# Python Programming and Machine Learning for Economists (August 2022)

Michael E. Rose, PhD

# Introduction

# Who am I?

- Senior Research Fellow, Max Planck Institute for Innovation and Competition, PhD in Econ (University of Cape Town)
- Writing code since 8th grade
- Author of 3 open-source projects: `pybliometrics`, `sosia`, `scholarmetrics`
- Teaching experience:
  - *This course* @ Kiel Institute for the World Economy (ASP), University of Zurich, ifo Institute Munich, LMU Munich, Scheller College of Business at Georgia Tech, TU Munich
  - Risk Management Computing Skills [Matlab, SQL, Excel, VBA] @ University of Cape Town
- Michael.Ernst.Rose@gmail.com

# Who are you?

- Name, Status
- Which languages, how long?
- Which operating system?
- Who is more in control, your computer or you?

# Course content

1. Empirical research using Python
2. Project management
3. Unsupervised Machine Learning
4. Supervised Machine Learning
5. Natural Language Processing

# Course Design

- Lecture in the morning, exercises in the afternoon
- Each exercise session starts with a Monty Python sketch
- 10 Minutes breaks after 50 Minutes of Teaching

# Exercises (= mini projects)

- Difficulty increases as the course progresses

Data sets
in tutorials

Data sets in
the wild

- Your grades depend on the exercises of days 3, 4 and 5
- The exercise on the 2nd day is optional, but recommend to all newbies

# Learning outcomes

- Programming part
  1. List some of the right basic tools for empirical research
  2. Use python independently
  3. Apply `pandas`, `seaborn`, `sklearn`
  4. Understand coding principles
  5. Use PyCharm
  6. Understand and use version control and use git
- Machine Learning
  1. Apply simple Neural Networks, clustering algorithms and Principal Component Analysis
  2. Interpret and evaluate any machine learning application
  3. Teach yourself how to apply machine learning algorithms we don't speak about

# Required Readings

- Shapiro, J. and M. Gentzkow: "Code and Data for the Social Sciences: A Practitioners Guide" - *Short paper on project management by Economists, read it all today*
- Athey, S. and G. Imbens (ARE 2019): "Machine Learning Methods That Economists Should Know About" - *Well-written overview that introduces all the technical terms for meachine learning, read it until 3rd day*
- Gentzkow, M., B. Kelly and M. Taddy (JEL 2019): "Text as Data" - *Well-written introduction to language processing, read it until last day*

# How to use Python

# Why Python?

- Interpreted, high-level, general-purpose programming language
- Can be object-oriented, imperative, functional and procedural
- Free (= no licenses)
- Large (= support and many packages)
- Centralized development
- Very good first language

# Why Python?

- Interpreted, high-level, general-purpose programming language
- Can be object-oriented, imperative, functional and procedural
- Free (= no licenses)
- Large (= support and many packages)
- Centralized development
- Very good first language

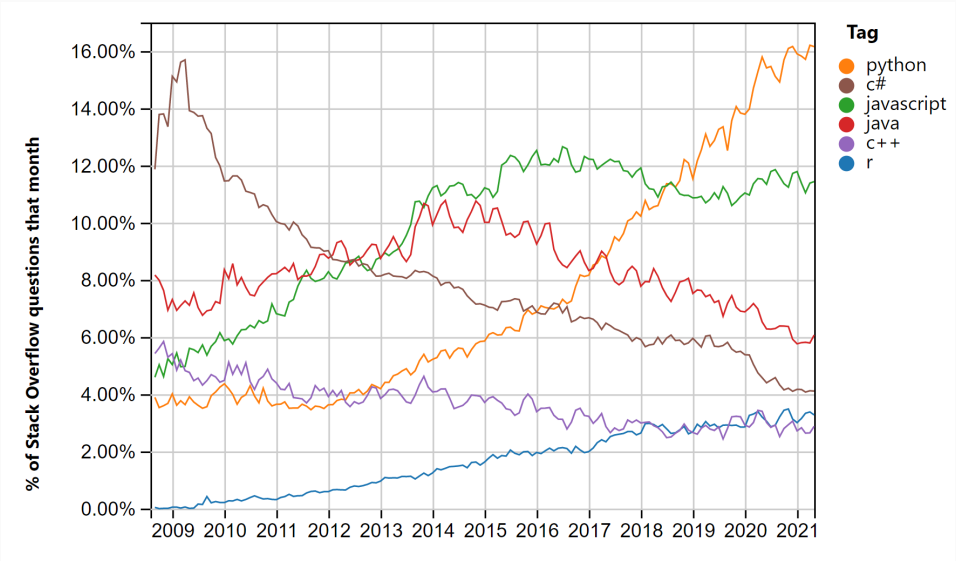
*There should be one– and preferably only one –obvious way to do it. Although that way may not be obvious at first unless you're Dutch.* (Tim Peters - The Zen of Python)

# Credit where Credit is due

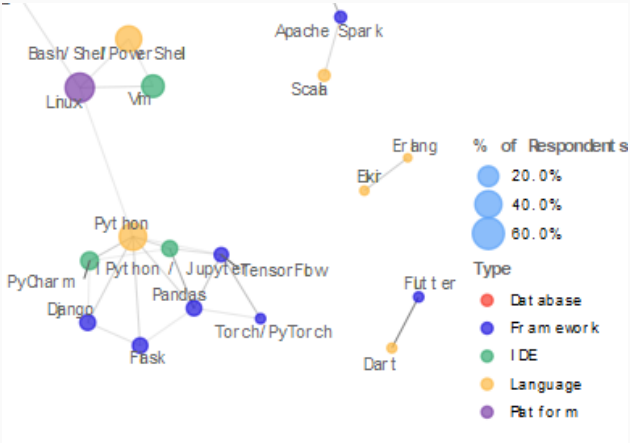
- Guido van Rossum
  created Python in his Christmas holidays 1989 as
  > *"a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus)."*

- Since 2019 5-member steering committee at the Python Foundation heads the development of Python

# Python is popular and increasing in popularity



# Python's local technology cluster




StackOverflow.com: "Developer Survey Results 2019"

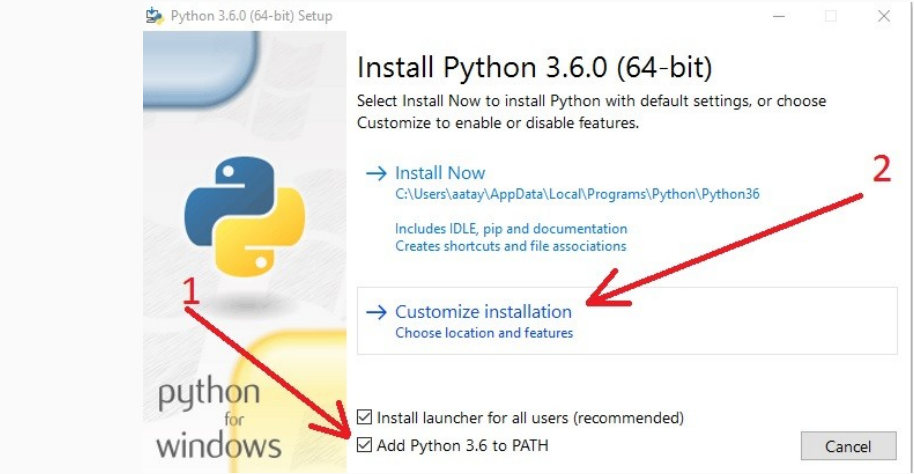
# Why I discourage anaconda

- packages provided by anaconda need to be installed with `conda install` (they will ONLY be in the conda environment)
- Main difference in the past: `conda` used to be a better package manager than `pip`
- packages part of conda might be outdated
- Overkill/Unnecessary software (RStudio)
- Jupyter and spyder run without anaconda as well
- Actually not *that* popular: 19% of Python installations via Anaconda[1]

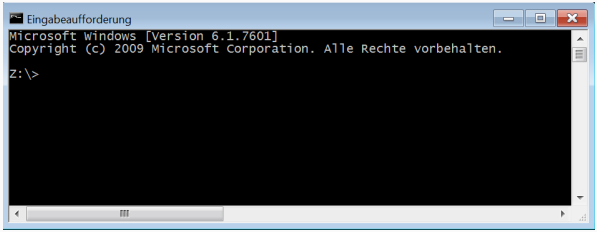
[1] Python Developers Survey 2020 Results

# Installing Python and pip

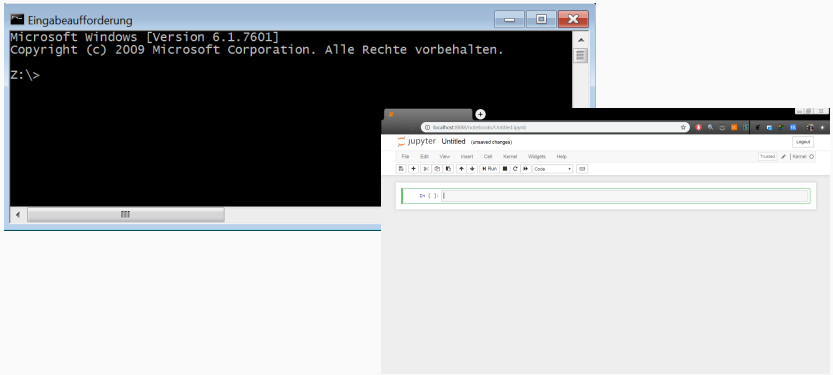
https://www.python.org/downloads/



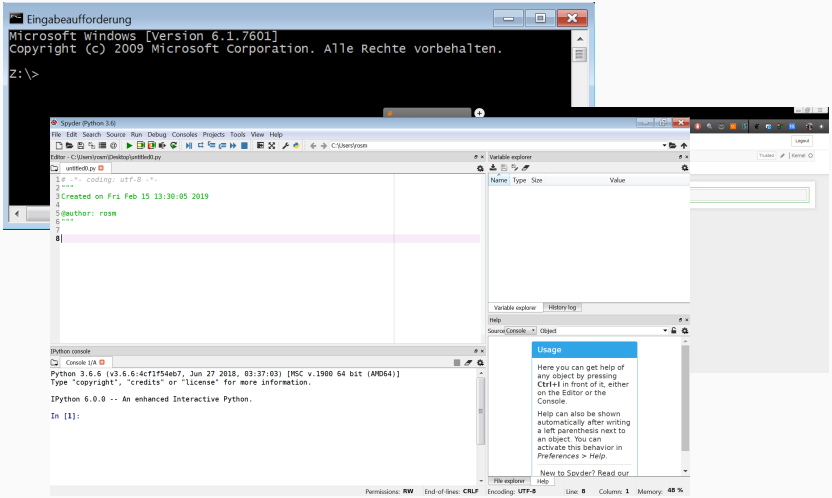
# Different ways to use Python



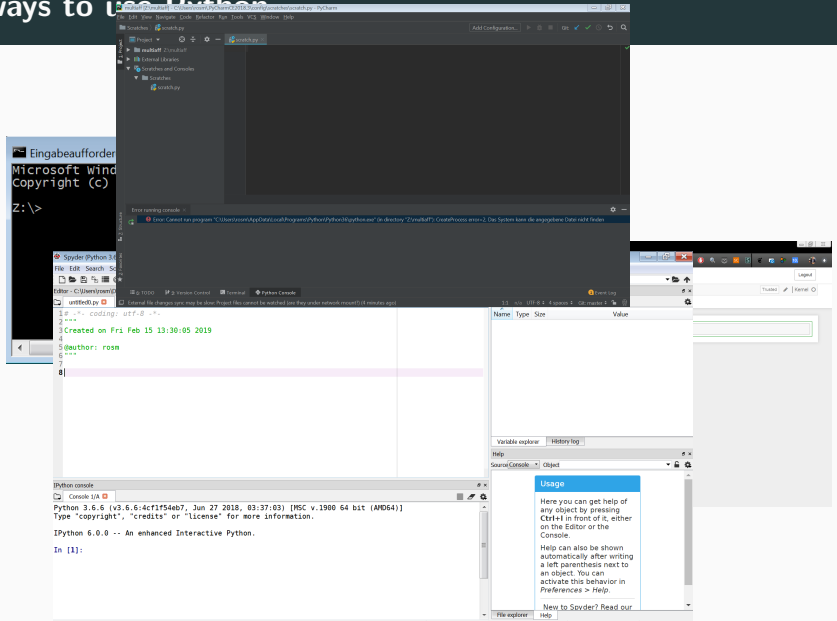
# Different ways to use Python



# Different ways to use Python



# Different ways to use Python



# Terminal/Console

- Console uses DOS language (Windows) or `shell` and `bash` (Linux and Apple)
- Starts python environment, Jupyter, and executes scripts

# Terminal/Console

- >_ Console uses DOS language (Windows) or `shell` and `bash` (Linux and macOS)
- >_ Starts python environment, Jupyter, and executes scripts
- >_ Install packages here:
  - Windows: `python -m pip install pandas seaborn`
  - Linux/macOS: `python3 -m pip install pandas seaborn`
- >_ Shortcut (which is not platform-independent)
  - Windows: `pip install pandas seaborn`
  - Linux/macOS: `pip3 install pandas seaborn`

# Jupyter Notebook on your computer

- Create a folder for this course and navigate there in your terminal (alternatively, open the "PowerShell" via context menu after ⇧+rightclick)

# Jupyter Notebook on your computer

- Create a folder for this course and navigate there in your terminal (alternatively, open the "PowerShell" via context menu after ⇧+rightclick)
- Install the jupyter notebook if necessary

```
python3 -m pip install notebook
jupyter notebook
```

- Your browser will fire up (i.e., you started your own server)

# Jupyter Notebook on your computer

- Create a folder for this course and navigate there in your terminal (alternatively, open the "PowerShell" via context menu after ⇧+rightclick)
- Install the jupyter notebook if necessary

```
python3 -m pip install notebook
jupyter notebook
```

- Your browser will fire up (i.e., you started your own server)
- Click on New in the upper right corner to start a new notebook

Notebooks will be saved in the folder where you invoked the jupyter server

# Jupyter notebook in the ☁

- colab.research.google.com: requires Google account; stores notebooks in your Drive; integrates with GitHub; potentially older packages
- kaggle.com/code: requires Kaggle account; allows for R as well
- mybinder.org: requires GitHub account; builds from a GitHub repository

# Recap some Python basics

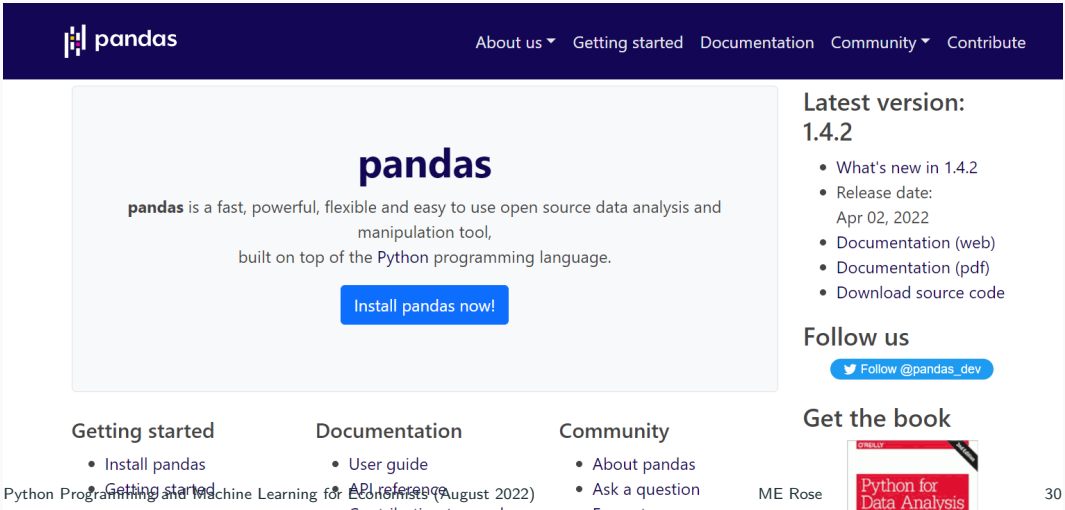
What matters in Python?

- Indentation is key (convention: four spaces)
- Case-sensitive
- Variables must not start with numbers
- It's a language, *not* a program

# Pandas

# pandas: the library for data manipulation

- Documentation: http://pandas.pydata.org/pandas-docs/stable/



# Let's start with a dataset on twins...

```
1 import pandas as pd
2
3 FNAME = "http://www.stat.ucla.edu/~rgould/datasets/twins.dat"
4
5 df = pd.read_csv(FNAME, sep='\t')
```

- Documentation at
  http://www.stat.ucla.edu/~rgould/datasets/twinsexplain.txt

# pandas functionality relevant for the course

- 10 minutes to pandas
- IO tools (text, CSV, HDF5, ...)
- Indexing and selecting data
- Reshaping and pivot tables
- Working with missing data
- Computational tools

# Let's inspect our data

```
df.shape  # Dimensions
df.head()  # First 5 lines (by default)
df.tail(7)  # Last 7 lines
df.columns  # List of variables
df.describe()  # Summary statistics
```

1. How many observations do you have?
2. How many variables do you have?
3. Which variables are numeric?
4. What is the mean of variable "DEDUC1"?

# Slicing the DataFrame

```python
# Selecting columns
df["DEDUC1"]  # Column by column name
df[["AGE", "LHRWAGEH"]]  # Columns by list of column names
df.iloc[:, 5:7]  # Column range by column indices

# Selecting rows
df.loc[0]  # Row by index name (also accepts lists)
df.iloc[0]  # Row by row number (also accepts lists)

# Selecting values
df.loc[18, "AGE"]  # Name of row and column
df.iloc[18, 2]  # Index of row and column
```

# Slicing the DataFrame

```python
# Selecting columns
df["DEDUC1"]  # Column by column name
df[["AGE", "LHRWAGEH"]]  # Columns by list of column names
df.iloc[:, 5:7]  # Column range by column indices

# Selecting rows
df.loc[0]  # Row by index name (also accepts lists)
df.iloc[0]  # Row by row number (also accepts lists)

# Selecting values
df.loc[18, "AGE"]  # Name of row and column
df.iloc[18, 2]  # Index of row and column
```

1. What is the 6th entry of the 5th column?
2. What is the 5th entry of column `"DTEN"`?
3. What is the last entry of column `"LHRWAGEL"`?

# Understanding dtypes

```
1 df.info()
```

# Understanding dtypes

```
1 df.info()
```

| Pandas | Python | Purpose |
|---|---|---|
| object | unicode | Text |
| int64 | int | Integers |
| float64 | float | Floating numbers |
| bool | bool | True & False values |
| datetime64 | | Date and time values |
| timedelta[ns] | | Differences between two datetimes |
| category | | Finite list of text values |

# Changing dtypes

```
df["WHITEH"] = df["WHITEH"].astype(bool)
df["DMARRIED"] = df["DMARRIED"].astype("category")
df["LHRWAGEH"] = pd.to_numeric(df["LHRWAGEH"], errors="coerce")
```

# Optimising dtypes

```
df.info(memory_usage=True)
```

# Optimising dtypes

```
df.info(memory_usage=True)
```

```
bools = ['WHITEH', 'MALEH', 'WHITEL', 'MALEL']
df[bools] = df[bools].astype(bool)
df['DMARRIED'] = df['DMARRIED'].astype('int8')
df.info(memory_usage=True)
```

# Boolean indexing

```
df[df["AGE"] > 20]  # One condition
df[(df["AGE"] > 20) & (df["WHITEL"] == 1)]  # Multiple conditions
df[~(df["AGE"] > 20)]  # Tilde inverses boolean
values = (20, 21, 22, 23)
df[df["AGE"].isin(values)]  # Select specific values
```

# Boolean indexing

```
df[df["AGE"] > 20]  # One condition
df[(df["AGE"] > 20) & (df["WHITEL"] == 1)]  # Multiple conditions
df[~(df["AGE"] > 20)]  # Tilde inverses boolean
values = (20, 21, 22, 23)
df[df["AGE"].isin(values)]  # Select specific values
```

1. How many observations have "WHITEL" equal to 0?
2. How many observations have "WHITEH" equal to 1 and "DEDUC1 unequal to 0?
3. In how many rows do the values for "WHITEH" and "WHITEL" differ?
4. What is the mean age of twins whose L-sibling is a non-white male with either 12 or 14 years of education? (Use "WHITEL", "MALEL" and "EDUCHL",)

# Aggregate data

```
df["WHITEL"].value_counts()
pd.crosstab(df["WHITEH"], df["WHITEL"])
```

# Aggregate data

```
df["WHITEL"].value_counts()
pd.crosstab(df["WHITEH"], df["WHITEL"])
```

1. What is the most common value in "EDUCL"?
2. What is the most common combination of "MALEH" and "MALEL"?

# Manipulation

```python
# Representation
df = df.sort_values(by='HRWAGEH')  # Sorting by column
df = df[sorted(df.columns)]  # Re-order columns alphabetically
# Work on columns
df = df.drop('AGESQ', axis=1)  # Drop a column
df['new'] = 9  # Add new column
df['AGETR'] = df['AGE']**3
df['combined'] = df['MALEH'] + df['EDUCH']
# Missing data
df["HRWAGEH_new"] = df["HRWAGEH"].fillna(0)  # Fill missings with 0
df = df.dropna(subset=["HRWAGEH"])  # Drop rows missing in "HRWAGEH"
```

# Grouping

```
grouped = df.groupby(['MALEH'])
print(grouped['AGE'].mean())
print(grouped['EDUCH'].agg(['mean', 'sum']))
print(grouped[['EDUCH', 'AGE']].agg(['mean', 'std']))
```

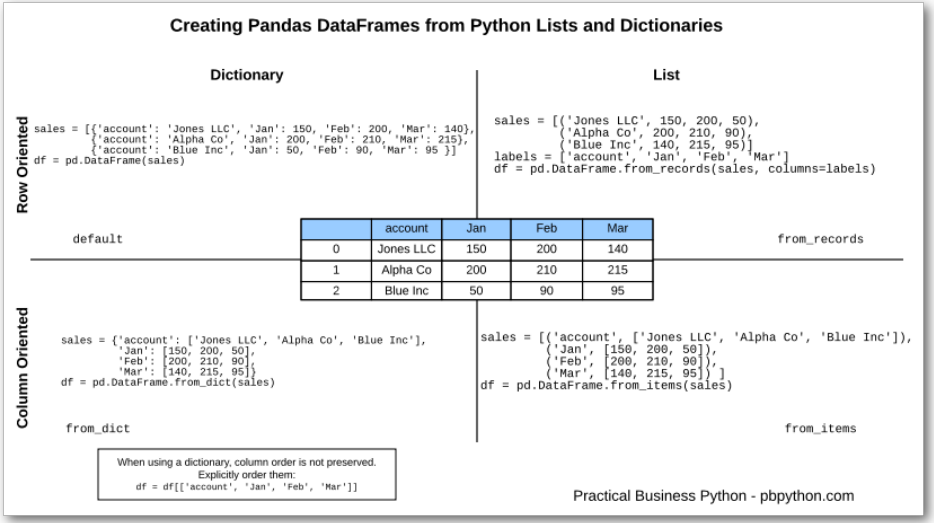
# Grouping

```
grouped = df.groupby(['MALEH'])
print(grouped['AGE'].mean())
print(grouped['EDUCH'].agg(['mean', 'sum']))
print(grouped[['EDUCH', 'AGE']].agg(['mean', 'std']))
```

→ Full list at https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html#aggregation

- What is the "AGE" variance for "MALEL" == 0 individuals?
- What are the second and the third quartile of years of schooling for female L-siblings? (Use "EUDCL" and "MALEL" == 0)
- What is the average "AGE" for twins where both siblings are female?

# Creating DataFrames from other objects



# To become a Master...

- 10 minutes to pandas
- Wes McKinney: "Python for Data Analysis. Data Wrangling with Pandas, NumPy, and IPython", O'Reilly (2017)
- Fabio Nelli: "Python Data Analytics. Data Analysis and Science Using Pandas, matplotlib, and the Python Programming Language", Apress (2015)

# Plotting w/ pandas (matplotlib), and w/ seaborn



# Visualization with pandas

- Straightforward plotting as DataFrame methods for all kinds: barplots, areas, histograms, violin plots, timeseries, etc.:
  https://pandas.pydata.org/pandas-docs/stable/visualization.html
- Has `matplotlib` under the hood - for aesthetics
  `import matplotlib.pyplot as plt`
- Set global styles with `plt.style.use('<style>')` (list all styles with `plt.style.available`)

! Beware: Have DataFrame in correct format (long vs. wide)

# Statistical plotting with seaborn

- seaborn: wrapper for `matplotlib`, optimized for quick statistical plotting: Error bars, distributions, regressions, etc.

- Use seaborn's toy datasets using `.load_dataset()`

☞ If downloading example datasets via `.load_dataset()` doesn't work:

- run `sudo /Applications/Python\ 3.10/Install\ Certificates.command` in Terminal
- get the data manually from github.com/mwaskom/seaborn-data (search the data file, open it, right-click on "Raw" and select "Save link as") and store them in `~./seaborn-data/`

# Seaborn's plotting philosophy

- Statistical relation between numeric values?
  - ➔ `relplot()` for Scatter and Line (→ Documentation)
- Categorical data?
  - ➔ `catplot()` for Scatter-like (Swarm and Strip), Distributions (Box, Violin, Boxen) and Estimations (Point, Bar, Count) (→ Documentation)
- Linear relationships?
  - ➔ `regplot()` (→ Documentation)

# Pandas plotting vs. seaborn

- In Jupyter, remember to write and execute `%matplotlib inline` in first cell to show figures
- Use pandas when you do the aggregations yourself
- Use seaborn when you use raw data – seaborn will aggregate itself

# Excourse: colormaps

List of named colors in matplotlib

Color maps in matplotlib

Color maps in seaborn

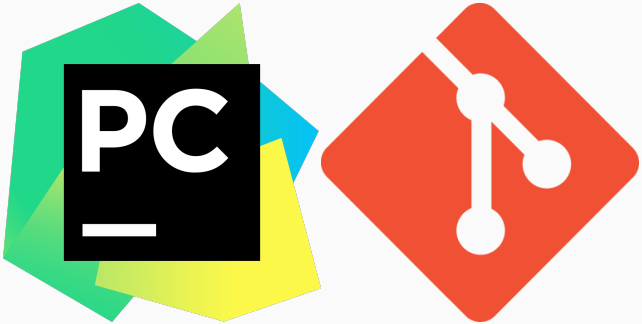
# To become a Master...

- Fabio Nelli: "Python Data Analytics. Data Analysis and Science Using Pandas, matplotlib, and the Python Programming Language", Apress (2015)
- matplotlib Tutorials
- seaborn User guide and tutorial

# Recap Day 1

- Use the Terminal/Console to install new packages, upgrade using `--upgrade` flag
- Consult the package's documentation for parameter names, defaults and examples
- Python is object-orientated: don't forget to reassign after working with an object

# Project Management with PyCharm and git



# Proper Data Management

- ... increasingly required by funders (as of 2021, ERC grant holders have to have a Research Data Management Plan in place)
- ... usually entails a backup system, maybe with versioning
- ... enables you to keep track of your progress
- ... facilitates working with others

# Proper Data Management

- ... increasingly required by funders (as of 2021, ERC grant holders have to have a Research Data Management Plan in place)
- ... usually entails a backup system, maybe with versioning
- ... enables you to keep track of your progress
- ... facilitates working with others

! Remember: You are your first re-user of your data
  - Documentation
  - Accuracy
  - Replicability

# Ten Simple Rules for Reproducible Computational Research

1. For Every Result, **Keep Track** of How It Was Produced
2. Avoid **Manual Data Manipulation** Steps
3. **Archive** the Exact Versions of All External Programs Used
4. **Version Control** All Custom Scripts
5. Record All **Intermediate Results**, When Possible in Standardized Formats
6. For Analyses That Include Randomness, Note Underlying **Random Seeds**
7. Always Store **Raw Data** behind Plots
8. Generate **Hierarchical Analysis Output**, Allowing Layers of Increasing Detail to Be Inspected
9. Connect Textual Statements to **Underlying Results**
10. Provide **Public Access** to Scripts, Runs, and Results

Geir K. Sandve et al. (2013): "Ten Simple Rules for Reproducible Computational Research", Plos ONE.

# More control for users

- Show file endings - How?
- Show hidden files - How?

# Simple rules for an Economist's project directory

- "Automate everything that can be automated."
- "Store code and data under version control."
- "Separate directories by function."
- "Separate files into inputs and outputs."
- "Manage tasks with a task management system."

# Why PyCharm?

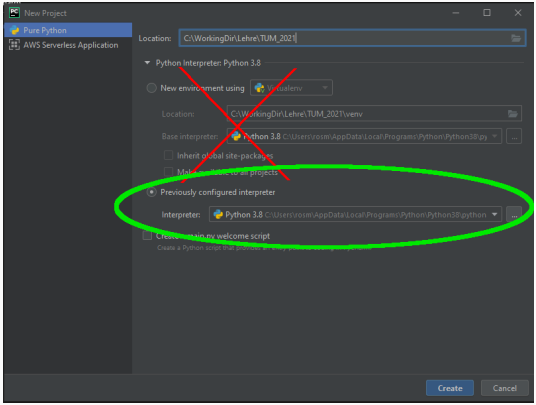
- Integrated Developer Environment (IDE), i.e. terminal, editor, object explorer, etc. in a single window
- Project-aware: Knows of usage of imported functions elsewhere etc.
- Integrates with version control systems and also Amazon Web Services (AWS)
- Community edition is free (→ Download)

🏆 Most used editor or IDE in 2020, with 33% of developers[2]

[2] Python Developers Survey 2020 Results

# Starting a project in PyCharm

1. (Install and )Open PyCharm
2. In the Welcome screen, click on "Open" and open the folder where you saved your notebook yesterday
3. Do **NOT** create a new/virtual environment (*venv*), rather (set and )use the system interpreter( to your python installation)
4. `main.py` Welcome Script not necessary



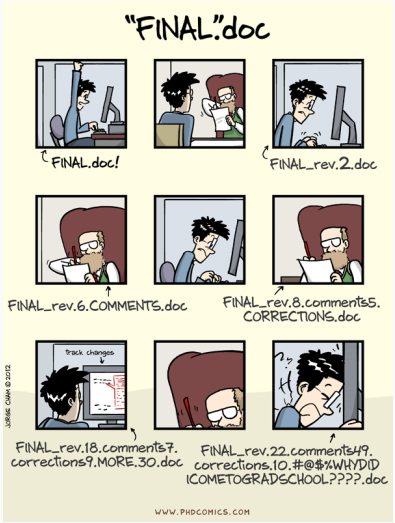

jetbrains.com/help/pycharm/creating-and-running-your-first-python-project.htmls

# Why does git exist?

- Git protects yourself and others from yourself and others
- You can modify/change/break/improve your code and data, secure in the knowledge that you can not ruin your work too badly
- **No** commercial software is written without Version Control!
- Lots of open-source projects as well:
  - pandas, scikit-learn, seaborn, ggplot2, ⋯
- Very handy to compare recent changes against history
- Almost all Python developers use version control at least sometimes[3]

[3] Python Developers Survey 2020 Results

# With git you *never* change the file name



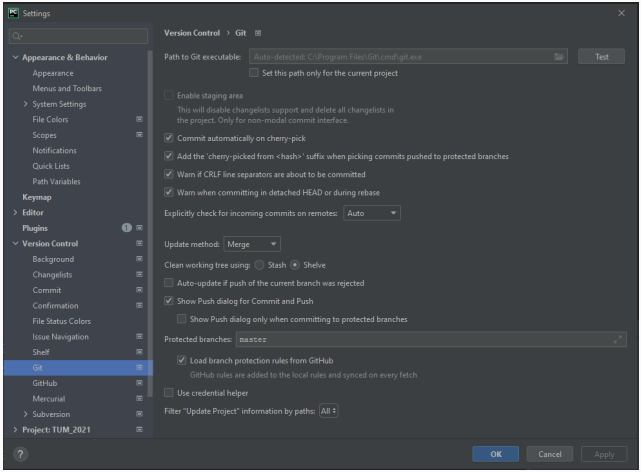
# How does git work?

1. You tell git which files to keep track of ("checking-in")
2. ... eventually to store snapshots of changes of tracked files ("committing")
3. ... on top of previous commits ("repository")

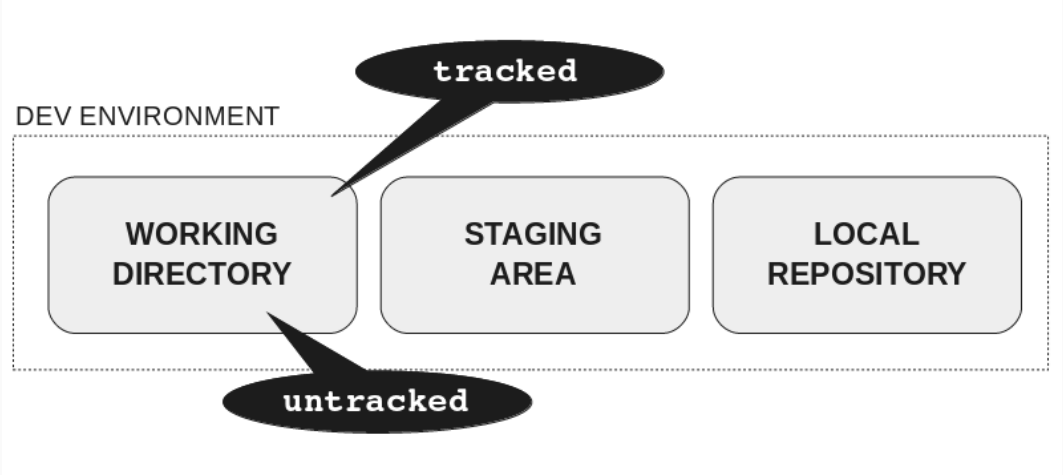
$\longrightarrow$ git manages changes to a project without overwriting any part of it

# Configuring git in PyCharm

1. (Install git from git-scm.com/download)
2. File | Settings (: PyCharm | Preferences) > Version Control > Git → Set "Path to Git executable" (often auto-detected)
3. VCS | Enable Version Control Integration → select "Git"
4. Click on green marker to open git dialogue



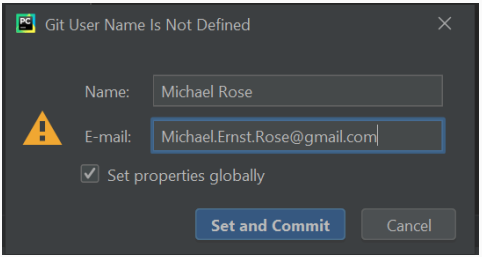
# git's architecture




from: Rachel Carmena (2018): "How to teach Git"

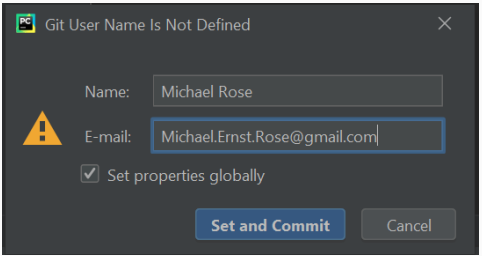
# Telling git who you are

On first commit, PyCharm prompts for name and email address



# Telling git who you are

On first commit, PyCharm prompts for name and email address




Alternatively, you may state your identity via the terminal:

```
$ git config --global user.name "<Your real name>"
$ git config --global user.email <Your real email address>
```

If you plan to use git outside of PyCharm also set the editor

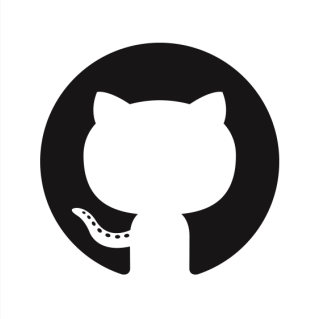
# The `.gitignore` file

- Small file to specify files and folders you do not want to track → Documentation
  - PyCharm's `.idea` folder
  - temp files from Stata, Python, R, etc.
  - Windows' database files
- Works best with regex → Templates
- Hidden on *nix systems; show with ctrl + h

# To become a Master...

- PyCharm's playlist Getting Started with PyCharm (13 videos)
- PyCharm's Knowledge Base

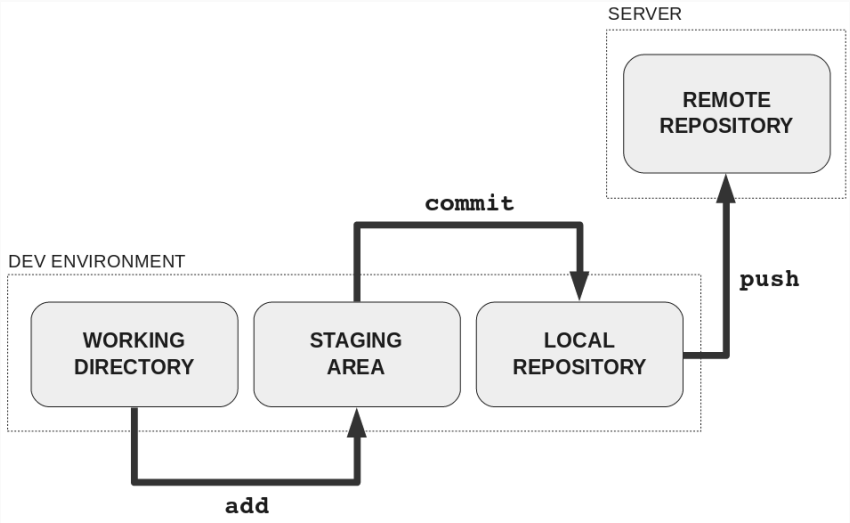
# Collaborating with GitHub and/or GitLab



# What's the difference?

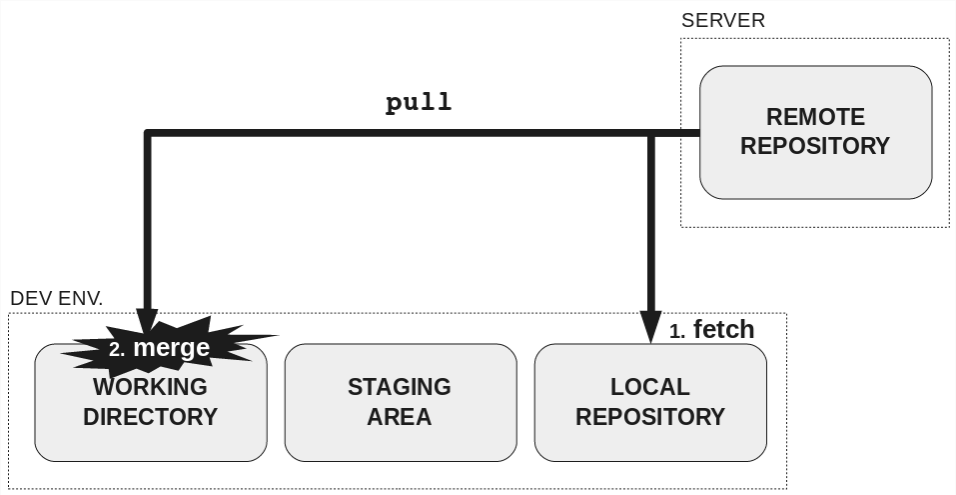
- git: Version control *on your machine*
- GitHub: Cloud storage accessible from git
- GitLab: GitHub for projects that require continuous integration (CI), i.e. web-apps

# How do your changes make it to GitHub/GitLab?




from: Rachel Carmena (2018): "How to teach Git"

# How do others' changes make it to your system?




from: Rachel Carmena (2018): "How to teach Git"

# Configuring GitHub in PyCharm

1. File | Settings > Version Control > Git → check "Credential Helper"
2. File | Settings > Version Control > GitHub → Click "Add Account"
   - Create an account, or
   - Sign in

⚠ If in future your commits don't make it to GitHub, verify on this page that you're still connected to GitHub

---

If you plan to use GitHub outside PyCharm:

```
$ git config --global credential.helper cache
$ git config --global user.password "<Your GitHub password>"
```

# Option 1: You have a local repo and want to have it on GitHub

1. Open PyCharm in the folder you want to have on GitHub
2. (Have at least one commit in repo)
3. Git | GitHub > Share Project on GitHub → Type repository name( and check Private)

👉 With GitLab this doesn't work (yet)

# Option 2: You have a repo on GitHub/GitLab and want it locally ("cloning")

1. Create a (preferably private) repository on github.com (click "+" top right)
2. Open PyCharm anywhere
3. Either click on
   - VCS | Get from Version Control
   - Git | Clone...
4. In the new window, select "GitHub <your account name>" on the left
5. From the list of repos, select the new one; then on the bottom set the location

👉 PyCharm creates a new folder, turns it into a projects and establishes the connection to GitHub

👉 Do not attempt to clone a remote repo into another local one!

# GitHub

- Repos have unlimited space but no file may be larger than 100MB
- Stars a repo on GitHub to save to your favorites and to say Thank you
- Get Pro benefits for free via GitHub Student Developer Pack) (Added benefit: GitHub hosts a simple private webpage)

# To become a Master...

- GitHub's Learning Lab

# Debugging

# Bad things that can happen to your code

- Syntax Errors: Prevent your code from running (i.e. pre-runtime)
- Runtime Error: Occur during runtime (Exception)
- Semantic Error: Code runs, but not the way you like (Bugs)

# Bad things that can happen to your code

- Syntax Errors: Prevent your code from running (i.e. pre-runtime)
- Runtime Error: Occur during runtime (Exception)
- Semantic Error: Code runs, but not the way you like (Bugs)

**?** Which one of these is a syntax error, which one is a bug, and which one will throw an exception?

1. Attempting to divide by 0
2. Not closing a parenthesis
3. Not dividing by 100 when computing a percentage

# Avoid bugs in the first place

- Write easy code
- Experiment to check your hypotheses
  - `print()` objects to see what they contain
  - `print(type())` objects to see what they are
- Scaffolding: Write, check, repeat (Get something working and keep it working)
- Think formally (unlike in natural languages)
  - No ambiguity
  - Less redundancy
  - Always literal

# Avoid bugs in the first place

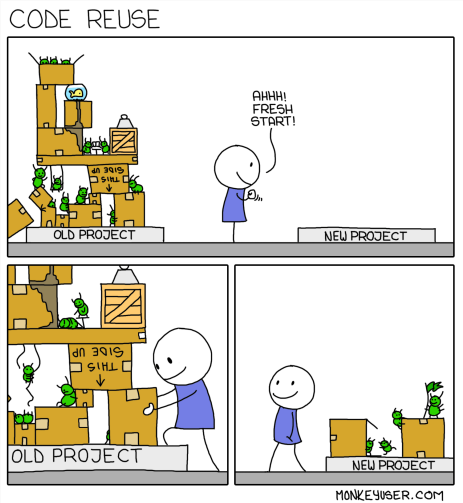
- Write easy code
- Experiment to check your hypotheses
  - `print()` objects to see what they contain
  - `print(type())` objects to see what they are
- Scaffolding: Write, check, repeat (Get something working and keep it working)
- Think formally (unlike in natural languages)
  - No ambiguity
  - Less redundancy
  - Always literal

⌨ The problem always sits behind the keyboard

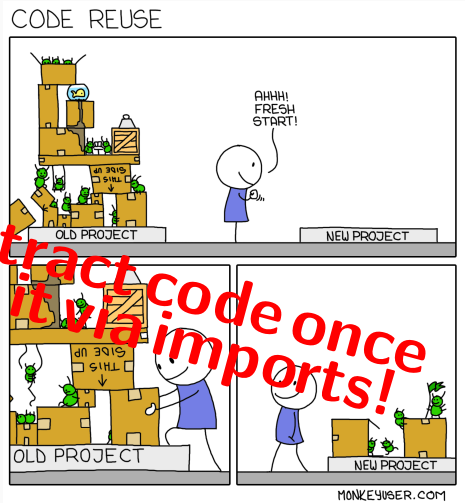
# How to hunt down the bug

- You will spend most of the time debugging
- It's detective work: Where does the bug come from, how to fix it w/o breaking other things
- Tracebacks help you: What kind of error & where (approximately)

# Avoid reusing bad code



# Avoid reusing bad code




**Write abstract code once and reuse it via imports!**

# Make use of tracebacks!

```
Traceback (most recent call last):
  File "./test.py", line 21, in <module>
    main()
  File "./test.py", line 14, in main
    data=tips, legend=False)
  File "/usr/local/lib/python3.6/dist-packages/seaborn/relational.py", line 1613, in relplot
    **plot_kws)
  File "/usr/local/lib/python3.6/dist-packages/matplotlib/__init__.py", line 1810, in inner
    return func(ax, *args, **kwargs)
  File "/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_axes.py", line 4300, in scatter
    collection.update(kwargs)
  File "/usr/local/lib/python3.6/dist-packages/matplotlib/artist.py", line 916, in update
    ret = [_update_property(self, k, v) for k, v in props.items()]
  File "/usr/local/lib/python3.6/dist-packages/matplotlib/artist.py", line 916, in <listcomp>
    ret = [_update_property(self, k, v) for k, v in props.items()]
  File "/usr/local/lib/python3.6/dist-packages/matplotlib/artist.py", line 912, in _update_property
    raise AttributeError('Unknown property %s' % k)
AttributeError: Unknown property xcol
```

# Inspecting the object

```
1 my_list = {'syntax': 10, 'runtime': 99}
2 print(type(my_list))
```

- What is the type of object `my_list`?

# Checking the version

Every decent package has a magic attribute `.__version__`:

```
import pandas as pd

pd.__version__
```

Useful to check whether your version is outdated; assure you're on the latest version before bothering developers

# Assertions

User `assert()` statements in runtime to test conditions *that should never happen*

- document code
- verify input data is not corrupt or results didn't change after data update
- verify the bug didn't occur at (or before) specific line

```
assert x => 0, "x became negative"
# throws an AssertionError if x is negative
```

# Know your error I

```
x = "9"
y = 1
z = x + y
```

# Know your error I

```
1 x = "9"
2 y = 1
3 z = x + y
```

- `TypeError`: you try to combine two objects that are not compatible

# Know your error II

```
currencies = ["dollar", "euro"]
print(currency)
```

# Know your error II

```
1 currencies = ["dollar", "euro"]
2 print(currency)
```

- `NameError`: you refer to an object that does not exist

# Know your error III

```
int("9.0")
```

# Know your error III

```
1 int("9.0")
```

- `ValueError`: the value you passed to a parameter does not pass the function's limitations on the value

# Know your error IV

```
1 marks = [1, 1, 4]
2 print(marks[4])
```

# Know your error IV

```
1 marks = [1, 1, 4]
2 print(marks[4])
```

- `IndexError`: you are referring to an element in a container that does not exist

# Know your error V

```
capitals = {'ger': 'berlin', 'aut': 'vienna'}
print(capitals['fra'])
```

# Know your error V

```
capitals = {'ger': 'berlin', 'aut': 'vienna'}
print(capitals['fra'])
```

- `KeyError`: you are referring to a key in a dict (or dict-like object) that does not exist

# Know your error VI

```
1 my_list = "dbcea"
2 my_list.sort()
```

# Know your error VI

```
1 my_list = "dbcea"
2 my_list.sort()
```

- `AttributeError`: what you want to do with an object is not possible (mostly: the object is not what you think it is)

# Handling exceptions with try-except clauses

To find out how your objects look like *exactly* when code fails, use a try-except clause

```
    try:
        average = sum(a_list) / len(a_list)
    except ZeroDivisionError:
        print(a_list)
```

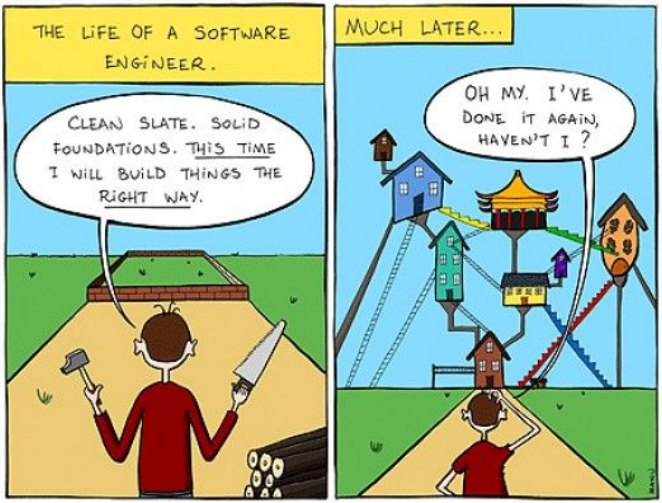
General rule: Catch only specific errors!

# Warnings

- Warnings are messages only
- Warnings do not break runtime
- Most of the time you have `DeprecationWarnings` and pandas' https://www.dataquest.io/blog/settingwithcopywarning/`SettingwithCopyWarning`

If you call me for help saying you have an *error* when in fact you have a *warning*, you owe me a beer

# Refactor as needed



# To become a Master...

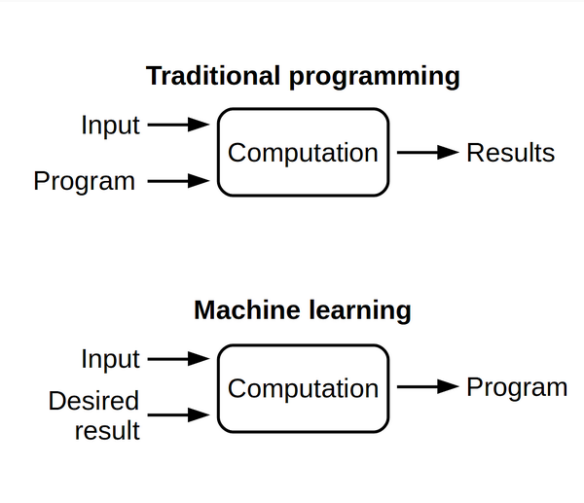
- Allen B. Downey: "Think Python 2e", Green Tea Press (2015)
- Arthur Turrell: "Coding for Economists" (2021)
- "How to Think Like a Computer Scientist: Interactive Edition"
- Garret Christensen, Jeremy Freese and Edward Miguel "Transparent and Reproducible Social Science Research: How to Do Open Science" UC Press (2019)

# Machine Learning for Economists

# Why should you know Machine Learning?

- To understand its impact on the economy
- To make use of text as data
- To create huge, fat datasets based on prediction
- To understand one's datasets better
- Useful for Econometrics

# Relation ML and traditional programming




from: Antti Ajanki (2018): "Differences between machine learning and software engineering"

# General considerations

- Potential to achieve **super-human capability** in learned tasks
- **High quality data** is key: Garbage in, Garbage out
- ML will **err**
- Do not **interpret** anything
- Both science and an **art**

"If a typical person can do a mental task with less than one second of thought, we can probably automate it using AI either now or in the near future."

*Andrew Ng*

# Some definitions and relationships

- Machine Learning: Learning from data
  1. Unsupervised ML: Finding patterns in the unknown
  2. Supervised ML: Predicting from what's known
     - Deep Learning: A multi-layer neural network
  3. Reinforcement Learning: Explore and exploit
- Natural Language Processing: Turning Text to Data
- Artificial Intelligence: ML + decision-making

# Translation: Econometrics to Machine Learning

| Term in Econometrics | Term in ML |
|---|---|
| Dependent variable | Label, Target |
| Variable | Feature |
| Variable construction | Feature engineering |
| estimate, fit | learn, fit |
| coefficient | weight |
| Numerical regression | Prediction |
| Logistic (Multinomial) regression | Classification |
| Dummy | One-hot encoding |
| Bias | Assumptions made to ease learning |
| <Greek letters in formulas> | Hyper-parameters |

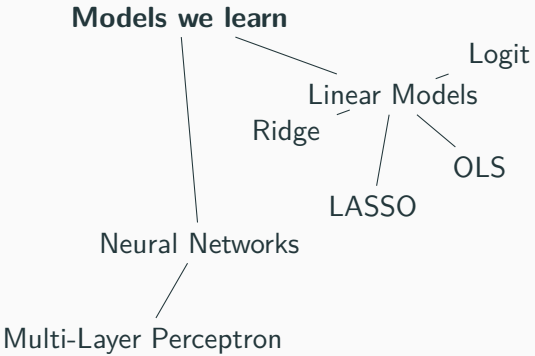
# Feature Engineering

1. Categories into dummies (One-Hot-Encoding)
2. Continuous variables into dummies representing groups (Binning and Discretization)
3. Polynomials
4. Combinations
5. Various moments of distributions

# Supervised Machine Learning

You want to extrapolate from some dataset with certain information

- Prediction tasks (YouTube Silicon Valley 4-4)



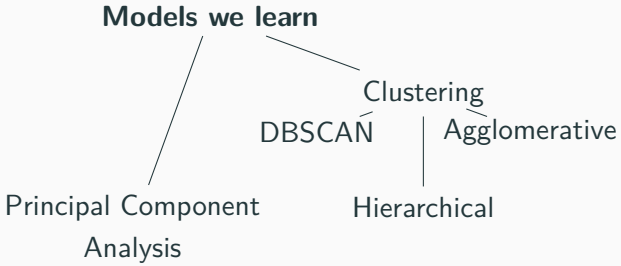
# Examples from Economics

- Policy prediction
  - Andini, Ciani, de Blasio, D'Ignazio & Salvestrini (JEBO 2018), "Targeting with machine learning: An application to a tax rebate program in Italy"
  - Knittel & Stolper (AEA P&P 2021), "Using Machine Learning to Target Treatment: The Case of Household Energy Use"
  - Mullainathan & Obermeyer (QJE 2022): "Diagnosing Physician Error: A Machine Learning Approach to Low-Value Health Care"
- Data generation
  - Blumenstock, Cadamuro & On (Science 2015): "Predicting Poverty and Wealth from Mobile Phone Metadata"
  - Jean, Burke, Xie, Davis, Lobell & Ermon (Science 2016): "Combining satellite imagery and machine learning to predict poverty"
- Experiments
  - Chernozhukov, Demirer, Duflow & Fernández-Val (2020): "Generic Machine Learning Inference on Heterogeneous Treatment Effects in Randomized Experiments, with an Application to Immunization in India"

# Unsupervised Machine Learning

You know nothing about the data

- cluster data to to find patterns and regularities
- reduce dimensions (fewer features), often pre-processing for supervised ML



# Examples in Economics

Dimensionality Reduction

- Nancy Kong, Uwe Dulleck, Shupeng Sun, Sowmya Vajjala and Adam B. Jaffe: "Linguistic Metrics for Patent Disclosure: Evidence from University Versus Corporate Patents," CESifo Working Paper No. 8571.

Clustering

- Marko Terviö (2011): "Divisions within Academia: Evidence from Faculty Hiring and Placement," The Review of Economics and Statistics 93(3), 1053–1062.
- Anil Chaturvedi, J. Douglas Carroll, Paul E. Green and John A. Rotondo (1997): "A Feature-Based Approach to Market Segmentation via Overlapping K-Centroids Clustering," Journal of Marketing Research 34(39), 370–377.

# A primer about `sklearn`

- Very high internal consistency, all models have the same methods
- Datasets stored in dictionaries with labels, explanations and data separated
  1. California House prices (continuous outcome)
  2. Breast cancer (binary outcome)
- For each prediction model there are two classes:
  1. Numerical predictions: Use the regressor class
  2. Categorical predictions: Use the classifier class

# Principal Component Analysis

# Principal Component Analysis

- Represent a large share of your data's variation using fewer features ("dimensionality")
- Algorithmic steps
  1. $\forall$ feature $k$ find linear function $\sum_{j=1}^{p} \alpha_{kj} x_j$ with maximum variance (Think of principal components as maximum variance directions)
  2. *Combine* features in all possible ways such that the combinations are *orthogonal* to each other which maximizes variance
- No hyper-parameter (→ Documentation)

\+ Reduce noise and redundancy

\+ Reduce dimension; For instance, instead of 100 features, use only 40 principal components to represent 95% of variance of original data

\- No interpretation possible

\- Pre-scaling necessary

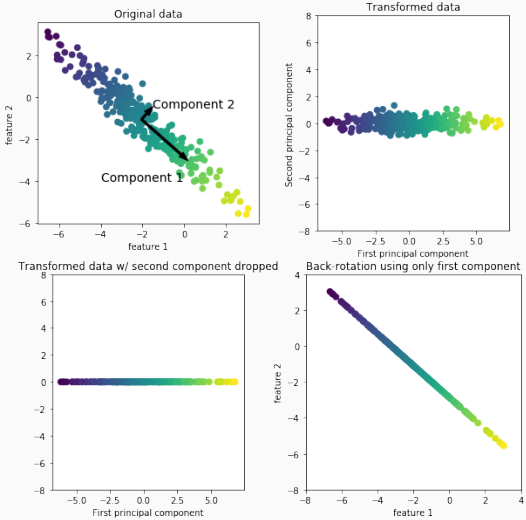
# Principal Component Analysis: Mathematical intuition

1. $\Sigma$ is variance-covariance matrix: $\frac{1}{n-1}\mathbf{X}'\mathbf{X}$
2. Constrained optimization problem: $\text{argmax}\, var(\alpha'_k \Sigma \alpha_k)$ s.t. $(\alpha'_k \alpha_k = 1)$
3. Lagrangian: $\alpha'_k \Sigma \alpha_k - \lambda_k(\alpha'_k \alpha_k - 1)$
4. After partial differentiation: $\Sigma \alpha_k = \lambda_k \alpha_k$

# Principal Component Analysis: Mathematical intuition

1. $\Sigma$ is variance-covariance matrix: $\frac{1}{n-1}\mathbf{X}'\mathbf{X}$
2. Constrained optimization problem: $\text{argmax}\, var(\alpha_k'\Sigma\alpha_k)$ s.t. $(\alpha_k'\alpha_k = 1)$
3. Lagrangian: $\alpha_k'\Sigma\alpha_k - \lambda_k(\alpha_k'\alpha_k - 1)$
4. After partial differentiation: $\Sigma\alpha_k = \lambda_k\alpha_k$
5. Solution: Use eigenvectors of the $k$ largest eigenvalues to form a new matrix $\mathbf{W}$
6. Transform onto subspace: $y = W' \times x$

# Principal Component Analysis: Graphical intuition




from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

# Principal Component Analysis

- Form of **partitional** Clustering
- Aims to minimize variance within a cluster
- Algorithmic steps
  1. Initialize $k$ points as cluster means randomly
  2. Assign each point to closest cluster center (in Euclidean distance)
  3. Reset cluster center as mean of points assigned to it
  4. Repeat 2 and 3 until convergence
- 1 main parameter (→ Documentation)
  1. How many clusters?

\+ Fast and transparent

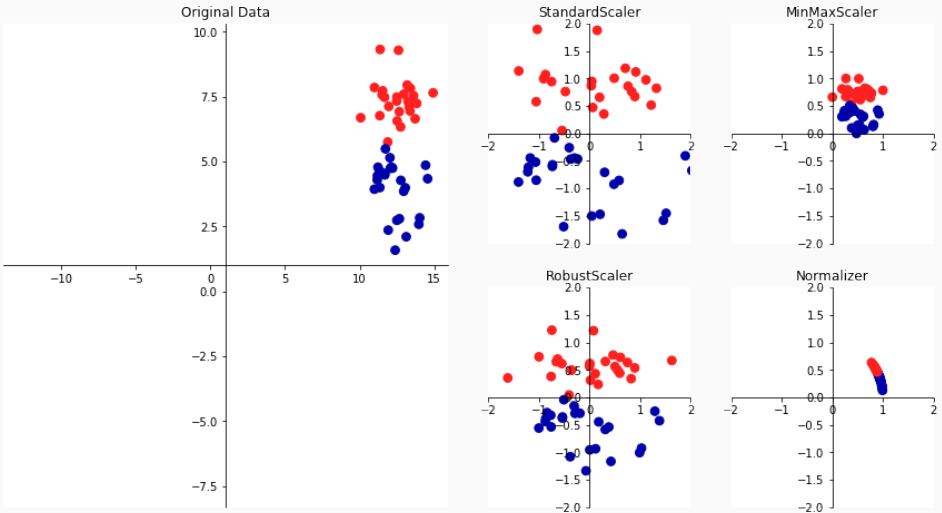
\- Works only with Euclidean distance

\- Performs badly for non-simple shapes (e.g. where clusters don't have same diameter)

# Four scaling classes in `sklearn`

1. `StandardScaler()`: Standarization (mean 0 and variance 1)
2. `RobustScaler()`: Removes median and scales according to inter-quartile range
3. `Normalizer()`: Projection on unit circle
4. `MinMaxScaler()`: Features shifted between 0 and 1
5. `MaxAbsScaler()`: Like `MinMaxScaler()` but works with negative values to ([0, 1], [-1, 0], [-1, 1])

# Four scaling classes in `sklearn`, cont.




from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

# To become a Master...

- Andreas Müller and Sarah Guido: "Introduction to Machine Learning with Python", O'Reilly (2016)

# Clustering

# $k$-Means Clustering

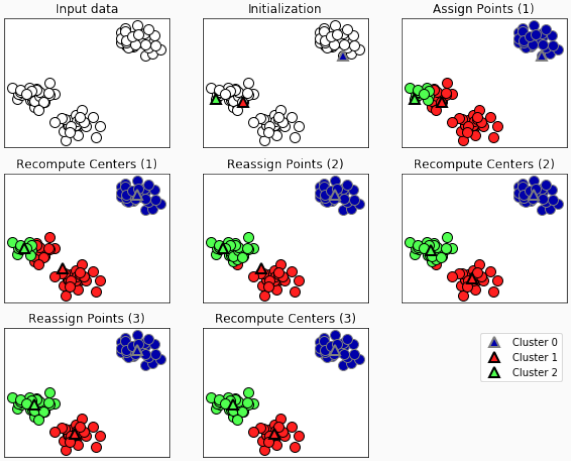
- Form of **partitional** Clustering
- Aims to minimize variance within a cluster
- Algorithmic steps
  1. Initialize $k$ points as cluster means randomly
  2. Assign each point to closest cluster center (in Euclidean distance)
  3. Reset cluster center as mean of points assigned to it
  4. Repeat 2 and 3 until convergence
- 1 main parameter (→ Documentation)
  1. How many clusters?

\+ Fast and transparent

\- Works only with Euclidean distance

\- Performs badly for non-simple shapes (e.g. where clusters don't have same diameter)

# $k$-Means Clustering, cont.




from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

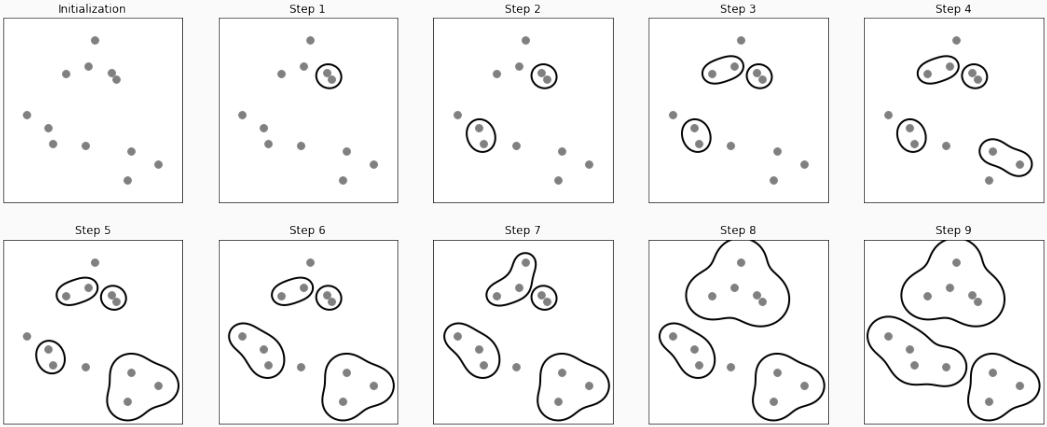
# Agglomerative Clustering

- Form of **hierarchical** clustering
- Algorithmic steps
  1. Make each point its own cluster
  2. Iteratively merge two closest clusters
  3. Stop when k clusters are left
- 3 main parameters (→ Documentation)
  1. Which number of clusters?
  2. Which clustering method?
  3. Which distance measure?

\+ Good for hierarchical data (= nested clusters)

\- No prediction, performs badly for non-simple shapes

# Agglomerative Clustering: Graphical intuition



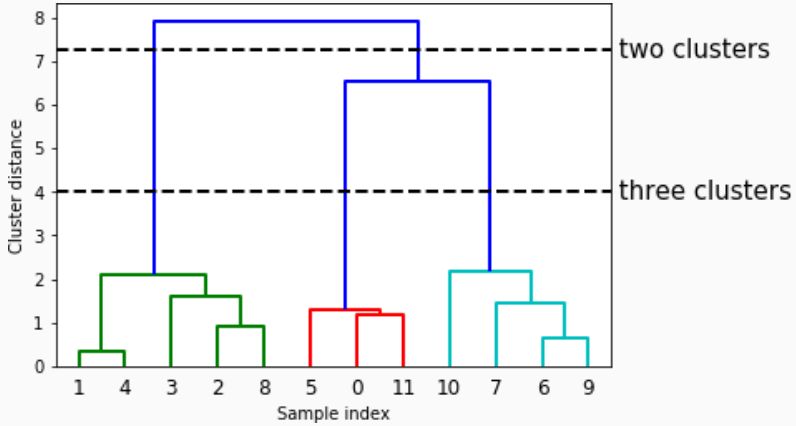

from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

# What is distance?

- Multiple ways to compute distance between two points in multi-dimensional space
- https://scikit-learn.org/0.24/modules/generated/sklearn.neighbors.DistanceMetric.html

# Use a dendrogram to find the optimal $k$

- Visualizes a linkage array, depicting distances between clusters




from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

# Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

- Find clusters satisfying specific conditions
- Algorithmic steps
  1. Pick an arbitrary observation
  2. Check neighborhood of observation based on parameters
  3. Observations in neighborhood become part of cluster and observation itself becomes core, *if* parametric conditions; otherwise observation becomes noise
  4. Repeat until all observations have been visited

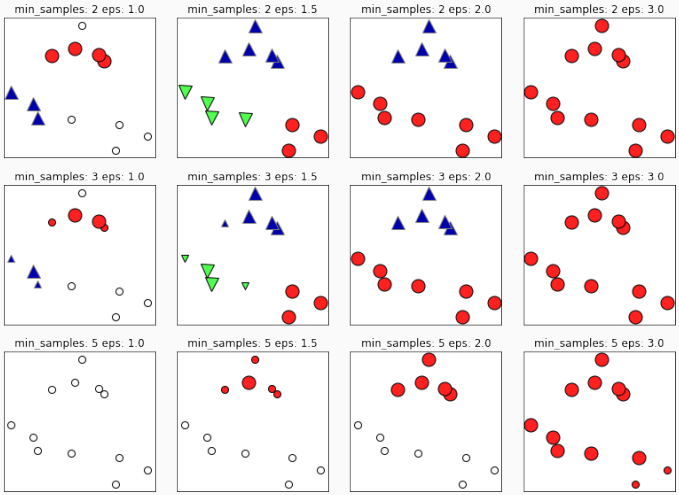
  3 main parameters (→ Documentation)
  1. How many observations in a cluster at least?
  2. How close at least?
  3. Which distance measure?

\+ No a priori number of clusters needed, captures complex shapes

\+ Extensions exist for e.g. geo-clustering

\- Slow

# DBSCAN: Graphical intuition




from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

# Evaluating clusters (in the absence of labels)

1. Silhouette Score → Documentation
   - Mean silhouettes of cluster; Silhouette: (1) compute mean distance to other points in cluster; (2) subtract the mean distance to points in nearest cluster; (3) normalize
   - Ranges between -1 (bad) and 1 (good)
2. Davies-Bouldin score → Documentation
   - Average similarity of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances
   - Ranges between 0 (good) and $\infty$ (bad)
3. Calinski-Harabasz score → Documentation
   - Compute difference for each point to its cluster's centroid and compare that to the difference of each centroid to the global centroid
   - Ranges between 0 (bad) and $\infty$ (good)

# Evaluating clusters (in the absence of labels)

1. Silhouette Score → Documentation
   - Mean silhouettes of cluster; Silhouette: (1) compute mean distance to other points in cluster; (2) subtract the mean distance to points in nearest cluster; (3) normalize
   - Ranges between -1 (bad) and 1 (good)
2. Davies-Bouldin score → Documentation
   - Average similarity of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances
   - Ranges between 0 (good) and $\infty$ (bad)
3. Calinski-Harabasz score → Documentation
   - Compute difference for each point to its cluster's centroid and compare that to the difference of each centroid to the global centroid
   - Ranges between 0 (bad) and $\infty$ (good)

! Remember: Clustering algorithms find clusters because that is what they do - not necessarily because there are clusters

# Should I standardize the data before clustering?

Q: Should different features (potentially with different units) have **equal** weight? E.g., on a feature measured in kilograms and another one in metres, is a 1 unit difference equally significant in both instances?

**No** You should standardize

**Yes** It doesn't hurt to standardize, eventually improves convergence

# To become a Master...

- Andreas Müller and Sarah Guido: "Introduction to Machine Learning with Python", O'Reilly (2016)

# Supervised Machine Learning

# Relation Supervised ML and Econo(metric)s

$$Y = f(X) + \epsilon = X\beta + \epsilon, \text{ with } E[\epsilon] = 0$$

- Economists: What is $\beta$?
- Machine Learner: What is $\widehat{Y}$?
- Both: $\widehat{Y} = \widehat{f(X)} = X\widehat{\beta}$

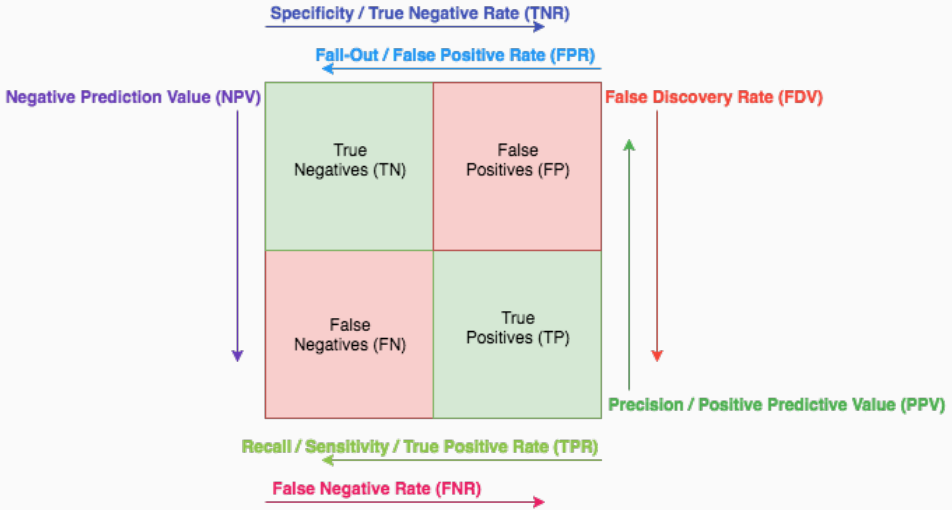
# The simple workflow

1. (Pre-process the data)
2. Split sample randomly into training set and test set
3. Train algorithm on training set
4. Evaluate on test set (= "generalization")
5. Tweak hyper-parameters, repeat 2 and 3
6. Predict labels of new data

# Most common evaluation metrics

- **Prediction accuracy metrics** (for regressions)
  - Mean absolute error
  - Root mean square error
  - $R^2$ (the default for regressors)
- **Decision support metrics** (for classifications)
  - Accuracy score (the default for classifiers)
  - Precision & Recall
  - F1 score
  - Area-under-the-curve
- Rank-aware evaluation metrics
  - Mean Reciprocal Rank
  - (Mean )Average Precision
  - Recall@k

# Confusion matrix




from: Sanyam Kapoor (2017): "Visualizing the Confusion Matrix"

# Precision and Recall

- Precision
  - What proportion of positive *identifications* was actually correct?
  - $\frac{TP}{FP+TP}$

# Precision and Recall

- Precision
  - What proportion of positive *identifications* was actually correct?
  - $\frac{TP}{FP+TP}$
- Recall
  - What proportion of *actual positives* was identified correctly?
  - $\frac{TP}{TP+FN}$

# Precision and Recall

- Precision
  - What proportion of positive *identifications* was actually correct?
  - $\frac{TP}{FP+TP}$
- Recall
  - What proportion of *actual positives* was identified correctly?
  - $\frac{TP}{TP+FN}$

Q: What happens with precision and recall when you predict all observations to be positive?

# Precision and Recall

- Precision
  - What proportion of positive *identifications* was actually correct?
  - $\frac{TP}{FP+TP}$
- Recall
  - What proportion of *actual positives* was identified correctly?
  - $\frac{TP}{TP+FN}$

Q: What happens with precision and recall when you predict all observations to be positive?

Possible solution: F1 score

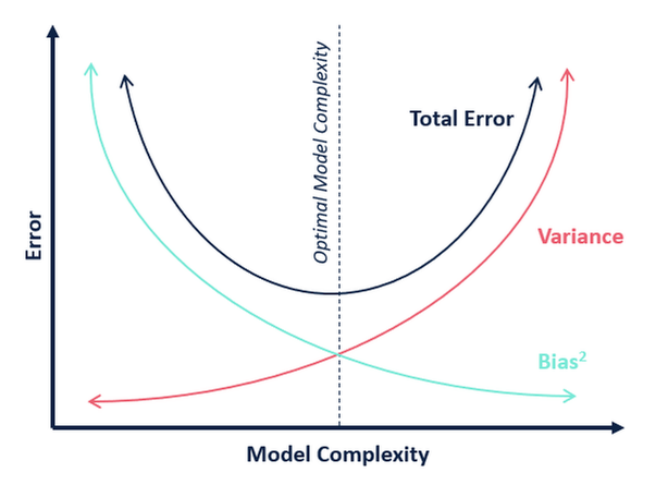
- Harmonic mean of precision and recall: $2 \times \frac{precision \times recall}{precision+recall}$
- See sklearn documentation

# Variance-Bias-Trade-Off

- Both Variance and Bias of an estimator are desired to be low
- OLS is unbiased but has huge variance, specifically when
  - ... features are highly correlated with each other
  - ... there are many predictors

→ Regularization: Reduce *variance* at the cost of introducing some *bias*, which improves prediction!

# Variance-Bias-Trade-Off: Graphical Intuition




from: AI Pool (2019): Bias-Variance Tradeoff in Machine Learning

# Pure regularizations

$\ell_1$ Ridge: stabilizes variance (multicollinearity!) and avoids extreme estimates

$$\ell_1(\widehat{\beta}) = \sum_{i=1}^{N}(y_i - x'\widehat{\beta})^2 + \alpha \sum_{j=1}^{m} \widehat{\beta}_j^2$$

$\ell_2$ Lasso: selects certain features (so-called sparse solutions)

$$\ell_2(\widehat{\beta}) = \sum_{i=1}^{N}(y_i - x'\widehat{\beta})^2 + \alpha \sum_{j=1}^{m} |\widehat{\beta}_j|$$

$\ell_3$ Firth: corrects small-sample bias [not part of `sklearn`]

$$\ell_3(\widehat{\beta}) = \sum_{i=1}^{N}(y_i - x'\widehat{\beta})^2 + \frac{1}{2}\log\det(I(\beta))$$

# Advanced regularizations

- Elastic net (Mixture of Ridge and Lasso): produces sparse solutions and can retain (or drop) groups of correlated variables

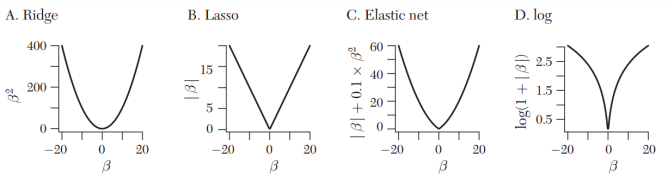



Fig. 1 of Gentzkow, Kelly and Taddy (JEL 2019): "Text as Data"

# Advanced regularizations

- Elastic net (Mixture of Ridge and Lasso): produces sparse solutions and can retain (or drop) groups of correlated variables

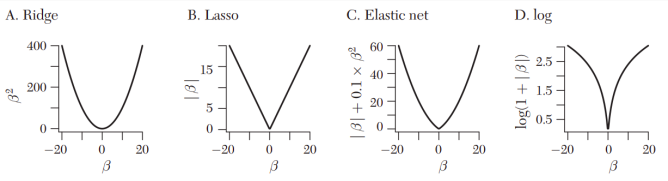



Fig. 1 of Gentzkow, Kelly and Taddy (JEL 2019): "Text as Data"

- Adaptive Lasso: selects variables consistently under weaker assumptions
- Square-root Lasso: Optimal $\alpha$ independent of the unknown error variance under homoskedasticity

# To become a Master...

- Andreas Müller and Sarah Guido: "Introduction to Machine Learning with Python", O'Reilly (2016)
- Fabio Nelli: "Python Data Analytics. Data Analysis and Science Using Pandas, matplotlib, and the Python Programming Language", Apress (2015)

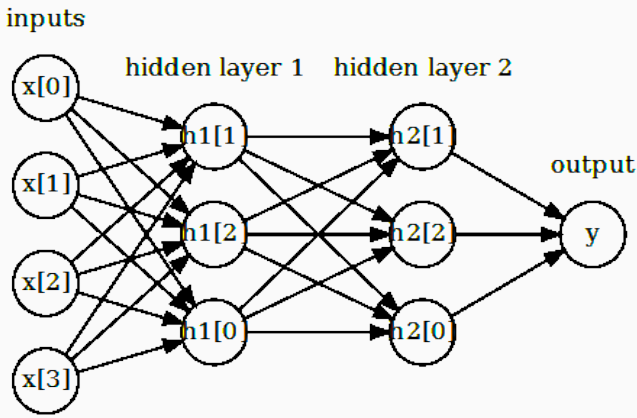
# Neural Networks

# What is a Neural Network?

YouTube Neural Networks explained in one minute

- One or more layers with nodes, links between all nodes of consecutive layers
- Linear regression with Regularization
- Activation function
- Scaled data

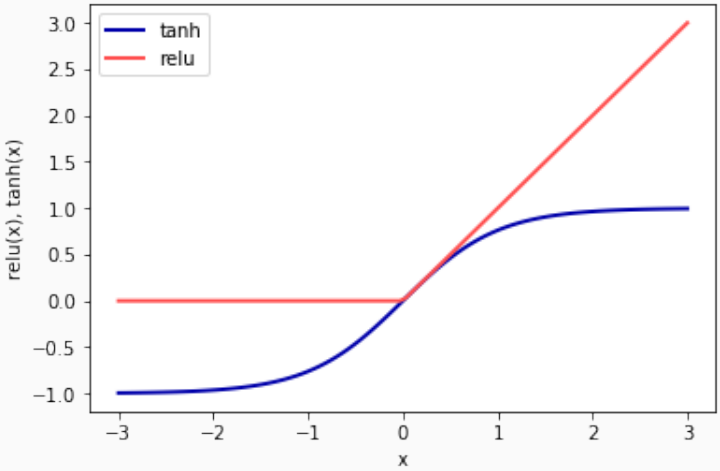
# Ingredients: The layers (and their math)




from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

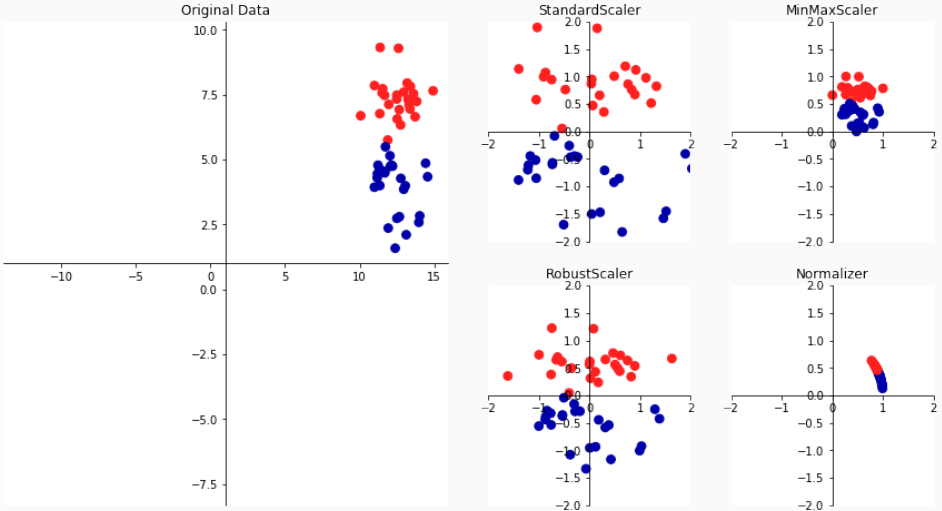
$$h1[1] = g\big(w_{1,0}x[0] + w_{1,1}x[1] + w_{1,2}x[2] + w_{1,3}x[3]\big)$$

# Ingredients: The Activation function




from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

# Ingredients: Scaling




from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

# MLP in sklearn

- Many hyper-parameters (→ Documentation)
  1. How many layers?
  2. How many units (nodes) (per layer)?
  3. Which activation function?
  4. Regularization strength?
  5. Underlying algorithm? (and their respective parameters)
  6. ...

\+ Can be infinitely complex, often beat other algorithms

\- Much slower than other algorithms

# Neural Network classes

1. Multi-layer Perceptron (MLP)
2. Convolutional Neural Networks (CNN)
3. Recurrent Neural Networks (RNN)
4. Auto encoders
5. ...

⟶ See the chart at towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

# To become a Master...

- Shai Shalev-Shwartz and Shai Ben-David: "Understanding Machine Learning: From Theory to Algorithms", Cambridge University Press (2014)

# Advanced Machine Learning workflow

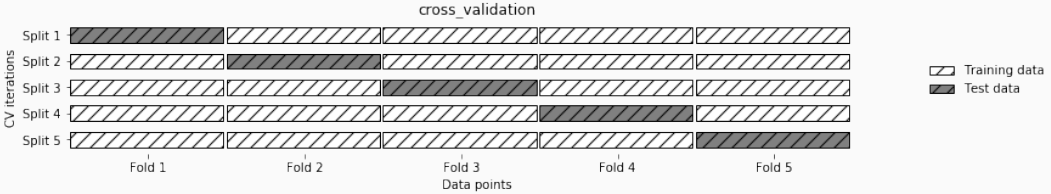
1. Cross Validation
2. Grid Search
3. Model Pipelines

# Why cross-validation?

- Learned weights likely specific to training set (even though random)
- Estimates of generalization affected by random split into training and test

➔ Solution: Repeat learning on different splits, i.e. do "Cross-validation" (CV)

# $k$-Fold Cross-validation




from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

Split sample evenly into $k$ data points, pick one as test set and the rest as training set, repeat $k$ times (data can be shuffled first)

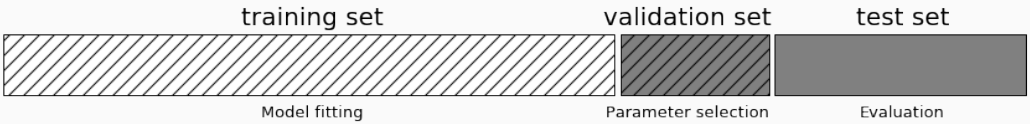
# Other Cross-validation strategies

- **Stratified k-Fold**: Split data $k$ times such that proportions between classes are similar across folds
- **Leave-one-out CV**: Set $K$ equal to the number of observations
- **Shuffle-split CV**: In each fold, split data into fixed shares for training and test set (which do not need add up to 1)

# Why Grid Search?

- Randomly or systematically loop over different combinations of parameters
- Keep the best performing parameter combination
- IMPORTANT: Don't evaluate parameters on training set, but on distinct *validation set*

# Validation set




from: Andreas Müller and Sarah Guido (2016): Introduction to Machine Learning with Python, O'Reilly

- Necessary to evaluate parameter combinations on unseen data
- ...for the same reason you do generalize on unseen data, too

# Grid Search with Cross-validation

- Two strategies:
  - Exhaustive: `GridSearchCV(estimator, param_grid)` (→ Documentation)
  - Random: `GridSearchCV(estimator, param_distributions)` (→ Documentation)
- `estimator` is model class (i.e. `MLPerceptron()`)
- `param_grid`/`param_distributions` is dict or list of dict
- Optionally specify desired evaluation score and CV strategy

# Grid Search with Cross-validation

- Two strategies:
  - Exhaustive: `GridSearchCV(estimator, param_grid)` (→ Documentation)
  - Random: `GridSearchCV(estimator, param_distributions)` (→ Documentation)
- `estimator` is model class (i.e. `MLPerceptron()`)
- `param_grid`/`param_distributions` is dict or list of dict
- Optionally specify desired evaluation score and CV strategy

How many computations do you have for a 5-fold Cross-Validation, 2 possibilities for one parameter and 3 for another parameters?

# Why Model Pipelines?

What's wrong with scaling, then folding and then selecting parameters?

# Why Model Pipelines?

What's wrong with scaling, then folding and then selecting parameters?

- The information used for scaling partly comes from the verification fold
- → Information leakage (see reproducible.cs.princeton.edu/

# Why Model Pipelines?

What's wrong with scaling, then folding and then selecting parameters?

- The information used for scaling partly comes from the verification fold
- → Information leakage (see reproducible.cs.princeton.edu/ Correct approach: Splitting/Folding before any pre-processing, i.e. in the cross-validation loop using `Pipeline()` (→ Documentation)

# Checklist

- ! Never go without cross-validation as e.g. in `GridSearchCV()`
- ! Put parameters into dictionary
- ! If you scale data, you *must* use `Pipeline()`

# To become a Master...

- Fabio Nelli: "Python Data Analytics. Data Analysis and Science Using Pandas, matplotlib, and the Python Programming Language", Apress (2015)
- Andreas Müller and Sarah Guido: "Introduction to Machine Learning with Python", O'Reilly (2016)

# Excourse:
# Machine Learning for Econometricians

# Why should Econometricians know Machine Learning?

- Prediction is part of 2SLS
- Systematic model selection
- Policy prediction

# Post-double-selection (PDS)

1. Estimate Lasso with all controls but *without* variable of interest
2. Estimate Lasso with all controls and variable of interest
3. Repeat using $K$-fold Cross-Validation to find optimal $\alpha$ (in Econ usually $\lambda$)
4. Use union of non-zero controls under optimal $\alpha$ in OLS

# Post-double-selection (PDS)

1. Estimate Lasso with all controls but *without* variable of interest
2. Estimate Lasso with all controls and variable of interest
3. Repeat using $K$-fold Cross-Validation to find optimal $\alpha$ (in Econ usually $\lambda$)
4. Use union of non-zero controls under optimal $\alpha$ in OLS

Stata `dsregress`, `poregress`, `xporegess` and `dslogit`, `pologit`, `xpologit` (see stata.com/features/overview/lasso-inferential-methods/)

R Use package `hdm`; see r-bloggers.com/2017/08/the-package-hdm-for-double-selection-inference-with-a-simple-example/

# Post-double-selection (PDS), cont.

- Belloni, Chernozhukov & Hansen (ReStud 2014): "Inference on Treatment Effects after Selection among High-Dimensional Controls"
- Urminsky, Hansen & Chernozhukov (2016): "Using Double-Lasso Regression for Principled Variable Selection"
- Angrist & Frandsen (JLE 2022): "Machine Labor"

# Post-regularization (CHS)

- Estimate Lasso with all controls but *without* variable of interest
- Estimate Lasso with all controls and variable of interest
- Repeat using $K$-fold Cross-Validation to find optimal $\alpha$ (in Econ usually $\lambda$)
- Use non-zero controls orthogonalized versions of the dependent variable in OLS

# Post-regularization (CHS)

- Estimate Lasso with all controls but *without* variable of interest
- Estimate Lasso with all controls and variable of interest
- Repeat using $K$-fold Cross-Validation to find optimal $\alpha$ (in Econ usually $\lambda$)
- Use non-zero controls orthogonalized versions of the dependent variable in OLS

Stata `pdslasso` (see statalasso.github.io/docs/pdslasso/pdslasso_demo/)

Chernozhukov, Hansen & Spindler (AER 2015): "Post-Selection and Post-Regularization Inference in Linear Models with Many Controls and Instruments"

# Causal Machine Learning

- Economists: Causal ML = ML used in econometrics
  - BUT: does not improve identification strategy
- Computer Scientists/Life Scientists: Causal ML = interventions and counterfactuals embedded in models
  - Kaddour, Lynch, Liu, Kusner & Silva (arXiv 2022): "Causal Machine Learning: A Survey and Open Problems"

# Natural Language Processing

# Examples in Economics

- Paul Tetlock (2007): "Giving Content to Investor Sentiment: The Role of Media in the Stock Market," The Journal of Finance 62(3).
- Christian Catalini, Nicola Lacetera & Alexander Oettl (2015): "The incidence and role of negative citations in science," Proceedings of the National Academy of Sciences, 112(45).
- Joshua Angrist, Pierre Azoulay, Glenn Ellison, Ryan Hill & Susan Feng Lu (2017): "Economic Research Evolves: Fields and Styles," American Economic Review, 107(5).

# Vocabulary on Vocabulary

Character

↓

Word/Term [ends on a whitespace or punctuation] → token [generated by WordPiece]

↓

Sentence [something that ends on specific interpunctuation]

↓

Paragraph [something that ends on a newline]

↓

Document

↓

Corpus

# Encoding hell

- Historically, ASCII provided space for 128 characters
  - Letter `A` on ordinal position 65 with byte `01000001`

❓ What's with `Ä`, `À`, `Ã`, `Ą`, etc.?

# Encoding hell

- Historically, ASCII provided space for 128 characters
  - Letter `A` on ordinal position 65 with byte `01000001`

❓ What's with `Ä`, `À`, `Ã`, `Ą`, etc.?

- Many languages came up with their own extensions with *conflicting* byte mappings
  - `ord("€".encode('latin1'))` (164)
  - `ord("€".encode('cp1252'))` (128)
  - `ord("€")` (8634)

# Encoding hell

- Historically, ASCII provided space for 128 characters
  - Letter `A` on ordinal position 65 with byte `01000001`

❓ What's with `Ä`, `À`, `Ã`, `Ą`, etc.?

- Many languages came up with their own extensions with *conflicting* byte mappings
  - `ord("€".encode('latin1'))` (164)
  - `ord("€".encode('cp1252'))` (128)
  - `ord("€")` (8634)

🧩 Unicode and UTF-8 solve all of this, are standard str representation in Python 3

👍 When opening files, you need to know their encoding!

# Vectorization and Encodings

NLTK

# Two approaches

1. Bag of Words, count matrix, tf-idf matrix
   - ✚ simple and fast
   - ━ stupid and wasteful
   - ➲ Baseline model useful in benchmarking

# Two approaches

1. Bag of Words, count matrix, tf-idf matrix
   - ✚ simple and fast
   - ━ stupid and wasteful
   - ➔ Baseline model useful in benchmarking
2. Transformers, esp. BERTs
   - ✚ smart and powerful
   - ━ slow and complex
   - ➔ The future

# Traditional approach ("bag of words")

1. Remove stopwords (e.g. ~~am~~, ~~you~~, ...)
2. Stem words (drinking → drink, drinks → drink)
3. Tokenize (via regular expression)
4. Remove punctuation and numbers
5. Eventually construct *n*-grams
6. Build vocabulary
7. Vectorize (words to counts)

# Excourse: Regular Expression

- Very powerful mini language
- Specify patterns to search for groups of characters
- Used in all programming languages, operating systems, search engines, etc.
- RegEx is f***ing fast

# Counting words = Vectorization

- Turning words into numbers
- Create $W \times D$ matrix $L$ for $W$ words and $D$ documents
- $L_{w,d}$ indicates how often document $d$ uses word $w$
- Optionally transform the matrix according to tfidf

# Vectorizing a document

- Document 1: burger, ketchup, beer, salad
- Document 2: kassler, sauerkraut, beer, salad

# Vectorizing a document, cont.

Obtain the count matrix:

| | | |
|---|---|---|
| beer | 1 | 1 |
| burger | 1 | 0 |
| kassler | 0 | 1 |
| ketchup | 1 | 0 |
| salad | 1 | 1 |
| sauerkraut | 0 | 1 |

# tfidf-transformation

tf: term frequency

idf: inverse document frequency

$$\text{tfidf}(w,d) = \underbrace{f_{w,d}}_{\text{tf}} \times \underbrace{\log\left(\frac{D+1}{D_w+1}+1\right)}_{\text{idf}}$$

- $f_{w,d}$: Count of word $w$ in $d$
- $D$: number of documents
- $D_w$: number of documents using $w$

# Vectorizing a document, cont.

Apply tfidf-transformation:

| | | |
|---|---|---|
| beer | 0.41 | 0.41 |
| burger | 0.58 | 0 |
| kassler | 0 | 0.58 |
| ketchup | 0.58 | 0 |
| salad | 0.41 | 0.41 |
| sauerkraut | 0 | 0.58 |

Cosine similarity is $1 - 0.664 \approx 0.336$

# Use count or tfidf matrix for:

- ... any clustering algorithm you like
- ... predicting text topics of new documents (after labelling)
- ... use term counts as features
- ...

# Why count/tfidf matrices are stupid and wasteful

- Loos all syntactic and relational information ("bag of words")
- Equal weight within sentence
- No idea about synonyms
- No idea about meaning depending on context
- Each new term changes matrix

$\longrightarrow$ Language models

# Language models

- Pre-Trained models
- Predict missing word by looking at previous word(s)

```
I arrived at the bank after _____ the river
```

# Language models

- Pre-Trained models
- Predict missing word by looking at previous word(s)

```
I arrived at the bank after _____ the river
```

- Generate embedding (vector for a word)
- Allows vector algebra: `King - Man + Woman = Queen` via "Word2Vec"

# Bidirectional Encoder Representations from Transformers (BERT)

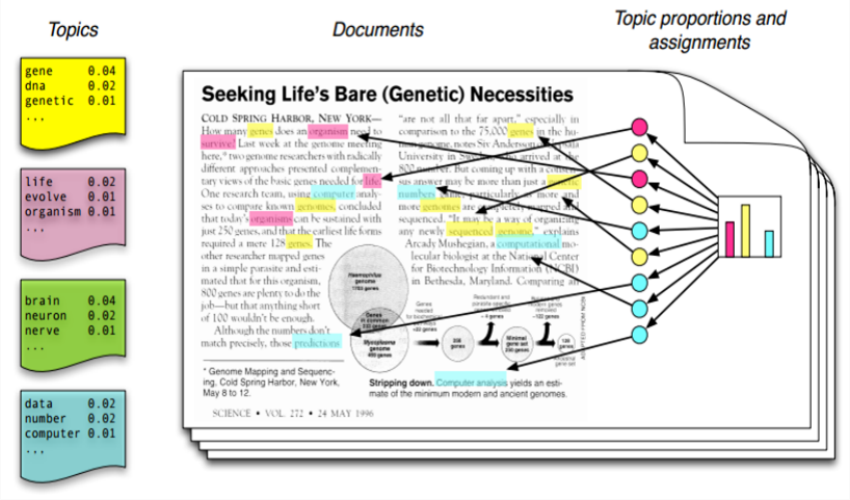
- Predict word based on sourrounding tokens; predict next sentence based on previous one
- Convert entire document to single vector
- Allows fine-tuning
- Many pre-trained models on huggingface.co/

- Useful for a wide range of tasks: natural language inference, sentiment analysis, question answering, paraphrase detection, etc.
- "BERT-Large": 24-layer, 1024-hidden-nodes, 16-attention-heads, 340M parameters; 16 TPUs for 4 days

# More vocabulary

- tokens: words or subsets of generated by WordPiece model, e.g. `eat`, `##ing`
- embeddings: map words to vectors of real numbers
- pre-trained: Usually Wikipedia or online newspaper corpora
- attention-based: pre-trained guess on which words in a sentence are relevant
- Transformer: attention mechanism that learns contextual relationships between words in a text [very complex!]
- encodings: Sequence of tokens, which are first converted into vectors and then processed in a neural network

# Exourse: Latent Dirichlet Analysis




from: Félix Revert (2018): "An overview of topics extraction in Python with LDA"

# Exourse: Latent Dirichlet Analysis, cont.

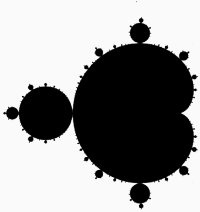
- Unstable and not replicable [due to probabilistic approach]
  - Jonas Rieger, Jörg Rahnenführer and Carsten Jentsch: Improving Latent Dirichlet Allocation: On Reliability of the Novel Method LDAPrototype
  - Ruidan He, Wee Sun Lee, Hwee Tou Ng and Daniel Dahlmeier: "An Unsupervised Neural Attention Model for Aspect Extraction"
- Biased towards successful (= large) topics
- Performs badly for short documents
  - George Ho: "Why Latent Dirichlet Allocation Sucks"
- Ignores potential model topic correlation
- Still: Parametric; ignores relative position

⟶ LDA is not usable right now, but some more development going on

# To become a Master...

- Steven Bird, Ewan Klein and Edward Loper: "Natural Language Processing with Python - Analyzing Text with the Natural Language Toolkit", O'Reilly 2009
- Andreas Müller and Sarah Guido: "Introduction to Machine Learning with Python", O'Reilly (2016)

# Useful text analysis tools



# TextBlob

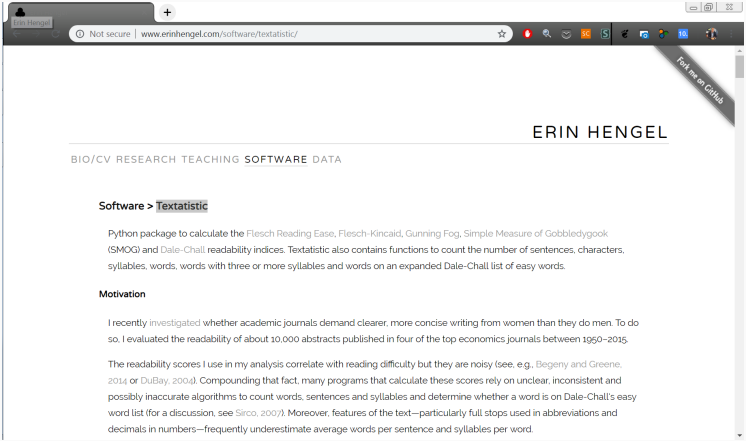

→ Documentation

# Pre-learned analysis with TextBlob

- Sentiment analysis
  - Whether a text is positive (+1), neutral (0) or negative (-1); and how subjective this classification is
  - Uses nltk's set of categorized words, and adds semantic information
- Noun phrases
  - "A small group of words standing together as a conceptual unit, typically forming a component of a clause."
  - based on speech tags
- Translation
  - via the Google Translate API

There is also a dedicated German version library called `textblob-de`

# Textatistic




→ Documentation

# Readability analysis with Textatistic

- Number of syllables, words, sentences
- Number of words in Dale-Chall list
- Multiple scores:
  - Dale-Chall score
  - Flesch Reading Ease
  - Flesch-Kincaid score
  - Gunning Fog score
  - SMOG score

Windows users need Visual Studio C++ Buildtools!

# Practical websites for researchers using NLP

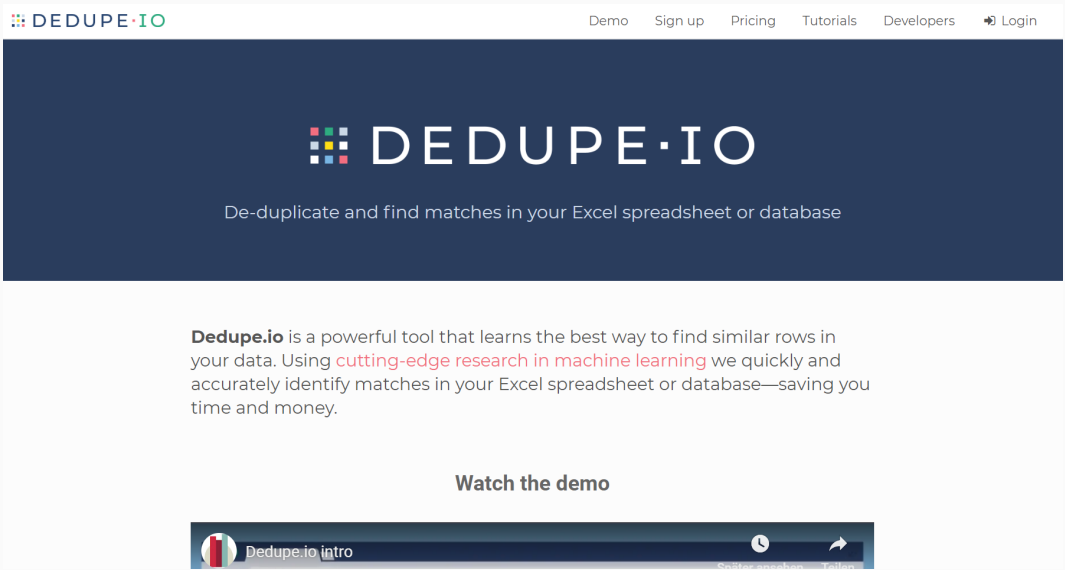
- hemingwayapp.com: Editor with embedded text readability analysis
- tldrthis.com/get-started: Get twitter-length summary of abstarct( and introduction)
- huggingface.co/spaces/ey211/PolisciTitleGenerator: Suggest title based on on abstract using a fine-tuned BERT

# Word clouds

```
import matplotlib.pyplot as plt
import nltk
from wordcloud import WordCloud

text = "..."
stops = nltk.corpus.stopwords.words('english')
wc = WordCloud(relative_scaling=1.0, stopwords=stops).generate(text)
plt.imshow(wc)  # Make plot active
plt.savefig('img/wordcloud.png')
```

# dedupe, a Python library to link records and deduplication



# How dedupe works

- Probability is weighted distance of field-entries
- Field weights are *learned* by algorithm
- Solve entries manually that are most uncertain of being duplicates (why?), then relearn weights → Active Learning

# How dedupe works

- Probability is weighted distance of field-entries
- Field weights are *learned* by algorithm
- Solve entries manually that are most uncertain of being duplicates (why?), then relearn weights → Active Learning
- Reduce number of pairs by grouping possible pairs after learning → blocking rules
- Cluster possible groups of pairs after estimating their matching probability
- Define matching threshold as F-score computed from → Precision and Recall

# Using dedupe as Programmer

- https://docs.dedupe.io/en/latest/
- `pip install dedupe`

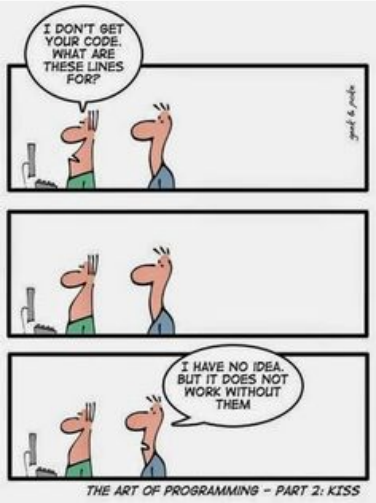
1. Instantiate with list of field definitions (dict)
2. Feed with data organized as index-oriented nested dict
3. Train
4. Match uncertain rows manually
5. Learn again
6. Merge back to data

# Script Style

# Our code is in the focus ...

- *Nature Editors* (2018) "Editorial: Does your code stand up to scrutiny?", Nature 555, 142.
- Sören Sonenburg et. al (2007): "The Need for Open Source Software in Machine Learning", Journal of Machine Learning Research 8.
- Simon Portegies Zwart (2018): "Computational astrophysics for the future", Science Perspective 361(6406).

# About readable, understandable code



# The Zen of Python

```
1 import this
```

*"Code is read much more often than it is written"* *(*Guido van Rossum*)*

# Python Style Guide (PEP8) is coding industry standard

Most important rules:

1. Use descriptive names for your objects
2. Limit each line to 79 characters
3. Indent with 4 spaces
4. Document code with complete plain English sentences, but not obvious things
5. Start module and functions with docstrings (surrounded by """)
6. Add 1 whitespace around binary operators (+, -, =)
7. Do not add whitespace around = assigning parameters
8. Avoid trailing whitespaces

# Variable naming convention

| Object | naming convention | example |
|---|---|---|
| Function | lowercase separated by underscores | `parse_patents()` |
| Variable | lowercase separated by underscores | `patent_text` |
| Class | camel case | `GridSearchCV()` |
| Method | lowercase separated by underscores | `.fit_model()` |
| Constant | uppercase | `URL, CONFIG_FILE` |
| Module | lowercase separated by underscore | `patent_parser.py` |

# Check your code automatically

- PyCharm normally inspects your code automatically (→ Documentation)
  - Problems are categorized and color-coded
  - There might be a clickable widget in the top right corner
  - Colored markers on the right hand side indicate the location and category of a problem
- Manually perform an analysis Via "Code | Inspect Code" (→ Documentation)

# Fix your code (semi-)automatically

PyCharm:

- For some problems, PyCharm offers to modify the script: missing blanks, trailing blanks, imports on top of module, etc.
- Use "Refactor | Rename" to safely rename variables/functions/module etc. safely

# Fix your code (semi-)automatically

PyCharm:

- For some problems, PyCharm offers to modify the script: missing blanks, trailing blanks, imports on top of module, etc.
- Use "Refactor | Rename" to safely rename variables/functions/module etc. safely

Git:

- Use pre-commit hooks: Little scripts that get execute immediately before `git commit` gets executed
- Most relevant ones for code style (full list at pre-commit.com/hooks.html:
  - check-docstring-first
  - end-of-file-fixer
  - trailing-whitespace
  - autopep8
  - Flake8
  - mdsub

# Relative paths

- Relative paths start from some working directory (i.e. where the script is executed)
- Absolute paths always point to the same location in a system

# Relative paths

- Relative paths start from some working directory (i.e. where the script is executed)
- Absolute paths always point to the same location in a system

```
1  import pandas as pd
2
3  FNAME = "C:\Users\rosm\Dropbox\science_project\input_file.csv"
4
5  df = pd.read_csv(FNAME)
```

- Will the code run on my coauthor's computer?
- How should I best write it (provided the script is in `C:\Users\rosm\Dropbox\science_project`)?

# The __main__ function

- dunder function to define top-level and sub-level namespace
- Relevant if you want to import from this file
- Good coding practice to always include it and have main code (but not functions) in it

# Good script layout

```python
#!/usr/bin/env python3
# Author:   Python Teacher <teacher@python.edu>
"""Teaches students to write nice scripts."""

from pathlib import Path

import pandas as pd
from numpy import nan

CONSTANT1 = Path("./some_relative_path/file.csv")


def main():
    ...


if __name__ == '__main__':
    main()
```

# Don't dump everything in one folder

```
                    ---C:/tv_and_potato/---
chips.csv       mergefiles.do          tv_potato_submission.pdf
cleandata.do    regressions_alt.do     tv_potato.tex
extract0B.xls   regressions_alt.log    tv.csv
fig1.eps        regressions.do         tvdata.dta
fig2.eps        regressions.log        rundirectory.bat
figures.do      tables.txt             export_to_csv.stc
```

# Don't dump everything in one folder

---C:/tv_and_potato/---

| | | |
|---|---|---|
| chips.csv | mergefiles.do | tv_potato_submission.pdf |
| cleandata.do | regressions_alt.do | tv_potato.tex |
| extractOB.xls | regressions_alt.log | tv.csv |
| fig1.eps | regressions.do | tvdata.dta |
| fig2.eps | regressions.log | rundirectory.bat |
| figures.do | tables.txt | export_to_csv.stc |

Separate directories by function!

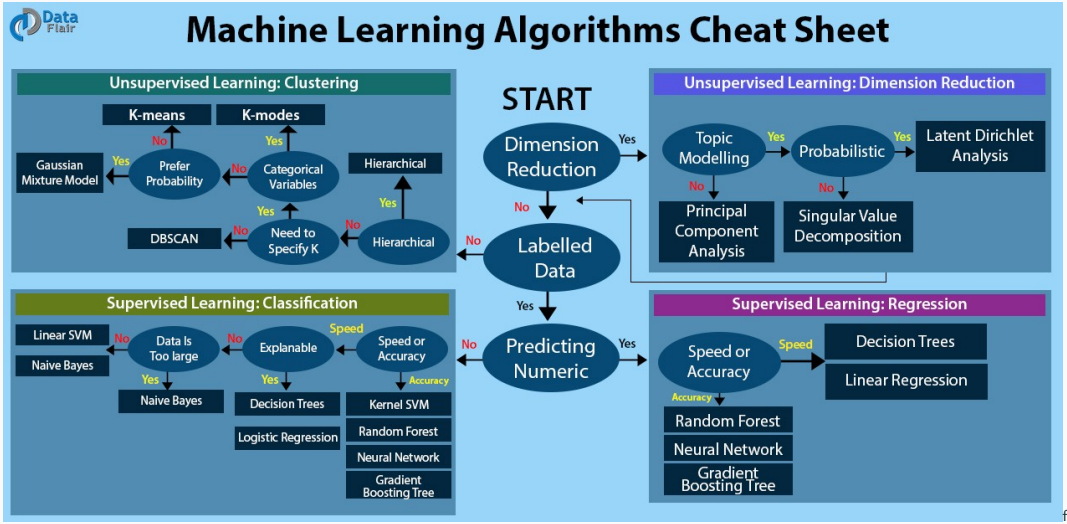
# Create a `requirements.txt`

`pipreqs . > requirements.txt` in project folder to list actually used packages

```
Genderize==0.3.1
matplotlib==3.1.2
networkx==2.6.3
numpy==1.17.4
pandas==1.4.1
pybliometrics==3.4.0
scholarmetrics==0.2.1
scikit_learn==1.1.2
scipy==1.8.0
seaborn==0.11.2
textatistic==0.0.1
tqdm==4.30.0
```

`pip install requirements.txt` will automatically install *exactly* your environment

# When to Use What?




from:

Himani Bansal (2019): "Beat The Heat with Machine Learning Cheat Sheet"

# When to Use What? (cont.)

- Distances:
  - Cosine for text
  - Haversine for geographic coordinates
- Models
  - Long short-term memory when working with text
  - Siamesic networks when looking for similarities

# Important packages for Machine Learning Pros

- More Neural Networks? → `tensor-flow` or `pytorch`
- More reinforcement learning? → `tensorforce` or `rl_coach`
- More topic modelling? → `gensim`
- More language processing? → huggingface.co