Data cleaning and preparation (Basics)

Prof. Dr. Axel Polleres
Dr. Stefan Sobernig
Dr. Jürgen Umbrich

Oct 15, 2018

Unit3

Unit3

Data cleaning and preparation:

- Data inspection (structure, interpretation)
- Data transformation (incl. reshaping)
- Data scanning (incl. sniffing)
- Data filtering
- Data sorting
- Data aggregation:
    a. grouping
    b. descriptive statistics

"Bread and Butter" for data scientists...

Cleaning & preparation

- Importance of cleaning & preparation follows from answering:
    - How to describe datasets (data structure and data semantics)?
    - What are common anomalies in datasets?
    - How to best reshape the data to facilitate analysis?
    - (How computationally expensive are the underlying procedures (transformation, filtering, sorting)?)
    - (How scalable are the underlying procedures to **really large** datasets?)

This is where most of the effort in data-science projects is spent, repeatedly (+80%)

Cleaning & preparation

Question.

Discuss: How would you describe the following two synthetic data sets?

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

- layout?
- rows?
- columns?

- cells
- labels?

Running example: EUROSTAT Urban audit

- Demographic data on European cities taken from EUROSTAT (1990-2016).
- Read more at Urban Audit (Navigate the database)
- TSV at `./data/urb_cpop1.tsv` list of European cities (city/country), population counts, and demographic indicators (gender, age groups).

Question.

How do you inspect this dataset? How do you characterize this dataset? What do we see when we look at this dataset?

Right questions at the right time

Questions we *could* ask right now:

1. Which one is the biggest city?
2. What are the (most recent) populations per country?
3. Which ones are the 10 biggest cities?
4. What was the population of the city with the code **AT004C1** in 2014?
5. What was the population of the city named "Innsbruck" in 2015?
6. ...

... , but we don't go there just yet

Interpretation of the data structure (1)

Consider first the following key notions:

- Dataset: Collection of *values* which describe *data objects* (e.g., units of observation) according to certain *variables* (a.k.a. attributes).
- Values can be *numeric* ("numbers") or *categorical* ("strings").
- Variables holding numeric values on data objects are *quantitative* variables.
- Variables holding categorical values on data objects are *qualitative* variables.
- Values are mapped to variables (attributes) of a data object via a (measurement) *scale.*
    - numeric values: interval, ratio
    - categorical values: nominal, ordinal

Interpretation of the data structure (2)

- Therefore, values are organized in two ways:
    - Every value belongs to a variable and a data object (observation)
    - A data object (observation) contains all values measured on the same unit of observation across *all* variables.
- Variables can be further discriminated by their *roles* in the analysis:
    - fixed variables ("dimensions"), in particular: identifier variables
    - measured variables
    - derived variables (e.g., mediators)

Interpretation of the data structure (3)

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

- Six observations
- Three variables: person, treatment, result
- 18 values (6 times 3)
- Person: nominal, fixed (identifier), three possible values
- Treatment: nominal, fixed (identifier), two possible values (a, b)
- Result: interval, measured, six possible values (incl. missing value, NA)

Running example: EUROSTAT Urban Audit

	CITIES,INDIC_UR\TIME	1990	1991	1992	1993	1994	1995	1996	1997	1998	...	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
0	BE,DE1001V	:	9 986 975	10 021 997	10 068 319	10 100 631	10 130 574	10 143 047	10 170 226	10 192 264	...	10 584 534	10 666 866	10 753 080	10 839 905	11 000 638	11 094 850	11 161 642	11 203 992	:	:
1	BE001C1,DE1001V	964 385	954 045	951 217	950 339	949 070	951 580	948 122	950 597	953 175	...	1 031 215	1 048 491	1 068 532	1 089 538	1 136 778	1 159 448	1 174 624	1 183 841	:	:
2	BE002C1,DE1001V	470 349	467 518	465 783	465 102	462 880	459 072	455 852	453 030	449 745	...	466 203	472 071	477 936	483 505	498 473	507 368	512 230	513 915	:	:
3	BE003C1,DE1001V	230 543	230 246	230 232	229 821	228 490	227 483	226 464	225 469	224 545	...	235 143	237 250	240 049	243 366	248 358	249 205	249 754	251 984	:	:
4	BE004C1,DE1001V	206 779	206 214	206 903	207 045	206 898	206 491	205 591	204 899	203 853	...	201 550	201 593	202 234	202 598	204 150	204 762	204 826	203 640	:	:
5	BE005C1,DE1001V	196 825	:	196 303	196 632	195 389	192 393	:	189 510	188 568	...	365 741	367 612	369 487	371 880	377 263	379 978	382 009	382 637	:	:
6	BE006C1,DE1001V	117 460	117 063	116 717	116 871	116 724	116 273	115 815	115 500	115 573	...	116 982	117 073	116 969	116 741	117 260	117 617	118 145	117 886	:	:
7	BE007C1,DE1001V	:	:	:	:	:	:	:	:	:	...	107 653	107 939	108 514	108 950	110 175	110 753	111 224	111 348	:	:
8	BE008C1,DE1001V	:	:	:	:	:	:	:	:	:	...	:	:	:	95 463	97 270	98 056	98 119	98 591	:	:
9	BE009C1,DE1001V	:	:	:	:	:	:	:	:	:	...	:	:	:	91 759	92 721	93 398	94 316	95 357	:	:

Running example: EUROSTAT Urban Audit

Question.

How would you describe the Urban Audit dataset using these key notions?

1. indic_ur,cities\time -> **AT,DE1001V**, **AT001C1,DE1001V**
    a. Indicators such as "population" use particular codes, e.g. **DE1001V** stands for "Population on the 1st of January, total"
        - indicator codes area available as another CSV at `./data/indic_ur.csv`
    b. Cities use particular codes... The codes are available in another file as RDF or as CSV
        - CSV `./data/cities.csv` list of cities incl their codes and names.
    c. Countries use ISO two-letter codes, e.g. available on datahub.io
        - CSV `./data/iso_3166_2_countries.csv` list of countries and country codes.
2. missing-value notation (NA, ":")
3. -> integers, **BUT:** 72959 b

Data transformation (1): Overview

Data transformation involves:

1. Modifying values contained by given variables and/ or
2. Adding variables (e.g., taken from previous) and/ or
3. Reshaping the dataset (i.e., its layout)

Permitted (value) transformations are indicated by the types of variables.

Data transformation (2): Goals

Datasets ("in the wild") may not be eligible:

- to run the intended value checks and value-based operations (e.g., numeric operations)
- to reshape the data layout
- to proceed with data preparation (scanning, filtering, sorting)

Some examples: When a dataset is consumed from a datasource as raw strings:

- it does not allow for number operations (e.g "5"+"5" != "10")
- it does not allow for comparison or sorting (e.g. "5" != 5, "11">"2", "2016-10-11" vs "11-10-2016")
- it does not allow for splitting & combining variables
- it does not allow for combining datasets (e.g., mixed letter cases as in "Wien" vs. "wien")

Data transformation (3): Value types

Let us first take a look at data types and how we can handle them in Python.

Python has the following "built-in", bit-representational ("primitive") datatypes:

- Numerical types: int , float, complex
- Boolean
- String (i.e., sequences of Unicode characters)
- (Collections: lists, tuples, dictionaries)

Other (structured) data types:

- Date, Datetime
- URL

Data transformation (4): Value types

Any (planned) transformation assumes introspection:

```
 type(variable)
 #e.g.
 >>> type(5)
<class 'int'>
```

```
 isinstance( x, t) //returns true if x is of type t, else false
 #e.g.
 >>> isinstance( 5, int)
 True
```

Data transformation (5): Number conversions

```
int (x) # Return an integer object constructed from a number or string x
float (x) # Return a floating point number constructed from a number or string x.
```

Examples

```
>>>float("   -12345\n")
-12345.0
>>> int(2.0)
2
```

Data transformation (6): Truth (boolean) values

```
bool( x)
```

Return a Boolean value, i.e. one of True or False. x is converted using the **standard truth testing** procedure

```
>>>bool(0)
False
>>>bool(10)
True
```

Data transformation (7): Truth-value checks

Any object can be tested for truth value, for use in an if or while condition or as operand of the Boolean operations below. The following values are considered **false**:

- None
- False
- zero of any numeric type, for example, 0, 0.0, 0j.
- any empty sequence, for example, '', (), [].
- any empty mapping, for example, {}.
- instances of user-defined classes, if the class defines a __bool__() or __len__() method, when that method returns the integer zero or bool value False. [1]

All other values are considered true — so objects of many types are always true.

Data transformation (7): Date/ datetime values

- Python offers with several options (modules) to deal and work with dates and datetime information, allowing for parsing, converting, comparing, and manipulating dates and times
- Official module

Available datetime types:

- date (year, month day)
- time (hour, minute, second, microsecond)
- datetime (year, month, day, hour, minute, second, microsecond)
- timedelta: A duration expressing the difference between two date, time, or datetime
- tzinfo: dealing with time zones
- timezone: dealing with time zones

Data transformation (8): Date/ datetime values

The **datetime.strptime()** class method creates a datetime object from

- a string representing a datetime *and from*
- a corresponding format string

```
>>> from datetime import datetime
>>> text = '2012-09-20'
>>> datetime.strptime(text, '%Y-%m-%d')
datetime.datetime(2012, 9, 20, 0, 0)
```

See the online documentation for a full list of variables for the string format

Data transformation (9): Date/ datetime values

The standard datetime Python module does not automatically detect and parse date/time strings and still requires to manually provide the format/ pattern string.

Options with (some) auto-detection:

- dateparser provides modules to easily parse localized dates in almost any string formats commonly found on web pages.

```
>>> import dateparser
>>> dateparser.parse('12/12/12')
datetime.datetime(2012, 12, 12, 0, 0)
```

- The dateutil module provides powerful extensions to the standard datetime module, available in Python.

```
>>> from dateutil.parser import parse
>>> parse("Today is January 1, 2047 at 8:21:00AM", fuzzy_with_tokens=True)
(datetime.datetime(2011, 1, 1, 8, 21), (u'Today is ', u' ', u'at '))
```

Notice.

!!Careful, such libraries might not necessarily detect the correct format but they cover properly 90% of all common cases.

Data transformation (10): String manipulation

- Converting (unicode) strings to some other value type is important to prepare and clean e.g. quantitative variables.
- Sometimes, transformations between strings is a preparatory step to a succesful type conversion.
- Commonly, strings themselves are the needed value representation (e.g., in qualitative variables), but:
  - ... they are not in the right or in an ambivalent format, e.g.:
    - "100,50": comma as the decimal mark, octal strings, etc.
    - "16-11-11" -> year-month-day vs, day-month-year ?
  - ... they contain (intended or unintended) artifacts
    - unintended: leading and trailing whitespace
    - intended: super- or subscripts (e.g., from presentation data)
- To clean up such strings, we need string manipulation methods

Data transformation (11): String manipulation

Python provides several functions to do to manipulate strings at the per-character level:

- functions to convert strings to upper or lower case
- **strip()** to remove leading and ending whitespaces
- **slicing** return a substring given one or two indices
- **split()** to split strings given a *"delimiter"*
- **replace(o,r)** to replace the occurrences of o with r

Mind the examples of assignment 1

For more functions, please see the official documentation for str objects

Data transformation (12): String slicing

Problem.

How to "split" strings, or extract substrings? For instance, "AT,DE1001V" -> AT and DE1001V

```
Use [ # : # ] to get set of letter
word[0]          #get one char of the word
word[0:3]        #get the first three char
word[-3:]        #get the last three char
```

Keep in mind that python, as many other languages, starts to count from 0!!

```
>>> word="AT,DE1001V"
>>> print(word[3:11])
DE1001V
```

Data transformation (13): String slicing

Some useful helper functions for dealing with strings and to find "index positions"

```
>>> word = "Data Processing"
>>> print(word.count('a'))      # count how many times l is in the string
2
>>> print(word.find("D") )      # find the word H in the string
0
>>> print( word.index("Data") ) # find the letters World in the string
0
>>> print( word.index("Pro") )
5
>>> print( len("Data") )
4
```

```
>>> word="AT,DE1001V"
>>> print(word[3:3+len("DE1001V")])
DE1001V
```

Data transformation (14): Substring search/replace

```
str.replace(old, new[, count])
Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.
```

```
>>>word="Data Processing"
>>>word.replace('Processing', 'Science')
Data Science

>>> float( "100,50".replace(",","."))
100.50
//while
>>>float( "100,50")
ValueError: could not convert string to float: '100,50'
```

Data transformation (15): Testing for character classes

```
word = "Data Processing"
word.isalnum()         #check if all char are alphanumeric
word.isalpha()         #check if all char in the string are alphabetic
word.isdigit()         #test if string contains digits
word.istitle()         #test if string contains title words
word.isupper()         #test if string contains upper case
word.islower()         #test if string contains lower case
word.isspace()         #test if string contains spaces
word.endswith('g')     #test if string endswith a g
word.startswith('D')   #test if string startswith D
```

Data transformation (15): Reshaping

- Reshaping can involve stacking or unstacking a dataset:
  - Stacking (melting): Turning columns into rows; typically for processing and analysis.
  - Unstacking: Turning rows into columns; typically for presentation.
- A "tidy dataset":https://vimeo.com/33727555 is one in which the abstract interpretation of a dataset (value, variable, observation) is reflected 1:1 by its structure.
  - Each variable forms a column.
  - Each observation forms a row.
  - Each type of data object (observation unit) forms a separate table.

Data transformation (16): Reshaping

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Data transformation (17): Reshaping

Question.

Discuss: Is the Urban Audit dataset tidy or messy?

	CITIES,INDIC_UR\TIME	1990	1991	1992	1993	1994	1995	1996	1997	1998	...	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
0	BE,DE1001V	:	9 986 975	10 021 997	10 068 319	10 100 631	10 130 574	10 143 047	10 170 226	10 192 264	...	10 584 534	10 666 866	10 753 080	10 839 905	11 000 638	11 094 850	11 161 642	11 203 992	:	:
1	BE001C1,DE1001V	964 385	954 045	951 217	950 339	949 070	951 580	948 122	950 597	953 175	...	1 031 215	1 048 491	1 068 532	1 089 538	1 136 778	1 159 448	1 174 624	1 183 841	:	:
2	BE002C1,DE1001V	470 349	467 518	465 783	465 102	462 880	459 072	455 852	453 030	449 745	...	466 203	472 071	477 936	483 505	498 473	507 368	512 230	513 915	:	:
3	BE003C1,DE1001V	230 543	230 246	230 232	229 821	228 490	227 483	226 464	225 469	224 545	...	235 143	237 250	240 049	243 366	248 358	249 205	249 754	251 984	:	:
4	BE004C1,DE1001V	206 779	206 214	206 903	207 045	206 898	206 491	205 591	204 899	203 853	...	201 550	201 593	202 234	202 598	204 150	204 762	204 826	203 640	:	:
5	BE005C1,DE1001V	196 825	:	196 303	196 632	195 389	192 393	:	189 510	188 568	...	365 741	367 612	369 487	371 880	377 263	379 978	382 009	382 637	:	:
6	BE006C1,DE1001V	117 460	117 063	116 717	116 871	116 724	116 273	115 815	115 500	115 573	...	116 982	117 073	116 969	116 741	117 260	117 617	118 145	117 886	:	:
7	BE007C1,DE1001V	:	:	:	:	:	:	:	:	:	...	107 653	107 939	108 514	108 950	110 175	110 753	111 224	111 348	:	:
8	BE008C1,DE1001V	:	:	:	:	:	:	:	:	:	...	:	:	:	95 463	97 270	98 056	98 119	98 591	:	:
9	BE009C1,DE1001V	:	:	:	:	:	:	:	:	:	...	:	:	:	91 759	92 721	93 398	94 316	95 357	:	:

Data transformation (18): Reshaping

Messy datasets result from violating these three main rules in different ways, for example:

- Column headers (labels) denote values, not variable names;
- Multiple variables are stored in one column;
- Variables are stored *both* in rows and columns;
- Multiple types of data objects are stored in the same dataset (e.g., regions and cities across years);
- A single observational unit is stored in multiple datasets (e.g., split sets by country);

Data scanning (1)

Scanning involves reading-in and processing a dataset in piecemeal manner, e.g.:

- row by row (in a messy dataset)
- observation by observation (in a tidy dataset)
- column by column (in a messy dataset)
- variable by variable (in a tidy dataset)
- value by value (per row/column, per observation/variable)

Python example ("row by row"):

```
with open('./data/urb_cpop1.tsv', 'r') as f:
    rows = f.readlines()
    for eachRow in rows:
        print(eachRow)
```

Data scanning (2)

For a given dimension (e.g., rows), scanning may be used to inspect on:

- the "head" of a dataset
- the "tail" of a dataset
- a "sample" (subset, slice) of a dataset
    - *random* vs. non-random
    - *ordered* vs. unordered

Data sniffing

Sniffing involves detecting in a guided, semi-automated manner:

- Details of a dataset layout, in particular:
    - headers
    - row labels
    - column separators
    - dimensions
- The data types of column values, e.g.:
    - Are columns homogeneous or heterogeneous?
    - Auto-detection of datetime formats
- Sniffing requires means of data scanning

Data filtering (1)

- Filtering: Removing or subsetting data objects (observations) based on a filter condition.
- Filtering can be considered as a conditional scanning.

Data filtering (2): Python basics

```
#filter out negative values
L=[0,1,2,-1,4,-6]
Lfiltered=[]
for i in L:
  if i>0:
    Lfiltered.append(i)
#another way to filter lists is to use list comprehension
Lfiltered=[ i for i in L if i>0] #same as above
```

Data filtering (3): Python basics

```
#filter out negative values
L=[ ['a',0],['b',1],['c',2],['d',-1],['e',4],['f',-6]]
Lfiltered=[]
for i in L:
  if i[1]>0:
    Lfiltered.append(i)
#another way to filter list is to use list comprehension
Lfiltered=[ i for i in L if i[1]>0] #same as above
```

Data filtering (4): Python basics

Dictionaries:

```
#filter out negative values
L=[ {'a':0},{'b':1},{'c':2},{'d':-1},{'e':4},{'f':-6}]
Lfiltered=[]
for d in L:
  for k,v in d.items():
    if v>0:
      Lfiltered.append(d)
# comprehension alternative
[i for i in L for k,v in i.items() if v > 0]
```

Data filtering (5): Applications

Data filtering has many applications:

1. "search" can be seen as filtering
2. focusing on only the *relevant* parts of the data
3. eliminating unnecessary content (e.g., removing unwanted data-object types in reshaping)
4. removing content which cannot be processed (e.g., structurally missing values)
5. reducing amount of data to to be processed at once, per job

(chunking data)

Data filtering (6): Urban-audit dataset

- Recall: We got multiple variables in the dataset.
- Task: Reduce it to one measure variable (population count)
    a. Observe: **indicator-identifier,2-letter-ISO-country-code** in the first column
    b. The identifier for population-count variable is **DE1001V**

Modify the scanning previous example to include a (filter) condition:

```
with open('./data/urb_cpop1.tsv', 'r') as f:
    rows = f.readlines()
    for eachRow in rows:
        # only output those lines containing a trailing 'DE1001V'
        colEnd = eachRow.find('\t')
        if (eachRow[colEnd-7:colEnd] == 'DE1001V')  :
            print(eachRow)
```

Question.

How could this be further improved?

- Observe: The indicators in the file *urb_cpop1.tsv* are *sorted* and **DE1001V** are the first indicators appearing!
- It is important to inspect the data before processing it!

Data filtering (7): Costs

Question.

How expensive is it to do filtering?

- For every filtering operation you need to do **one scan**...
- ... but several conjunctive filtering conditions can be combined (keyword: `and`) in one scan
- ... sometimes (e.g when file is sorted, or when you search only for a single result) you can stop before having to scan the whole file!

Data sorting (1)

- Sorting: Changing the order of data objects (observations) depending on the ordinal values of one or several of their variables (attributes).
- In-place sorting: Python lists have a built-in **list.sort()** method that modifies the list in-place.
- Out-place sorting: There is also a **sorted()** built-in function that builds a new sorted list from an iterable.
- See also the official documentation

Data sorting (2): Basics

```
sorted([5, 2, 3, 1, 4])
[1, 2, 3, 4, 5]
# the parameter 'reverse' can be set for descending order:
sorted([5, 2, 3, 1, 4], reverse = True)
[5, 4, 3, 2, 1]
```

```
a = [5, 2, 3, 1, 4]
```

```
a.sort()
a
```

Data sorting (3): List of lists

```
l = [[0, 1, 'f'], [4, 2, 't'], [9, 4, 'afsd']]
l.sort(key=lambda x: x[2])
print(l)
[[9, 4, 'afsd'], [0, 1, 'f'], [4, 2, 't']]
```

Data sorting (4): Dictionaries by key

- Note that dictionaries are typically **unordered**.
- So the output dictionary must be an order-preserving one: `OrderedDict`

```
orig = {2: 3, 1: 89, 4: 5, 3: 0}

import OrderedDict from collections
out = OrderedDict(sorted(orig.items(), key=lambda t: t[0]))
print(out)
```

Data sorting (5): Dictionaries by value

```
orig = {"aa": 3, "bb": 4, "cc": 2, "dd": 1}
from collections import OrderedDict
out = OrderedDict(sorted(orig.items(), key=lambda t: t[1]))
print(out)
```

Data sorting (6): List of tuples

```
student_tuples = [
    ('john', 'A', 15),
    ('jane', 'B', 12),
    ('dave', 'B', 10),
  ]
sorted(student_tuples, key=lambda student: student[2])   # sort by age
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

Data filtering and sorting: EUROSTAT Urban Audit

Question.

Discussion: Which of the following problems needs filtering? Which one needs sorting?

1. Which one is the biggest city?
2. What are the (most recent) populations per country?
3. Which ones are the 10 biggest cities?
4. What was the population of the city with the code **AT004C1** in 2014?
5. What was the population of the city named "Innsbruck" in 2015?
6. How many cities per country does that dataset contain?
7. Which country/ies has the most cities listed?
8. Which city/ies grew fastest over the past 10 years?

Data aggregation (1)

- Aggregation: Collapsing multiple values into a single value by
  a. *grouping* values by certain variables or variable levels
  b. computing *aggregates* of the so formed value groups.
- Objective:
  - Compress datasets to allow more expensive analysis steps (less memory or processing time)
  - Change in scope or in scale of the analysis, by presenting a high-level view on a dataset
  - Data aggregates are more stable than individual observations (prediction, variance).
- There are several ways to group items in Python.
  a. use a dictionary (esp. defaultdict
  b. (use itertools groupby)
  c. (pandas)

Data aggregation (2): Dictionary-based grouping

```
data = [
['Vienna','Austria',11,12,13], ['Salzburg','Austria',12,22,23],
['Stuttgart','Germany',12,22,23], ['Berlin','Germany',12,22,23],
['Paris','France',12,22,23]
]

# verbose tactic
groupby={}
for item in data:
    group=item[1]
    if group not in groupby:
        groupby[group]=[]
    groupby[group].append(item)

print(groupby)

# more compact tactic
from collections import defaultdict
groupby = defaultdict(list)
for row in data:
   groupby[row[1]].append(row)
```

Data aggregation (3): Dictionary-based grouping

```
Austria
[['Vienna', 'Austria', 11, 12, 13], ['Salzburg', 'Austria', 12, 22, 23]]
Germany
[['Stuttgart', 'Germany', 12, 22, 23], ['Berlin', 'Germany', 12, 22, 23]]
France
[['Paris', 'France', 12, 22, 23]]
```

Data aggregation (4): Dictionary-based grouping

```
data = [("animal", "bear"), ("animal", "duck"), ("plant", "cactus"), ("vehicle", "speed boat"), ("vehicle", "school bus")]

from collections import defaultdict
groupby = defaultdict(list)
for row in data:
   groupby[row[0]].append(row[1])

print(groupby.items())
for key, values in groupby.items():
    for thing in values:
        print("A "+thing+" is a "+key)
    print(" ")
```

```
A bear is a animal.
A duck is a animal.

A cactus is a plant.

A speed boat is a vehicle.
A school bus is a vehicle.
```

Data aggregation (5): Computing groupwise aggregates

- Typical tasks you want to do on lists or on groups: provide summary descriptors (statistics).
- The kind of summary descriptor computable depends on the variable type (quantitative, qualitative)
- **Frequency**: Counting the elements contained by a group (qualitative variables; absolute/ relative)

- **Mode**: The value of the highest frequency (qualitative variables)
- **Location**: mean and median (quantitative variables)
- **Spread**: range and variance (quantitative variables)

Data aggregation (6): Computing groupwise aggregates

```
quant  = [['a', 5], ['a', 1], ['b', 1], ['a', 2], ['b',3], ['b',1], ['a',4]]

from collections import defaultdict
groupby = defaultdict(list)
for row in quant:
   groupby[row[0]].append(row[1])
print(groupby.items())

# element count (group size)
{i: len(v) for i,v in groupby.items()}
# sum
{i: sum(v) for i,v in groupby.items()}
# location:
from statistics import mean
{i: mean(v) for i,v in groupby.items()}
from statistics import median
{i: median(v) for i,v in groupby.items()}
```

Data aggregation (6): Computing groupwise aggregates

```
qual  = ['a', 'c', 'a', 'c', 'b', 'e', 'a', 'c', 'b', 'e','b', 'e', 'a', 'd']

# frequency (absolute)
from collections import defaultdict
freq = defaultdict(int)
for el in qual:
   freq[el] += 1

# frequency (relative)
print({i: v/len(qual) for i,v in freq.items()})
```

References

- Pang-Ning Tan, Michael Steinbach, Vipin Kumar (2006): "Introduction to Data Mining", **Chapter 2**: "Data", Pearson.
- Hadley Wickham (2014): "Tidy data", The Journal of Statistical Software (59), DOI: 10.18637/jss.v059.i10