      'url': 'https://ro.wikipedia.org/wiki/Gl%C4%83di%C8%9B%C4%83',
      'langname': 'Romanian',
      '*': 'Glădiță'},
     {'lang': 'ru',
      'url': 'https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%B5%D0%B4%D0%B8%D1%87%D0%B8%D1%8F_%D1%82%D1%80%D1%91
      'langname': 'Russian',
      '*': 'Гледичия трёхколючковая'},
     {'lang': 'sr',
      'url': 'https://sr.wikipedia.org/wiki/%D0%A2%D1%80%D0%BD%D0%BE%D0%B2%D0%B0%D1%86_(%D0%B1%D0%B8%D1%99%D0%B
      'langname': 'Serbian',
      '*': 'Трновац (биљка)'},
     {'lang': 'sv',
      'url': 'https://sv.wikipedia.org/wiki/Gleditsia_triacanthos',
      'langname': 'Swedish',
      '*': 'Gleditsia triacanthos'},
     {'lang': 'uk',
      'url': 'https://uk.wikipedia.org/wiki/%D0%93%D0%BB%D0%B5%D0%B4%D0%B8%D1%87%D1%96%D1%8F_%D0%BA%D0%BE%D0%BB
      'langname': 'Ukrainian',
      '*': 'Гледичія колюча'},
     {'lang': 'vi',
      'url': 'https://vi.wikipedia.org/wiki/B%E1%BB%93_k%E1%BA%BFt_ba_gai',
      'langname': 'Vietnamese',
      '*': 'B□ k□t ba gai'},
     {'lang': 'war',
      'url': 'https://war.wikipedia.org/wiki/Gleditsia_triacanthos',
      'langname': 'Waray',
      '*': 'Gleditsia triacanthos'},
     {'lang': 'zh',
      'url': 'https://zh.wikipedia.org/wiki/%E7%BE%8E%E5%9B%BD%E7%9A%82%E8%8D%9A',
      'langname': 'Chinese',
      '*': '□□□□'}],
    'description': 'species of tree',
    'descriptionsource': 'central'}}}}
```

```
In [18]: languages = ['fr', 'ru', 'ar']
```

```
         # add new columns to cadastre table
         for lang in languages:
             df['Name_' + lang] = pd.Series([''] * df.shape[0], index=df.index)

         for tree in top_trees.index.to_list():
             if tree not in tree_translations:
                 continue
             for _id, result in tree_translations[tree]['query']['pages'].items():
                 for lang in languages:
                     for langlink in result['langlinks']:
                         if langlink['lang'] in languages:
                             # print(tree, langlink)
                             # add the translation to the table
                             df.loc[df['Name_lat']==tree, 'Name_' + langlink['lang']] = langlink['*']
```

```
In [19]: name_cols = ['Name_lat', 'Name_dt', *['Name_' + lang for lang in languages]]

         tree_name_translation = df.loc[df['Name_lat'].isin(top_trees.index), name_cols]
         tree_name_translation['count'] = 1
         tree_name_translation.groupby(name_cols).sum().sort_values('count', ascending=False)
```

```
Out[19]:
         Name_lat                              Name_dt                          Name_fr                    Name_ru
         Platanus x acerifolia                 Platane                          Platane commun             Платан кленол
         Betula pendula                        Birke, Sand-Birke                Bouleau verruqueux         Берёза повисл
         Quercus robur                         Eiche, Stiel-Eiche, Sommer-Eiche Chêne pédonculé            Дуб черешчаты
         Fraxinus excelsior                    Esche, Esche gemeine             Frêne élevé                Ясень обыкнов
         Tilia cordata                         Linde, Winter-Linde              Tilleul à petites feuilles Липа сердцеви
         Malus domestica                       Kultur-Apfel                     Pommier domestique         Яблоня домашн
         Salix alba                            Weide, Silber-Weide              Salix alba                 Ива белая
         Acer platanoides                      Ahorn, Spitz-Ahorn               Érable plane               Клён остролис
         Acer pseudoplatanus                   Ahorn, Berg-Ahorn                Érable sycomore            Клён белый
         Pyrus communis                        Birne, Holz-Birne                Poirier commun             Груша обыкнов
         Carpinus betulus                      Weißbuche, Hainbuche             Charme commun              Граб обыкнове
         Acer campestre                        Ahorn, Feld-Ahorn                Érable champêtre           Клён полевой
         Juglans regia                         Nussbaum, Walnuss                Noyer commun               Орех грецкий
         Aesculus hippocastanum                Rosskastanie                     Aesculus hippocastanum     Конский кашта
         Fagus sylvatica                       Buche, Rotbuche                  Hêtre commun               Бук европейск
         Fraxinus excelsior 'Westhof's Glorie' Straßen-Esche                    Frêne élevé                Ясень обыкнов
         Tilia platyphyllos                    Linde, Sommer-Linde              Tilleul à grandes feuilles Липа крупноли
         Prunus avium                          Kirsche, Vogel-Kirsche           Prunus avium               Черешня
         Tilia cordata 'Greenspire'            Linde "Greespire"                Tilleul à petites feuilles Липа сердцеви
         Gleditsia triacanthos 'Inermis'       Dornenlose Gleditschie           Févier d'Amérique          Гледичия трёх
```

2.4.2 Remark: Advanced JSON processing with `jq`

Processing deeply nested JSON is cumbersome because the Pythone code may also require nested loops or recursive function calls. The JSON processor jq allows for easy processing (filter and transform) of JSON data. There exist Python bindings but it is primarily a command-line tool:

1. download one tree record from Wikispecies using curl:

```
curl 'https://species.wikimedia.org/w/api.php?action=query&format=json&prop=iwlinks|langlinks|description&lllimit=
    >data/wikispecies-quercus-robur.json
```

2. inspect the JSON result (nicely formatted):

   ```
   jq . <data/wikispecies-quercus-robur.json
   ```

3. step by step drill down to extract the data

   ```
   jq -r '.["query"]["pages"][]["langlinks"][] | [.["lang","*"]] | join("\t")' \
       <data/quercus_robur-wikimedia-species.json \
     | head
   ```

   which will extract a map <language,name_of_tree>:

   ```
   af      Steeleik
   ar      □□□□□□ □□□
   arz     □□□□□□ □□□
   ast     Quercus robur
   az      Yay palıdı
   azb     □□□ □□□□□□
   bat-smg Ōžouls
   be      Дуб звычайны
   bg      Обикновен дъб
   bs      Hrast lužnjak
   ```

Using the jq Python bindings you could extract the data by …

```
In [20]: import jq

         q = jq.compile('.["query"]["pages"][]["langlinks"][] | [.["lang","*"]]')
         translations_quercus_robur = dict(
             q.input(
                 json.load(
                     open('data/quercus_robur-wikimedia-species.json'))).all())
         translations_quercus_robur['fr']

Out[20]: 'Chêne pédonculé'
```

2.5 Mapping Geographic Data

To show the trees on the map we use the package Folium. See also the quickstart and API docs.

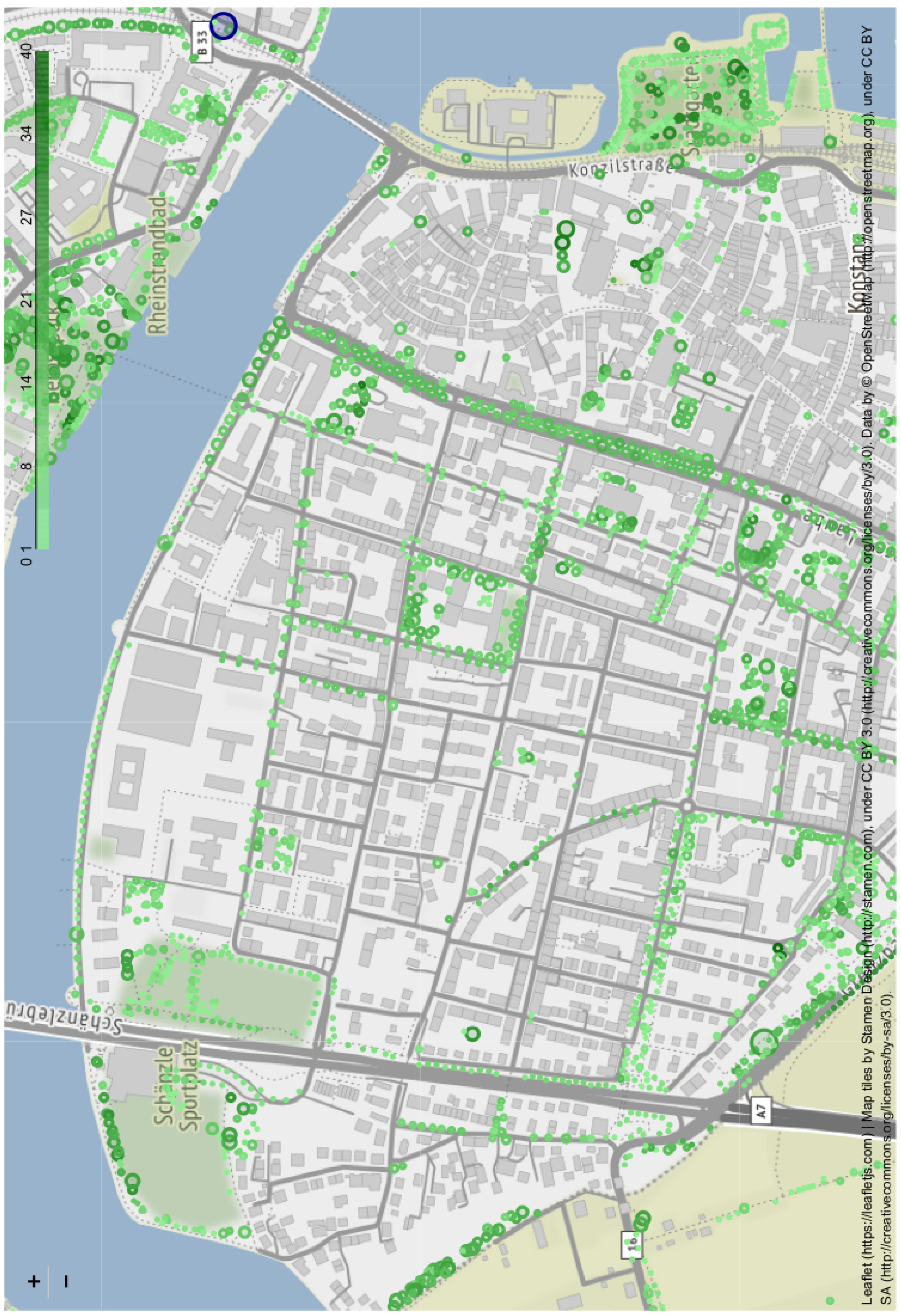
```
In [21]: import folium
         import math
         import branca.colormap as cm

         map = folium.Map(location=[47.66336, 9.17598],
                          tiles = 'Stamen Terrain',
                          zoom_start=16)

         colormap = cm.LinearColormap(colors=['lightgreen','darkgreen'],
```

```
                                        vmin=1, vmax=40).to_step(n=12)

def color_height(height):
    if 1.0 <= height <= 40.0:
        return colormap(height)
    else:
        return 'darkblue'

def map_tree(row):
    marker = folium.CircleMarker(
                location=(row['Y'], row['X']),
                tooltip=folium.Tooltip(row['Name_lat']),
                radius=row['treetop diameter (m)']/4,
                fill=True,
                color=color_height(row['height (m)']),
              )
    marker.add_to(map)

# for development: select a subset because plotting 16k trees takes long
#   df[df['location']=='Münsterplatz (27)']
#   df.head(500)

df.apply(map_tree, axis=1)

map.add_child(colormap, name='height (m)')
map
```

```
Out[21]: <folium.folium.Map at 0x7fe9766db070>
```

2.6 Links and References

- Pandas getting started
- matplotlib cheatsheet (beginners sheet)
- processing JSON data from the course "Data Analysis and Visualization with Python for Social Scientists" (https://datacarpentry.org/python-socialsci/)



Leaflet (https://leafletjs.com) | Map tiles by Stamen Design (http://stamen.com), under CC BY 3.0 (http://creativecommons.org/licenses/by/3.0). Data by © OpenStreetMap (http://openstreetmap.org), under CC BY SA (http://creativecommons.org/licenses/by-sa/3.0).

3 The Twitter API

- what is an API?
- get access to the Twitter API
- use a client: DocNow/twarc
- tweets, user timelines, followers, trends
- text statistics, language, sentiment

3.1 What is an API?

The Application Programming Interface (API) allows computer programs to interact with software libraries (the pandas API) or services (eg. Twitter or Mediawiki) in a similar way a user interface allows humans to interact with computers.

3.2 Why social media and why Twitter?

Social media is an important data source for social science research:

> social media platforms are, in one sense, vast collections of freely available unscripted opinions, experiences and insights on any number of topics“ (Phillip D. Brooker Section **??**)

The Twitter API is easy to set up and usage is less restrictive compared to the APIs of other social media platforms.

3.3 Get Access to the Twitter API

Before apply for access you definitely should read about the restrictions on using and sharing Twitter data. You may also start browsing the API documentation.

After having registered for an API account, you need to follow the documentation about getting started.

Note that

- the registration and setup process requires some time
- the examples given below can only replayed if you have registered for the Twitter API

3.4 Install and Setup Twarc

Twarc is

> a command line tool and Python library for archiving Twitter JSON data. Each tweet is represented as a JSON object that is exactly what was returned from the Twitter API. Tweets are stored as line-oriented JSON. twarc will handle Twitter API's rate limits for you. In addition to letting you collect tweets twarc can also help you collect users, trends and hydrate tweet ids. (from the Twarc documentation)

Installation and setup is done in just two steps:

- install

  ```
  pip install twarc
  ```

- configure twarc to use your Twitter API credentials

  ```
  twarc configure
  ```

  or for version 2 of the API

  ```
  twarc2 configure
  ```

See the Twarc documentation for more details and also for first examples to work with Twarc.

We will use twarc2 to access version 2 of the Twitter API. We focus on the command-line tool only - there is no need to use the Twarc API unless there are very specific requirements or using Twarc is part of a more complex data acquisition process.

First, we call `twarc2 --help` to figure out which options and commands are provided:

```
In [1]: !twarc2 --help
```

```
Usage: twarc2 [OPTIONS] COMMAND [ARGS]…

  Collect data from the Twitter V2 API.

Options:
  --consumer-key TEXT         Twitter app consumer key (aka "App Key")
  --consumer-secret TEXT      Twitter app consumer secret (aka "App Secret")
  --access-token TEXT         Twitter app access token for user
                              authentication.
  --access-token-secret TEXT  Twitter app access token secret for user
                              authentication.
  --bearer-token TEXT         Twitter app access bearer token.
  --app-auth / --user-auth    Use application authentication or user
                              authentication. Some rate limits are higher with
                              user authentication, but not all endpoints are
                              supported.  [default: app-auth]
  -l, --log TEXT
  --verbose
  --metadata / --no-metadata  Include/don't include metadata about when and
                              how data was collected.  [default: metadata]
  --config FILE               Read configuration from FILE.
```

```
  --help                      Show this message and exit.

Commands:
  configure      Set up your Twitter app keys.
  conversation   Retrieve a conversation thread using the tweet id.
  conversations  Fetch the full conversation threads that the input…
  counts         Return counts of tweets matching a query.
  flatten        "Flatten" tweets, or move expansions inline with tweet…
  followers      Get the followers for a given user.
  following      Get the users who are following a given user.
  hydrate        Hydrate tweet ids.
  mentions       Retrieve max of 800 of the most recent tweets mentioning…
  sample         Fetch tweets from the sample stream.
  search         Search for tweets.
  stream         Fetch tweets from the live stream.
  stream-rules   List, add and delete rules for your stream.
  timeline       Retrieve recent tweets for the given user.
  timelines      Fetch the timelines of every user in an input source of…
  tweet          Look up a tweet using its tweet id or URL.
  users          Get data for user ids or usernames.
  version        Return the version of twarc that is installed.
```

… and to get the command-specific options:

```
In [2]: !twarc2 timeline --help
```

```
Usage: twarc2 timeline [OPTIONS] USER_ID [OUTFILE]

  Retrieve recent tweets for the given user.

Options:
  --limit INTEGER                 Maximum number of tweets to return
  --since-id INTEGER              Match tweets sent after tweet id
  --until-id INTEGER              Match tweets sent prior to tweet id
  --exclude-retweets              Exclude retweets from timeline
  --exclude-replies               Exclude replies from timeline
  --start-time [%Y-%m-%d|%Y-%m-%dT%H:%M:%S]
                                  Match tweets created after time (ISO
                                  8601/RFC 3339), e.g.  2021-01-01T12:31:04
  --end-time [%Y-%m-%d|%Y-%m-%dT%H:%M:%S]
                                  Match tweets sent before time (ISO 8601/RFC
                                  3339)
  --use-search                    Use the search/all API endpoint which is not
                                  limited to the last 3200 tweets, but
                                  requires Academic Product Track access.
  --hide-progress                 Hide the Progress bar. Default: show
                                  progress, unless using pipes.
  --help                          Show this message and exit.
```

3.5 Analyzing Tweets from a User Timeline

For a first trial we download 500 tweets from the timeline of [@EXCInequality](https://twitter.com/EXCInequality) and save it to a file:

```
twarc2 timeline EXCInequality --limit 500 >data/twitter/timeline.EXCInequality.jsonl
```

Note that the Twitter developer terms of use do not allow to share the content of tweets. That's why not tweet data is included in this repository, or only in aggregations on the level of words. You need to apply for API access in order to replay the examples.

```
In [3]: import json
        import pandas as pd

        def load_tweets(file):
            tweets = []
            with open(file) as stream:
                for line in stream:
                    api_response = json.loads(line)
                    for tweet in api_response['data']:
                        tweets.append(tweet)
            return tweets

        tweets = load_tweets('data/twitter/timeline.EXCInequality.jsonl')

        len(tweets)
```

```
Out[3]: 500
```

Let's look into the one of the tweets to understand the data structure and compare this with the tweet object model documentation.

```
In [4]: #tweets[1]
```

Note: it's possible to load the tweets into a pandas dataframe but some cells still contain nested JSON elements:

```
df = pd.DataFrame(tweets)
```

Pandas provides normalization routines to flatten nested data.

But we will work with the JSON data directly and first extract which hashtags are frequently used in the Tweets of [@EXCInequality](https://twitter.com/EXCInequality):

```
In [5]: from collections import Counter

        aggregation_on = ('hashtags', 'tag')

        # instead of hashtags count other items in the `entities` object:
        # aggregation_on = ('annotations', 'normalized_text')
        # aggregation_on = ('mentions', 'username')
        # aggregation_on = ('urls', 'url')
        counts = Counter()


        for t in tweets:
            if 'entities' not in t:
                continue
            if aggregation_on[0] in t['entities']:
                for obj in t['entities'][aggregation_on[0]]:
                    counts[obj[aggregation_on[1]]] += 1

        counts.most_common()[0:20]
```

```
Out[5]: [('inequality', 35),
         ('UniKonstanz', 22),
         ('jobsinscience', 22),
         ('ClusterColloquium', 21),
         ('jobsinacademia', 21),
         ('COVID19', 18),
         ('PolicyPaper', 11),
         ('ThePoliticsOfInequality', 9),
         ('InequalityMagazine', 9),
         ('FunFriday', 9),
         ('Konstanz', 8),
         ('Homeoffice', 7),
         ('unikonstanz', 7),
         ('outsoon', 6),
         ('research', 5),
         ('PGS21', 4),
         ('Ungleichheit', 4),
         ('NewPublication', 4),
         ('Exzellenzcluster', 4),
         ('EqualPayDay', 4)]
```

3.5.1 Find the Most Commonly Used Words in Tweets

We will now look into the tweets itself and - split the text into words - count word occurrences and - generate a word cloud to visualize word frequencies or the "importance" of words

```
In [6]: words = Counter()

        for t in tweets:
            for word in t['text'].split(' '):
                words[word] += 1

        words.most_common()[0:10]
```

```
Out[6]: [('the', 313),
         ('of', 256),
         ('to', 230),
         ('in', 228),
         ('and', 226),
         ('RT', 199),
         ('a', 178),
         ('on', 128),
         ('for', 121),
         ('is', 103)]
```

This initial attempt shows that we need to skip over the most common functional words, in text processing called "stop words".

```
In [7]: from stop_words import get_stop_words

        stop_words = set(get_stop_words('en'))
        stop_words.update(get_stop_words('de'))

        def word_counts(tweets):
            words = Counter()
            for t in tweets:
                for word in t['text'].split(' '):
                    word = word.lower()
                    if word in stop_words:
                        continue
                    words[word] += 1
            return words

        word_counts(tweets).most_common()[0:25]
```

```
Out[7]: [('rt', 199),
         ('&amp;', 81),
         ('-', 73),
         ('@unikonstanz', 55),
         ('@unikonstanz:', 52),
         ('cluster', 48),
         ('new', 45),
         ('research', 45),
         ('@excinequality', 30),
         ('talk', 29),
         ('work', 28),
         ('just', 27),
         ('us', 27),
         ('#inequality', 27),
         ('project', 26),
         ('–', 26),
         ('can', 24),
         ('one', 24),
         ('policy', 23),
         ('#unikonstanz', 23),
         ('social', 22),
         ('paper', 21),
         ('great', 21),
         ('inequality', 21),
         ('political', 20)]
```

… and we also need to skip mentions, hashtags, URLs and everything which does not look like a word. We simply skip all words containing any other characters except letters (alphabetical characters). Note that this approach is simple and effective but it will also remove words such as “Covid-19”.

```
In [8]: stop_words.add('rt') # retweet


        def word_counts(tweets):
            words = Counter()
            for t in tweets:
                for word in t['text'].split(' '):
                    word = word.lower()
                    if word in stop_words:
                        continue
                    if not word.isalpha():
                        # skip words containing non-alphabetical characters
                        continue
                    words[word] += 1
            return words


        word_counts(tweets).most_common()[0:25]
```

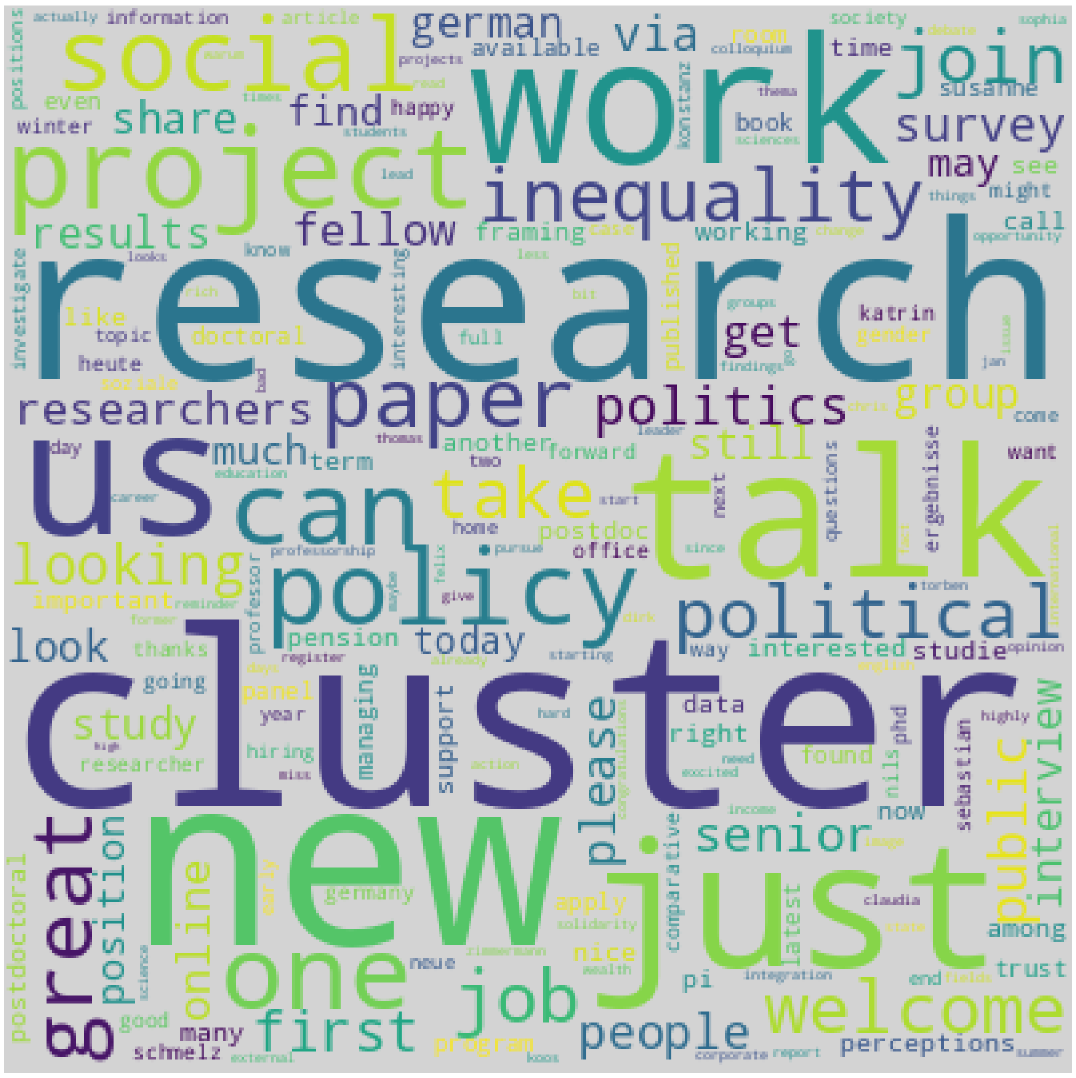
```
Out[8]: [('cluster', 48),
         ('new', 45),
         ('research', 45),
         ('talk', 29),
         ('work', 28),
         ('just', 27),
         ('us', 27),
         ('project', 26),
         ('can', 24),
         ('one', 24),
         ('policy', 23),
         ('social', 22),
         ('paper', 21),
         ('great', 21),
         ('inequality', 21),
         ('political', 20),
         ('welcome', 20),
         ('join', 20),
         ('job', 20),
         ('take', 18),
         ('looking', 18),
         ('first', 18),
         ('public', 16),
         ('politics', 16),
         ('senior', 15)]
```

Word clouds are generated using the wordcloud package, see also: - API docs of the WordCloud class - more examples

```
In [9]: from wordcloud import WordCloud
```

```
wordcloud = WordCloud(width=400, height=400,
                      background_color='lightgrey') \
            .generate_from_frequencies(word_counts(tweets))

wordcloud.to_image()
```

Out[9]:



3.5.2 Words Used by the Official Twitter Accounts of German Political Parties

Let's download tweets from the official Twitter accounts of the political parties currently. We wrap the calls of Twarc into a loop in the command-line shell and limit the download to a single month and max. 50k tweets:

```
mkdir -p data/twitter/ppart/timeline/
for pp in CDU CSU spdde Die_Gruenen dieLinke AfD; do
```

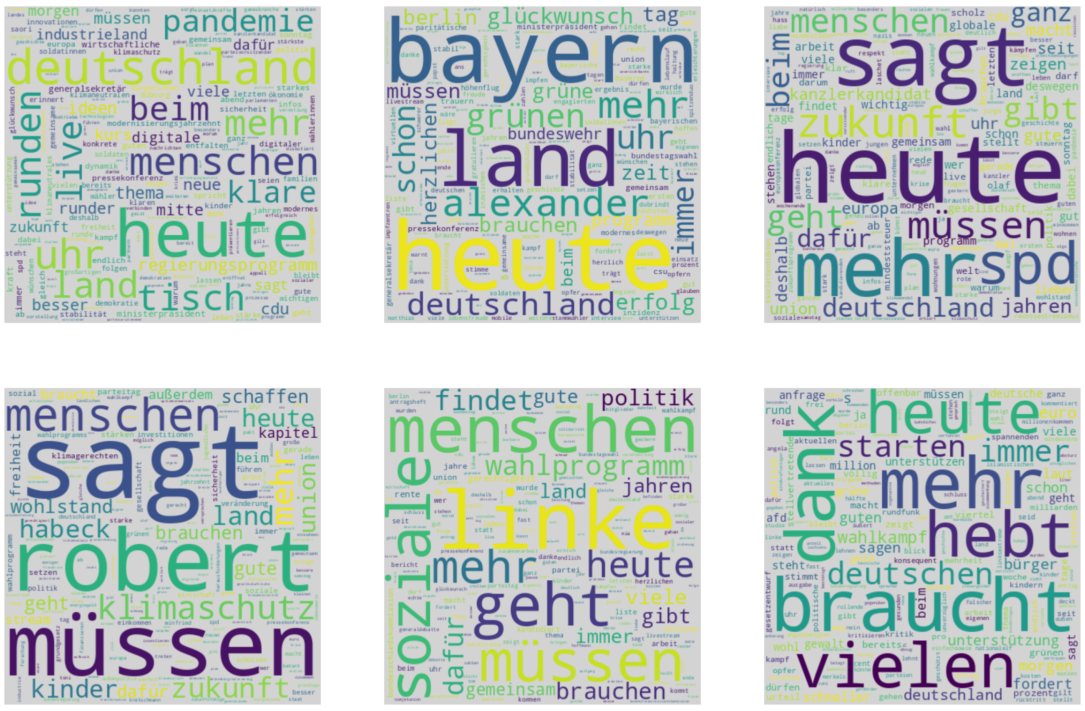
```
    twarc2 timeline $pp \
           --start-time 2021-06-01 \
           --end-time 2021-07-01 \
           --limit 50000 \
           >data/twitter/ppart/timeline/$pp.jsonl
done
```

Then we load the data in Python, extract the word counts and generate the word clouds...

```
In [10]: parties = 'CDU CSU spdde Die_Gruenen dieLinke AfD'.split()

         words = {}

         for party in parties:
             tweets = load_tweets('data/twitter/ppart/timeline/%s.jsonl' % party)
             words[party] = word_counts(tweets)
             # show some stats
             print(party, len(tweets), 'tweets')
             print('\t', word_counts(tweets).most_common()[0:3])
```

```
CDU 188 tweets
         [('heute', 21), ('deutschland', 19), ('uhr', 13)]
CSU 179 tweets
         [('heute', 18), ('bayern', 16), ('land', 12)]
spdde 765 tweets
         [('heute', 72), ('sagt', 53), ('mehr', 46)]
Die_Gruenen 280 tweets
         [('sagt', 32), ('müssen', 25), ('robert', 24)]
dieLinke 444 tweets
         [('linke', 33), ('menschen', 28), ('soziale', 23)]
AfD 206 tweets
         [('braucht', 14), ('mehr', 13), ('dank', 12)]
```

```
In [11]: import matplotlib.pyplot as plt



         fig, axes = plt.subplots(nrows=2, ncols=3, figsize=[36,24])

         n = 0
         for party in parties:
             wordcloud = WordCloud(width=400, height=400,
                                   background_color='lightgrey') \
                         .generate_from_frequencies(words[party])
             axis = axes[int(n/3),n%3]
             axis.imshow(wordcloud)
             axis.axis('off') # do not show x/y scale
             n += 1

         plt.show()
```



3.6 Links and References

- Phillip Brooker's book Programming with Python for Social Scientists includes a chapter about using the Twitter API
- https://developer.twitter.com/en/products/twitter-api
- https://twitter.com/TwitterAPI
- https://developer.twitter.com/en/use-cases/do-research
- https://developer.twitter.com/en/products/twitter-api/academic-research
- https://twarc-project.readthedocs.io/en/latest/
- https://scholarslab.github.io/learn-twarc/
- https://github.com/DocNow/twarc/tree/main/utils (for JSON data downloaded using the v1 API)