A Practical Introduction to Machine Learning in Python

Day 1 - Monday

»Introduction«

Damian Trilling
Anne Kroon

d.c.trilling@uva.nl, @damian0604
a.c.kroon@uva.nl, @annekroon

Gesis

March 9, 2020

Today

1. Introducing...
   - ...the people
2. Defining CSS
   - ...some definitions
   - Are we doing Big Data research?
   - Computational social science
3. CSS project workflow
   - step-by-step
   - A good workflow
4. Best practices
   - Open science
   - Clean, high-quality code
   - Exercise
   - datatypes
   - Generators
   - Scaling up

Introducing. . .

. . . the people

Introducing. . .
Damian

dr. Damian Trilling
Assistant Professor Political Communication & Journalism

- interested in political communication and journalism in a changing media environment and in innovative (digital, large-scale, computational) research methods

@damian0604 |d.c.trilling@uva.nl |`www.damiantrilling.net`

Introducing. . .
Anne

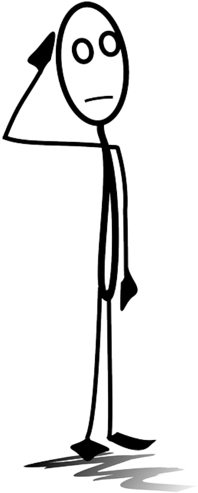
dr. Anne Kroon
Assistant Professor Corporate Communication

- Research focus on biased AI in recruitment, and media bias regarding minorities
- text analysis using automated approaches, word embeddings

@annekroon |a.c.kroon@uva.nl |`http://www.uva.nl/profiel/k/r/a.c.kroon/a.c.kroon.html`

Introducing...

You



Your name?
Your background?
Your reason to follow this course?
Do you have a dataset you are working on?

Short poll

Do you need

- ⓐ an intro
- ⓑ a brief refresher
- ⓒ nothing

on

- ⓘ datatypes (int, float, string, lists, dictionaries)
- ⓘⓘ control flow statements (for, if, try/except)
- ⓘⓘⓘ ways to run your code (notebooks vs IDE's vs text editors)

?

We will try do adapt today's programme to your needs!

What is Big Data?

Dan Ariely

6 januari 2013 ·

Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it...

Rachel Daxtor en 2,9 d. anderen 144 opmerkingen 1,3 d. keer gedeeld

Leuk Opmerking plaatsen Delen

BIG DATA ANALYTICS

Is Big Data Dying?

By **Aravind Sekar** — Last updated **Nov 30, 2018**

Share

What is Big Data?

What is Big Data?

A simple technical definition could be:

Everything that needs so much computational power and/or storage that you cannot do it on a regular computer.

What is Big Data?

What is Big Data?

Vis, 2013

- “commercial” definition (Gartner): “’Big data’ is high-volume, -velocity and -variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making”

What is Big Data?

Vis, 2013

- boyd & Crawford definition:
  1. Technology: maximizing computation power and algorithmic accuracy to gather, analyze, link, and compare large data sets.
  2. Analysis: drawing on large data sets to identify patterns in order to make economic, social, technical, and legal claims.
  3. Mythology: the widespread belief that large data sets offer a higher form of intelligence and knowledge that can generate insights that were previously impossible, with the aura of truth, objectivity, and accuracy.

What is Big Data?

Vis, 2013

- "commercial" definition (Gartner): "'Big data' is high-volume, -velocity and -variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making"
- boyd & Crawford definition:
  1. Technology: maximizing computation power and algorithmic accuracy to gather, analyze, link, and compare large data sets.
  2. Analysis: drawing on large data sets to identify patterns in order to make economic, social, technical, and legal claims.
  3. Mythology: the widespread belief that large data sets offer a higher form of intelligence and knowledge that can generate insights that were previously impossible, with the aura of truth, objectivity, and accuracy.

Implications & criticism

Implications & criticism

boyd & Crawford, 2012

1. Big Data changes the definition of knowledge
2. Claims to objectivity and accuracy are misleading
3. Bigger data are not always better data
4. Taken out of context, Big Data loses its meaning
5. Just because it is accessible does not make it ethical
6. Limited access to Big Data creates new digital divides

APIs, researchers and tools *make* Big Data

APIs, researchers and tools *make* Big Data

Vis, 2013

Inevitable influences of:

- APIs
- filtering, search strings, . . .
- changing services over time
- organizations that provide the data

Epistemologies and paradigm shifts

Kitchin, 2014

Epistemologies and paradigm shifts

Kitchin, 2014

- (Reborn) empiricism: purely inductive, correlation is enough

Epistemologies and paradigm shifts

Kitchin, 2014

- (Reborn) empiricism: purely inductive, correlation is enough
- Data-driven science: knowledge discovery guided by theory

Epistemologies and paradigm shifts

Kitchin, 2014

- (Reborn) empiricism: purely inductive, correlation is enough
- Data-driven science: knowledge discovery guided by theory
- Computational social science and digital humanities: employ Big Data research within existing epistemologies
  - DH: descriptive statistics, visualizations
  - CSS: prediction and simulation

Are we doing Big Data research in this course?

Are we doing Big Data research in this course?

Depends on the definition

- Not if we take a definition that *only* focuses on computing power and the amount of data

Are we doing Big Data research in this course?

Depends on the definition

- Not if we take a definition that *only* focuses on computing power and the amount of data
- **But:** We are using the same techniques. And they *scale* well.

Are we doing Big Data research in this course?

Depends on the definition

- Not if we take a definition that *only* focuses on computing power and the amount of data
- **But:** We are using the same techniques. And they *scale* well.
- Oh, and about that high-performance computing in the cloud: We actually *do* have access to that, so if someone has a really great idea...

Our epistomological underpinnings

Computational Social Science

Computational Social Science

“It is an approach to social inquiry defined by (1) the use of large, complex datasets, often—though not always— measured in terabytes or petabytes; (2) the frequent involvement of “naturally occurring” social and digital media sources and other electronic databases; (3) the use of computational or algorithmic solutions to generate patterns and inferences from these data; and (4) the applicability to social theory in a variety of domains from the study of mass opinion to public health, from examinations of political events to social movements”

Shah, D. V., Cappella, J. N., & Neuman, W. R. (2015). Big Data, digital media, and computational social science: Possibilities and perils. T*he ANNALS of the American Academy of Political and Social Science, 659*(1), 6–13. doi:10.1177/0002716215572084

Computational Social Science

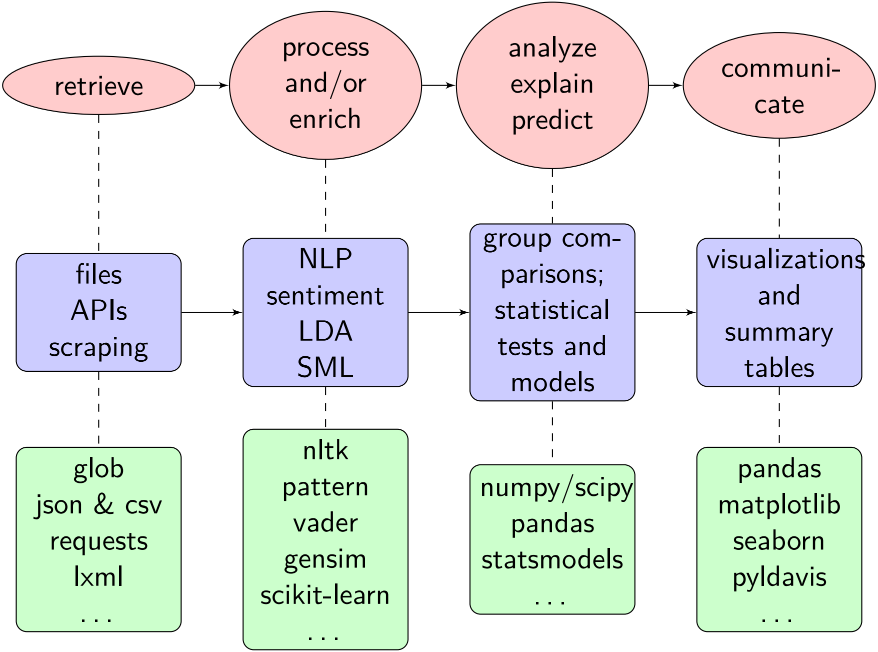
"[. . .] the computational social sciences employ the scientific method, complementing descriptive statistics with inferential statistics that seek to identify associations and causality. In other words, they are underpinned by an epistemology wherein the aim is to produce sophisticated statistical models that explain, simulate and predict human life."

Kitchin, R. (2014). Big Data, new epistemologies and paradigm shifts. *Big Data & Society, 1*(1), 1–12. doi:10.1177/2053951714528481

Steps of a CSS project

Different techniques for:

- retrieving data (previous week)
- processing data (previous week)
- analyzing data (main part of this week)
- visualising data (a bit on Friday)


retrieve
process and/or enrich
analyze explain predict
communi-cate
files APIs scraping
NLP sentiment LDA SML
group com-parisons; statistical tests and models
visualizations and summary tables
glob json & csv requests lxml ...
nltk pattern vader gensim scikit-learn ...
numpy/scipy pandas statsmodels ...
pandas matplotlib seaborn pyldavis ...

A good workflow

The big picture

Start with pen and paper

1. Draw the Big Picture

The big picture

Start with pen and paper

1. Draw the Big Picture
2. Then work out what components you need

Maximize transparency

Maximizing transparency of code and data

- Use openly accessible repository (e.g., Github)

Maximize transparency

Maximizing transparency of code and data

- Use openly accessible repository (e.g., Github)
- Store and preserve (pseudonymised) data at a secure environment (e.g., OSF)

Maximize transparency

Maximizing transparency of code and data

- Use openly accessible repository (e.g., Github)
- Store and preserve (pseudonymised) data at a secure environment (e.g., OSF)
- Create reusable workflows

Maximize transparency

Maximizing transparency of code and data

- Use openly accessible repository (e.g., Github)
- Store and preserve (pseudonymised) data at a secure environment (e.g., OSF)
- Create reusable workflows

Maximize transparency

Maximizing transparency of code and data

- Use openly accessible repository (e.g., Github)
- Store and preserve (pseudonymised) data at a secure environment (e.g., OSF)
- Create reusable workflows

Advantages

- Reusable data and code

Maximize transparency

Maximizing transparency of code and data

- Use openly accessible repository (e.g., Github)
- Store and preserve (pseudonymised) data at a secure environment (e.g., OSF)
- Create reusable workflows

Advantages

- Reusable data and code
- Efficiency and credibility

Maximize transparency

Maximizing transparency of code and data

- Use openly accessible repository (e.g., Github)
- Store and preserve (pseudonymised) data at a secure environment (e.g., OSF)
- Create reusable workflows

Advantages

- Reusable data and code
- Efficiency and credibility
- Recognition of tools and data

Develop components separately

One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)

Develop components separately

One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)
- Makes it easier to re-use your code or apply it to other data

Develop components separately

One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)
- Makes it easier to re-use your code or apply it to other data

Develop components separately

One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)
- Makes it easier to re-use your code or apply it to other data

Start small, then scale up

- Take your plan and solve *one* problem at a time (e.g., parsing a review page; or getting the URLs of all review pages)

Develop components separately

One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)
- Makes it easier to re-use your code or apply it to other data

Start small, then scale up

- Take your plan and solve *one* problem at a time (e.g., parsing a review page; or getting the URLs of all review pages)
- (for instance, by using functions [next slides])

Develop components separately

If you copy-paste code, you are doing something wrong

- Write loops!

Develop components separately

If you copy-paste code, you are doing something wrong

- Write loops!
- If something takes more than a couple of lines, write a function!

Copy-paste approach
(ugly, error-prone, hard to scale up)

```
allreviews = []

response = requests.get(‘http://xxxxx’)
tree = fromstring(response.text)
reviewelements = tree.xpath(‘//div[@class="review"]’)
reviews = [e.text for e in reviewelements]
allreviews.extend(reviews)

response = requests.get(‘http://yyyyy’)
tree = fromstring(response.text)
reviewelements = tree.xpath(‘//div[@class="review"]’)
reviews = [e.text for e in reviewelements]
allreviews.extend(reviews)
```

Better: for-loop
(easier to read, less error-prone, easier to scale up (e.g., more URLs, read URLs from a file or existing list)

```
allreviews = []

urls = [‘http://xxxxx’, ‘http://yyyyy’]

for url in urls:
    response = requests.get(url)
    tree = fromstring(response.text)
    reviewelements = tree.xpath(‘//div[@class="review"]’)
    reviews = [e.text for e in reviewelements]
    allreviews.extend(reviews)
```

Even better: for-loop with functions
(main loop is easier to read, function can be re-used in multiple contexts)

```
def getreviews(url):
    response = requests.get(url)
    tree = fromstring(response.text)
    reviewelements = tree.xpath(‘//div[@class="review"]’)
    return [e.text for e in reviewelements]


urls = [‘http://xxxxx’, ‘http://yyyyy’]

allreviews = []

for url in urls:
    allreviews.extend(getreviews(url))
```

Exercises

Did you pass?

- Think of a way to determine for a list of grades whether they are a pass (>5.5) or fail.
- Can you make that program robust enough to handle invalid input (e.g., a grade as 'ewghjieh')?
- How does your program deal with impossible grades (e.g., 12 or -3)?
- ...

Datatypes

Low-level: Native python datatypes

- Booleans, integers, floats, strings, bytes, byte arrays

Datatypes

Low-level: Native python datatypes

- Booleans, integers, floats, strings, bytes, byte arrays
- Lists, typles, sets, dictionaries

Datatypes

Low-level: Native python datatypes

- Booleans, integers, floats, strings, bytes, byte arrays
- Lists, typles, sets, dictionaries

Datatypes

Low-level: Native python datatypes

- Booleans, integers, floats, strings, bytes, byte arrays
- Lists, typles, sets, dictionaries

Advantages

- fast, flexible

Datatypes

Low-level: Native python datatypes

- Booleans, integers, floats, strings, bytes, byte arrays
- Lists, typles, sets, dictionaries

Advantages

- fast, flexible
- allows for nested, unstructured data

Datatypes

Low-level: Native python datatypes

- Booleans, integers, floats, strings, bytes, byte arrays
- Lists, typles, sets, dictionaries

Advantages

- fast, flexible
- allows for nested, unstructured data

Datatypes

Low-level: Native python datatypes

- Booleans, integers, floats, strings, bytes, byte arrays
- Lists, typles, sets, dictionaries

Advantages

- fast, flexible
- allows for nested, unstructured data

Disadvantages

- can be more cumbersome: e.g., inserting a column

Datatypes

Low-level: Native python datatypes

- Booleans, integers, floats, strings, bytes, byte arrays
- Lists, typles, sets, dictionaries

Advantages

- fast, flexible
- allows for nested, unstructured data

Disadvantages

- can be more cumbersome: e.g., inserting a column
- less consistency checks

Datatypes

Higher-level: importing modules

- e.g., numpy, pandas, seaborn

Datatypes

Higher-level: importing modules

- e.g., numpy, pandas, seaborn

Datatypes

Higher-level: importing modules

- e.g., numpy, pandas, seaborn

Advantages

- useful convenience functionality, works very intuitively (for tabular data)

Datatypes

Higher-level: importing modules

- e.g., numpy, pandas, seaborn

Advantages

- useful convenience functionality, works very intuitively (for tabular data)
- easy, allows for pretty visualization

Datatypes

Higher-level: importing modules

- e.g., numpy, pandas, seaborn

Advantages

- useful convenience functionality, works very intuitively (for tabular data)
- easy, allows for pretty visualization

Datatypes

Higher-level: importing modules

- e.g., numpy, pandas, seaborn

Advantages

- useful convenience functionality, works very intuitively (for tabular data)
- easy, allows for pretty visualization

Disadvantages

- not suited for one-dimensional or messy / deeply nested data

Datatypes

Higher-level: importing modules

- e.g., numpy, pandas, seaborn

Advantages

- useful convenience functionality, works very intuitively (for tabular data)
- easy, allows for pretty visualization

Disadvantages

- not suited for one-dimensional or messy / deeply nested data
- when your data is very large (machine learning!!)

Datatypes in this course

In this week, we will mainly work with lower-level datatypes (as opposed to, for instance, pandas dataframes)

- Often, ML algorithms require native data types as input (i.e., lists, genertors)

Datatypes in this course

In this week, we will mainly work with lower-level datatypes (as opposed to, for instance, pandas dataframes)

- Often, ML algorithms require native data types as input (i.e., lists, genertors)
- We have to seriously consider memory:

Datatypes in this course

In this week, we will mainly work with lower-level datatypes (as opposed to, for instance, pandas dataframes)

- Often, ML algorithms require native data types as input (i.e., lists, genertors)
- We have to seriously consider memory:
- Maybe size does not apply to your project yet, but in the future you might want to scale up.

Generators

Generators

- We will work with *generators* to deal with memory issues

Generators

Generators

- We will work with *generators* to deal with memory issues
- Generators behave like iterators: loops through elements of an object.

Generators

Generators

- We will work with *generators* to deal with memory issues
- Generators behave like iterators: loops through elements of an object.

Generators

Generators

- We will work with *generators* to deal with memory issues
- Generators behave like iterators: loops through elements of an object.

Behavior of a generator

- Does not hold results in memory

Generators

Generators

- We will work with *generators* to deal with memory issues
- Generators behave like iterators: loops through elements of an object.

Behavior of a generator

- Does not hold results in memory
- Only computes results at the moment you need them (i.e. lazy')

Generators

Generators

- We will work with *generators* to deal with memory issues
- Generators behave like iterators: loops through elements of an object.

Behavior of a generator

- Does not hold results in memory
- Only computes results at the moment you need them (i.e. lazy')
- You can only loop over your object ONCE.

Creating generators: Example 1

```
def my_generator(my_list):
    for i in my_list:
        yield i
example_list = [1, 2, 3, 4]
gen1 = my_generator(example_list)
next(gen1)
```

Creating generators: Example 2 (shorter)

```
my_list = [1,2,3,4]
gen = (i for i in my_list)
```

Scaling up

When considering datatypes, consider re-usability, scalability

- Use functions and classes to make code more readable and re-usable
- Avoid re-calculating values
- Think about how to minimize memory usage (e.g., Generators)
- Do not hard-code values, file names, etc., but take them as arguments

Make it robust

You cannot foresee every possible problem.
Most important: Make sure your program does not fail and loose all data just because something goes wrong at case 997/1000.

- Use `try/except` to explicitly tell the program how to handle errors
- Write data to files (or database) in between
- Use `assert len(x) == len(y)` for sanity checks

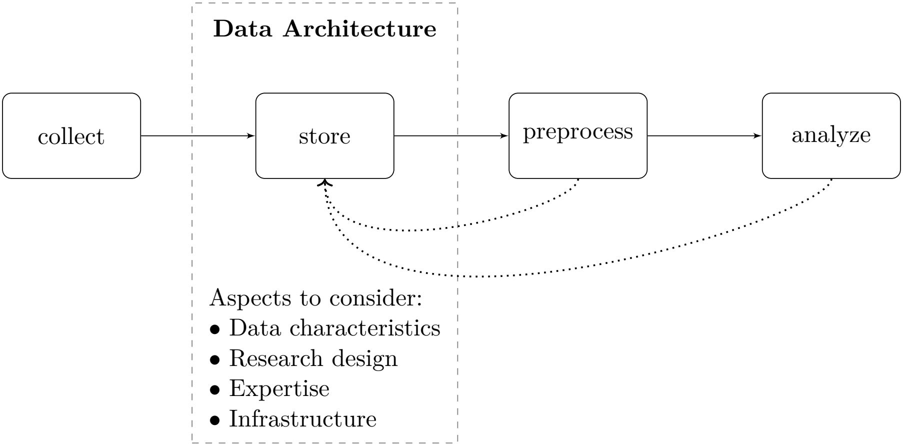
Storing data

Use of databases

Storing data

- We can store our data in files (often, one CSV or JSON file)
- But that's not very efficient if we have large datasets; especially if we want to select subsets later on
- SQL-databases to store tables (e.g., MySQL)
- NoSQL-databases to store less structured data (e.g., JSON with unknown keys) (e.g., MongoDB, ElasticSearch)
- $\Rightarrow$ Günther, E., Trilling, D., & Van de Velde, R.N. (2018). But how do we store it? (Big) data architecture in the social-scientific research process. In: *Stuetzer, C.M., Welker, M., & Egger, M. (eds.): Computational Social Science in the Age of Big Data. Concepts, Methodologies, Tools, and Applications*. Cologne, Germany: Herbert von Halem.

Storing data



From retrieved data to enriched data

1. Retrieved

```
<div>
<span id='author'
class='big'>
Author: John
Doe </span>
<span id='viewed'>
seen 42 times
</span>
</div>
```

2. Structured

```
{
"author":
"Author: John Doe",
"viewed":
"seen 42 times"
}
```

3. Cleaned

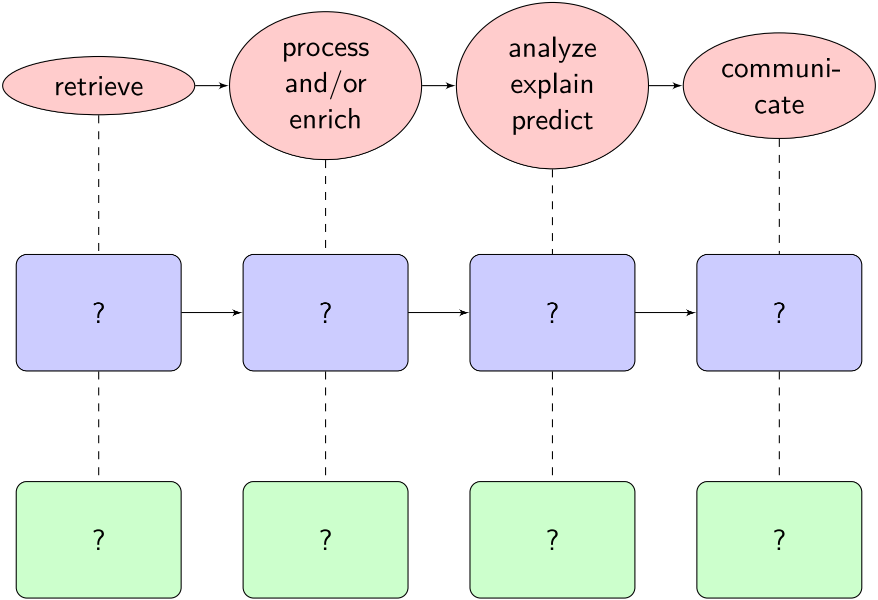
```
{
"author":
"john doe",
"viewed":
42,
}
```

4. Enriched

```
{
"author":
"john doe",
"author_gender":
"M",
"viewed":
42,
}
```

Looking forward

Try to fill in the blanks for your personal CSS project


retrieve
process and/or enrich
analyze explain predict
communi-cate
?
?
?
?
?
?
?
?

Long story short:
Don't forget to plan the bigger picture
We will focus on machine learning this week. But for each technique we cover, think about how it fits in *your* workflow.

Long story short:
Don't forget to plan the bigger picture
We will focus on machine learning this week. But for each technique we cover, think about how it fits in *your* workflow.
. . . and now lets get started!

Types of Automated Content Analysis

Methodological approach

	Counting and Dictionary	*Supervised Machine Learning*	*Unsupervised Machine Learning*
Typical research interests and content features	visibility analysis sentiment analysis subjectivity analysis	frames topics gender bias	frames topics
Common statistical procedures	string comparisons counting	support vector machines naive Bayes	principal component analysis cluster analysis latent dirichlet allocation semantic network analysis

deductive **inductive**

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset.

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset.

Unsupervised machine learning

You have no labels.

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset.

Unsupervised machine learning

You have no labels. (You did not measure y)

Some terminology

Unsupervised machine learning

You have no labels.

Again, you already know some techniques to find out how `x1, x2,...x_i` co-occur from other courses:

- Principal Component Analysis (PCA)
- Cluster analysis
- ...

This afternoon

Getting started

Getting started with the IMBD dataset

Backup slides in case we need to do more fundamentals

Basics]The very, very, basics of programming with Python

The very, very, basics of programming

See also Chapter 4.

Python lingo

Basic datatypes (variables)

int	`32`
float	`1.75`
bool	`True`, `False`
string	`"Damian"`

Python lingo

Basic datatypes (variables)

int	`32`
float	`1.75`
bool	`True`, `False`
string	`"Damian"`
(variable name	`firstname`)

"firstname" and firstname is not the same.

Python lingo

Basic datatypes (variables)

int	`32`
float	`1.75`
bool	`True`, `False`
string	`"Damian"`
(variable name	`firstname`)

"firstname" and firstname is not the same.
"5" and 5 is not the same.
But you can transform it: `int("5")` will return 5.
You cannot calculate `3 * "5"` (In fact, you can. It's `"555"`).
But you can calculate `3 * int("5")`

Python lingo

More advanced datatypes

Python lingo

More advanced datatypes

list
```
firstnames = ['Damian','Lori','Bjoern']
lastnames =
['Trilling','Meester','Burscher']
```

Note that the elements of a list, the keys of a dict, and the values of a dict can have any datatype! (Better to be consistent, though!)

Python lingo

More advanced datatypes

list `firstnames = ['Damian','Lori','Bjoern']`
`lastnames = ['Trilling','Meester','Burscher']`

list `ages = [18,22,45,23]`

Note that the elements of a list, the keys of a dict, and the values of a dict can have any datatype! (Better to be consistent, though!)

Python lingo

More advanced datatypes

list firstnames = ['Damian','Lori','Bjoern']
lastnames =
['Trilling','Meester','Burscher']

list ages = [18,22,45,23]

dict familynames= {'Bjoern': 'Burscher',
'Damian': 'Trilling', 'Lori': 'Meester'}

dict {'Bjoern': 26, 'Damian': 31, 'Lori':
25}

Note that the elements of a list, the keys of a dict, and the values of a dict can have any datatype! (Better to be consistent, though!)

Python lingo

Functions

Python lingo

Functions

functions Take an input and return something else
`int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.

Python lingo

Functions

functions Take an input and return something else
`int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.

methods are similar to functions, but directly associated with an object. `"SCREAM".lower()` returns the string "scream"

Python lingo

Functions

functions Take an input and return something else
`int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.

methods are similar to functions, but directly associated with an object. `"SCREAM".lower()` returns the string "scream"

Both functions and methods end with `()`. Between the `()`, *arguments* can (sometimes have to) be supplied.

Writing own functions

You can write an own function:

```
def addone(x):
    y = x + 1
    return y
```

Functions take some input ("argument") (in this example, we called it `x`) and *return* some result.
Thus, running

```
addone(5)
```

returns `6`.

Modifying lists and dictionaries

Modifying lists

Appending to a list

```
mijnlijst = ["element 1", "element 2"]
anotherone = "element 3" # note that this is a string, not a list!
mijnlijst.append(anotherone)
print(mijnlijst)
```

gives you:

```
["element 1", "element 2", "element 3"]
```

Modifying lists

Merging two lists (= extending)

```
mijnlijst = ["element 1", "element 2"]
anotherone = ["element 3", "element 4"]
mijnlist.extend(anotherone)
print(mijnlijst)
```

gives you:

```
["element 1", "element 2", "element 3", "element 4]
```

Modifying dicts

Adding a key to a dict (or changing the value of an existing key)

```
mydict = {"whatever": 42, "something": 11}
mydict["somethingelse"] = 76
print(mydict)
```

gives you:

```
{'whatever': 42, 'somethingelse': 76, 'something': 11}
```

If a key already exists, its value is simply replaced.

Indention]Indention: The Python way of structuring your program

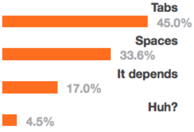
Indention: The Python way of structuring your program

Indention

Structure

The program is structured by TABs or SPACEs about 10% of developers still don't use it.

VII. TABS VS. SPACES



25,807 responses

After millennia of heated debate, mercifully, at long last, we have an answer. **Most developers prefer tabs to spaces.**

Upon closer examination of the data, a trend emerges: Developers increasingly prefer spaces as they gain experience. Stack Overflow reputation correlates with a preference for spaces, too: users who have 10,000 rep or more prefer spaces to tabs at a ratio of 3 to 1.

Indention

Structure

The program is structured by TABs or SPACEs

```
firstnames=['Damian','Lori','Bjoern']
age={'Bjoern': 27, 'Damian': 32, 'Lori': 26}
print ("The names and ages of these people:")
for naam in firstnames:
    print (naam,age[naam])
```

Indention

Structure

The program is structured by TABs or SPACEs

```
firstnames=['Damian','Lori','Bjoern']
age={'Bjoern': 27, 'Damian': 32, 'Lori': 26}
print ("The names and ages of these people:")
for naam in firstnames:
    print (naam,age[naam])
```

Don't mix up TABs and spaces! Both are valid, but you have to be consequent!!! Best: always use 4 spaces!

Indention

Structure

The program is structured by TABs or SPACEs

```
print ("The names and ages of all these people:")
for naam in firstnames:
    print (naam,age[naam])
    if naam=="Damian":
        print ("He teaches this course")
    elif naam=="Lori":
        print ("She is a former assistant")
    elif naam=="Bjoern":
        print ("He helped teaching this course in the past")
    else:
        print ("No idea who this is")
```

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (`if`, `elif`, and `else` statements)

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (`if`, `elif`, and `else` statements)
- an alternative block should be executed if an error occurs (`try` and `except` statements)

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (`if`, `elif`, and `else` statements)
- an alternative block should be executed if an error occurs (`try` and `except` statements)
- a file is opened, but should be closed again after the block has been executed (`with` statement)

Exercise

We'll now together do some simple exercises . . .

Exercises

1. Warming up

- Create a list, loop over the list, and do something with each value (you're free to choose).

2. Did you pass?

- Think of a way to determine for a list of grades whether they are a pass (>5.5) or fail.
- Can you make that program robust enough to handle invalid input (e.g., a grade as 'ewghjieh')?
- How does your program deal with impossible grades (e.g., 12 or -3)?
- ...