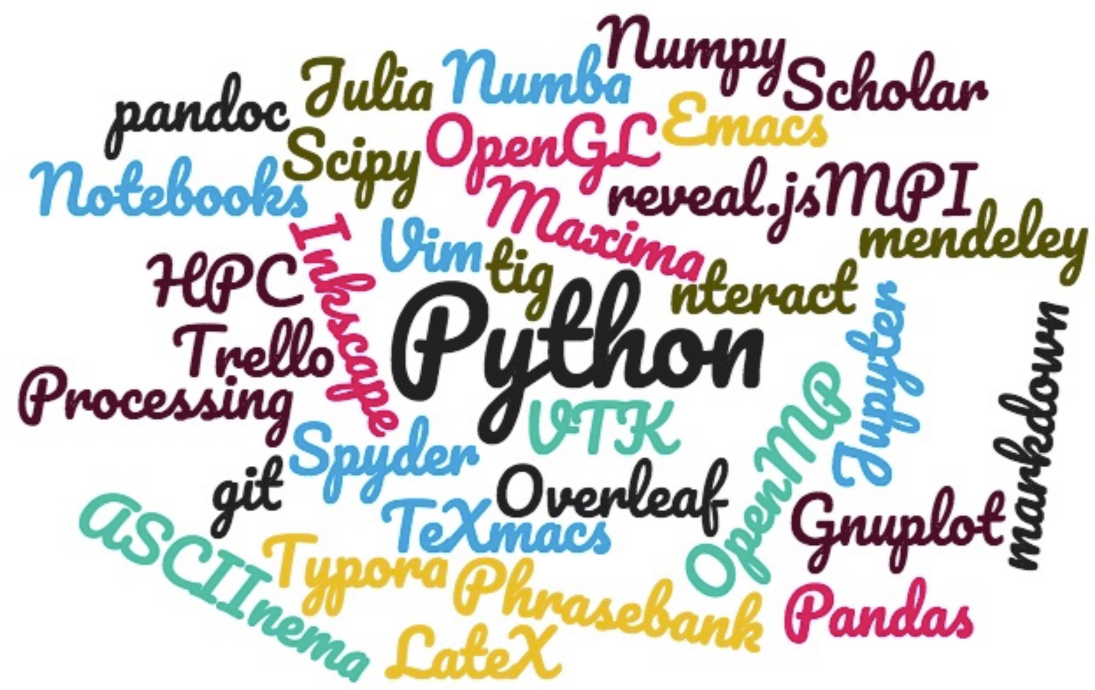
The efficient toolbox of the Computational Scientist



Dr. Gábor Závodszky - _Computational Science Lab_ https://github.com/gzavo/CS_Assignment

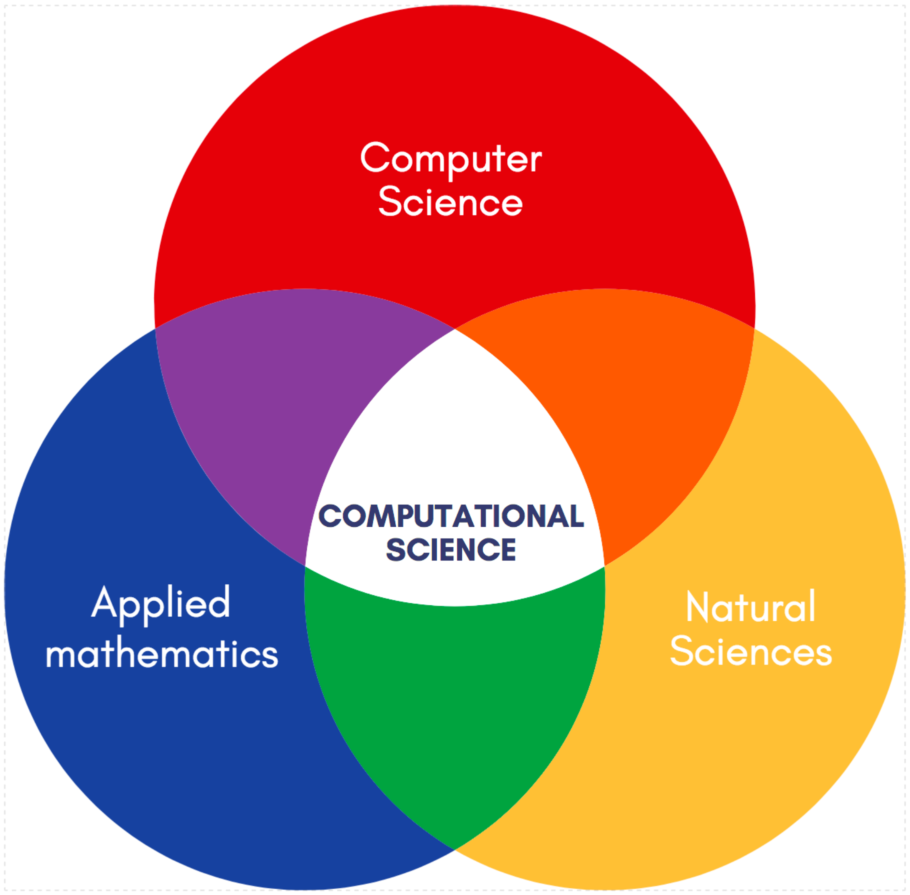
> "Never do a live demo" -- Every presenter ever

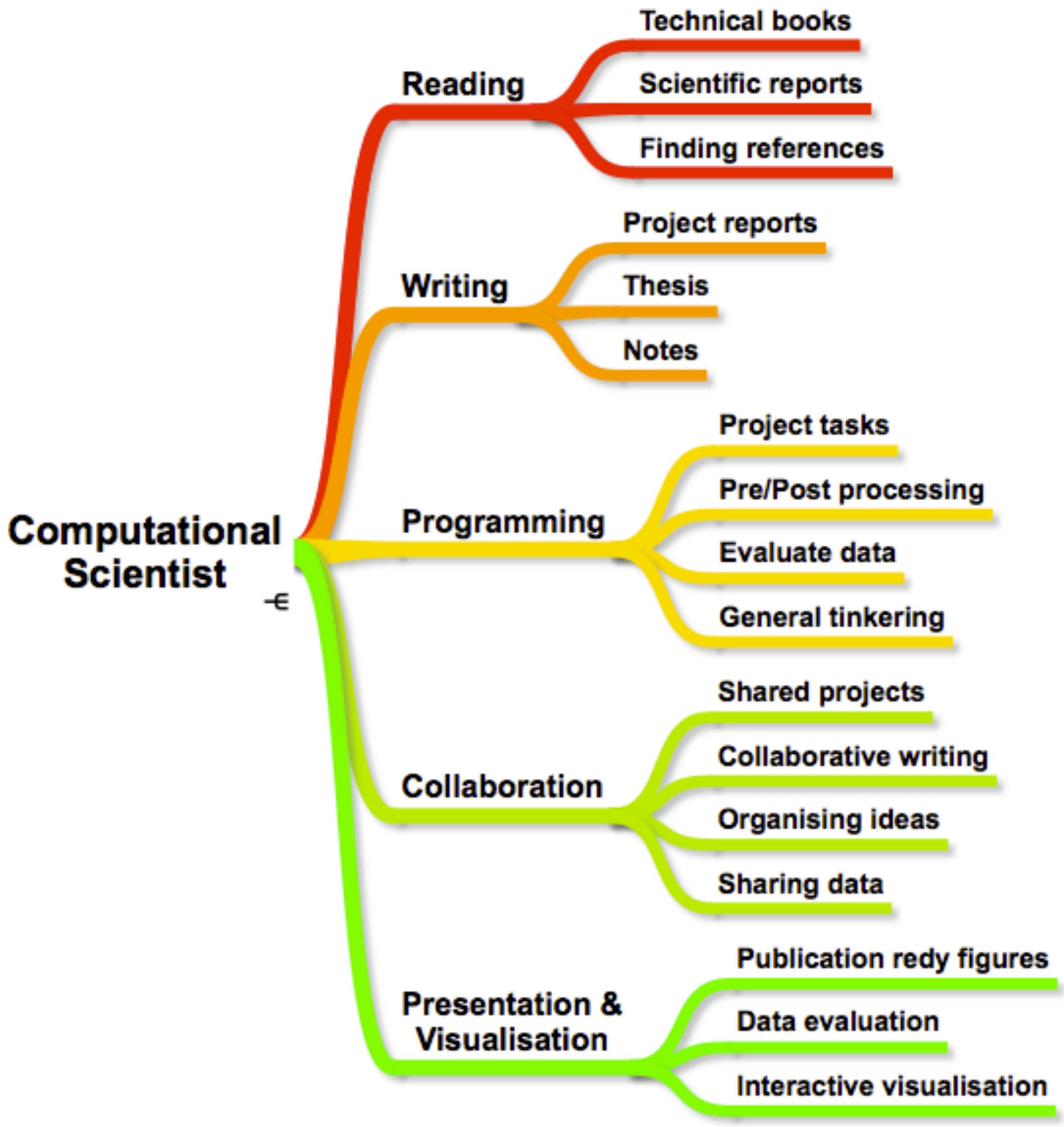
Structure of the lecture

- Domains of "The Computational Scientist"
- Scientific writing and reading
- Programming and visualization basics
- Software development in a collaborative environment
- Homework assingment

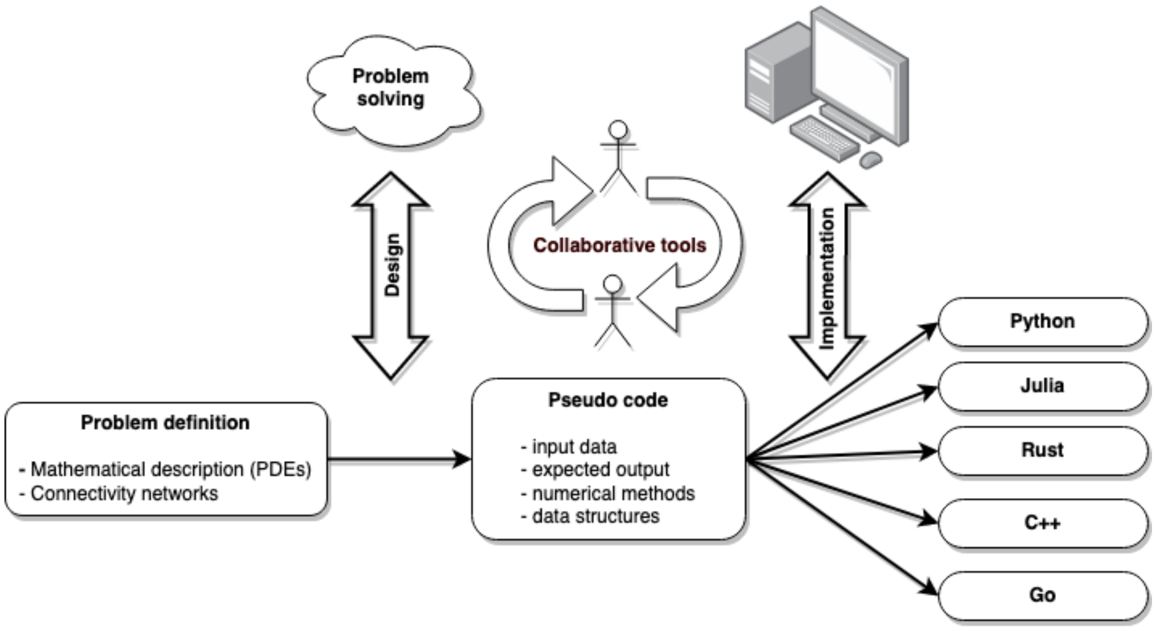
- Can be an information overload!
- The slides contain the very basics.
- Due to the structure of this MSc programme, you might not need *everything*
- The most important things are marked: (!)
- Some things are there to hear at least once.
- You can use the pdf version as a list of useful tools.

Who is the Computational Scientist? - Domains and fundamental skills




Computational Scientist
Reading
Technical books
Scientific reports
Finding references
Writing
Project reports
Thesis
Notes
Programming
Project tasks
Pre/Post processing
Evaluate data
General tinkering
Collaboration
Shared projects
Collaborative writing
Organising ideas
Sharing data
Presentation & Visualisation
Publication redy figures
Data evaluation
Interactive visualisation

General workflow of the Computational Scientist



Reading

- Library (!) (http://uba.uva.nl/en)
  - You can also use it to get access to non-open access journal papers.

- Google Scholar (!) - https://scholar.google.com
  - Search for papers
  - export references
  - look at researcher profiles
  - E.g.: "Carbon monoxide, boldly goes where..."
- Web of Science, Scopus, ...

- Managing references: Mendeley, Zotero (!), Papers, etc.
  - Sync. paper database
  - Annotate pdfs, add notes and sync them as well
- Most students use Zotero.
- Mendeley gives more cloud space, but its less flexible.

Writing

- LaTeX
    - Overleaf (!) - https://www.overleaf.com/
    - TexMaker
    - TeXmacs
    - latex2png - http://latex2png.com/

-> Why is LaTeX useful?

- Not a WYSIWYG solution
- But often WYGIWYN
- Literally a programming language for word processing

nature International weekly journal of science

Home | News & Comment | Research | Careers & Jobs | Current Issue | Archive | Audio & Video | For Authors

Archive > Volume 514 > Issue 7520 > Toolbox > Article

NATURE | TOOLBOX

Scientific writing: the online cooperative

Collaborative browser-based tools aim to change the way researchers write and publish their papers.

Jeffrey M. Perkel

01 October 2014 | Clarified: 06 October 2014

What matters in science — and why — free in your inbox every weekday.

Sign up

Listen

Nature Podcast

Our award-winning show features highlights from the week's edition of *Nature*, interviews with the people behind the science, and in-depth commentary and analysis from journalists around the world.

Science jobs from naturejobs

South China Normal University sincerely invite oversea talented scholars to apply for the Recruitment Program for Young Professionals

from IPython.display import IFrame

IFrame("https://www.nature.com/news/scientific-writing-the-online-cooperative-1.16039", "100%", 600)

- Markdown (!)
  - Use Pandoc (!) to turn it to .doc, .pdf, .html, you name it (https://pandoc.org/)
  - Typora (https://typora.io/)
  - Mark Text (https://github.com/marktext/marktext)
  - Basically everywhere (websites, forums, editors...really, everywhere)
  - Supersets / alternatives (ASCIIDOC, ...)

Markdown

Write equations: $e^{i\pi} + 1 = 0$

Embed code:

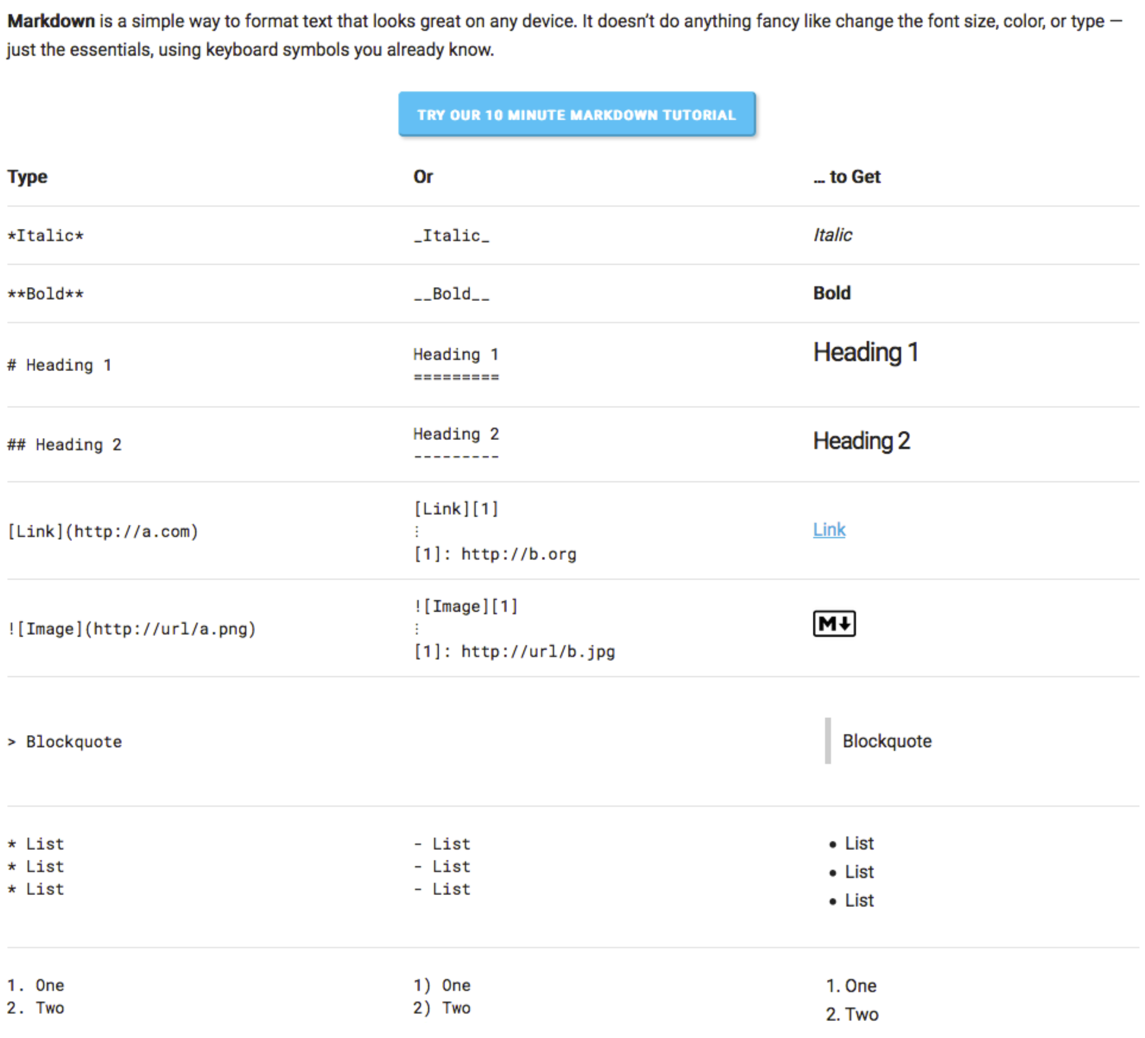
```
def f(x):
    """docstring of this very useful function"""
    return x**2
```

Add tables:

This	is
a	table

Create lists:

- list item 1
- list item 2
- list item 3



Markdown is a simple way to format text that looks great on any device. It doesn't do anything fancy like change the font size, color, or type — just the essentials, using keyboard symbols you already know.

Type	Or	... to Get
`*Italic*`	`_Italic_`	*Italic*
`**Bold**`	`__Bold__`	**Bold**

`# Heading 1`	`Heading 1`
	`=========`
		Heading 1

`## Heading 2`	`Heading 2`
	`---------`
		Heading 2

`[Link](http://a.com)`	`[Link][1]`
	`⋮`
	`[1]: http://b.org`
		Link

`![Image](http://url/a.png)`	`![Image][1]`
	`⋮`
	`[1]: http://url/b.jpg`
		M↓

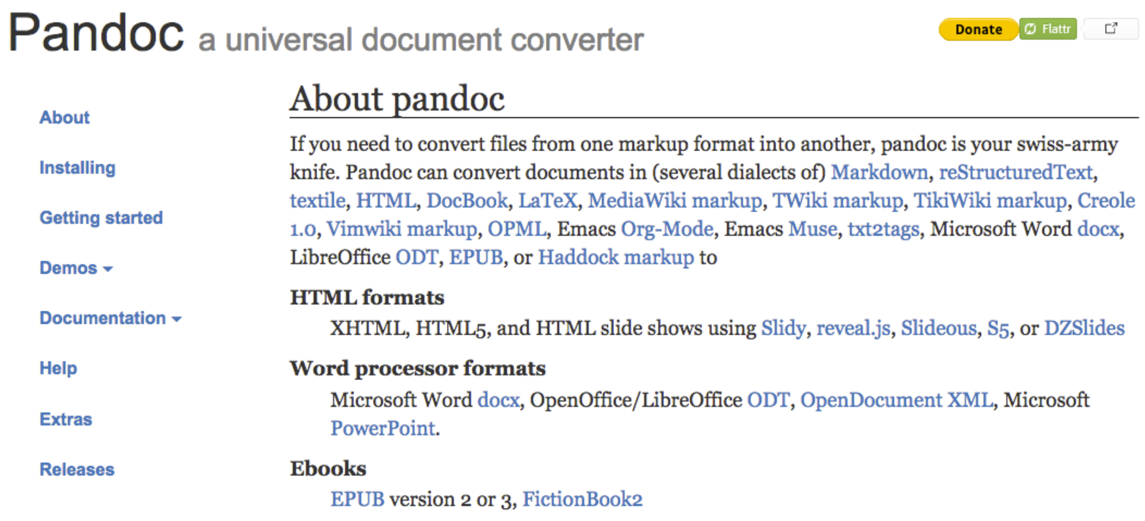
`> Blockquote`		Blockquote
`* List` `* List` `* List`	`- List` `- List` `- List`	• List • List • List

`1. One`
`2. Two`
	`1) One`
	`2) Two`
		1. One
		2. Two

from IPython.display import IFrame

IFrame("http://commonmark.org/help/", "100%", 600)

Writing - cont.



Pandoc a universal document converter

Donate Flattr

About
Installing
Getting started
Demos
Documentation
Help
Extras
Releases

About pandoc

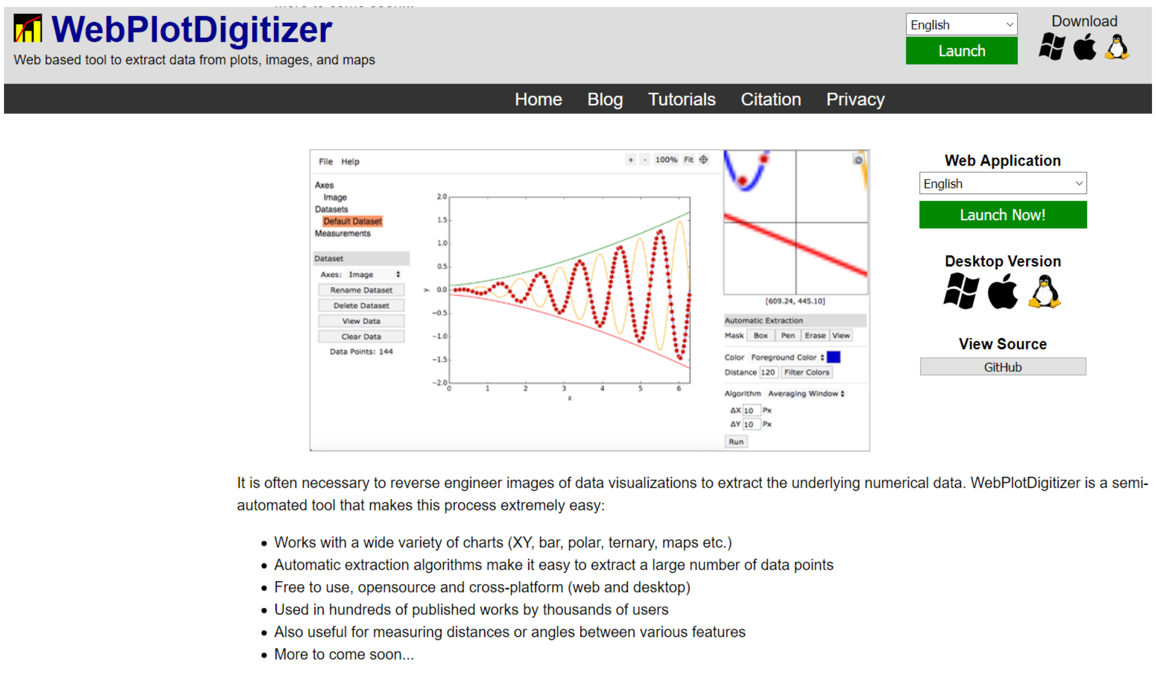
If you need to convert files from one markup format into another, pandoc is your swiss-army knife. Pandoc can convert documents in (several dialects of) Markdown, reStructuredText, textile, HTML, DocBook, LaTeX, MediaWiki markup, TWiki markup, TikiWiki markup, Creole 1.0, Vimwiki markup, OPML, Emacs Org-Mode, Emacs Muse, txt2tags, Microsoft Word docx, LibreOffice ODT, EPUB, or Haddock markup to

HTML formats
XHTML, HTML5, and HTML slide shows using Slidy, reveal.js, Slideous, S5, or DZSlides

Word processor formats
Microsoft Word docx, OpenOffice/LibreOffice ODT, OpenDocument XML, Microsoft PowerPoint.

Ebooks
EPUB version 2 or 3, FictionBook2

https://automeris.io/WebPlotDigitizer/



WebPlotDigitizer
Web based tool to extract data from plots, images, and maps

English
Launch
Download

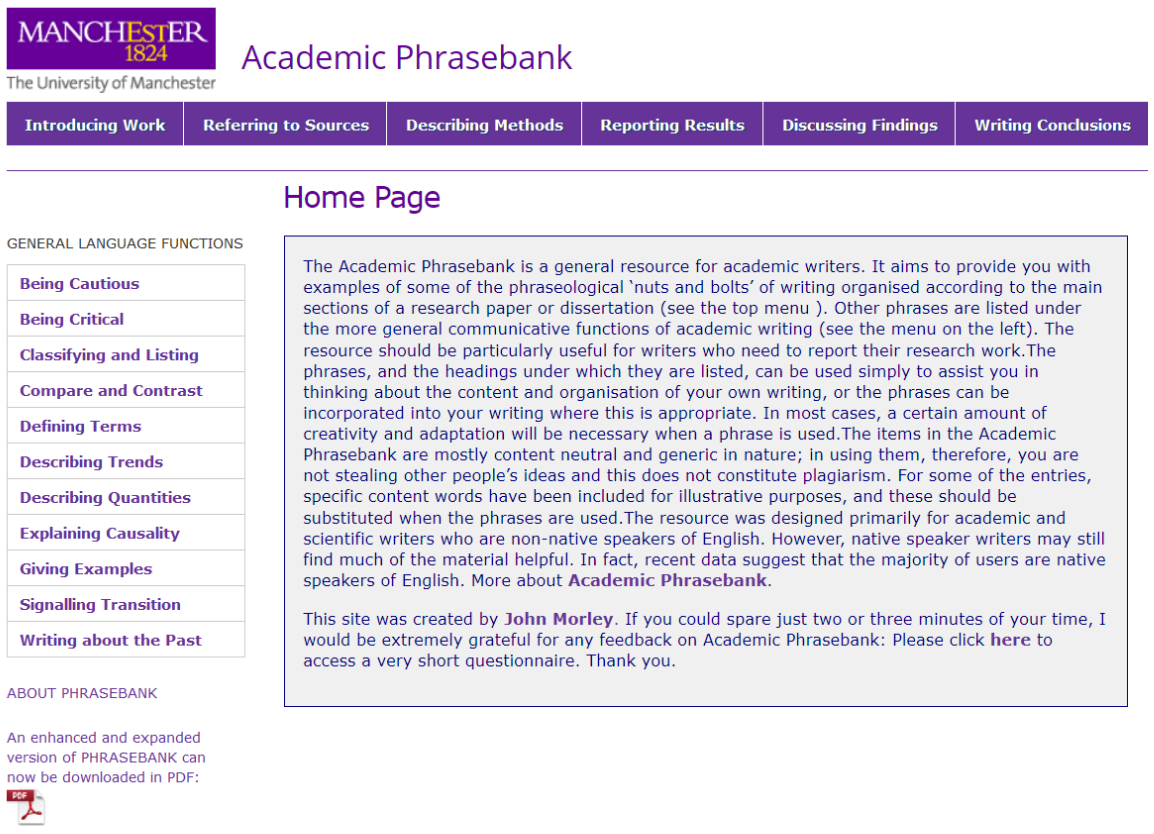
Home Blog Tutorials Citation Privacy

Web Application
English
Launch Now!
Desktop Version
View Source
GitHub

It is often necessary to reverse engineer images of data visualizations to extract the underlying numerical data. WebPlotDigitizer is a semi-automated tool that makes this process extremely easy:

- Works with a wide variety of charts (XY, bar, polar, ternary, maps etc.)
- Automatic extraction algorithms make it easy to extract a large number of data points
- Free to use, opensource and cross-platform (web and desktop)
- Used in hundreds of published works by thousands of users
- Also useful for measuring distances or angles between various features
- More to come soon...

http://www.phrasebank.manchester.ac.uk/



Academic Phrasebank

Home Page

The Academic Phrasebank is a general resource for academic writers. It aims to provide you with examples of some of the phraseological 'nuts and bolts' of writing organised according to the main sections of a research paper or dissertation (see the top menu). Other phrases are listed under the more general communicative functions of academic writing (see the menu on the left). The resource should be particularly useful for writers who need to report their research work.The phrases, and the headings under which they are listed, can be used simply to assist you in thinking about the content and organisation of your own writing, or the phrases can be incorporated into your writing where this is appropriate. In most cases, a certain amount of creativity and adaptation will be necessary when a phrase is used.The items in the Academic Phrasebank are mostly content neutral and generic in nature; in using them, therefore, you are not stealing other people's ideas and this does not constitute plagiarism. For some of the entries, specific content words have been included for illustrative purposes, and these should be substituted when the phrases are used.The resource was designed primarily for academic and scientific writers who are non-native speakers of English. However, native speaker writers may still find much of the material helpful. In fact, recent data suggest that the majority of users are native speakers of English. More about **Academic Phrasebank**.

This site was created by **John Morley**. If you could spare just two or three minutes of your time, I would be extremely grateful for any feedback on Academic Phrasebank: Please click **here** to access a very short questionnaire. Thank you.

An enhanced and expanded version of PHRASEBANK can now be downloaded in PDF:

Programming

- programming languages
- editors
- examples

Programming languages

- Python (!)
- C (!) (performance, hardware access)
- C++ (!) (If you need performance + OO)
- Julia (C + Python + Lisp structures)
- Rust (performance and safety)
- Kotlin (the new popular kid, mobile development)
- R (statistics)
- Scala (data science)
- Go (Google's try)
- Javascript (too popular to leave out)
- Racket (for the gourmet)
- +1 Lobster lang. (http://strlen.com/lobster/)

> Application domains can overlap, choose 2-3 and learn them well!
>
> For instance, C/C++ and Python is a quite versatile combination.
>
> Computer Language Benchmark Game (https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html)
>
> Rosetta Code (http://www.rosettacode.org)

```
In [ ]:
```

General editors

- Visual Studio Code (!)
- Sublime Text
- **Jupyter / JupyterLab**
- Vim
- Emacs
- micro
- Atom
- ... wide palett of other editors ...

Specific editors / IDEs

- PyCharm
- Spyder (!)
- Juno
- Lazarus
- Code::Blocks
- **Pluto** (for Julia) (!)

Why is Jupyter important?

- HTML5 platform (kernel as backend, can run remotely on a different machine)
- Compatible with several languages (C++, Python, Julia, Haskell, ...)
- Important step in hosting and sharing codes (reproducability)
- Towards reproducable science! (also see Jupyter Lab)

> Careful: non-linear state, see next example! (Check out Pluto for linear state solution)

- Additional benefits of the separate backend: parallel execution, cluster management

In [1]:
```
b=2
print(b)
```
```
2
```

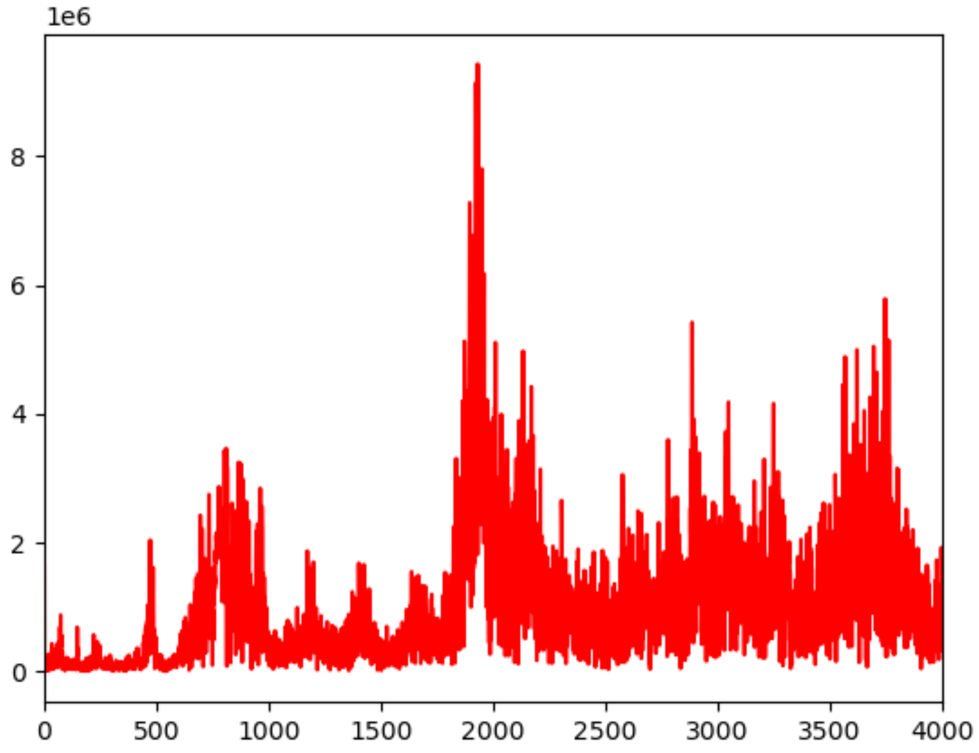
In [2]:
```
print(b)
b=3
```
```
2
```

In [3]:
```
import sympy as sp
sp.init_printing(use_latex='mathjax')
x,y,z = sp.symbols('x,y,z')
f = sp.sin(x*y)+sp.cos(y*z)
sp.integrate(f,x)
```

Out[3]:

$$x\cos(yz) + \begin{cases} -\frac{\cos(xy)}{y} & \text{for } y \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

In [4]:
```
import matplotlib.pyplot as plt
%matplotlib notebook
from scipy.fftpack import fft; from scipy.io import wavfile
fs, data = wavfile.read('sound/sample.wav') # load the data, 16 bit, 44.1 kH
c = fft(data.T) # calculate fourier transform (complex numbers list)
d = len(c)//2  # you only need half of the fft list (real signal symmetry)
plt.plot(abs(c[:(d-1)]),'r'); plt.xlim((0,4000))
```

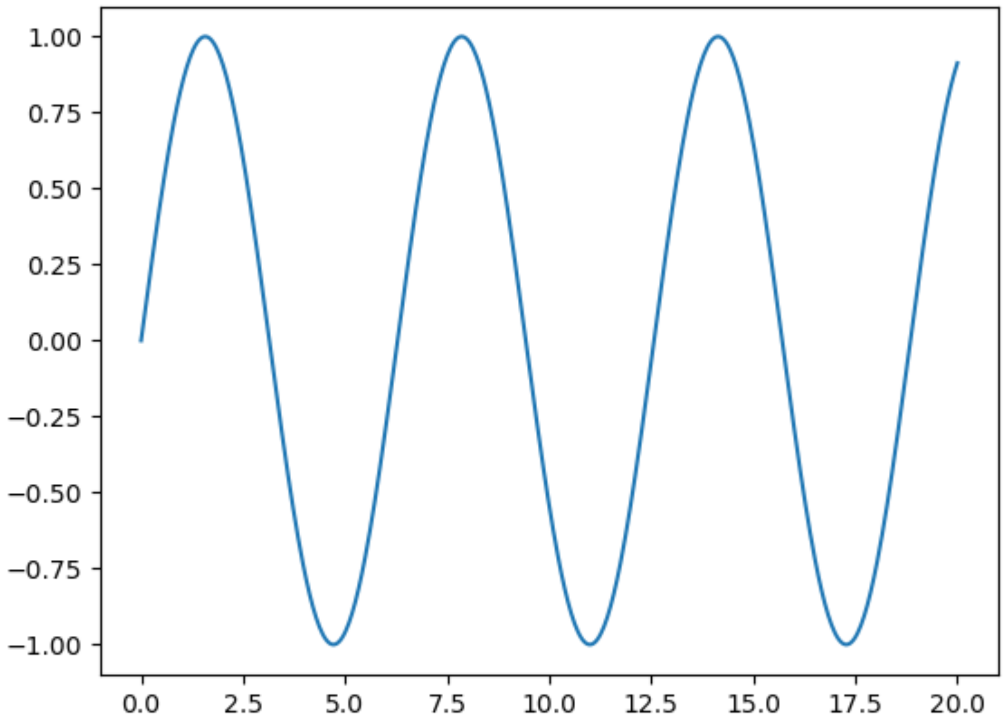


Out[4]: (0.0, 4000.0)

In [5]:
```
import IPython
# IPython.display.Audio('sound/sample440.wav')
# IPython.display.Audio('sound/sample_heroic.wav')
IPython.display.Audio('sound/sample.wav')
```

Out[5]: 0:00 / 0:01

In [6]:
```
import matplotlib.pyplot as plt
%matplotlib notebook
import numpy as np
x = np.linspace(0,20,500)
plt.plot(x, np.sin(x))
```



```
Out[6]: [<matplotlib.lines.Line2D at 0x7f88c19d54f0>]
```

```
In [7]: import matplotlib.pyplot as plt
        %matplotlib notebook
        from ipywidgets import interact, IntSlider
        from IPython.display import display,clear_output

        def f(freq):
            x = np.linspace(0,20,500)
            plt.plot(x, np.sin(x*freq))
            display(plt.figure)

        interact(f, freq=IntSlider(min=1,max=5,step=1,value=1));
```

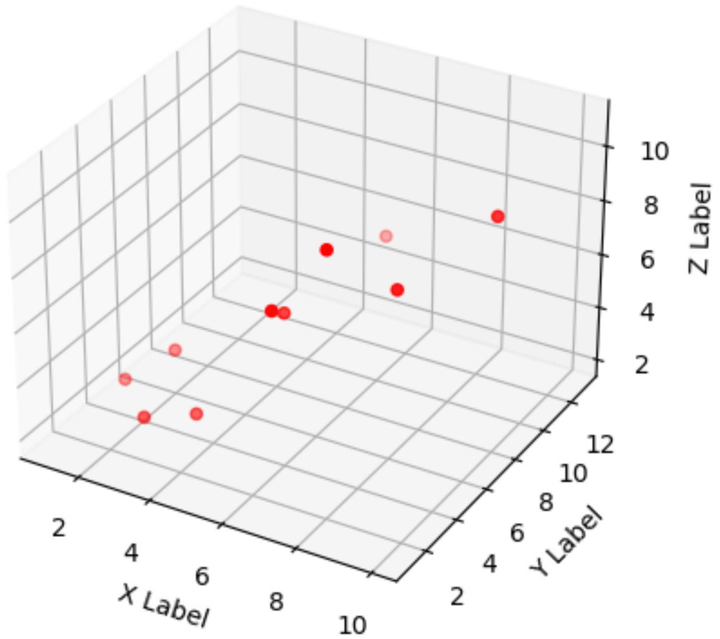
```
interactive(children=(IntSlider(value=1, description='freq', max=5, min=1),
Output()), _dom_classes=('widget-i…
```

```
In [8]: from IPython.display import display, Javascript
        import ipywidgets as widgets

        L = widgets.Label("Hello World")
        display(L)
```

```
Label(value='Hello World')
```

```
In [9]: L.value = "Howdy World"
```

In [10]:
```
from mpl_toolkits.mplot3d import Axes3D; import matplotlib.pyplot as plt;
%matplotlib notebook

fig = plt.figure(); ax = fig.add_subplot(111, projection='3d')
x =[1,2,3,4,5,6,7,8,9,10]; y =[5,6,2,3,13,4,1,2,4,8]; z =[2,3,3,3,5,7,9,11,9
ax.scatter(x, y, z, c='r', marker='o'); ax.set_xlabel('X Label'); ax.set_yla
```



Out[10]:
```
Text(0.5, 0, 'Z Label')
```

In [ ]:
```
# It does not work with the current nbextensions in Firefox :/
import ipyvolume as ipv
import numpy as np
import ipyvolume.datasets
stream = ipyvolume.datasets.animated_stream.fetch()
fig = ipv.figure()
# instead of doing x=stream.data[0], y=stream.data[1], ... vz=stream.data[5]
# limit to 50 timesteps to avoid having a huge notebook
q = ipv.quiver(*stream.data[:,0:50,:200], color="red", size=7)
ipv.style.use("dark") # looks better
ipv.animation_control(q, interval=200)
ipv.show()
```

In [1]:
```
import matplotlib.pyplot as plt
%matplotlib notebook
import pandas as pd
import seaborn as sns
df = sns.load_dataset("anscombe") #Anscombe's quarter, there are others (Tit
df.head()
```

Out[1]:

	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33

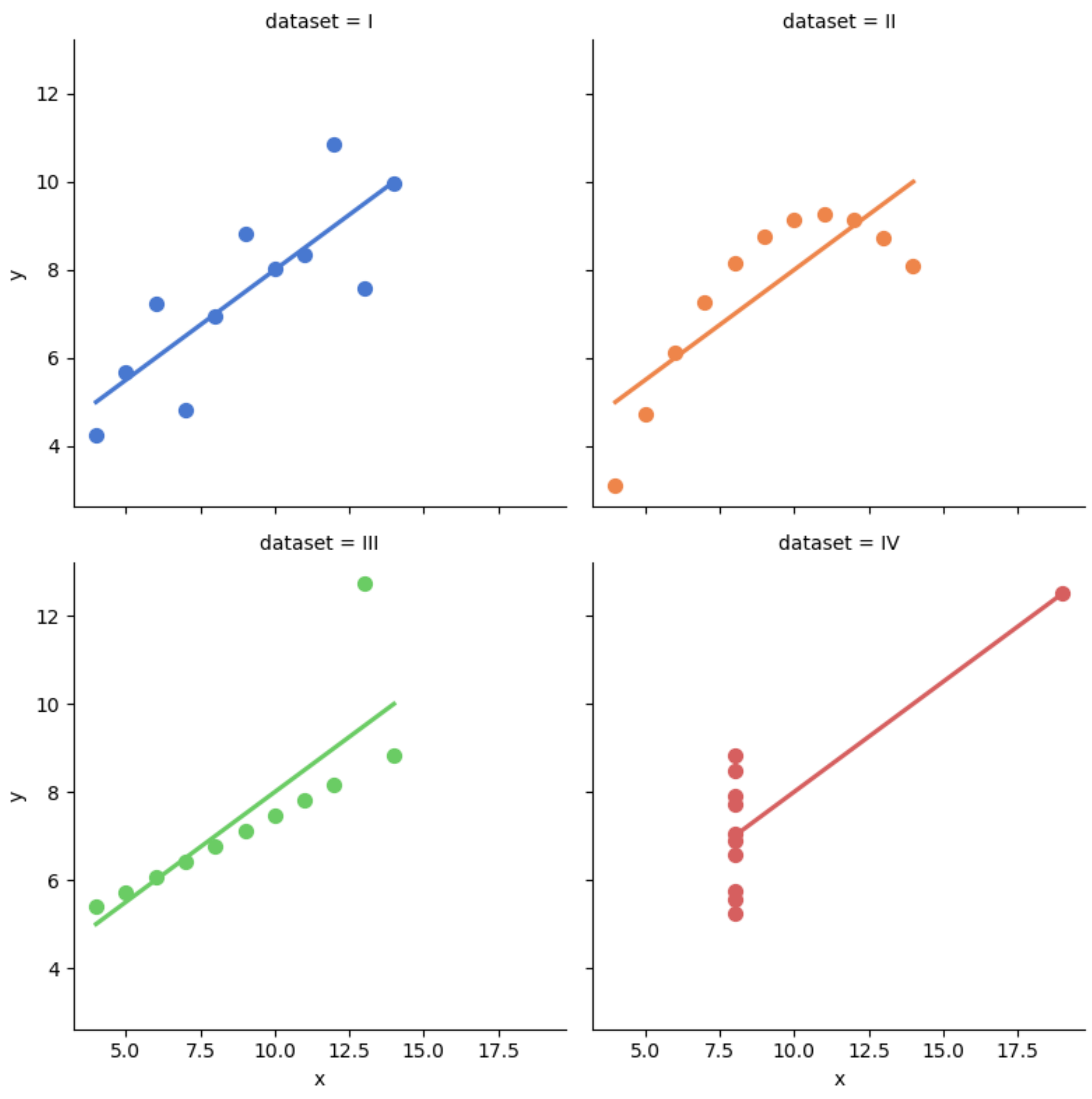
In [2]:
```python
df.groupby(df.dataset).describe()
```

Out[2]:

	x											
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**	**std**	
dataset												
I	11.0	9.0	3.316625	4.0	6.5	9.0	11.5	14.0	11.0	7.500909	2.031568	4
II	11.0	9.0	3.316625	4.0	6.5	9.0	11.5	14.0	11.0	7.500909	2.031657	3
III	11.0	9.0	3.316625	4.0	6.5	9.0	11.5	14.0	11.0	7.500000	2.030424	5
IV	11.0	9.0	3.316625	8.0	8.0	8.0	8.0	19.0	11.0	7.500909	2.030579	5

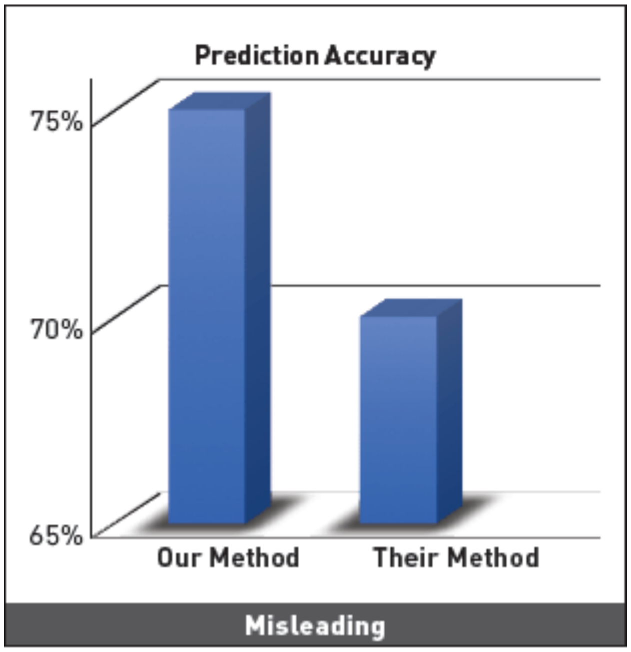
In [3]:
```python
sns.lmplot(x="x", y="y", col="dataset", hue="dataset", data=df, col_wrap=2,
           scatter_kws={"s": 50, "alpha": 1}); plt.show()
```

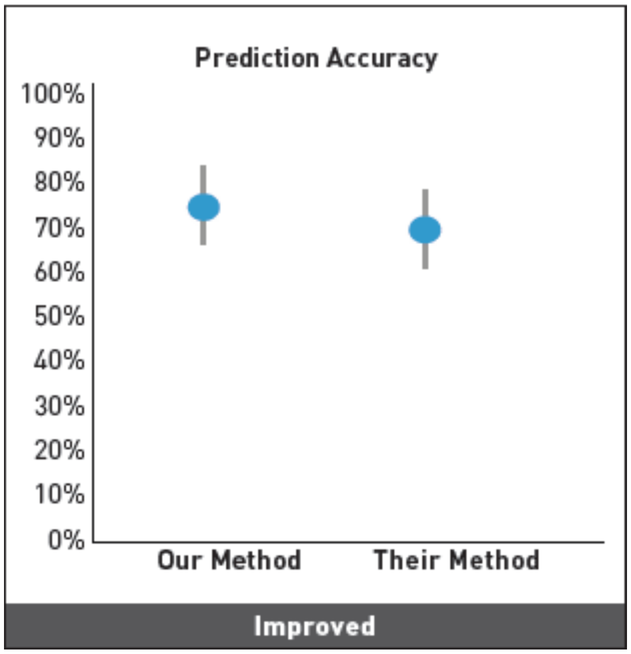
```
/Users/gzavo/opt/anaconda3/lib/python3.9/site-packages/seaborn/regression.p
y:581: UserWarning: The `size` parameter has been renamed to `height`; plea
se update your code.
  warnings.warn(msg, UserWarning)
```

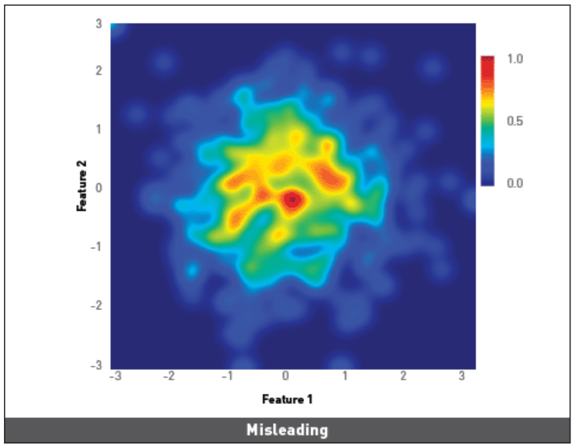


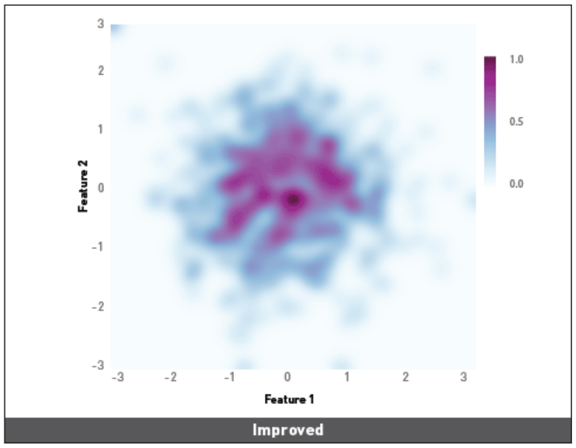
Visualization practices

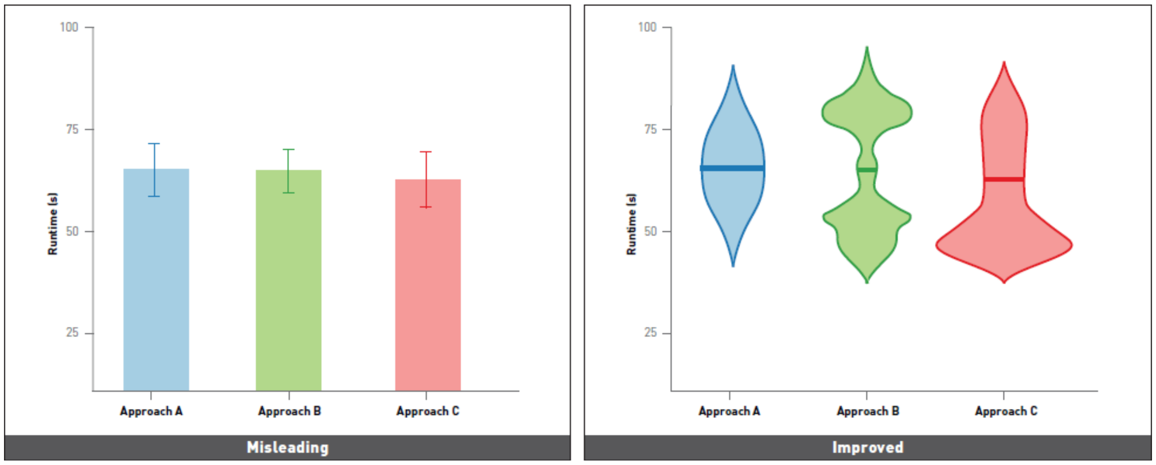
https://interactions.acm.org/archive/view/july-august-2018/the-good-the-bad-and-the-biased


Prediction Accuracy
75%
70%
65%
Our Method
Their Method
Misleading


Prediction Accuracy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Our Method
Their Method
Improved


Feature 1
Feature 2
1.0
0.5
0.0
Misleading


Feature 1
Feature 2
1.0
0.5
0.0
Improved



Further visualization tools

Because our simulations produce a lot of data (big data?!), but the purpose of the computation is not numbers, but insight.

- python matplotlib (!) - (https://matplotlib.org/)
- ParaView (!) - VTK based parallel 3D visualization tool (https://www.paraview.org/)
- MayaVi - ParaView alternative, written in python (embeddable!) (https://docs.enthought.com/mayavi/mayavi/)
- gnuplot (!) - Swiss army knif of plotters (http://www.gnuplot.info/)
- vedo - Simple visualization 2D/3D, simulation friendly (https://github.com/marcomusy/vedo)
- Processing - Programming language designed for 3D visualizations (https://processing.org/)
- Inkscape - For that nice poster. (https://inkscape.org/)
- Blender - (https://www.blender.org/)

Presentation tools

- MS PowerPoint (!)
- Reveal.js (https://github.com/hakimel/reveal.js/)
- LaTex (e.g. beamer)
- Jupyter Notebook (sort of works...)

Further things to look at in the Python world

- https://github.com/barbagroup/CFDPython
- http://mbakker7.github.io/exploratory_computing_with_python
- https://github.com/vinta/awesome-python

Other tools to mention

- Desktop jupyter: nteract (https://nteract.io/)
- Calculator: SpeedCrunch (https://speedcrunch.org/)
- CAS: Maxima (http://maxima.sourceforge.net/)
- Worksheet: SMathStudio (https://en.smath.com/view/SMathStudio/summary)
- Recording terminal session: ASCIInema (https://asciinema.org/)
- Collection of command line terminal tools (https://www.wezm.net/technical/2019/10/useful-command-line-tools/)

Software development in a collaborative environment

- Student-ware ('agile' method?)
- Guidelines PEP8
- Version control
- Tests

Software development as a student

The boundary conditions are a bit different, but aim to adopt good practices!

- most often no preliminary design
- "let's see what happens" first version
- then the code is "grown" in incremental steps
- "backups" is some old version on a pendrive, or sent in email
- this is a natural process given the boundary conditions

Pro.:

- nothing really
- maybe time efficiency on the short-run (*maybe*)

Con.:

- difficult collaboration
- not future proof
- quickly leads to decaying efficiency

What can be improved?

Coding style guides:

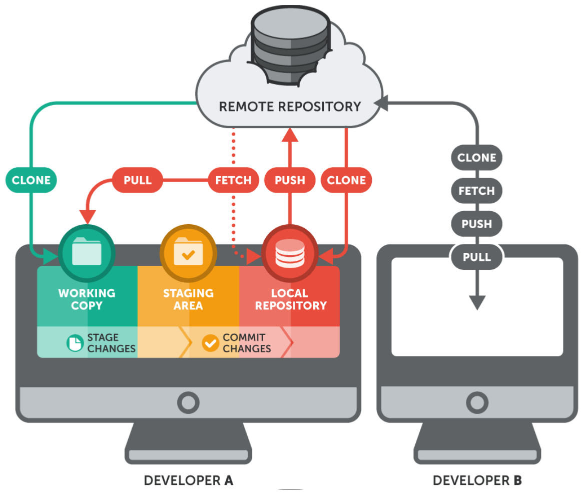
- Python PEP8 (https://www.python.org/dev/peps/)
- C++ Google Style guide (https://google.github.io/styleguide/cppguide.html)
- C - Many, e.g. Rob Pike's (https://www.maultech.com/chrislott/resources/cstyle/pikestyle.html)

See the code on the next slide.

In [14]:
```
print("".join(map(lambda x: chr((lambda p, x: int(sum(map(lambda i: p[i]*x**
```

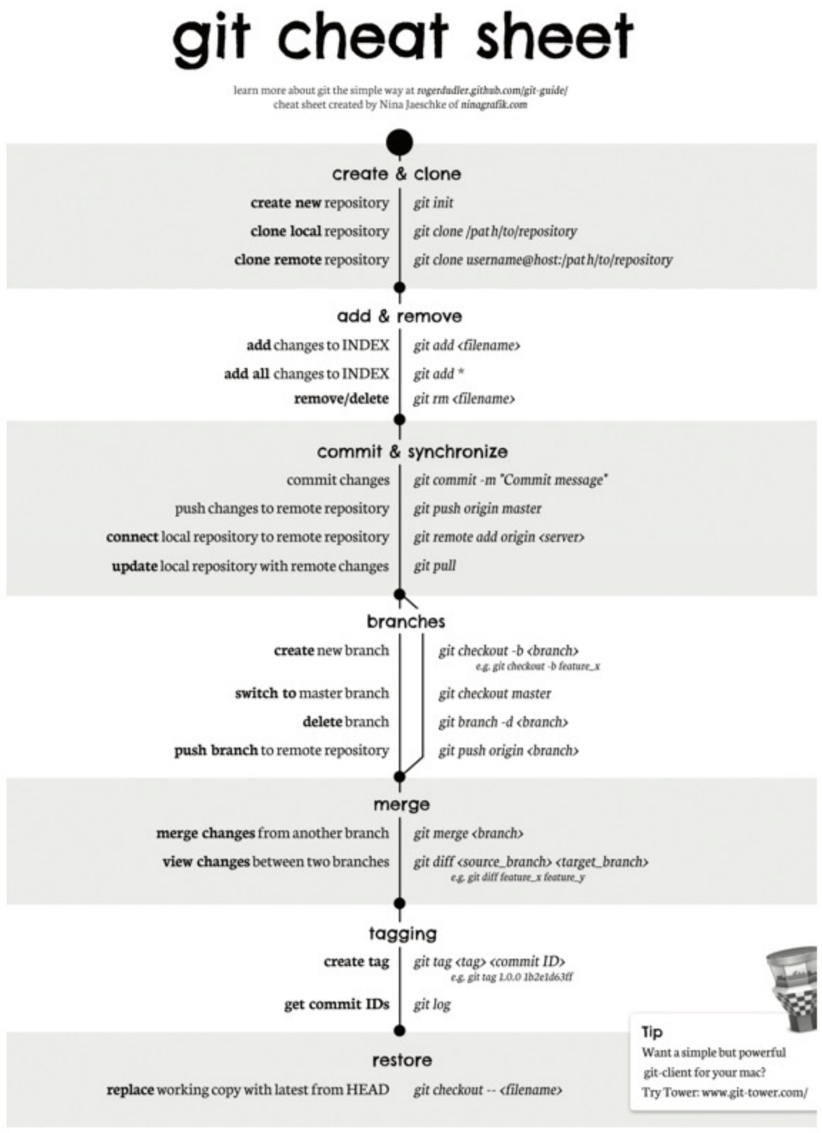
```
Hello world!
```

Version control - git (!)



In [15]:
```
from IPython.display import IFrame
IFrame("doc/git_cheat_sheet.jpg", width=800, height=600) # from http://roger
```

Out[15]:



git cheat sheet

learn more about git the simple way at *rogerdudler.github.com/git-guide/*
cheat sheet created by Nina Jaeschke of *ninagrafik.com*

create & clone

create new repository	*git init*
clone local repository	*git clone /path/to/repository*
clone remote repository	*git clone username@host:/path/to/repository*

add & remove

add changes to INDEX	*git add <filename>*
add all changes to INDEX	*git add **
remove/delete	*git rm <filename>*

commit & synchronize

commit changes	*git commit -m "Commit message"*
push changes to remote repository	*git push origin master*
connect local repository to remote repository	*git remote add origin <server>*
update local repository with remote changes	*git pull*

branches

create new branch	*git checkout -b <branch>* *e.g. git checkout -b feature_x*
switch to master branch	*git checkout master*
delete branch	*git branch -d <branch>*
push branch to remote repository	*git push origin <branch>*

merge

merge changes from another branch	*git merge <branch>*
view changes between two branches	*git diff <source_branch> <target_branch>* *e.g. git diff feature_x feature_y*

tagging

create tag	*git tag <tag> <commit ID>* *e.g. git tag 1.0.0 1b2e1d63ff*
get commit IDs	*git log*

restore

replace working copy with latest from HEAD	*git checkout -- <filename>*

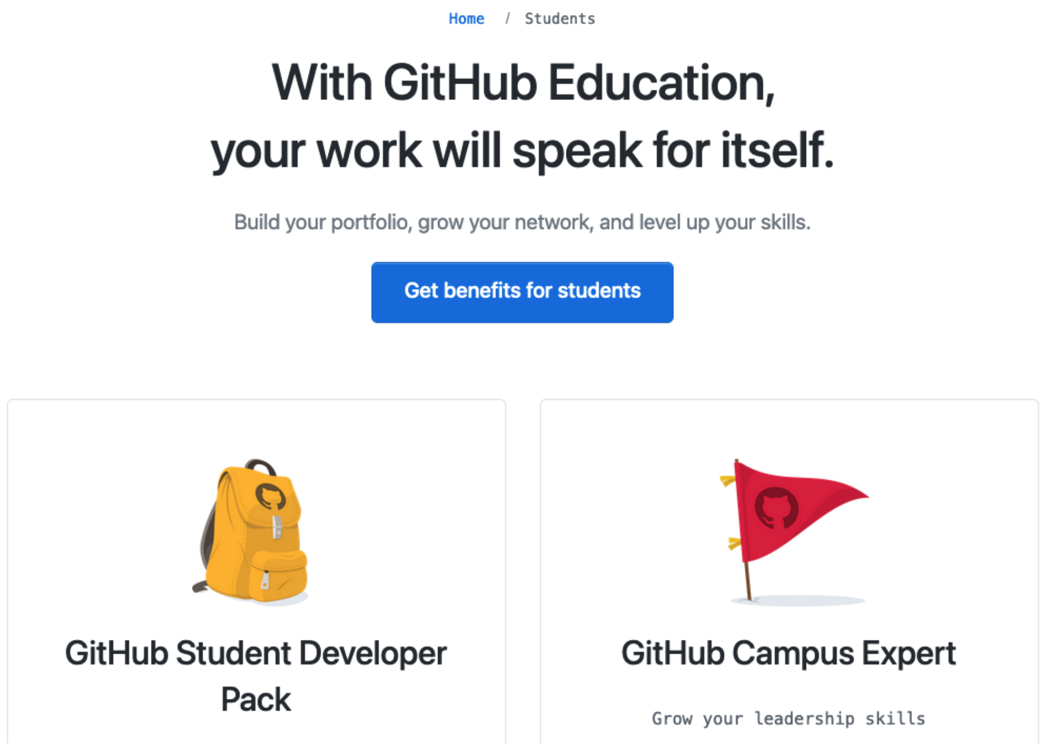
Tip
Want a simple but powerful git-client for your mac?
Try Tower: www.git-tower.com/

Tools for git

- tig (https://github.com/jonas/tig)
- sourcetree (https://www.sourcetreeapp.com/)
- GitUp (mac only :/) (https://gitup.co/)
- gitKraken (not free, but part of GitHub Education :/) (https://www.gitkraken.com/)
- most IDEs have built in support for git
- Free git service: github, bitbucket, gitlab

GitHub

https://education.github.com/students



With GitHub Education,
your work will speak for itself.

Build your portfolio, grow your network, and level up your skills.

GitHub Student Developer Pack

GitHub Campus Expert

Grow your leadership skills

Testing

- Unit tests (!)
- doc tests
- CI (continuous integration) / CD (continuous delivery)

In [16]:
```python
import unittest

def fun(x):
    return x + 1

class MyTest(unittest.TestCase):
    def test(self):
        self.assertEqual(fun(3), 4)

# Testing
# Normally: unittest.main()
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
```

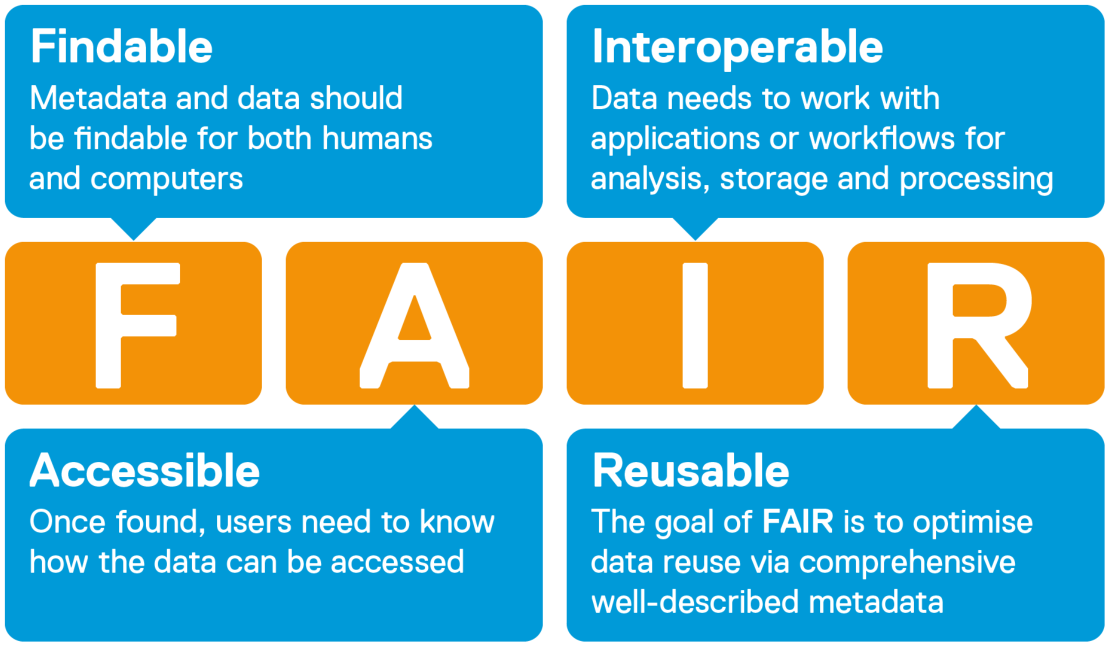
Out[16]:
```
<unittest.main.TestProgram at 0x7f88c2899910>
```

In [17]:
```python
def square(x):
    """Return the square of x.

    >>> square(3)
    9
    >>> square(-2)
    4
    """

    return x * x

# Testing
import doctest
doctest.testmod()
```

Out[17]:
```
TestResults(failed=0, attempted=2)
```

Data handling - FAIR principles

Publication in Nature Data Science - https://doi.org/10.1038/sdata.2016.18


Findable
Metadata and data should be findable for both humans and computers
Interoperable
Data needs to work with applications or workflows for analysis, storage and processing
F
A
I
R
Accessible
Once found, users need to know how the data can be accessed
Reusable
The goal of FAIR is to optimise data reuse via comprehensive well-described metadata

Use case - HemoCell (www.hemocell.eu)

0

Points of interest:

- Developped in C++ with processing scripts in Python and Bash
- The source code is version controlled under git (GitHub)
- Uses 2 CI/CD servers
- Edited mostly in Visual Studio Code, Sublime Text, and NetBeans
- The documentation is written in Markdown(-ish) text
- The publications are written in Overleaf
- Data visualization through ParaView and Blender
- Data evaluation through Python (+ HDF5, VTK, ...)
- Presentations, posters done in PowerPoint
- Illustrative graphics in Inkscape

Take home message

1. Always aim for work quality suitable for collaboration!
2. Write tidy, well commented code. I.e.: "Will I understand my code completely by reading this 2 years from now?"
3. Write documentation! You'll be glad you did few years down the line.
4. Use version control, even if you are the only developer.
5. Try to cover with tests as much as possible.
6. Visualize whenever possible, develop fast pipelines for visualization.
7. Use multiple visualization, observe it from different angles before you decide how will you present it.
8. Every statement you write down requires evidence! I.e., data produced by you or references (scientific ones, not wikipedia).

Homework assignement

https://github.com/gzavo/CS_Assignment

1. Create a free github account if you don't already have one.
2. Fork this repository.
3. Create a markdown (.md) named "solution_.md" file that will contain the following:
4. The title of the following papers pivotal to our knowledge:
   - MCC Van Dyke et al., 2019
   - JT Harvey, Applied Ergonomics, 2002
   - DW Ziegler et al., 2005
5. Create 1 plot from the dataset "istherecorrelation.csv", with DPI=300. The objective is to visualize the data as you see fit. Include the resulting image in the markdown file (and you can also write a few lines of interpretation if you like).
6. Commit and push these two files to your fork.
7. Create a pull request for me to this (original) repo. (Hint: use "compare across forks").

Thank you for your attention!

Questions?