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.value

            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)
```

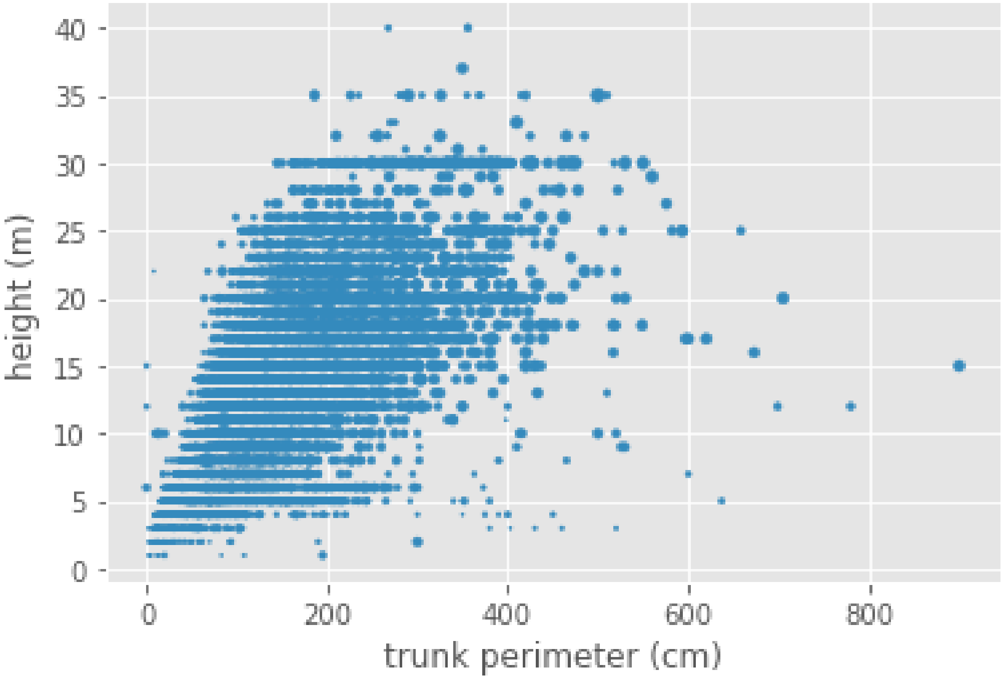
```
Out[8]:                                                                 count
        Name_lat                              Name_dt
        Platanus x acerifolia                 Platane                     887
        Betula pendula                        Birke, Sand-Birke           809
        Quercus 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)')
```

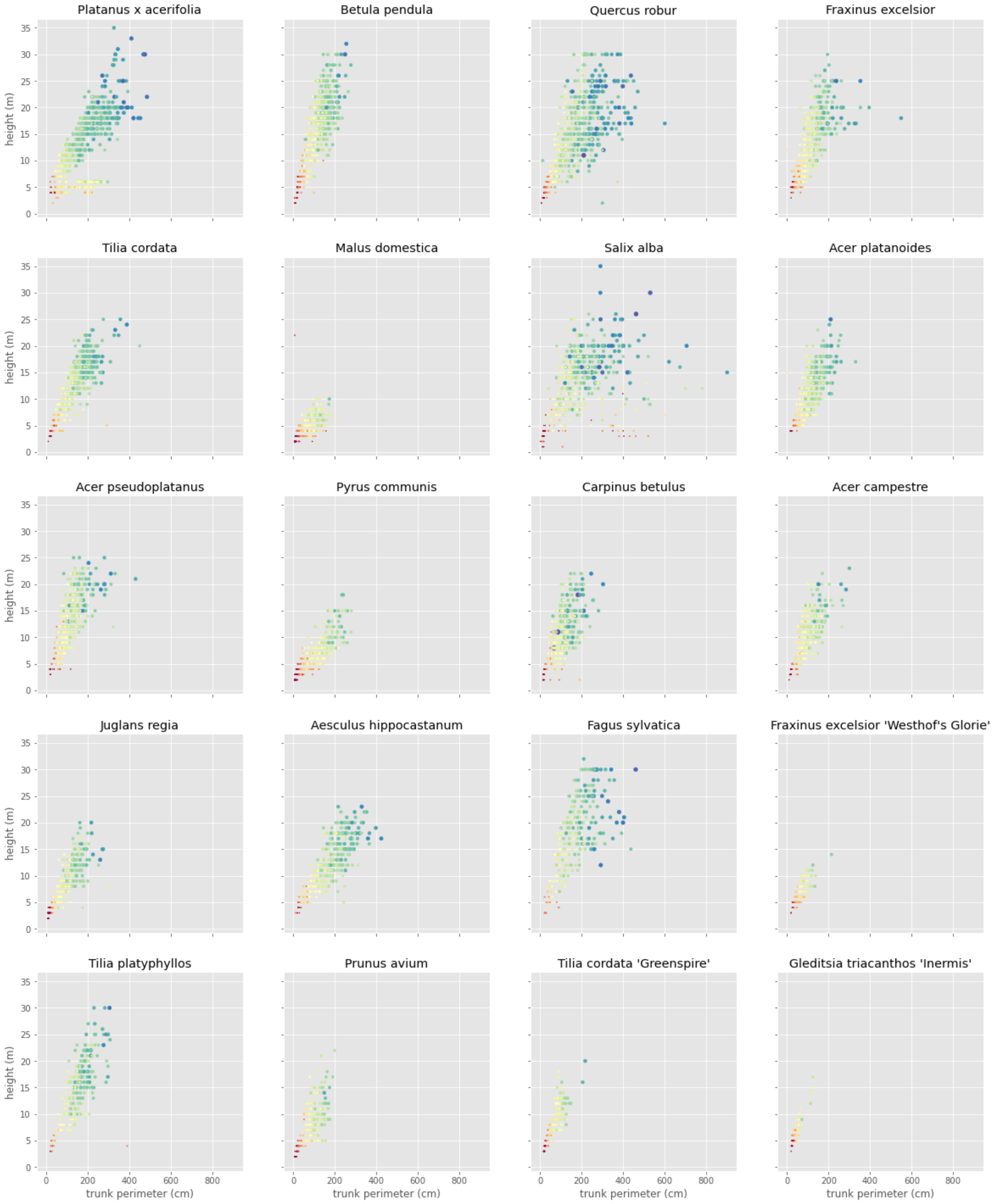
```
Out[9]: <AxesSubplot:xlabel='trunk perimeter (cm)', ylabel='height (m)'>
```



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')
```



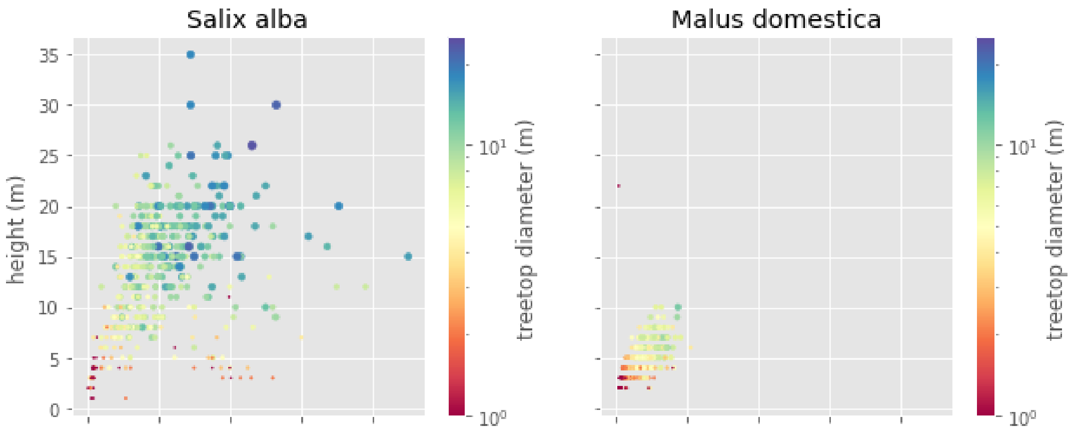
Notes about choosing the colormap for the treetop diameter: - the point size is hard to catch, while color is easier to discriminate (if not colorblind) - a spectral color map represents a continuous scale and allows for maximum discrimination - the range 1m - 25m (few trees reach 30m) is mapped on a 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 gledicîo'},
 {'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

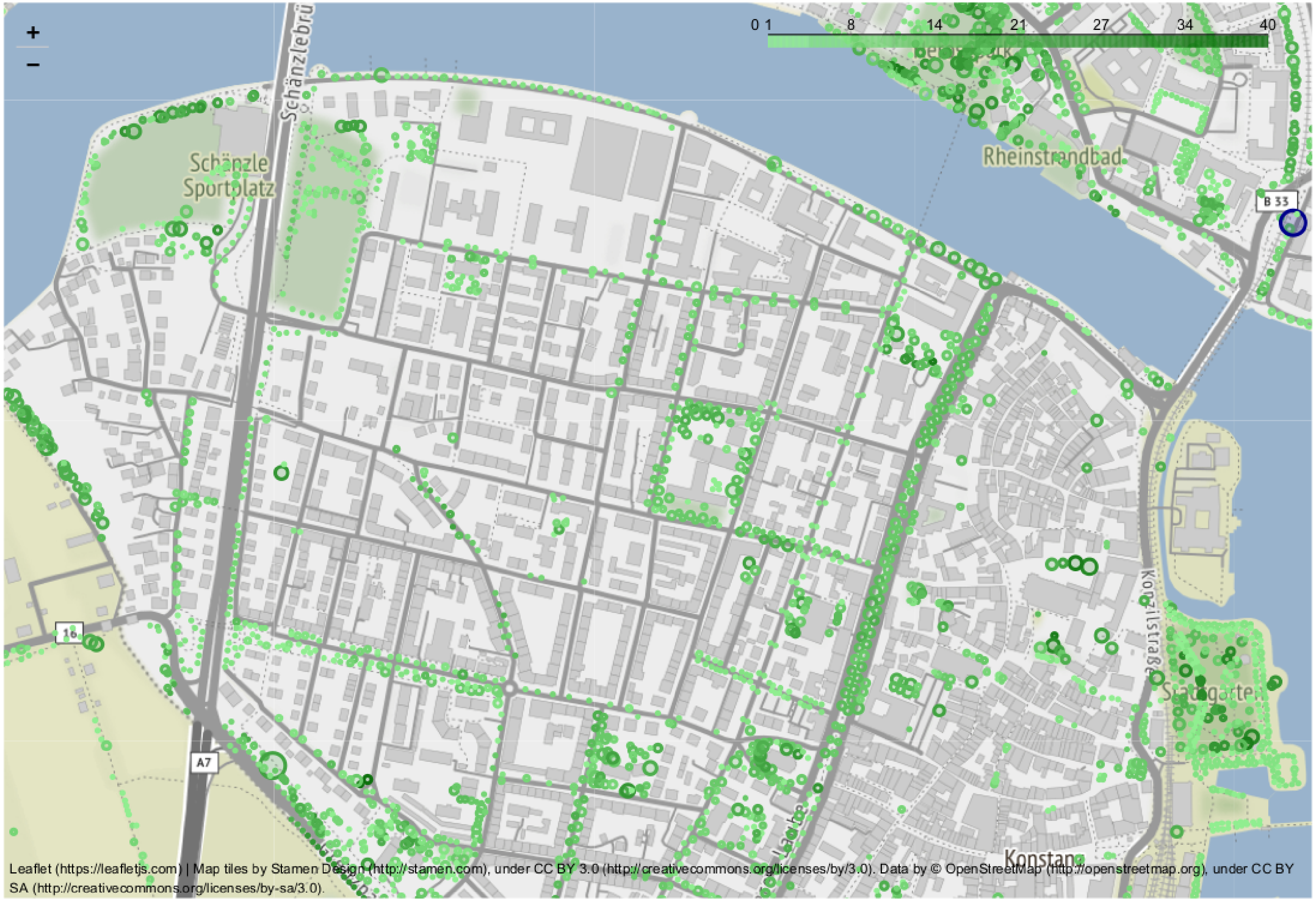
To show the trees on the map we use the package Folium. See also the quickstart and API docs.

```
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]
```

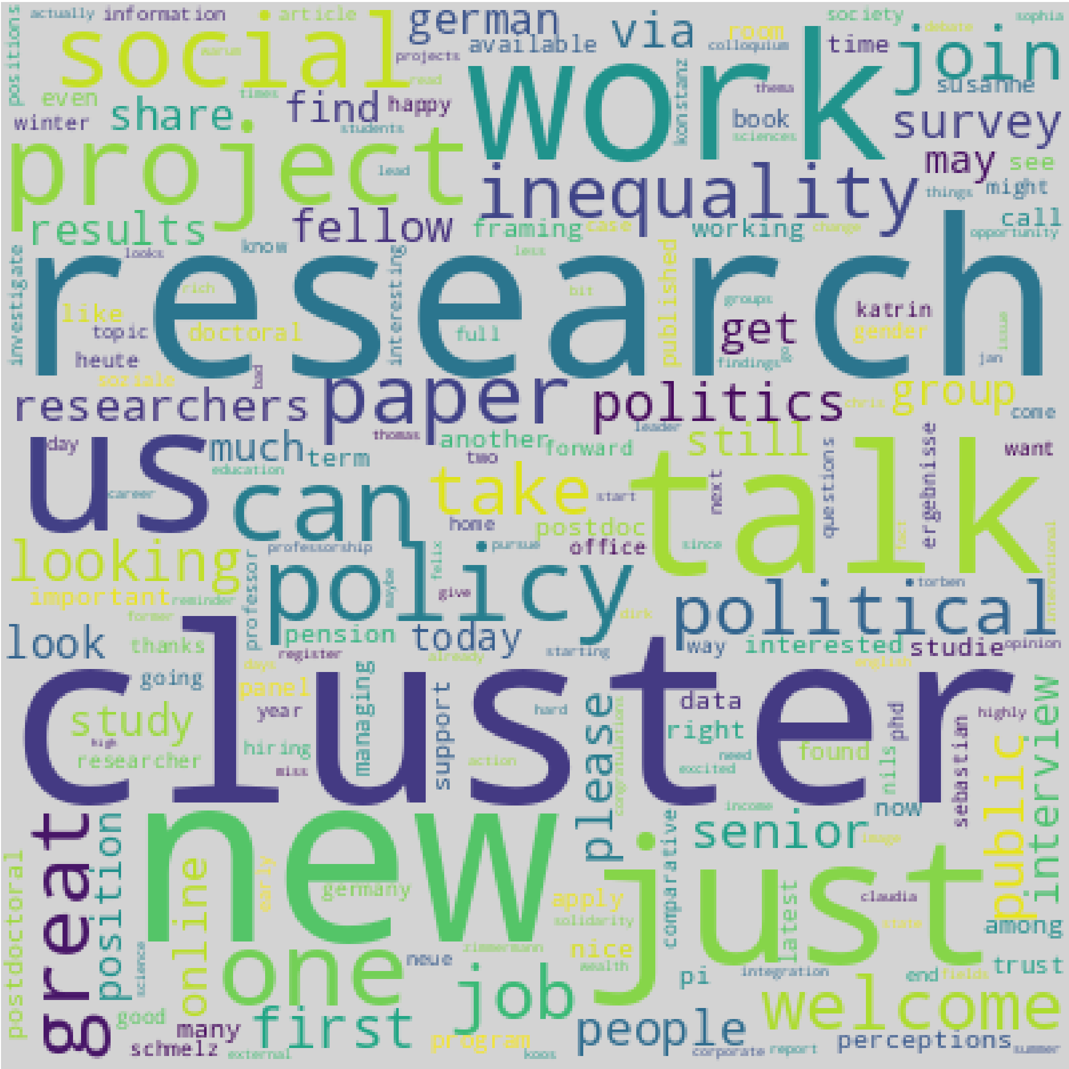
```
Out[8]: [('cluster', 48),
         ('new', 45),
         ('research', 45),
         ('talk', 29),
         ('work', 28),
         ('just', 27),
         ('us', 27),
         ('project', 26),
         ('can', 24),
         ('one', 24),
         ('policy', 23),
         ('social', 22),
         ('paper', 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
```

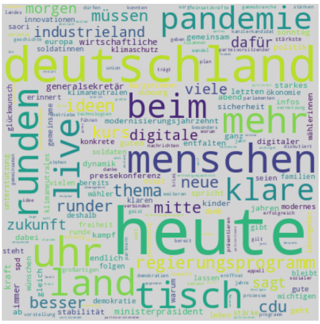
```
    twarc2 timeline $pp \
           --start-time 2021-06-01 \
           --end-time 2021-07-01 \
           --limit 50000 \
           >data/twitter/ppart/timeline/$pp.jsonl
done
```

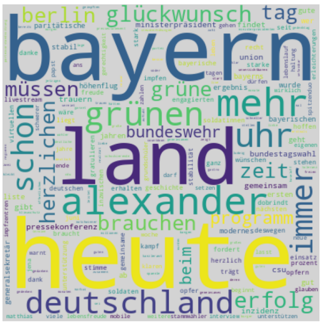
Then we load the data in Python, extract the word counts and generate the word clouds...

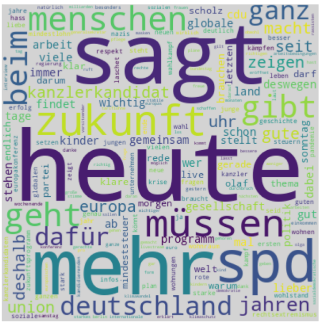
```
In [10]: parties = 'CDU CSU spdde Die_Gruenen dieLinke AfD'.split()

         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])
```

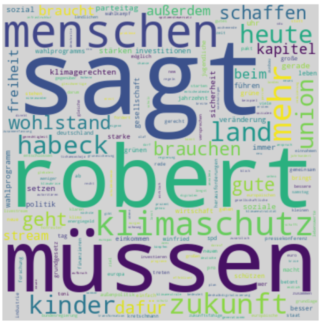
```
CDU 188 tweets
         [('heute', 21), ('deutschland', 19), ('uhr', 13)]
CSU 179 tweets
         [('heute', 18), ('bayern', 16), ('land', 12)]
spdde 765 tweets
         [('heute', 72), ('sagt', 53), ('mehr', 46)]
Die_Gruenen 280 tweets
         [('sagt', 32), ('müssen', 25), ('robert', 24)]
dieLinke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

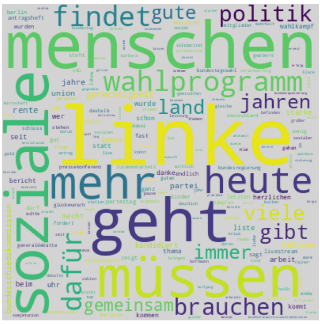
```
In [11]: import matplotlib.pyplot as plt


         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()
```

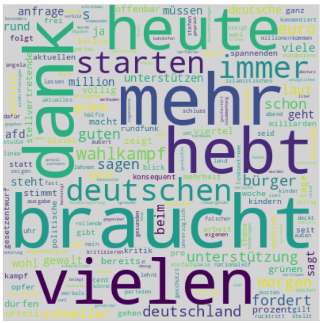












3.6 Links and References

- 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)

4 Web Scraping

- HTTP requests
- HTML, XML, DOM, CSS selectors, XPath
- browser automation
- cleanse and export extracted data

Web-based (or browser-based) user interfaces are ubiquitous

- web browser as universal platform to run software (at least, the user interface)
- if a human is able to access information in WWW using a web browser, also a computer program can access the same information and automatically extract it
- challenges: navigate a web page, execute user interaction (mouse clicks, forms)
- real challenges: login forms, captchas, IP blocking, etc.
  - not covered here
  - also: ethical considerations whether or not to get around access blocking
- well-defined technology stack
  - HTTP
  - HTML / XML
  - DOM
  - CSS
  - XPath
  - JavaScript

4.1 Web Browser

- render HTML page to make it readable for humans
- basic navigation in the WWW (follow links)
- text-based browsers

  ```
  lynx https://www.bundestag.de/parlament/fraktionen/cducsu
  ```

- modern graphical browsers
  - interpret JavaScript
  - show multi-media content
  - run "web applications"
- headless vs. headful browsers
  - headful: graphical user interface attached
  - headless
    * controlled programmatically or via command-line
    * interaction but no mandatory page rendering (saves resources: CPU, RAM)

4.1.1 Tip: Extract Text and Links Using a Text-Based Browser

Tip: text-based browsers usually have an option to “dump” the text and/or link lists into a file, e.g.

```
lynx -dump https://www.bundestag.de/parlament/fraktionen/cducsu \
    >data/bundestag/fraktionen.cducsu.txt
```

4.1.2 Tip: Explore Web Pages and Web Technologies using the Developer Tool of your Web Browser

Modern web browsers (Firefox, Chromium, IE, etc.) include a set of web development tools. Originally addressed to web developers to test and debug the code (HTML, CSS, Javascript) used to build a web site, the browser web developer tools are the easiest way to explore and understand the technologies used to build a web site. The initial exploration later helps to scrape data from the web site.

4.1.3 Browser Automation

- load a page by URL including page dependencies (CSS, Javascript, images, media)
- simulate user interaction (clicks, input, scrolling)
- take screenshots
- access the DOM tree or the HTML modified by executed Javascript and user interactions from/in the browser to extract data

4.2 Process HTML Pages in Python

- requests to fetch pages via HTTP
- beautifulsoup to parse HTML

```
In [1]: import requests

        request_url = 'https://www.bundestag.de/parlament/fraktionen/cducsu'
        response = requests.get(request_url)

        response

Out[1]: <Response [200]>

In [2]: response.headers

Out[2]: {'date': 'Tue, 13 Jul 2021 13:51:15 GMT', 'content-type': 'text/html;charset=UTF-8', 'content-length': '23620',

In [3]: response.status_code

Out[3]: 200

In [4]: !pip install beautifulsoup4
```

```
Requirement already satisfied: beautifulsoup4 in ./.venv/lib/python3.9/site-packages (4.9.3)
Requirement already satisfied: soupsieve>1.2 in ./.venv/lib/python3.9/site-packages (from beautifulsoup4) (2.2.1)
```

```
In [5]: from bs4 import BeautifulSoup

        html = BeautifulSoup(response.text)

        html.head.title  # tree-style path addressing of HTML elements
```

```
Out[5]: <title>Deutscher Bundestag - CDU/CSU-Fraktion</title>
```

Note: the HTML document can be represented as a tree structure aka. DOM tree:

```
html
├── head
|   ├── meta
|   |   └── @charset=utf-8
|   └── title
|       └── ...(text)
└── body
    └── ...
```

The tree above is an equivalent representation for the HTML snippet

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Deutscher Bundestag - CDU/CSU-Fraktion</title>
</head>
<body>
  ...
</body>
</html>
```

```
In [6]: # access the plain text of an HTML element
        # (inside the opening and closing tag)
        html.head.title.text
```

```
Out[6]: 'Deutscher Bundestag - CDU/CSU-Fraktion'
```

```
In [7]: # beautifulsoup also allows to select elements by tag name without a tree-like path

        html.find('title').text
```

```
Out[7]: 'Deutscher Bundestag - CDU/CSU-Fraktion'
```

```
In [8]: # or if a tag is expected to appear multiple times:
        # select all `a` elements and show the first three
        html.findAll('a')[0:3]
```

```
Out[8]: [<a class="sr-only sr-only-focusable" href="#main" title="Direkt zum Hauptinhalt springen">Direkt zum Hauptinhal
         <a class="sr-only sr-only-focusable" href="#main-menu" title="Direkt zum Hauptmenü springen">Direkt zum Hauptme
         <a href="https://www.bundestag.de/webarchiv" hreflang="de" lang="de" title="Archiv" xml:lang="de">
         <span class="sr-only-sm-down">Archiv</span>
         <span class="visible-xs-inline">Archiv</span>
         </a>]
```

```
In [9]: # selection by CSS class name
        html.find(class_='bt-standard-content')
```

```
Out[9]: <article class="bt-artikel col-xs-12 col-md-6 bt-standard-content">
        <h3 class="bt-artikel__title">CDU/CSU-Fraktion</h3>
        <div class="bt-bild-standard bt-bild-max" data-nosnippet="true">
        <img alt="Beschilderung einer Tür im Bereich der CDU/CSU-Fraktion auf der Fraktionsebene im Reichstagsgebäude."
        <span class="bt-bild-info-icon"><i aria-hidden="true" class="icon-info-1"></i>
        </span>
        <div class="bt-bild-info-text">
        <span class="bt-bild-info-close" tabindex="0">
        <i aria-hidden="true" class="icon-close"></i>
        </span>
        <p>
                Logo der CDU/CSU-Fraktion</p>
        <p>© DBT/Axel Hartmann Fotografie</p>
        </div>
        </div>
        <div class="bt-standard-content">
        <h4>Fraktionsvorsitzender:</h4><p><a href="/abgeordnete/biografien/B/brinkhaus_ralph-518692" target="_self">Ralp
        </article>
```

```
In [10]: html.find(class_='bt-standard-content').findAll('a')
```

```
Out[10]: [<a href="/abgeordnete/biografien/B/brinkhaus_ralph-518692" target="_self">Ralph Brinkhaus</a>,
          <a href="/abgeordnete/biografien/D/dobrindt_alexander-519076" target="_self">Alexander Dobrindt</a>,
          <a href="/abgeordnete/biografien/C/connemann_gitta-518902" target="_self">Gitta Connemann</a>,
          <a href="/abgeordnete/biografien/F/frei_thorsten-519532" rel="noopener" target="_blank">Thorsten Frei</a>,
          <a href="/abgeordnete/biografien/G/groehe_hermann-519870" target="_self">Hermann Gröhe</a>,
          <a href="https://www.bundestag.de/webarchiv/abgeordnete/biografien18/J/jung_andreas-258538" target="_self">And
          <a href="/abgeordnete/biografien/L/lange_ulrich-521486" target="_self">Ulrich Lange</a>,
          <a href="/abgeordnete/biografien/L/leikert_katja-521554" target="_self">Dr. Katja Leikert</a>,
          <a href="/abgeordnete/biografien/L/linnemann_carsten-521654" target="_self">Dr. Carsten Linnemann</a>,
          <a href="/abgeordnete/biografien/S/schoen_nadine-523428" target="_self">Nadine Schön</a>,
          <a href="/abgeordnete/biografien/S/stracke_stephan-523926" rel="noopener" target="_blank">Stephan Stracke</a>,
          <a href="/abgeordnete/biografien/V/vaatz_arnold-524242" target="_self">Arnold Vaatz</a>,
          <a href="/abgeordnete/biografien/W/wadephul_johann-524332" target="_self">Dr. Johann David Wadephul</a>,
          <a href="/abgeordnete/biografien/G/grosse_broemer_michael-519894" target="_self" title="Michael Grosse-Brömer"
          <a href="/abgeordnete/biografien/M/mueller_stefan-522218" target="_self">Stefan Müller</a>,
          <a href="/abgeordnete/biografien/B/brehmer_heike-518658" target="_self">Heike Brehmer</a>,
          <a href="/abgeordnete/biografien/G/grund_manfred-519946" target="_self">Manfred Grund</a>,
          <a href="/abgeordnete/biografien/S/schnieder_patrick-523412" target="_self">Patrick Schnieder</a>,
          <a href="/abgeordnete/biografien/R/rehberg_eckhardt-522826" target="_self">Eckhardt Rehberg</a>,
          <a href="/abgeordnete/biografien/B/brand_michael-518618" target="_self">Michael Brand</a>,
          <a href="/abgeordnete/biografien/D/doett_marie_luise-519098" target="_self">Marie-Luise Dött</a>,
          <a href="/abgeordnete/biografien/G/gienger_eberhard-519728" target="_self">Eberhard Gienger</a>,
          <a href="/abgeordnete/biografien/H/hahn_florian-520046" target="_self">Florian Hahn</a>,
          <a href="/abgeordnete/biografien/H/hardt_juergen-520110" target="_self">Jürgen Hardt</a>,
          <a href="/abgeordnete/biografien/K/klein_volkmar-521070" target="_self">Volkmar Klein</a>,
          <a href="/abgeordnete/biografien/L/lehrieder_paul-521542" target="_self">Paul Lehrieder</a>,
          <a href="/abgeordnete/biografien/M/maag_karin-521780" target="_self">Karin Maag</a>,
          <a href="/abgeordnete/biografien/M/middelberg_mathias-522026" target="_self">Dr. Mathias Middelberg</a>,
          <a href="/abgeordnete/biografien/M/motschmann_elisabeth-522132" target="_self">Elisabeth Motschmann</a>,
          <a href="/abgeordnete/biografien/O/otte_henning-522506" target="_self">Henning Otte</a>,
          <a href="/abgeordnete/biografien/P/pfeiffer_joachim-522616" target="_self">Dr. Joachim Pfeiffer</a>,
          <a href="/abgeordnete/biografien/R/rehberg_eckhardt-522826" target="_self">Eckhardt Rehberg</a>,
          <a href="/abgeordnete/biografien/R/rupprecht_albert-523090" target="_self">Albert Rupprecht</a>,
          <a href="/abgeordnete/biografien/S/schipanski_tankred-523278" target="_self">Tankred Schipanski</a>,
          <a href="/abgeordnete/biografien/S/stegemann_albert-523834" target="_self">Albert Stegemann</a>,
          <a href="/abgeordnete/biografien/S/storjohann_gero-523920" target="_self">Gero Storjohann</a>,
          <a href="/abgeordnete/biografien/T/tillmann_antje-524124" target="_self">Antje Tillmann</a>,
          <a href="/abgeordnete/biografien/W/wegner_kai-524464" target="_self">Kai Wegner</a>,
          <a href="/abgeordnete/biografien/W/weinberg_marcus-524490" target="_self">Marcus Weinberg</a>,
          <a href="/abgeordnete/biografien/W/weiss_peter-524514" target="_self">Peter Weiß</a>,
          <a href="/abgeordnete/biografien/W/winkelmeier_becker_elisabeth-524618" target="_self">Elisabeth Winkelmeier-B
          <a href="/abgeordnete/biografien/H/haase_christian-519998" target="_self">Christian Haase</a>,
          <a href="/abgeordnete/biografien/Z/zeulner_emmi-524762" rel="noopener" target="_blank">Emmi Zeulner</a>,
          <a href="/abgeordnete/biografien/M/magwas_yvonne-521800" target="_self">Yvonne Magwas</a>,
          <a href="/abgeordnete/biografien/P/pols_eckhard-522682" target="_self">Eckhard Pols</a>,
          <a href="/abgeordnete/biografien/S/schummer_uwe-523544" target="_self">Uwe, Schummer</a>,
          <a href="/abgeordnete/biografien/S/stetten_christian-523882" target="_self">Christian Freiherr von Stetten</a>
          <a href="/abgeordnete/biografien/F/fischer_axel-519454" target="_self">Axel E. Fischer</a>,
          <a href="/abgeordnete/biografien/G/gutting_olav-519978" target="_self">Olav Gutting</a>,
          <a href="/abgeordnete/biografien/G/guentzler_fritz-519962" target="_self">Fritz Güntzler</a>,
          <a href="/abgeordnete/biografien/H/heider_matthias-520192" target="_self">Matthias Heider</a>,
          <a href="/abgeordnete/biografien/H/520204-520204" target="_self">Thomas Heilmann</a>,
          <a href="/abgeordnete/biografien/H/helfrich_mark-520256" target="_self">Mark Helfrich</a>,
          <a href="/abgeordnete/biografien/H/henke_rudolf-520294" target="_self">Rudolf Henke</a>,
          <a href="/abgeordnete/biografien/H/holmeier_karl-520494" target="_self">Karl Holmeier</a>,
          <a href="/abgeordnete/biografien/K/kiesewetter_roderich-520990" target="_self">Roderich Kiesewetter</a>,
          <a href="/abgeordnete/biografien/M/metzler_jan-521988" target="_self">Jan Metzler</a>,
          <a href="/abgeordnete/biografien/M/michelbach_hans-522016" target="_self">Dr. h. c. Hans Michelbach</a>,
          <a href="/abgeordnete/biografien/M/mueller_carsten-522166" target="_self">Carsten Müller</a>,
          <a href="/abgeordnete/biografien/N/noll_michaela-522382" target="_self">Michaela Noll</a>,
          <a href="/abgeordnete/biografien/R/roering_johannes-522980" target="_self">Johannes Röring</a>,
          <a href="/abgeordnete/biografien/S/schimke_jana-523268" target="_self">Jana Schimke</a>,
          <a href="/abgeordnete/biografien/S/sorge_tino-523744" target="_self">Tino Sorge</a>,
          <a href="/abgeordnete/biografien/Z/zimmer_matthias-524810" target="_self">Prof. Dr. Matthias Zimmer</a>]
```

In [11]:
```python
# but we are also interested in the function of the members:
html.find(class_='bt-standard-content').findAll('h4')
```

Out[11]:
```
[<h4>Fraktionsvorsitzender:</h4>,
 <h4>Erster Stellvertretender Fraktionsvorsitzender:</h4>,
 <h4>Stellvertretende Fraktionsvorsitzende:</h4>,
 <h4>Erster Parlamentarischer Geschäftsführer:</h4>,
 <h4>Stellvertreter des Ersten Parlamentarischen Geschäftsführers:</h4>,
 <h4>Parlamentarische Geschäftsführer:</h4>,
 <h4>Sprecher der CDU-Landesgruppen:</h4>,
 <h4>Vorsitzende der Arbeitsgruppen/Sprecher/Obleute:</h4>,
 <h4>Vorsitzende der sechs soziologischen Gruppen:</h4>,
 <h4>Beisitzer:</h4>]
```

```
In [12]: from urllib.parse import urljoin

         for role_node in html.find(class_='bt-standard-content').findAll('h4'):

             role = role_node.text.rstrip(':')

             for link_node in role_node.next_sibling.findAll('a'):
                 name = link_node.text
                 link = urljoin(request_url, link_node.get('href'))
                 print(role, name, link)
```

```
Fraktionsvorsitzender Ralph Brinkhaus https://www.bundestag.de/abgeordnete/biografien/B/brinkhaus_ralph-518692
Erster Stellvertretender Fraktionsvorsitzender Alexander Dobrindt https://www.bundestag.de/abgeordnete/biografien/D/dobr
Stellvertretende Fraktionsvorsitzende Gitta Connemann https://www.bundestag.de/abgeordnete/biografien/C/connemann_gitta-
Stellvertretende Fraktionsvorsitzende Thorsten Frei https://www.bundestag.de/abgeordnete/biografien/F/frei_thorsten-5195
Stellvertretende Fraktionsvorsitzende Hermann Gröhe https://www.bundestag.de/abgeordnete/biografien/G/groehe_hermann-519
Stellvertretende Fraktionsvorsitzende Andreas Jung https://www.bundestag.de/webarchiv/abgeordnete/biografien18/J/jung_an
Stellvertretende Fraktionsvorsitzende Ulrich Lange https://www.bundestag.de/abgeordnete/biografien/L/lange_ulrich-521486
Stellvertretende Fraktionsvorsitzende Dr. Katja Leikert https://www.bundestag.de/abgeordnete/biografien/L/leikert_katja-
Stellvertretende Fraktionsvorsitzende Dr. Carsten Linnemann https://www.bundestag.de/abgeordnete/biografien/L/linnemann_
Stellvertretende Fraktionsvorsitzende Nadine Schön https://www.bundestag.de/abgeordnete/biografien/S/schoen_nadine-52342
Stellvertretende Fraktionsvorsitzende Stephan Stracke https://www.bundestag.de/abgeordnete/biografien/S/stracke_stephan-
Stellvertretende Fraktionsvorsitzende Arnold Vaatz https://www.bundestag.de/abgeordnete/biografien/V/vaatz_arnold-524242
Stellvertretende Fraktionsvorsitzende Dr. Johann David Wadephul https://www.bundestag.de/abgeordnete/biografien/W/wadeph
Erster Parlamentarischer Geschäftsführer Michael Grosse-Brömer https://www.bundestag.de/abgeordnete/biografien/G/grosse_
Stellvertreter des Ersten Parlamentarischen Geschäftsführers Stefan Müller https://www.bundestag.de/abgeordnete/biografi
Parlamentarische Geschäftsführer Heike Brehmer https://www.bundestag.de/abgeordnete/biografien/B/brehmer_heike-518658
Parlamentarische Geschäftsführer Manfred Grund https://www.bundestag.de/abgeordnete/biografien/G/grund_manfred-519946
Parlamentarische Geschäftsführer Patrick Schnieder https://www.bundestag.de/abgeordnete/biografien/S/schnieder_patrick-5
Sprecher der CDU-Landesgruppen Eckhardt Rehberg https://www.bundestag.de/abgeordnete/biografien/R/rehberg_eckhardt-52282
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Michael Brand https://www.bundestag.de/abgeordnete/biografien/B/brand_mi
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Marie-Luise Dött https://www.bundestag.de/abgeordnete/biografien/D/doett
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Eberhard Gienger https://www.bundestag.de/abgeordnete/biografien/G/gieng
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Florian Hahn https://www.bundestag.de/abgeordnete/biografien/H/hahn_flor
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Jürgen Hardt https://www.bundestag.de/abgeordnete/biografien/H/hardt_jue
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Volkmar Klein https://www.bundestag.de/abgeordnete/biografien/K/klein_vo
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Paul Lehrieder https://www.bundestag.de/abgeordnete/biografien/L/lehried
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Karin Maag https://www.bundestag.de/abgeordnete/biografien/M/maag_karin-
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Dr. Mathias Middelberg https://www.bundestag.de/abgeordnete/biografien/M
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Elisabeth Motschmann https://www.bundestag.de/abgeordnete/biografien/M/m
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Henning Otte https://www.bundestag.de/abgeordnete/biografien/O/otte_henn
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Dr. Joachim Pfeiffer https://www.bundestag.de/abgeordnete/biografien/P/p
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Eckhardt Rehberg https://www.bundestag.de/abgeordnete/biografien/R/rehbe
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Albert Rupprecht https://www.bundestag.de/abgeordnete/biografien/R/ruppr
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Tankred Schipanski https://www.bundestag.de/abgeordnete/biografien/S/sch
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Albert Stegemann https://www.bundestag.de/abgeordnete/biografien/S/stege
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Gero Storjohann https://www.bundestag.de/abgeordnete/biografien/S/storjo
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Antje Tillmann https://www.bundestag.de/abgeordnete/biografien/T/tillman
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Kai Wegner https://www.bundestag.de/abgeordnete/biografien/W/wegner_kai-
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Marcus Weinberg https://www.bundestag.de/abgeordnete/biografien/W/weinbe
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Peter Weiß https://www.bundestag.de/abgeordnete/biografien/W/weiss_peter
Vorsitzende der Arbeitsgruppen/Sprecher/Obleute Elisabeth Winkelmeier-Becker https://www.bundestag.de/abgeordnete/biogra
Vorsitzende der sechs soziologischen Gruppen Christian Haase https://www.bundestag.de/abgeordnete/biografien/H/haase_chr
Vorsitzende der sechs soziologischen Gruppen Emmi Zeulner https://www.bundestag.de/abgeordnete/biografien/Z/zeulner_emmi
Vorsitzende der sechs soziologischen Gruppen Yvonne Magwas https://www.bundestag.de/abgeordnete/biografien/M/magwas_yvon
Vorsitzende der sechs soziologischen Gruppen Eckhard Pols https://www.bundestag.de/abgeordnete/biografien/P/pols_eckhard
Vorsitzende der sechs soziologischen Gruppen Uwe, Schummer https://www.bundestag.de/abgeordnete/biografien/S/schummer_uw
Vorsitzende der sechs soziologischen Gruppen Christian Freiherr von Stetten https://www.bundestag.de/abgeordnete/biograf
Beisitzer Axel E. Fischer https://www.bundestag.de/abgeordnete/biografien/F/fischer_axel-519454
Beisitzer Olav Gutting https://www.bundestag.de/abgeordnete/biografien/G/gutting_olav-519978
Beisitzer Fritz Güntzler https://www.bundestag.de/abgeordnete/biografien/G/guentzler_fritz-519962
Beisitzer Matthias Heider https://www.bundestag.de/abgeordnete/biografien/H/heider_matthias-520192
Beisitzer Thomas Heilmann https://www.bundestag.de/abgeordnete/biografien/H/520204-520204
Beisitzer Mark Helfrich https://www.bundestag.de/abgeordnete/biografien/H/helfrich_mark-520256
Beisitzer Rudolf Henke https://www.bundestag.de/abgeordnete/biografien/H/henke_rudolf-520294
Beisitzer Karl Holmeier https://www.bundestag.de/abgeordnete/biografien/H/holmeier_karl-520494
Beisitzer Roderich Kiesewetter https://www.bundestag.de/abgeordnete/biografien/K/kiesewetter_roderich-520990
Beisitzer Jan Metzler https://www.bundestag.de/abgeordnete/biografien/M/metzler_jan-521988
Beisitzer Dr. h. c. Hans Michelbach https://www.bundestag.de/abgeordnete/biografien/M/michelbach_hans-522016
Beisitzer Carsten Müller https://www.bundestag.de/abgeordnete/biografien/M/mueller_carsten-522166
Beisitzer Michaela Noll https://www.bundestag.de/abgeordnete/biografien/N/noll_michaela-522382
Beisitzer Johannes Röring https://www.bundestag.de/abgeordnete/biografien/R/roering_johannes-522980
Beisitzer Jana Schimke https://www.bundestag.de/abgeordnete/biografien/S/schimke_jana-523268
Beisitzer Tino Sorge https://www.bundestag.de/abgeordnete/biografien/S/sorge_tino-523744
Beisitzer Prof. Dr. Matthias Zimmer https://www.bundestag.de/abgeordnete/biografien/Z/zimmer_matthias-524810
```

Now we put everything together, so that we can run this for all factions of the parliament: - we use a function to - fetch the page of the faction and - extract the members from the page content - iterate over all factions - store the list of faction roles and MPs in a data frame and CSV

```
In [13]: %%script false --no-raise-error
         # uncomment the above instruction to run this code
         # note: do not run the cell by default
         # because sending 6 HTTP requests may take long

         import requests
         from time import sleep
         from urllib.parse import urljoin
         from bs4 import BeautifulSoup
         import pandas as pd

         request_base_url = 'https://www.bundestag.de/parlament/fraktionen/'
         factions = 'cducsu spd fdp linke gruene afd'.split()

         def get_members_of_faction(faction):
             global request_base_url
```

```
    url = request_base_url + faction

    response = requests.get(url)
    if not response.ok:
        return

    result = []

    html = BeautifulSoup(response.text)

    for role_node in html.find(class_='bt-standard-content').findAll('h4'):

        role = role_node.text.strip().rstrip(':')

        if not role_node.next_sibling:
            continue

        for link_node in role_node.next_sibling.findAll('a'):
            name = link_node.text
            link = urljoin(url, link_node.get('href'))
            result.append([name, faction, role, link])

    return result


faction_roles = []

for faction in factions:
    if faction_roles:
        # be polite and wait before the next request
        sleep(5)
    faction_roles += get_members_of_faction(faction)

df_faction_roles = pd.DataFrame(faction_roles, columns=['name', 'faction', 'role', 'link'])

df_faction_roles.to_csv('data/bundestag/faction_roles.csv')
```

```
In [14]: import pandas as pd

         df_faction_roles = pd.read_csv('data/bundestag/faction_roles.csv')
         df_faction_roles.value_counts('faction')
```

```
Out[14]: faction
         cducsu    64
         spd       37
         linke     22
         gruene    12
         afd       12
         fdp       11
         dtype: int64
```

```
In [15]: # not all members of the parliament have a role in their faction
         # and are listed on the landing page of the faction
         df_faction_roles.shape

Out[15]: (158, 5)

In [16]: df_faction_roles[df_faction_roles['role'].str.startswith('Fraktionsvorsitz')]

Out[16]:      Unnamed: 0                   name faction                   role  \
         0             0        Ralph Brinkhaus  cducsu  Fraktionsvorsitzender
         64           64         Rolf Mützenich     spd  Fraktionsvorsitzender
         101         101      Christian Lindner     fdp  Fraktionsvorsitzender
         112         112      Amira Mohamed Ali   linke   Fraktionsvorsitzende
         113         113     Dr. Dietmar Bartsch   linke   Fraktionsvorsitzende
         134         134  Katrin Göring-Eckardt  gruene   Fraktionsvorsitzende
         135         135    Dr. Anton Hofreiter  gruene   Fraktionsvorsitzende
         146         146  Dr. Alexander Gauland     afd   Fraktionsvorsitzende
         147         147       Dr. Alice Weidel     afd   Fraktionsvorsitzende

                                                          link
         0    https://www.bundestag.de/abgeordnete/biografie…
         64   https://www.bundestag.de/abgeordnete/biografie…
         101  https://www.bundestag.de/abgeordnete/biografie…
         112  https://www.bundestag.de/abgeordnete/biografie…
         113  https://www.bundestag.de/abgeordnete/biografie…
         134  https://www.bundestag.de/abgeordnete/biografie…
         135  https://www.bundestag.de/abgeordnete/biografie…
         146  https://www.bundestag.de/abgeordnete/biografie…
         147  https://www.bundestag.de/abgeordnete/biografie…

In [17]: # now let's try whether we can fetch the biography and other information of a single MP

         member_url = df_faction_roles.loc[df_faction_roles['name']=='Andreas Jung','link'].values[0]
         member_response = requests.get(member_url)
         member_html = BeautifulSoup(member_response.text)

         # let's try first using the CSS class "bundestag-standard-content"
         for node in member_html.findAll(class_='bt-standard-content'):
             print(node.text)




Abgeordnetenbüro
Deutscher BundestagPlatz der Republik 111011 Berlin
```

Geboren am 13. Mai 1975 in Freiburg im Breisgau; aufgewachsen in Stockach am Bodensee; katholisch; verheiratet; ein Kind

Baden-Württemberg

Wahlkreis 287: Konstanz

Vorsitzender

Parlamentarischer Beirat für nachhaltige Entwicklung

Ordentliches Mitglied

Ausschuss für Wirtschaft und Energie

Parlamentarischer Beirat für nachhaltige Entwicklung

Stellvertretendes Mitglied

Ausschuss für Umwelt, Naturschutz, Bau und Reaktorsicherheit

Ordentliches Mitglied

Funktionen in Unternehmen

```
Mainau GmbH, Insel Mainau, Mitglied des Beirates
Funktionen in Vereinen, Verbänden und Stiftungen

Bundesverband Erneuerbare Energie e.V. (BEE), Berlin, Vorsitzender des Parlamentarischen Beirates

Deutsch-Französisches Institut e.V. (dfi), Ludwigsburg, Mitglied des Vorstandes

Fondation Entente Franco-Allemande (FEFA), Straßburg, Mitglied des Verwaltungsrates

GLOBE Deutschland, Berlin, Präsident

Stiftung Energie & Klimaschutz Baden-Württemberg, Karlsruhe, Mitglied des Kuratoriums

Vereinigung Deutsch-Französischer Gesellschaften für Europa e.V. (VDFG), Mainz, Mitglied des Parlamentarischen Beirates
Für die Höhe der Einkünfte sind nach den Verhaltensregeln die geleisteten Bruttobeträge einschließlich Entschädigungs-,
Zum Inhalt der veröffentlichungspflichtigen Angaben im Übrigen siehe auch die Hinweise zur Veröffentlichung der Angaben
```

4.2.1 Automatic Cleansing of Text

A trivial extraction of all text in the body of web page would include a lot of unwanted content (navigation menus, header, footer, side bars), the "main" content could be even only a small part in the middle of the page. There are heuristics and algorithms for automatic removal of "boilerplate" content:

- Mozilla Readability: the reader view of the Firefox browser
  - originally implemented in JavaScript, see Readability.js
  - but there is a Python port - ReadabiliPy or ReadabiliPy on pypi
- jusText or jusText on pypi

Here an example usage of ReadabiliPy with the latest fetched page (without any manual selection of elements by CSS class):

```
In [18]: !pip install readabilipy

         from readabilipy import simple_json_from_html_string

         article = simple_json_from_html_string(member_response.text, use_readability=True)
```

```
Requirement already satisfied: readabilipy in ./.venv/lib/python3.9/site-packages (0.2.0)
Requirement already satisfied: lxml in ./.venv/lib/python3.9/site-packages (from readabilipy) (4.6.3)
Requirement already satisfied: html5lib in ./.venv/lib/python3.9/site-packages (from readabilipy) (1.1)
Requirement already satisfied: regex in ./.venv/lib/python3.9/site-packages (from readabilipy) (2021.7.6)
Requirement already satisfied: beautifulsoup4>=4.7.1 in ./.venv/lib/python3.9/site-packages (from readabilipy) (4.9.3)
Requirement already satisfied: soupsieve>1.2 in ./.venv/lib/python3.9/site-packages (from beautifulsoup4>=4.7.1->readabi
Requirement already satisfied: webencodings in ./.venv/lib/python3.9/site-packages (from html5lib->readabilipy) (0.5.1)
Requirement already satisfied: six>=1.9 in ./.venv/lib/python3.9/site-packages (from html5lib->readabilipy) (1.15.0)
```

```
In [19]: for paragraph in article['plain_text']:
             print(paragraph['text'])
             print()
```

```
Geboren am 13. Mai 1975 in Freiburg im Breisgau; aufgewachsen in Stockach am Bodensee; katholisch; verheiratet; ein Kind

1981 bis 1985 Grundschule Stockach; 1985 bis 1994 Nellenburggymnasium Stockach; 1994 bis 2000 Studium der Rechtswissensc

1990 bis 2010 Mitglied der Jungen Union; 1991 bis 1993 Ortsvorsitzender der Jungen Union Stockach; 1993 bis 1999 Kreisvo

Seit 1993 Mitglied der CDU; 1995 bis 2011 gewähltes Mitglied im Kreisvorstand des CDU Kreisverbandes Konstanz; 2007 bis

18. September 2005 Wahl zum Bundestagsabgeordneten für den Wahlkreis Konstanz; Ordentliches Mitglied im Ausschuss für Um

27. September 2009 Wiederwahl zum Bundestagsabgeordneten für den Wahlkreis Konstanz; Ordentliches Mitglied im Ausschuss

22. September 2013 Wiederwahl zum Bundestagsabgeordneten für den WahlkreisKonstanz, Ordentliches Mitglied im Ausschuss f

Seit 6. Februar 2015 Vorsitzender der Deutsch-Französischen Parlamentariergruppe.

Seit 4. Juli 2016 Vorsitzender der CDU-Landesgruppe Baden-Württemberg im Deutschen Bundestag.
```

```
In [20]: # but there's also a "readable" and simple HTML snippet
         # (shown as rendered HTML in the output)
         from IPython.core.display import HTML

         HTML(article['plain_content'])

Out[20]: <IPython.core.display.HTML object>
```

4.3 Processing XML

The Open Data portal of the German parliament offers a zip file “Stammdaten aller Abgeordneten seit 1949 im XML-Format (Stand 12.03.2021)” for free download. Most likely we should get the information about all PMs from this source. But how do we process XML?

Assumed the zip archive has been downloaded, unzipped and the files are all placed in `data/bundestag/`, we can simply read the file and pass it to beautifulsoup which will parse it. But we request a specific parser feature (`lxml-xml`) so that the casing of XML elements is preserved.

```
In [21]: from bs4 import BeautifulSoup

         xml = BeautifulSoup(open('data/bundestag/MDB_STAMMDATEN.XML').read(),
                             features='lxml-xml')

         xml.MDB
```

Out[21]:

```
<MDB>
<ID>11000001</ID>
<NAMEN>
<NAME>
<NACHNAME>Abelein</NACHNAME>
<VORNAME>Manfred</VORNAME>
<ORTSZUSATZ/>
<ADEL/>
<PRAEFIX/>
<ANREDE_TITEL>Dr.</ANREDE_TITEL>
<AKAD_TITEL>Prof. Dr.</AKAD_TITEL>
<HISTORIE_VON>19.10.1965</HISTORIE_VON>
<HISTORIE_BIS/>
</NAME>
</NAMEN>
<BIOGRAFISCHE_ANGABEN>
<GEBURTSDATUM>20.10.1930</GEBURTSDATUM>
<GEBURTSORT>Stuttgart</GEBURTSORT>
<GEBURTSLAND/>
<STERBEDATUM>17.01.2008</STERBEDATUM>
<GESCHLECHT>männlich</GESCHLECHT>
<FAMILIENSTAND>keine Angaben</FAMILIENSTAND>
<RELIGION>katholisch</RELIGION>
<BERUF>Rechtsanwalt, Wirtschaftsprüfer, Universitätsprofessor</BERUF>
<PARTEI_KURZ>CDU</PARTEI_KURZ>
<VITA_KURZ/>
<VEROEFFENTLICHUNGSPFLICHTIGES/>
</BIOGRAFISCHE_ANGABEN>
<WAHLPERIODEN>
<WAHLPERIODE>
<WP>5</WP>
<MDBWP_VON>19.10.1965</MDBWP_VON>
<MDBWP_BIS>19.10.1969</MDBWP_BIS>
<WKR_NUMMER>174</WKR_NUMMER>
<WKR_NAME/>
<WKR_LAND>BWG</WKR_LAND>
<LISTE/>
<MANDATSART>Direktwahl</MANDATSART>
<INSTITUTIONEN>
<INSTITUTION>
<INSART_LANG>Fraktion/Gruppe</INSART_LANG>
<INS_LANG>Fraktion der Christlich Demokratischen Union/Christlich - Sozialen Union</INS_LANG>
<MDBINS_VON/>
<MDBINS_BIS/>
<FKT_LANG/>
<FKTINS_VON/>
<FKTINS_BIS/>
</INSTITUTION>
</INSTITUTIONEN>
</WAHLPERIODE>
<WAHLPERIODE>
```

```
<WP>6</WP>
<MDBWP_VON>20.10.1969</MDBWP_VON>
<MDBWP_BIS>22.09.1972</MDBWP_BIS>
<WKR_NUMMER>174</WKR_NUMMER>
<WKR_NAME/>
<WKR_LAND>BWG</WKR_LAND>
<LISTE/>
<MANDATSART>Direktwahl</MANDATSART>
<INSTITUTIONEN>
<INSTITUTION>
<INSART_LANG>Fraktion/Gruppe</INSART_LANG>
<INS_LANG>Fraktion der Christlich Demokratischen Union/Christlich - Sozialen Union</INS_LANG>
<MDBINS_VON/>
<MDBINS_BIS/>
<FKT_LANG/>
<FKTINS_VON/>
<FKTINS_BIS/>
</INSTITUTION>
</INSTITUTIONEN>
</WAHLPERIODE>
<WAHLPERIODE>
<WP>7</WP>
<MDBWP_VON>13.12.1972</MDBWP_VON>
<MDBWP_BIS>13.12.1976</MDBWP_BIS>
<WKR_NUMMER>174</WKR_NUMMER>
<WKR_NAME/>
<WKR_LAND>BWG</WKR_LAND>
<LISTE/>
<MANDATSART>Direktwahl</MANDATSART>
<INSTITUTIONEN>
<INSTITUTION>
<INSART_LANG>Fraktion/Gruppe</INSART_LANG>
<INS_LANG>Fraktion der Christlich Demokratischen Union/Christlich - Sozialen Union</INS_LANG>
<MDBINS_VON/>
<MDBINS_BIS/>
<FKT_LANG/>
<FKTINS_VON/>
<FKTINS_BIS/>
</INSTITUTION>
</INSTITUTIONEN>
</WAHLPERIODE>
<WAHLPERIODE>
<WP>8</WP>
<MDBWP_VON>14.12.1976</MDBWP_VON>
<MDBWP_BIS>04.11.1980</MDBWP_BIS>
<WKR_NUMMER>174</WKR_NUMMER>
<WKR_NAME/>
<WKR_LAND>BWG</WKR_LAND>
<LISTE/>
<MANDATSART>Direktwahl</MANDATSART>
<INSTITUTIONEN>
```

```
<INSTITUTION>
<INSART_LANG>Fraktion/Gruppe</INSART_LANG>
<INS_LANG>Fraktion der Christlich Demokratischen Union/Christlich - Sozialen Union</INS_LANG>
<MDBINS_VON/>
<MDBINS_BIS/>
<FKT_LANG/>
<FKTINS_VON/>
<FKTINS_BIS/>
</INSTITUTION>
</INSTITUTIONEN>
</WAHLPERIODE>
<WAHLPERIODE>
<WP>9</WP>
<MDBWP_VON>04.11.1980</MDBWP_VON>
<MDBWP_BIS>29.03.1983</MDBWP_BIS>
<WKR_NUMMER>174</WKR_NUMMER>
<WKR_NAME/>
<WKR_LAND>BWG</WKR_LAND>
<LISTE/>
<MANDATSART>Direktwahl</MANDATSART>
<INSTITUTIONEN>
<INSTITUTION>
<INSART_LANG>Fraktion/Gruppe</INSART_LANG>
<INS_LANG>Fraktion der Christlich Demokratischen Union/Christlich - Sozialen Union</INS_LANG>
<MDBINS_VON/>
<MDBINS_BIS/>
<FKT_LANG/>
<FKTINS_VON/>
<FKTINS_BIS/>
</INSTITUTION>
</INSTITUTIONEN>
</WAHLPERIODE>
<WAHLPERIODE>
<WP>10</WP>
<MDBWP_VON>29.03.1983</MDBWP_VON>
<MDBWP_BIS>18.02.1987</MDBWP_BIS>
<WKR_NUMMER>174</WKR_NUMMER>
<WKR_NAME/>
<WKR_LAND>BWG</WKR_LAND>
<LISTE/>
<MANDATSART>Direktwahl</MANDATSART>
<INSTITUTIONEN>
<INSTITUTION>
<INSART_LANG>Fraktion/Gruppe</INSART_LANG>
<INS_LANG>Fraktion der Christlich Demokratischen Union/Christlich - Sozialen Union</INS_LANG>
<MDBINS_VON/>
<MDBINS_BIS/>
<FKT_LANG/>
<FKTINS_VON/>
<FKTINS_BIS/>
</INSTITUTION>
```

```
</INSTITUTIONEN>
</WAHLPERIODE>
<WAHLPERIODE>
<WP>11</WP>
<MDBWP_VON>18.02.1987</MDBWP_VON>
<MDBWP_BIS>20.12.1990</MDBWP_BIS>
<WKR_NUMMER>174</WKR_NUMMER>
<WKR_NAME/>
<WKR_LAND>BWG</WKR_LAND>
<LISTE/>
<MANDATSART>Direktwahl</MANDATSART>
<INSTITUTIONEN>
<INSTITUTION>
<INSART_LANG>Fraktion/Gruppe</INSART_LANG>
<INS_LANG>Fraktion der Christlich Demokratischen Union/Christlich - Sozialen Union</INS_LANG>
<MDBINS_VON>18.02.1987</MDBINS_VON>
<MDBINS_BIS>20.12.1990</MDBINS_BIS>
<FKT_LANG/>
<FKTINS_VON/>
<FKTINS_BIS/>
</INSTITUTION>
</INSTITUTIONEN>
</WAHLPERIODE>
</WAHLPERIODEN>
</MDB>
```

```
In [22]: len(xml.findAll('MDB'))

Out[22]: 4089

In [23]: from collections import Counter

         mp_acad_title = Counter()

         mp_with_acad_title, mp_total = 0, 0

         for mp in xml.findAll('MDB'):

             mp_total += 1

             has_academic_title = False
             for nn in mp.findAll("NAME"):
                 if nn.AKAD_TITEL.text:
                     has_academic_title = True
                     mp_acad_title[nn.AKAD_TITEL.text] += 1

             if has_academic_title:
                 # count a title only once (in case of multiple names)
                 mp_with_acad_title += 1

         mp_with_acad_title / mp_total

Out[23]: 0.2582538517975055
```

```
In [24]: mp_acad_title.most_common()

Out[24]: [('Dr.', 930),
          ('Prof. Dr.', 81),
          ('Dr. h. c.', 42),
          ('Dr. Dr. h. c.', 17),
          ('Prof.', 13),
          ('Dr. - Ing.', 11),
          ('Prof. Dr. h. c.', 3),
          ('Dipl. - Ing.', 3),
          ('Dr. Dr.', 3),
          ('Prof. Dr. Dr. h. c.', 3),
          ('Dr. - Ing. e. h.', 2),
          ('Prof. Dr. Dr.', 2),
          ('Dr. - Ing. Dr. h. c.', 1),
          ('Prof. h. c.', 1),
          ('Prof. Dr. - Ing.', 1),
          ('Dr. h. c. Dr. - Ing. e. h.', 1),
          ('Dr. - Ing. Dr. - Ing. e. h. Dr. h. c.', 1),
          ('Dr. h. c. Dr. e. h.', 1),
          ('Prof. h. c. Dr.', 1),
          ('Dr. h. c. (Univ Kyiv)', 1),
          ('HonD', 1),
          ('Dr. h. c. (NUACA)', 1)]
```

A final note: Reading the XML file describing the members of the German parliament into a tabular data structure will be painful (similar as for JSON data source) because of - the nested structure - some list-like data, for example the fact that one MP can have multiple names

Instead of coding the conversion in Python: with XSLT there is a dedicated language for transforming XML documents into other document formats.

The Open Discourse projects hosts the proceedings of the German parliament and also a list of MPs in data formats easy to consume. See the Open Discourse data sets page.

4.4 Browser automation with Python

- Selenium
  - nice example: impf-botpy
- Playwright
  - Playwright on pypi including nice examples (some cited below)
  - Python API docs

Note: Playwright does not run in a Jupyter notebook. We'll run the scripts directly in the Python interpreter.

Installation:

```
pip install playwright
playwright install
```

Take a screenshot using two different browsers:

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    for browser_type in [p.chromium, p.firefox]:
        browser = browser_type.launch()
        page = browser.new_page()
        page.goto('http://whatsmyuseragent.org/')
        _ = page.screenshot(path=f'figures/example-{browser_type.name}.png')
        browser.close()
```

Just run the script `scripts/playwright_whatsmyuseragent_screenshot.py` in the console / shell:

```
python ./scripts/playwright_whatsmyuseragent_screenshot.py
```

The screenshots are then found in the folder `figures/` for chromium and firefox.

Playwright can record user interactions (mouse clicks, keyboard input) and create Python code to replay the recorded actions:

```
playwright codegen https://www.bundestag.de/abgeordnete/biografien
```

The created Python code is then modified, here to loop over all overlays showing the members of the parliament:

```
from time import sleep

from playwright.sync_api import sync_playwright

def run(playwright):
    browser = playwright.chromium.launch(headless=False)
    context = browser.new_context(viewport={'height': 1080, 'width': 1920})
    page = context.new_page()
    page.goto("https://www.bundestag.de/abgeordnete/biografien")
    while True:
        try:
            sleep(3)
            page.click("button:has-text(\"Vor\")")
        except Exception:
            break

with sync_playwright() as p:
    run(p)
```

Again: best run the replay script in the console:

```
python ./scripts/playwright_replay.py
```

5 Text Processing and Machine Learning

- pre-processing and tokenization (splitting text into words)
- n-grams, vectorization and word embeddings
- train and evaluate a text classifier
- a short look into Hugging Face's transformers library

5.1 Natural Language Processing

Natural language processing (NLP) is about programming computers to process and analyze natural language data (text and speech).

During the text classification training we touch only some aspects of NLP, namely

- tokenization or splitting a text into words (aka. tokens)
- the representation of words in a vector space (word embeddings)

NLP modules for Python:

- spaCy or spaCy on pypi
- NLTK or NLTK on pypi

5.2 Machine Learning

The field of machine learning is too broad to be introduced here. Please, see Google's machine learning crash course.

5.3 fastText

fastText is a software library for text classification and word representation learning. See the fastText tutorials for

- text classification
- word representation learning

We will now follow the fastText text classification tutorial (cf. documentation of the Python module "fasttext") to train and apply a text classifier.

The fastText tutorial uses the StackExchange cooking data set. We will use the Kaggle Toxic Comment Classification Challenge data set. In order to download the data set, you need to register at Kaggle.com.

After the data set is downloaded and unpacked into the folder `data/kaggle-jigsaw-toxic`, you should see the tree files `train.csv`, `test.csv` and `test_labels.csv` in the mentioned folder.

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

        df_train = pd.read_csv('data/kaggle-jigsaw-toxic/train.csv')

        #df.head()
```

```
In [2]: labels = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']

        df_train[labels].mean()
```

```
Out[2]: toxic            0.095844
        severe_toxic     0.009996
        obscene          0.052948
        threat           0.002996
        insult           0.049364
        identity_hate    0.008805
        dtype: float64
```

Only 10% of the comments are toxic. What does it mean for building a classifier?

```
In [3]: # tokenize the comments
        import string

        from nltk.tokenize import TweetTokenizer

        tweet_tokenizer = TweetTokenizer(reduce_len=True)

        def tokenize(text):
            global tweet_tokenizer
            words = tweet_tokenizer.tokenize(text)
            words = filter(lambda w: w != ''
                                     and w not in string.punctuation, words)
            words = map(lambda w: w.lower(), words)
            return ' '.join(words)

        tokenize("You're a hero! http://example.com/index.html")
```

```
Out[3]: "you're a hero http://example.com/index.html"
```

```
In [4]: # write data to fastText train file

        train_file = 'data/kaggle-jigsaw-toxic/train.txt'

        def write_line_fasttext(fp, row):
            global labels
            line = ''
            for label in labels:
                if row[label] == 1:
                    if line:
                        line += ' '
                    line += '__label__' + label
            if line:
                line += ' '
            else:
                line += '__label__none '
            line += tokenize(row['comment_text'])
            fp.write(line)
            fp.write('\n')

        with open(train_file, 'w') as fp:
            df_train.apply(lambda row: write_line_fasttext(fp, row), axis=1)
```

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

```
In [5]: # train a model

        import fasttext

        model = fasttext.train_supervised(input=train_file, wordNgrams=2, minCount=2)
```

```
In [6]: model.predict(tokenize("This is a well-written article."))
        # model.predict(tokenize("Fuck you!"), k=5)

Out[6]: (('__label__none',), array([0.99993789]))
```

```
In [7]: # looking into the underlying word embeddings

        model.get_nearest_neighbors('idiot', k=20)

Out[7]: [(0.9997914433479309, 'stupid'),
         (0.9996288418769836, 'moron'),
         (0.9995864033699036, 'jerk'),
         (0.9993796348571777, 'arrogant'),
         (0.9993292093276978, 'ignorant'),
         (0.999278724193573, 'stupidity'),
         (0.9992066025733948, 'coward'),
         (0.9992029070854187, 'disgusting'),
         (0.9991973638534546, 'idiotic'),
         (0.9990672469139099, 'pathetic'),
         (0.9990224242210388, 'fool'),
         (0.9989080429077148, 'morons'),
         (0.9989030957221985, 'losers'),
         (0.9988322854042053, 'hell'),
         (0.9988279342651367, 'jackass'),
         (0.9987922310829163, 'fascist'),
         (0.9987281560897827, 'idiots'),
         (0.9987263679504395, 'dirty'),
         (0.9987045526504517, 'sucked'),
         (0.998673141002655, 'bloody')]
```

```
In [8]: # save the model
        model_file = 'data/kaggle-jigsaw-toxic/model.bin'

        model.save_model(model_file)
```

```
In [9]: df_test = pd.read_csv('data/kaggle-jigsaw-toxic/test.csv')
        df_test_labels = pd.read_csv('data/kaggle-jigsaw-toxic/test_labels.csv')

        # join both tables
        df_test = df_test.merge(df_test_labels, on='id')

        # skip rows not labelled / not used
        df_test = df_test[df_test['toxic'] != -1]

        test_file = 'data/kaggle-jigsaw-toxic/test.txt'

        # write test set for fastText
        with open(test_file, 'w') as fp:
            df_test.apply(lambda row: write_line_fasttext(fp, row), axis=1)
```

5.3.1 Model Validation

See also: precision and recall

```
In [10]: model.test(test_file)

Out[10]: (63978, 0.9303666885491888, 0.8240416430163499)

In [11]: res_per_label = model.test_label(test_file)


         for label in res_per_label.items():
             print(label)

('__label__threat', {'precision': nan, 'recall': 0.0, 'f1score': 0.0})
('__label__identity_hate', {'precision': nan, 'recall': 0.0, 'f1score': 0.0})
('__label__severe_toxic', {'precision': 0.275, 'recall': 0.05994550408719346, 'f1score': 0.09843400447427293})
('__label__insult', {'precision': 0.7333333333333333, 'recall': 0.0032098044937262913, 'f1score': 0.006391632771644393})
('__label__obscene', {'precision': 0.9406952965235174, 'recall': 0.12462747222974803, 'f1score': 0.22009569377990432})
('__label__toxic', {'precision': 0.5887384176764077, 'recall': 0.6781609195402298, 'f1score': 0.6302937809996184})
('__label__none', {'precision': 0.9737668280742829, 'recall': 0.950896336710834, 'f1score': 0.9621956990378043})


In [12]: # in case the fastText command-line tool is installed: it has a nice output formatter
         !fasttext test-label \
            data/kaggle-jigsaw-toxic/model.bin \
            data/kaggle-jigsaw-toxic/test.txt

F1-Score : 0.962196  Precision : 0.973767  Recall : 0.950896   __label__none
F1-Score : 0.630294  Precision : 0.588738  Recall : 0.678161   __label__toxic
F1-Score : 0.220096  Precision : 0.940695  Recall : 0.124627   __label__obscene
F1-Score : 0.006392  Precision : 0.733333  Recall : 0.003210   __label__insult
F1-Score : 0.098434  Precision : 0.275000  Recall : 0.059946   __label__severe_toxic
F1-Score : 0.000000  Precision : --------  Recall : 0.000000   __label__identity_hate
F1-Score : 0.000000  Precision : --------  Recall : 0.000000   __label__threat
N        63978
P@1        0.930
R@1        0.824
```

5.4 Transformers

- https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)
- Hugging Face's transformers library: unique interface and provisioning of various transformer language models
  - see https://huggingface.co/course

```
In [14]: !pip install transformers
         !pip install tensorflow
         !pip install "transformers[sentencepiece]"
```

```
In [15]: from transformers import pipeline

         p = pipeline('fill-mask', model='bert-base-german-cased')
```

```
Some weights of the model checkpoint at bert-base-german-cased were not used when initializing BertForMaskedLM: ['cls.se
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model trained on another task or wit
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model that you expect to be exac
```

```
In [16]: for s in p("Er arbeitet als [MASK]."): print(s)
```

```
{'sequence': 'Er arbeitet als Rechtsanwalt.', 'score': 0.09919334203004837, 'token': 6143, 'token_str': 'Rechtsanwalt'}
{'sequence': 'Er arbeitet als Trainer.', 'score': 0.07836302369832993, 'token': 3674, 'token_str': 'Trainer'}
{'sequence': 'Er arbeitet als Journalist.', 'score': 0.0628521665930748, 'token': 10486, 'token_str': 'Journalist'}
{'sequence': 'Er arbeitet als Anwalt.', 'score': 0.05725342780351639, 'token': 6938, 'token_str': 'Anwalt'}
{'sequence': 'Er arbeitet als Schauspieler.', 'score': 0.05046413466334343, 'token': 5607, 'token_str': 'Schauspieler'}
```

```
In [17]: pipeline_fill_mask = pipeline('fill-mask', model='bert-base-german-cased')

         def fill_mask(cloze):
             global pipeline_fill_mask
             for s in pipeline_fill_mask(cloze):
                 print('%-20s\t%.5f' % (s['token_str'], s['score']))
```

```
Some weights of the model checkpoint at bert-base-german-cased were not used when initializing BertForMaskedLM: ['cls.se
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model trained on another task or wit
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model that you expect to be exac
```

```
In [18]: fill_mask("Er arbeitet als [MASK] in einer Klinik.")
```

```
Arzt                    0.61843
Angestellter            0.04225
Koch                    0.03064
Assistent               0.02001
Mediziner               0.01900
```

```
In [19]: fill_mask("Er arbeitet als [MASK] in einer Lungenklinik.")
```

```
Arzt                        0.69560
Angestellter                0.03423
Chemiker                    0.02711
Facharzt                    0.02113
Mediziner                   0.02024
```

```
In [20]: fill_mask("Er arbeitet als [MASK] bei BMW.")
```

```
Ingenieur                   0.18871
Berater                     0.17160
Manager                     0.15090
Geschäftsführer             0.07775
Trainer                     0.04951
```

```
In [21]: fill_mask("Er arbeitet als [MASK] an der Universität Konstanz.")
```

```
Professor                   0.74687
Dozent                      0.11445
Hochschullehrer             0.08565
Wissenschaftler             0.00667
Assistent                   0.00427
```

```
In [22]: fill_mask("Sie arbeitet als [MASK] an der Universität Konstanz.")
```

```
Professor                   0.52318
Lehrerin                    0.09859
Dozent                      0.08542
Professur                   0.04144
Richterin                   0.02292
```

```
In [23]: fill_mask("Sie ist wirklich [MASK].")
```

```
schön                       0.11005
jung                        0.06098
glücklich                   0.05704
toll                        0.05053
gut                         0.03495
```

```
In [24]: fill_mask("Er ist wirklich [MASK].")
```

```
gut                         0.05452
glücklich                   0.05183
da                          0.03765
jung                        0.03233
tot                         0.03229
```

```
In [25]: help(pipeline)
```

```
Help on function pipeline in module transformers.pipelines:

pipeline(task: str, model: Optional = None, config: Union[str, transformers.configuration_utils.PretrainedConfig, NoneTy
    Utility factory method to build a :class:`~transformers.Pipeline`.

    Pipelines are made of:

        - A :doc:`tokenizer <tokenizer>` in charge of mapping raw textual input to token.
        - A :doc:`model <model>` to make predictions from the inputs.
        - Some (optional) post processing for enhancing model's output.

    Args:
        task (:obj:`str`):
            The task defining which pipeline will be returned. Currently accepted tasks are:

            - :obj:`"feature-extraction"`: will return a :class:`~transformers.FeatureExtractionPipeline`.
            - :obj:`"text-classification"`: will return a :class:`~transformers.TextClassificationPipeline`.
            - :obj:`"sentiment-analysis"`: (alias of :obj:`"text-classification") will return a
              :class:`~transformers.TextClassificationPipeline`.
            - :obj:`"token-classification"`: will return a :class:`~transformers.TokenClassificationPipeline`.
            - :obj:`"ner"` (alias of :obj:`"token-classification"): will return a
              :class:`~transformers.TokenClassificationPipeline`.
            - :obj:`"question-answering"`: will return a :class:`~transformers.QuestionAnsweringPipeline`.
            - :obj:`"fill-mask"`: will return a :class:`~transformers.FillMaskPipeline`.
            - :obj:`"summarization"`: will return a :class:`~transformers.SummarizationPipeline`.
            - :obj:`"translation_xx_to_yy"`: will return a :class:`~transformers.TranslationPipeline`.
            - :obj:`"text2text-generation"`: will return a :class:`~transformers.Text2TextGenerationPipeline`.
            - :obj:`"text-generation"`: will return a :class:`~transformers.TextGenerationPipeline`.
            - :obj:`"zero-shot-classification:`: will return a :class:`~transformers.ZeroShotClassificationPipeline`.
            - :obj:`"conversational"`: will return a :class:`~transformers.ConversationalPipeline`.
        model (:obj:`str` or :obj:`~transformers.PreTrainedModel` or :obj:`~transformers.TFPreTrainedModel`, `optional`)
            The model that will be used by the pipeline to make predictions. This can be a model identifier or an
            actual instance of a pretrained model inheriting from :class:`~transformers.PreTrainedModel` (for PyTorch)
            or :class:`~transformers.TFPreTrainedModel` (for TensorFlow).

            If not provided, the default for the :obj:`task` will be loaded.
        config (:obj:`str` or :obj:`~transformers.PretrainedConfig`, `optional`):
            The configuration that will be used by the pipeline to instantiate the model. This can be a model
            identifier or an actual pretrained model configuration inheriting from
            :class:`~transformers.PretrainedConfig`.

            If not provided, the default configuration file for the requested model will be used. That means that if
            :obj:`model` is given, its default configuration will be used. However, if :obj:`model` is not supplied,
            this :obj:`task`'s default model's config is used instead.
        tokenizer (:obj:`str` or :obj:`~transformers.PreTrainedTokenizer`, `optional`):
            The tokenizer that will be used by the pipeline to encode data for the model. This can be a model
            identifier or an actual pretrained tokenizer inheriting from :class:`~transformers.PreTrainedTokenizer`.

            If not provided, the default tokenizer for the given :obj:`model` will be loaded (if it is a string). If
            :obj:`model` is not specified or not a string, then the default tokenizer for :obj:`config` is loaded (if
            it is a string). However, if :obj:`config` is also not given or not a string, then the default tokenizer
            for the given :obj:`task` will be loaded.
        feature_extractor (:obj:`str` or :obj:`~transformers.PreTrainedFeatureExtractor`, `optional`):
            The feature extractor that will be used by the pipeline to encode data for the model. This can be a model
            identifier or an actual pretrained feature extractor inheriting from
            :class:`~transformers.PreTrainedFeatureExtractor`.

            Feature extractors are used for non-NLP models, such as Speech or Vision models as well as multi-modal
            models. Multi-modal models will also require a tokenizer to be passed.

            If not provided, the default feature extractor for the given :obj:`model` will be loaded (if it is a
            string). If :obj:`model` is not specified or not a string, then the default feature extractor for
            :obj:`config` is loaded (if it is a string). However, if :obj:`config` is also not given or not a string,
            then the default feature extractor for the given :obj:`task` will be loaded.
        framework (:obj:`str`, `optional`):
            The framework to use, either :obj:`"pt"` for PyTorch or :obj:`"tf"` for TensorFlow. The specified framework
            must be installed.

            If no framework is specified, will default to the one currently installed. If no framework is specified and
            both frameworks are installed, will default to the framework of the :obj:`model`, or to PyTorch if no model
            is provided.
        revision(:obj:`str`, `optional`, defaults to :obj:`"main"`):
            When passing a task name or a string model identifier: The specific model version to use. It can be a
            branch name, a tag name, or a commit id, since we use a git-based system for storing models and other
            artifacts on huggingface.co, so ``revision`` can be any identifier allowed by git.
        use_fast (:obj:`bool`, `optional`, defaults to :obj:`True`):
            Whether or not to use a Fast tokenizer if possible (a :class:`~transformers.PreTrainedTokenizerFast`).
        use_auth_token (:obj:`str` or `bool`, `optional`):
            The token to use as HTTP bearer authorization for remote files. If :obj:`True`, will use the token
            generated when running :obj:`transformers-cli login` (stored in :obj:`~/.huggingface`).
            revision(:obj:`str`, `optional`, defaults to :obj:`"main"`):
        model_kwargs:
            Additional dictionary of keyword arguments passed along to the model's :obj:`from_pretrained(…,
            **model_kwargs)` function.
        kwargs:
            Additional keyword arguments passed along to the specific pipeline init (see the documentation for the
            corresponding pipeline class for possible values).

    Returns:
        :class:`~transformers.Pipeline`: A suitable pipeline for the task.

    Examples::

        >>> from transformers import pipeline, AutoModelForTokenClassification, AutoTokenizer

        >>> # Sentiment analysis pipeline
        >>> pipeline('sentiment-analysis')

        >>> # Question answering pipeline, specifying the checkpoint identifier
        >>> pipeline('question-answering', model='distilbert-base-cased-distilled-squad', tokenizer='bert-base-cased')

        >>> # Named entity recognition pipeline, passing in a specific model and tokenizer
        >>> model = AutoModelForTokenClassification.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")
        >>> tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
        >>> pipeline('ner', model=model, tokenizer=tokenizer)
```

```
In [26]: p = pipeline('sentiment-analysis')

         p("I'm happy.")

Out[26]: [{'label': 'POSITIVE', 'score': 0.9998724460601807}]

In [27]: p("I'm sad.")

Out[27]: [{'label': 'NEGATIVE', 'score': 0.9994174242019653}]

In [28]: p("I'm not happy.")

Out[28]: [{'label': 'NEGATIVE', 'score': 0.9998021125793457}]

In [31]: import transformers

         p = pipeline('ner', aggregation_strategy=transformers.pipelines.AggregationStrategy.SIMPLE)

         p("""We would like to belatedly welcome Ulrich Glassmann of the Europa-Universität
           Flensburg (#EUF), who is currently a guest at the Cluster. Ulrich has just decided
           to extend his stay until the end of June, welcome news indeed!""")

Out[31]: [{'entity_group': 'PER',
           'score': 0.9996402,
           'word': 'Ulrich Glassmann',
           'start': 35,
           'end': 51},
          {'entity_group': 'ORG',
           'score': 0.8913957,
           'word': 'Europa - Universität Flensburg',
           'start': 59,
           'end': 89},
          {'entity_group': 'ORG',
           'score': 0.988505,
           'word': 'EUF',
           'start': 92,
           'end': 95},
          {'entity_group': 'ORG',
           'score': 0.6957305,
           'word': 'Cluster',
           'start': 130,
           'end': 137},
          {'entity_group': 'PER',
           'score': 0.9996954,
           'word': 'Ulrich',
           'start': 139,
           'end': 145}]
```

```
In [32]: p = pipeline('translation', model='facebook/wmt19-de-en')

         p("""Nicht nur unterschiedliche Berechnungen bereiten Kopfzerbrechen.
           Bei der Eigenwahrnehmung zeigt sich: In Deutschland gibt es massive
           Missverständnisse über Ausmaß und Art von Ungleichheit.""")
```

```
/home/wastl/.local/lib/python3.9/site-packages/torch/_tensor.py:575: UserWarning: floor_divide is deprecated, and will b
To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), or for actual floor division, use torch.div(a,
  return torch.floor_divide(self, other)
```

```
Out[32]: [{'translation_text': 'It is not only different calculations that cause headaches. Self-perception shows that i
```

```
In [33]: p = pipeline('translation', model='facebook/wmt19-en-de')

         p("""We would like to belatedly welcome Ulrich Glassmann of the Europa-Universität
           Flensburg (#EUF), who is currently a guest at the Cluster. Ulrich has just decided
           to extend his stay until the end of June, welcome news indeed!""")
```

```
Out[33]: [{'translation_text': 'Mit Verspätung begrüßen wir Ulrich Glassmann von der Europa-Universität Flensburg (# EUF
```

```
In [34]: p = pipeline('text-generation')

         p("In Germany there are massive misunderstandings about the extent and type of inequality.")
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

```
Out[34]: [{'generated_text': 'In Germany there are massive misunderstandings about the extent and type of inequality. Fo
```

```
In [36]: p("some in Germany feel they have reached greater levels of economic equality without having")
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

```
Out[36]: [{'generated_text': 'some in Germany feel they have reached greater levels of economic equality without having
```

Transformers can be “fine-tuned” to a specific task, see training of transformers. Adding a task-specific head to a transformer pre-trained on large amounts of training data (usually 100 GBs or even TBs of text) saves resources spent for training and can overcome the problem of not enough training data. Manually labelling training data is expensive and naturally puts a limit on the amount of training data.There’s a good chance that the pre-trained transformer has seen words which are not in the small task-specific training data set about words not present