Analiza danych

prof. UAM dr hab. Tomasz Górecki

tomasz.gorecki@amu.edu.pl
http://drizzt.home.amu.edu.pl

Zakład Statystyki Matematycznej i Analizy Danych
Wydział Matematyki i Informatyki
Uniwersytet im. Adama Mickiewicza w Poznaniu

Literatura


Biecek, P. (2016). *Odkrywać! Ujawniać! Objaśniać!* Wydawnictwo UW.


Biecek, P. (2017). *Przewodnik po pakiecie R*. GiS.


Deisenroth, M.P., Faisal, A.A., Ongm C.S. (2022). *Matematyka w uczeniu maszynowym*. Helion.


Gągolewski, M. (2016). *Programowanie w języku R*. PWN.


Ghatak, A. (2019). *Deep Learning with R*. Springer.


Gillespie, C., Lovelace, R. (2018). *Wydajne programowanie w R*. O'Reilly Media.


Górecki, T. (2011). *Podstawy statystyki z przykładami w R*. BTC.


Hastie, T., Tibshirani, R., Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.

Literatura


James, G. Witten, D., Hastie, T., Tibshirani, R. (2017). *An Introduction to Statistical Learning with Applications in R*. Springer.


Krzyśko, M., Wołyński, W., Górecki, T., Skorzybut, M. (2008). *Systemy uczące się*. WNT.


Lander, J.P. (2018). *Język R dla każdego.* APN Promise.


Nwanganga, F., Chapple, M. (2022). *Praktyczne uczenie maszynowe w języku R.* APN Promise.


Muraszkiewicz, M., Nowak, R. (2022). *Sztuczna inteligencja dla inżynierów. Metody ogólne.* Oficyna Wydawnicza PW.


Muraszkiewicz, M., Nowak, R. (2023). *Sztuczna inteligencja dla inżynierów. Istotne obszary i zastosowania.* Oficyna Wydawnicza PW.


Szeliga, M. (2017). *Data science i uczenie maszynowe*. PWN.

Literatura


Wickham, H., Grolemund, G. (2018). *Język R*. Helion.


Winke, C.O. (2020). *Podstawy wizualizacji danych.* Helion.

Data Scientist

Data Scientist

Uses Statistics and Advanced Machine Learning algorithms to predict and provide key answer keys to business questions

Skills: Math, Programming, Statistics

Tech: SQL, Python, R, Cloud

Data Engineer

Develops systems for collecting, storing, and analyzing data at a large scale

Skills: Programming, BigData & Cloud

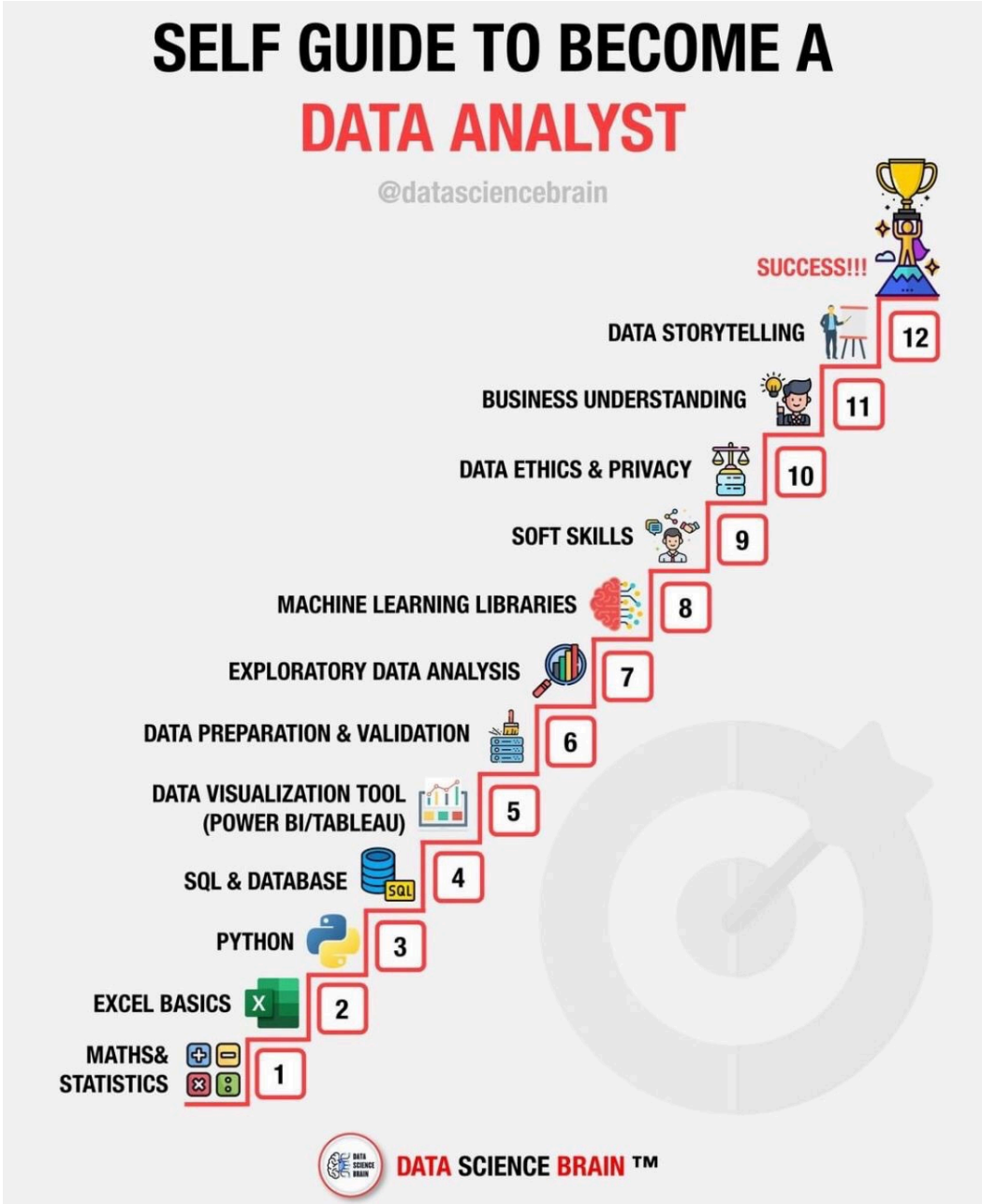
Tech: SQL, Python, Cloud, Distributed Computing

Data Analyst

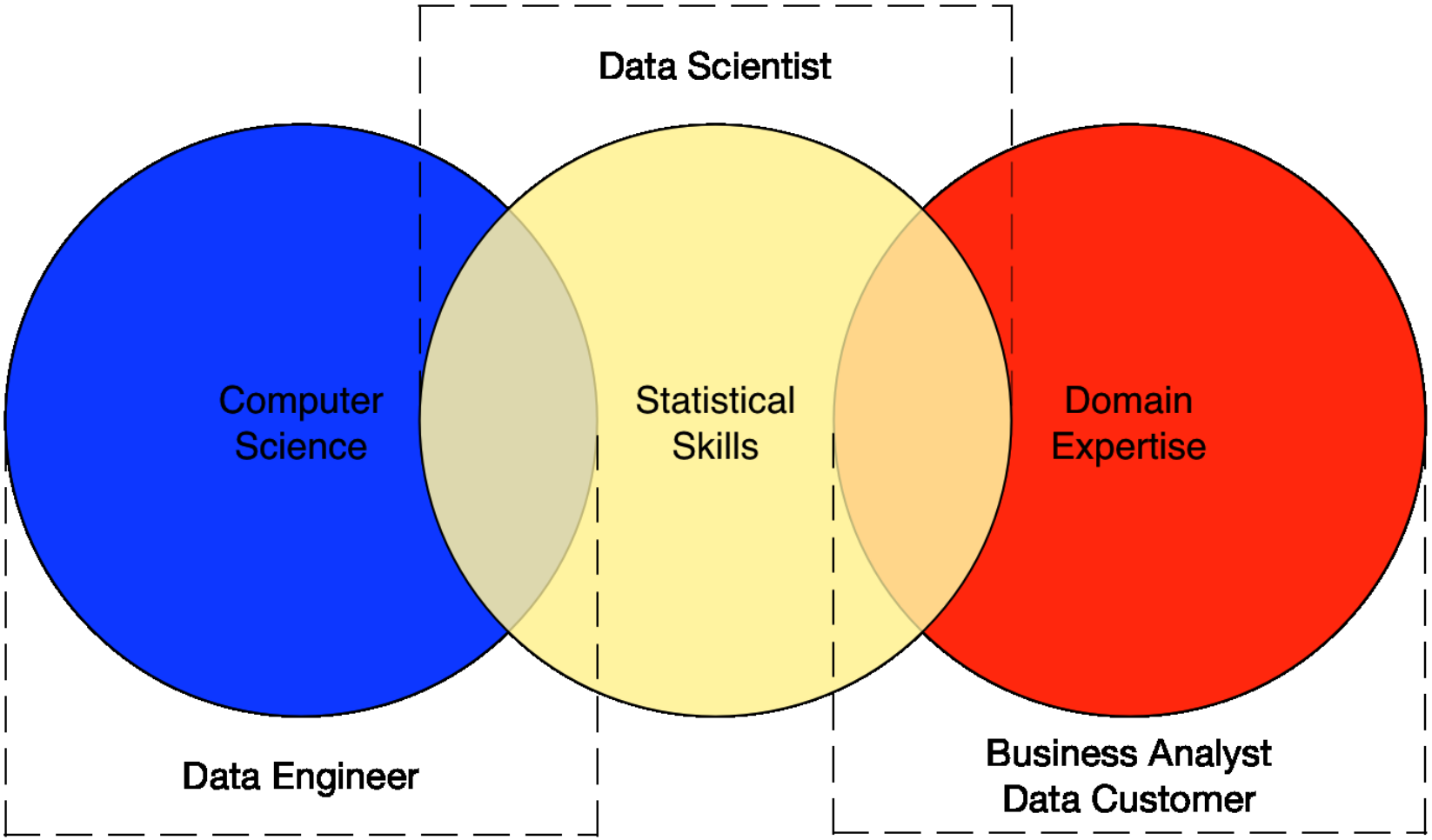
Examines information that help an organization to provide teams to develop insights and business strategies

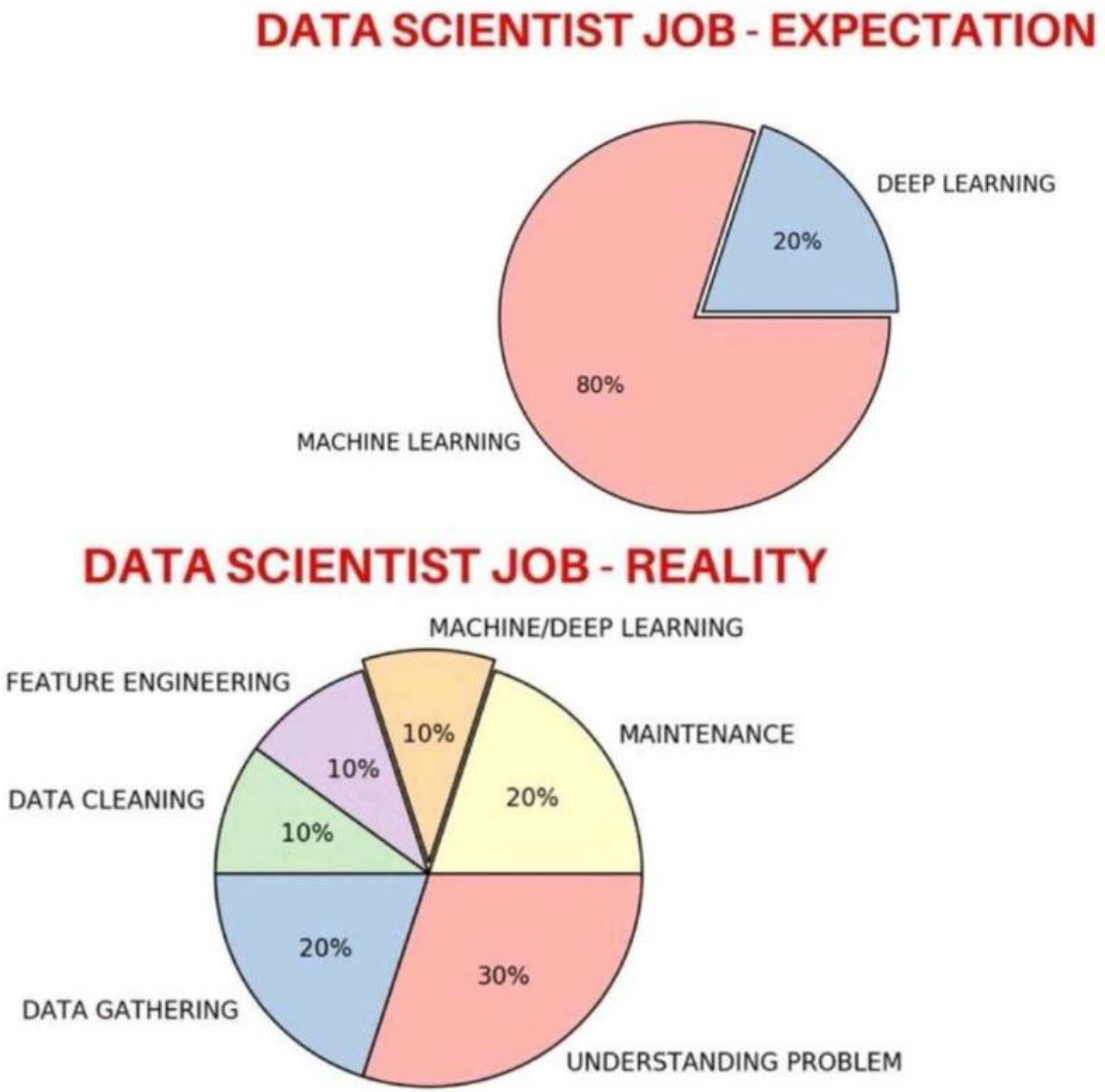
Skills: Communication, Business Knowledge

Tech: SQL, Excel, Tableau


SELF GUIDE TO BECOME A
DATA ANALYST
@datasciencebrain
SUCCESS!!!
DATA STORYTELLING
12
BUSINESS UNDERSTANDING
11
DATA ETHICS & PRIVACY
10
SOFT SKILLS
9
MACHINE LEARNING LIBRARIES
8
EXPLORATORY DATA ANALYSIS
7
DATA PREPARATION & VALIDATION
6
DATA VISUALIZATION TOOL
(POWER BI/TABLEAU)
5
SQL & DATABASE
4
PYTHON
3
EXCEL BASICS
2
MATHS&
STATISTICS
1
DATA SCIENCE BRAIN ™

Data Scientist




DATA SCIENTIST JOB - EXPECTATION
DEEP LEARNING
20%
80%
MACHINE LEARNING
DATA SCIENTIST JOB - REALITY
MACHINE/DEEP LEARNING
FEATURE ENGINEERING
10%
MAINTENANCE
10%
20%
DATA CLEANING
10%
20%
30%
DATA GATHERING
UNDERSTANDING PROBLEM

Data Mining

It is the process of automatically discovering useful information in large data repositories.

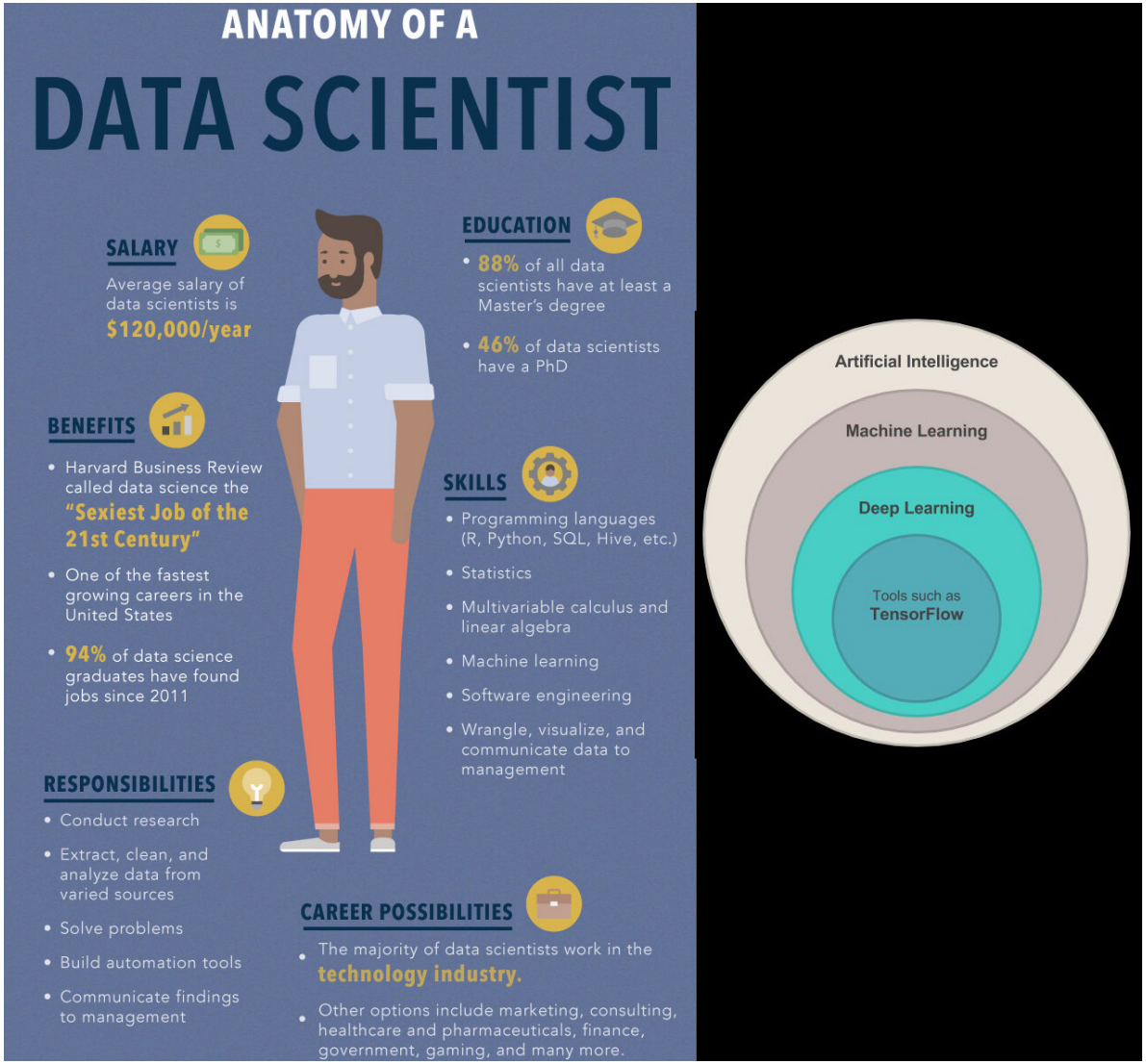
Machine Learning

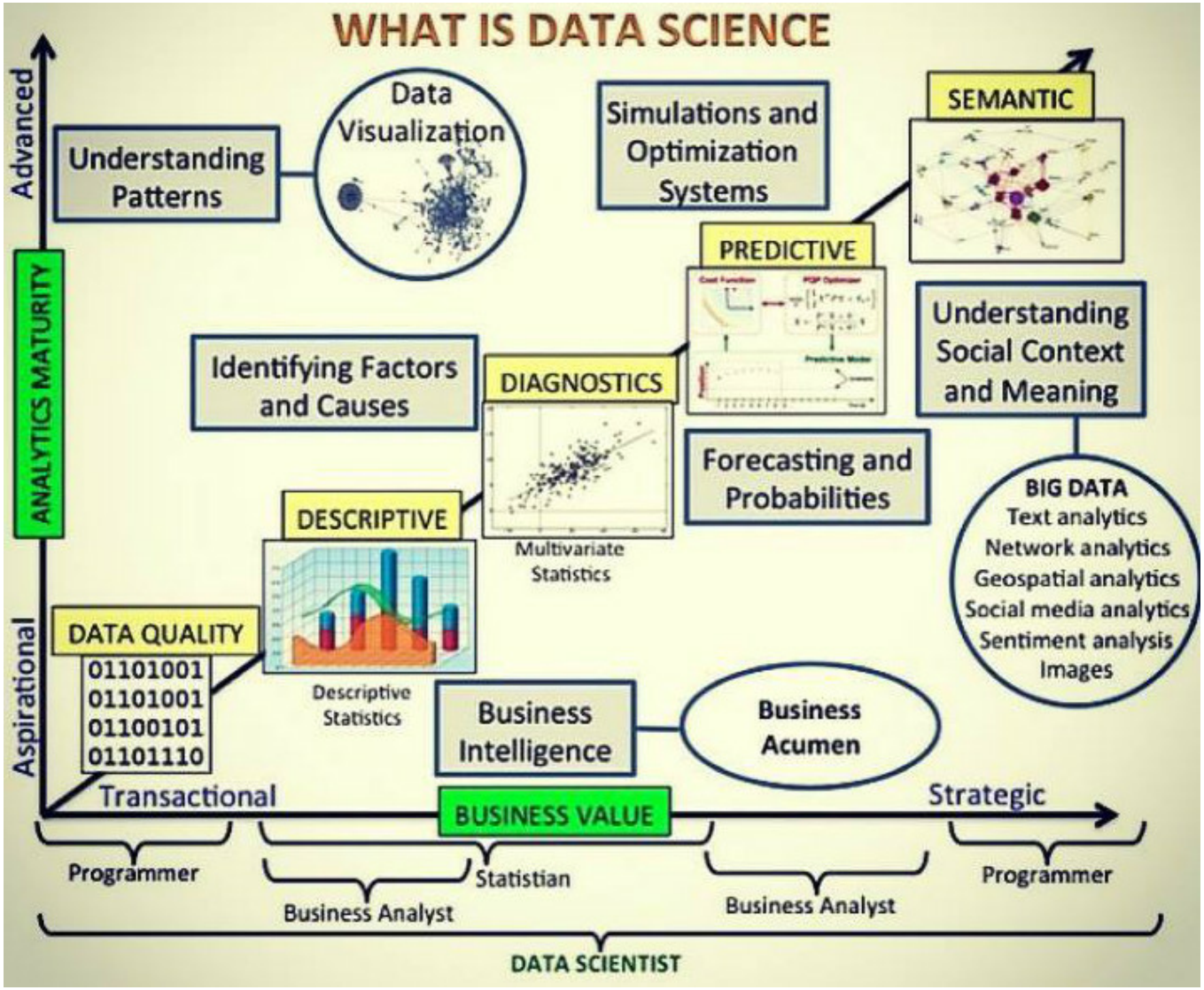
It is a set of techniques, which help in dealing with vast data in the most intelligent fashion (by developing algorithms or set of logical rules) to derive actionable insights (delivering search for users in this case)

Machine Learning and AI

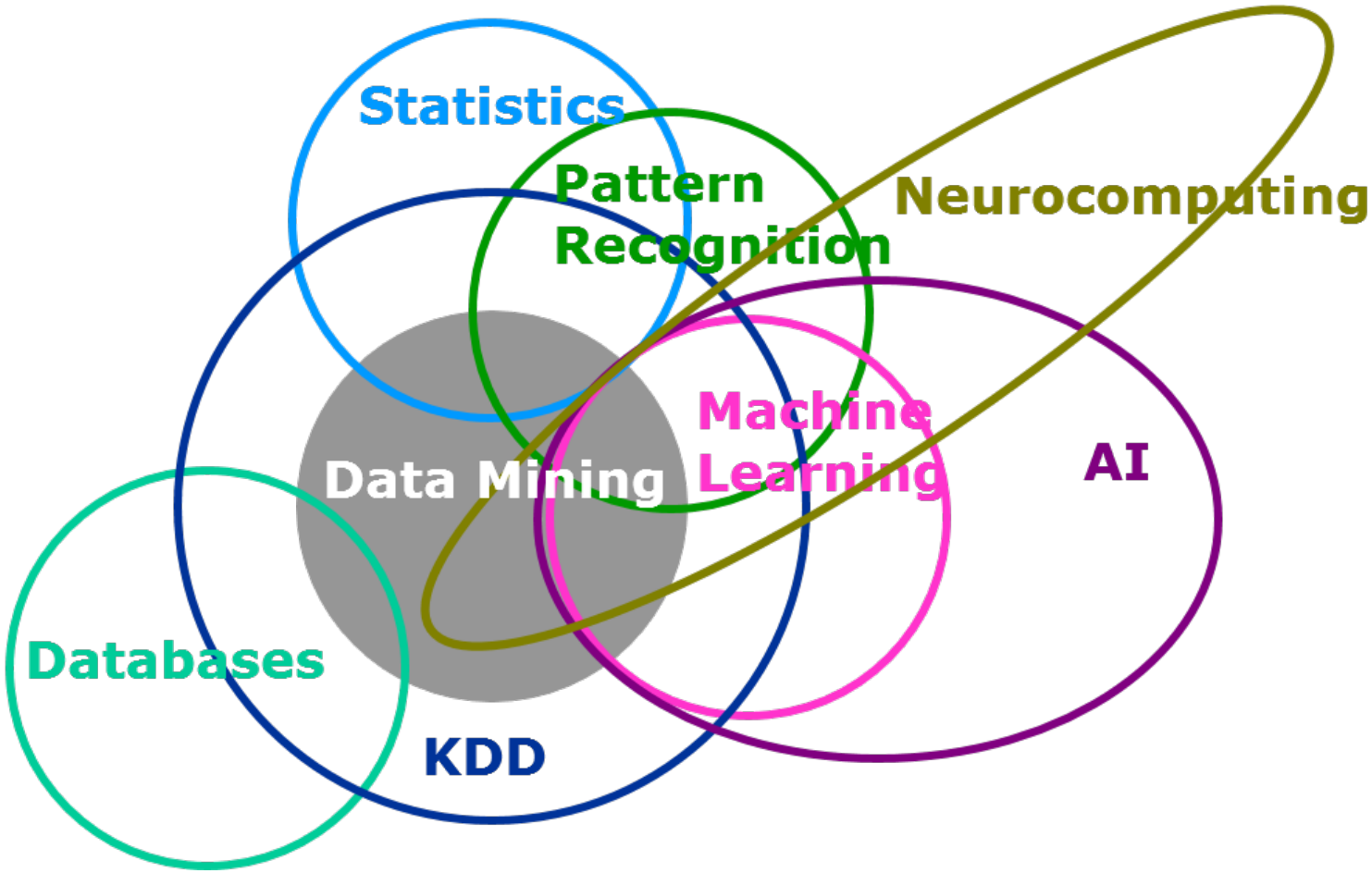
- ...what we want is a machine that can learn from experience – Alan Turing (1947).
- Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience – Tom Mitchell (1997).
- Artificial intelligence is the science and engineering of making computers behave in ways that, until recently, we thought required human intelligence – Andrew Moore (2008).

Alan Turing (1912-1954)

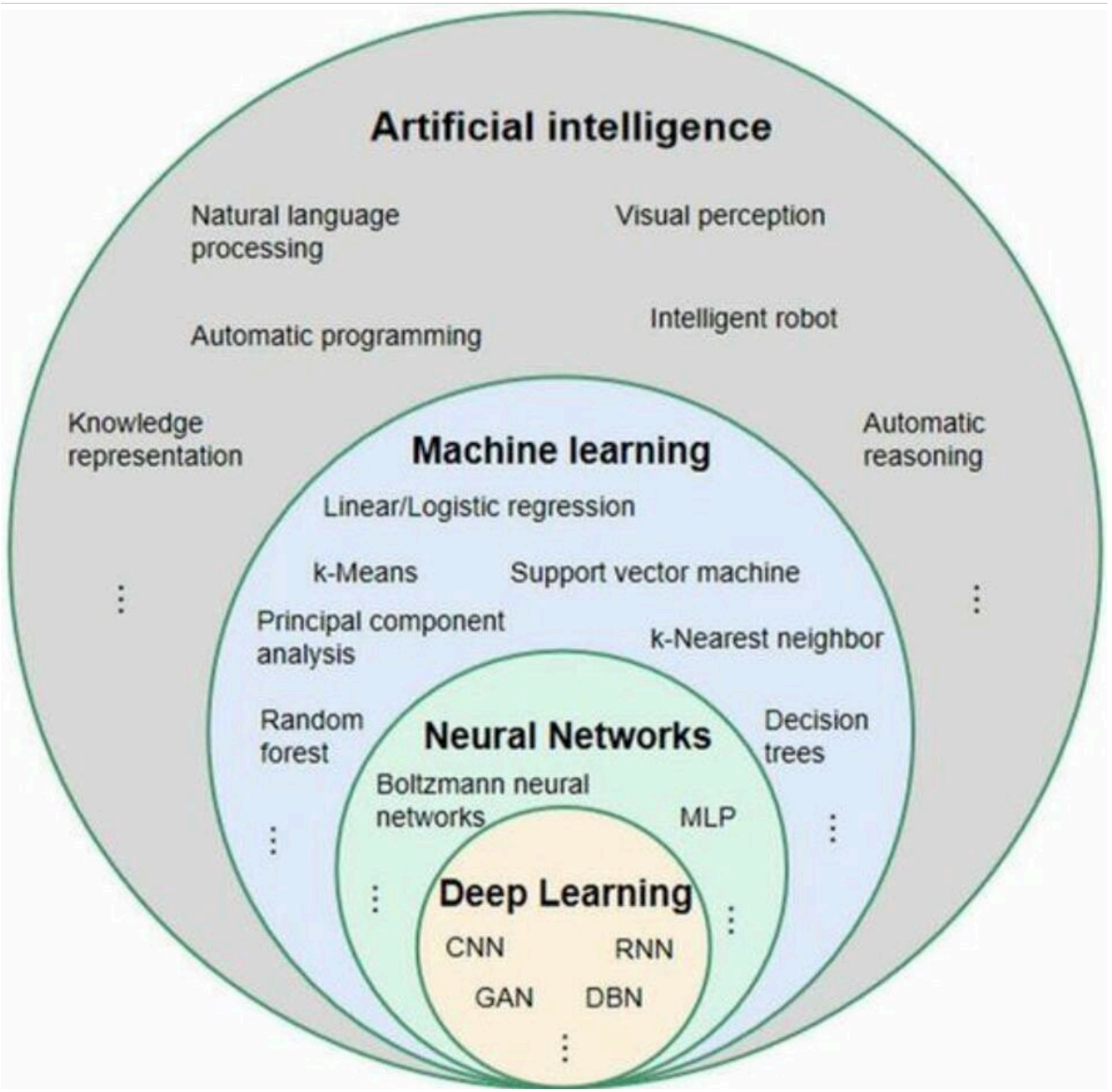

ANATOMY OF A
DATA SCIENTIST
SALARY
Average salary of data scientists is
$120,000/year
EDUCATION
88% of all data scientists have at least a Master's degree
46% of data scientists have a PhD
BENEFITS
Harvard Business Review called data science the "Sexiest Job of the 21st Century"
One of the fastest growing careers in the United States
94% of data science graduates have found jobs since 2011
SKILLS
Programming languages (R, Python, SQL, Hive, etc.)
Statistics
Multivariable calculus and linear algebra
Machine learning
Software engineering
Wrangle, visualize, and communicate data to management
RESPONSIBILITIES
Conduct research
Extract, clean, and analyze data from varied sources
Solve problems
Build automation tools
Communicate findings to management
CAREER POSSIBILITIES
The majority of data scientists work in the technology industry.
Other options include marketing, consulting, healthcare and pharmaceuticals, finance, government, gaming, and many more.
Artificial Intelligence
Machine Learning
Deep Learning
Tools such as TensorFlow

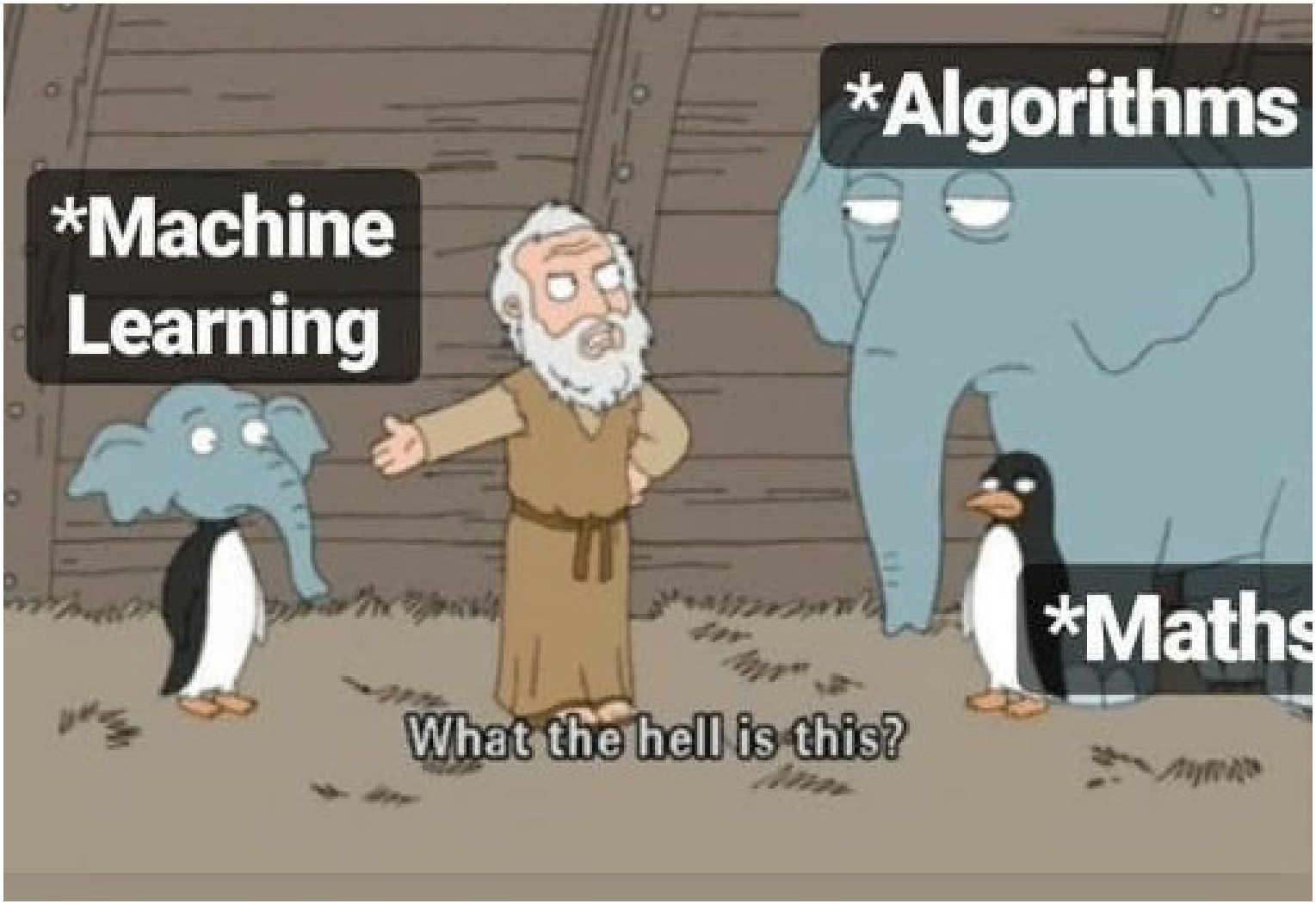

WHAT IS DATA SCIENCE
Advanced
ANALYTICS MATURITY
Aspirational
Understanding Patterns
Data Visualization
Simulations and Optimization Systems
SEMANTIC
PREDICTIVE
Understanding Social Context and Meaning
Identifying Factors and Causes
DIAGNOSTICS
Multivariate Statistics
Forecasting and Probabilities
BIG DATA
Text analytics
Network analytics
Geospatial analytics
Social media analytics
Sentiment analysis
Images
DESCRIPTIVE
Descriptive Statistics
DATA QUALITY
01101001
01101001
01100101
01101110
Business Intelligence
Business Acumen
Transactional
BUSINESS VALUE
Strategic
Programmer
Statistian
Business Analyst
Business Analyst
Programmer
DATA SCIENTIST

Data Science, Machine Learning & Artificial Intelligence



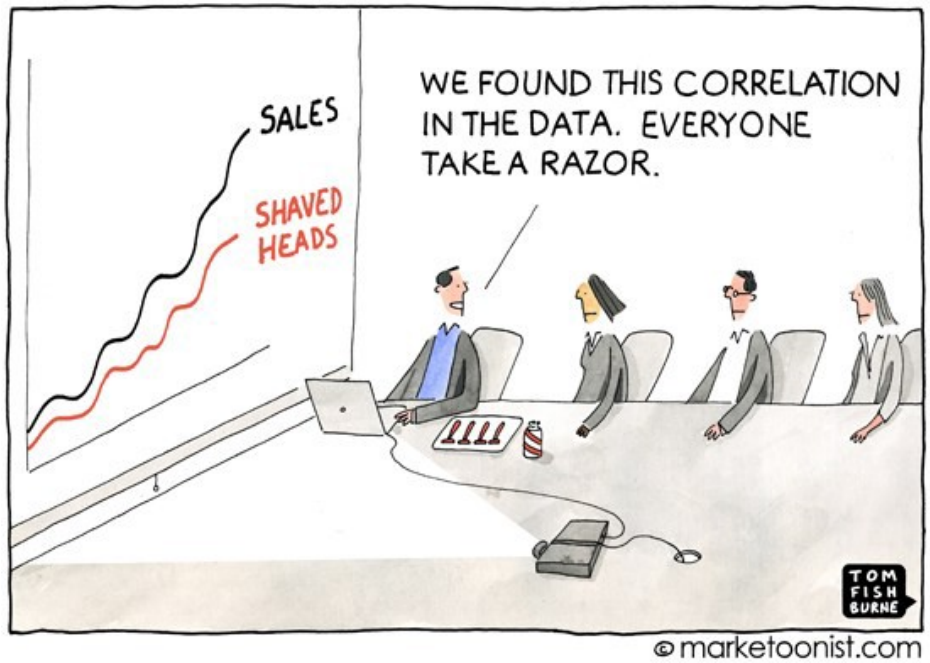
Data Science, Machine Learning & Artificial Intelligence




*Machine Learning
*Algorithms
*Maths
What the hell is this?

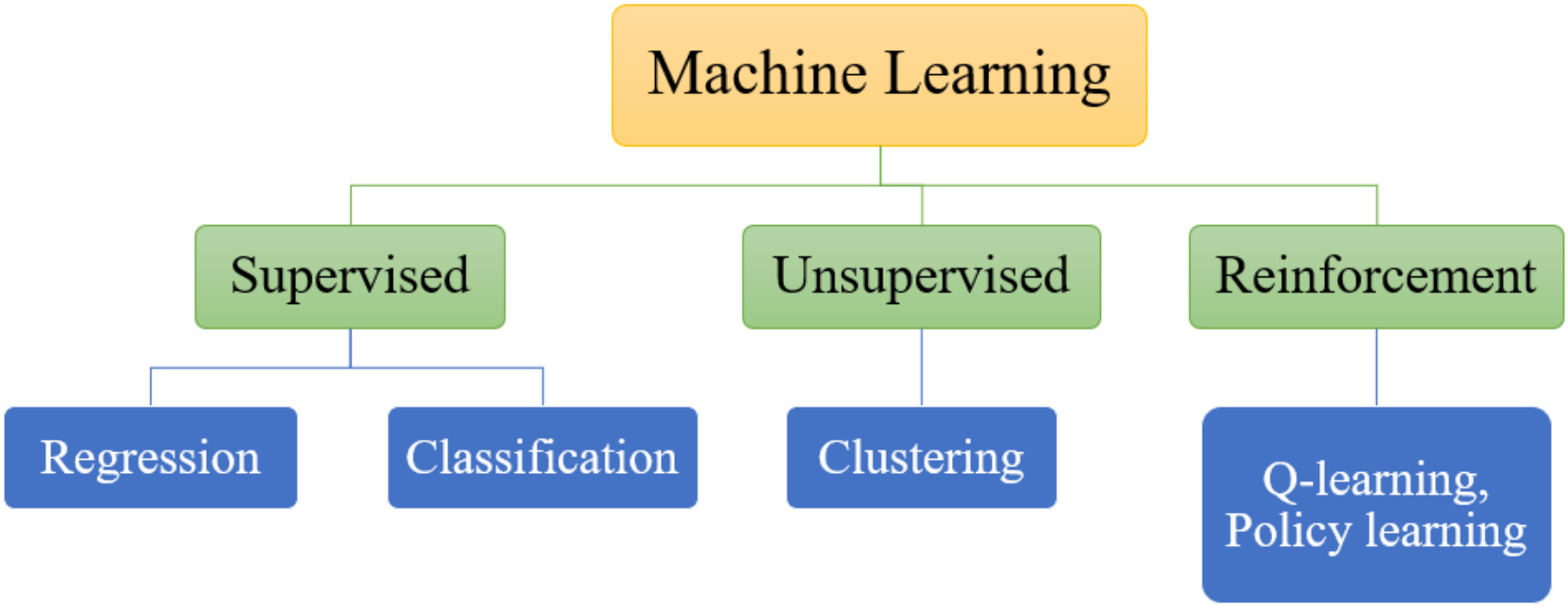
Prawo Twymana (*ang. Twyman's law*)

- Any figure that looks interesting or different is usually wrong.
- Any statistics that appears interesting is almost certainly a mistake (double check all findings).
- The more unusual or interesting the data, the more likely they are to have been the result of an error of one kind or another.

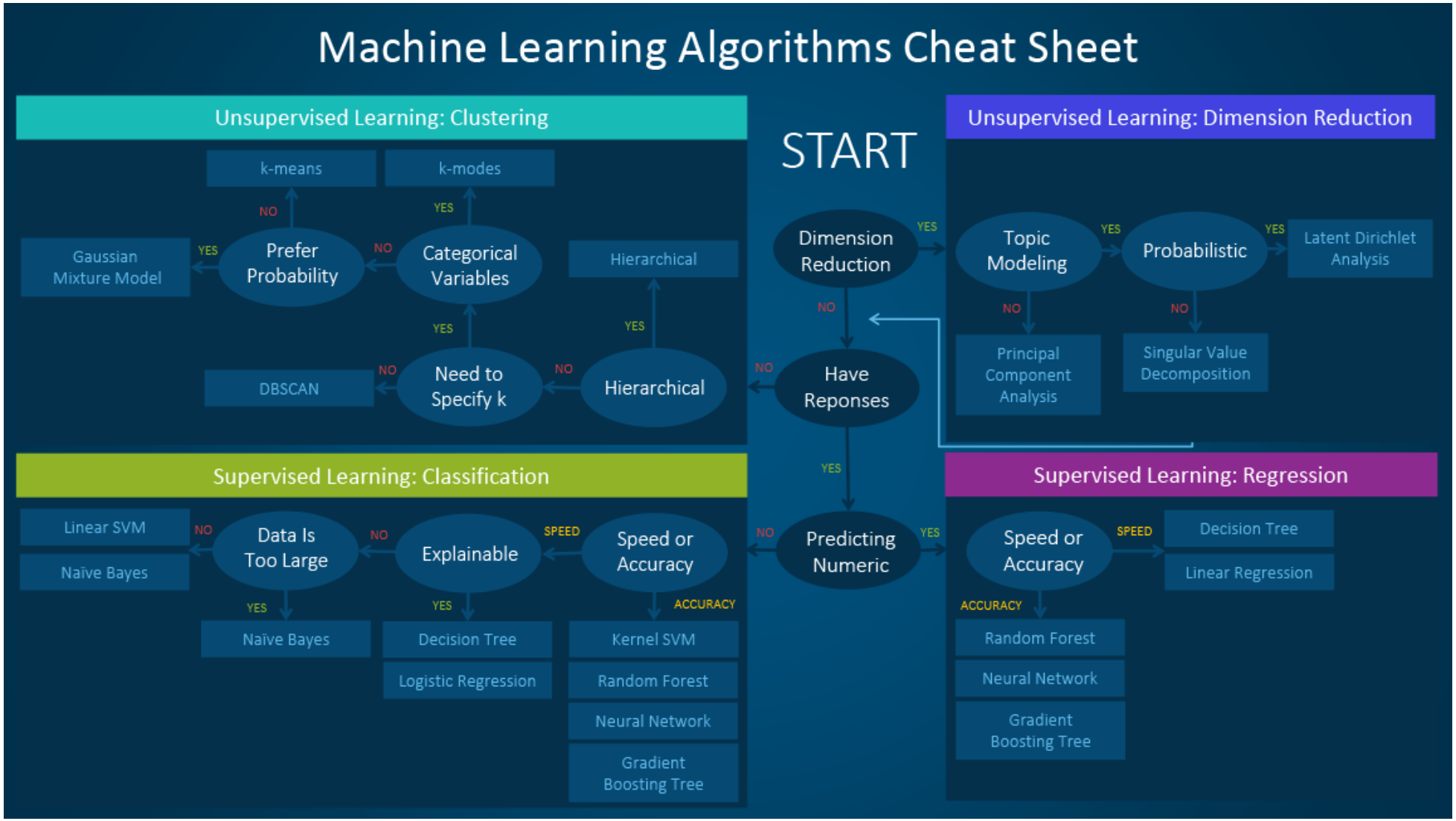


Ehrenberg, A.S.C. (1975). The Teaching of Statistics: Corrections and Comments. *Journal of the Royal Statistical Society. Series A* 138(4):543–545.

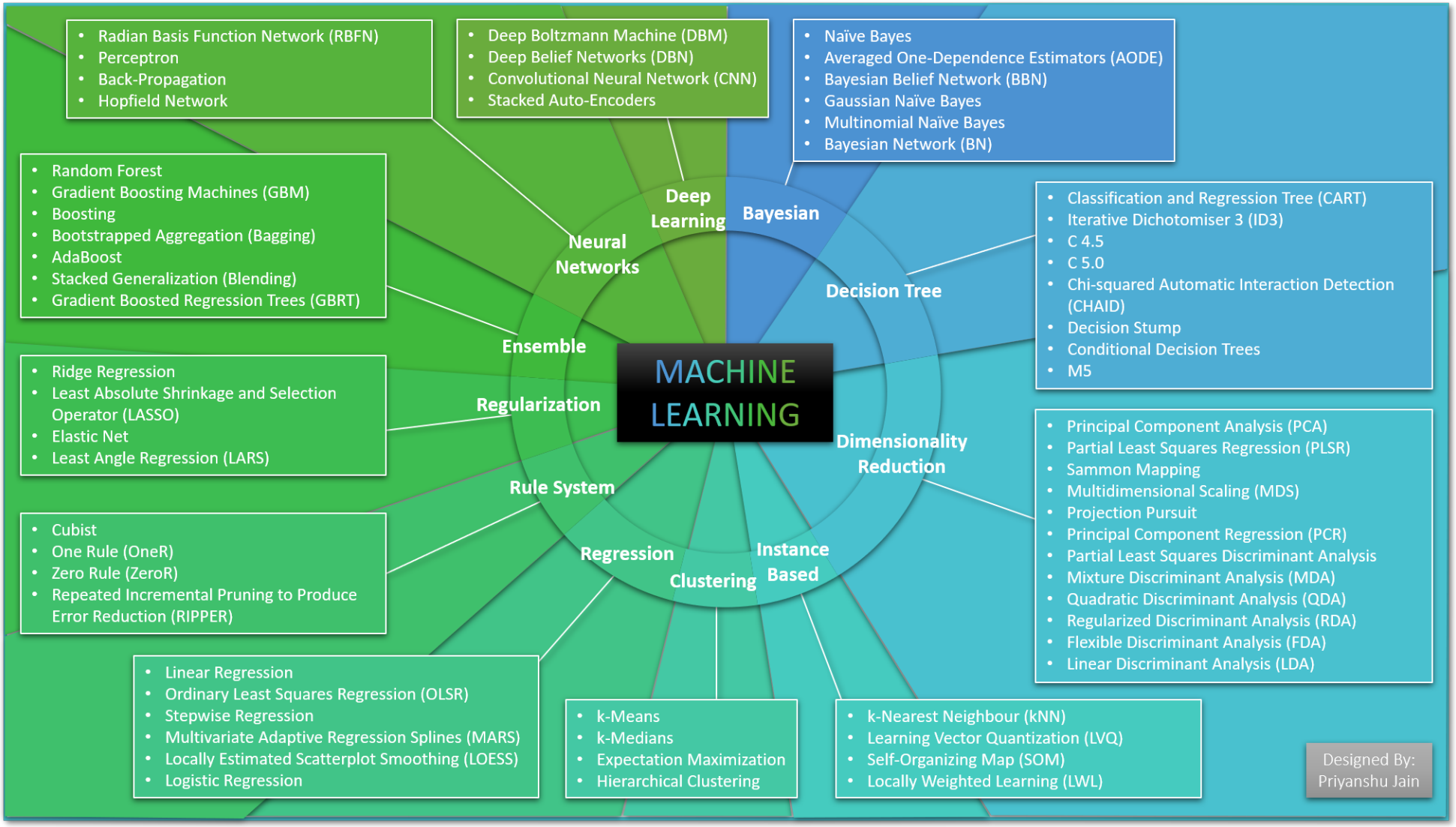
Podział algorytmów ML

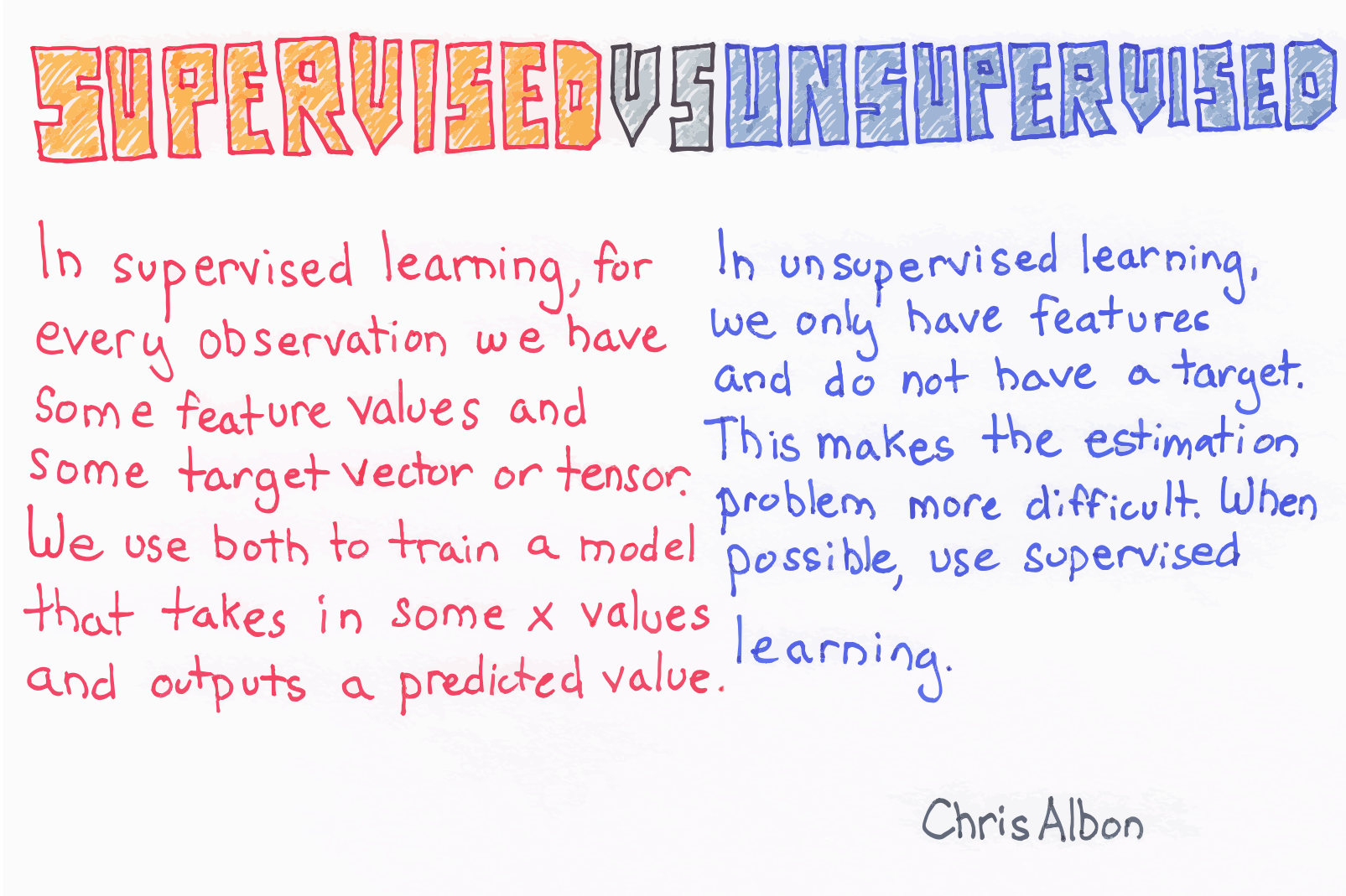


Podział algorytmów ML

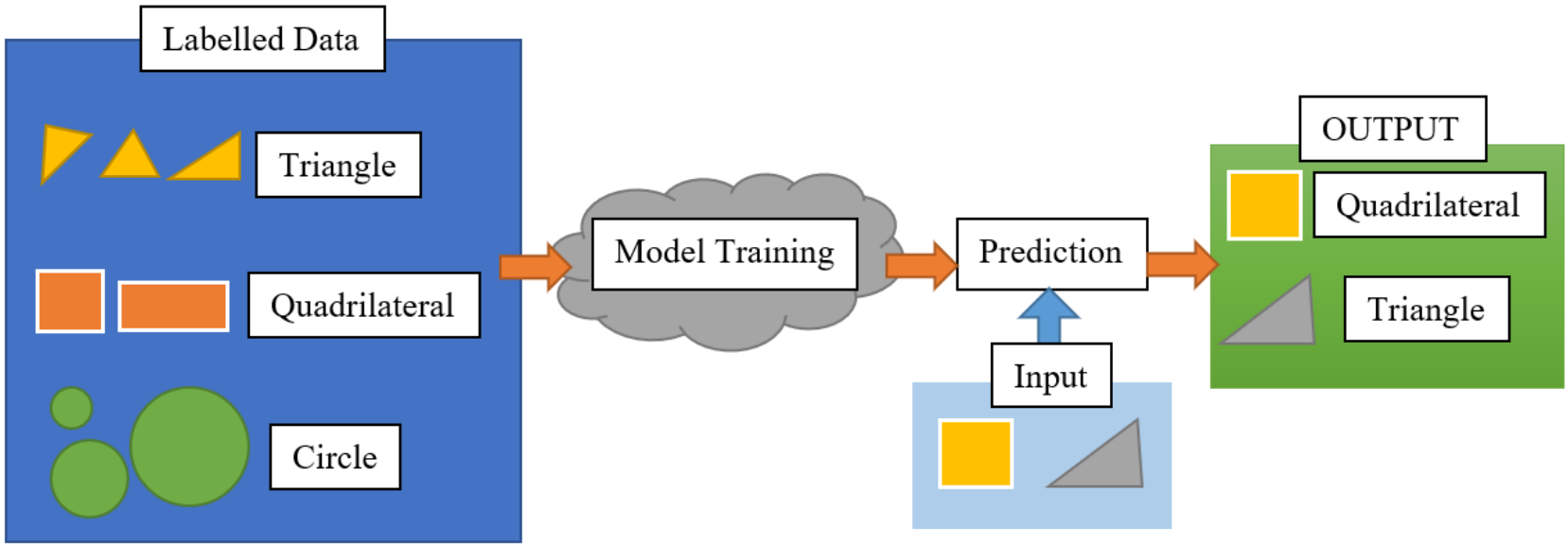


Podział algorytmów ML

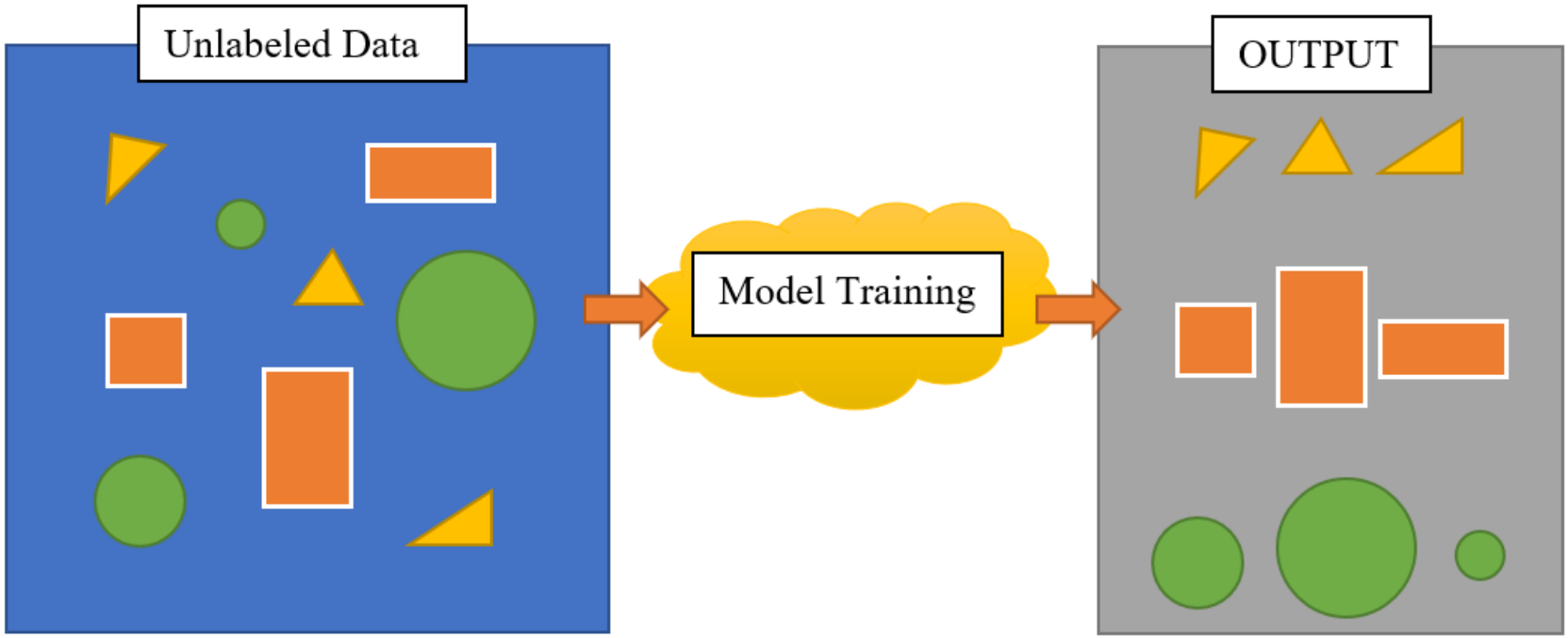



SUPERVISED VS UNSUPERVISED
In supervised learning, for every observation we have some feature values and some target vector or tensor. We use both to train a model that takes in some x values and outputs a predicted value.
In unsupervised learning, we only have features and do not have a target. This makes the estimation problem more difficult. When possible, use supervised learning.
Chris Albon

Podział algorytmów ML

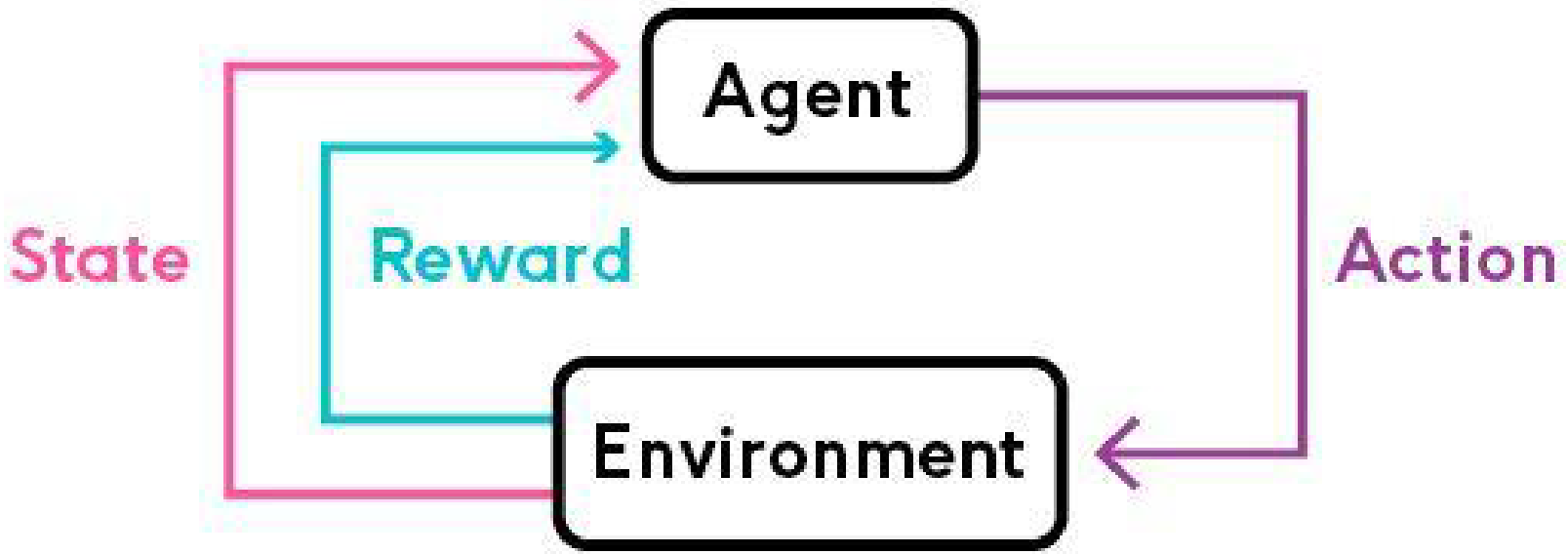


Podział algorytmów ML



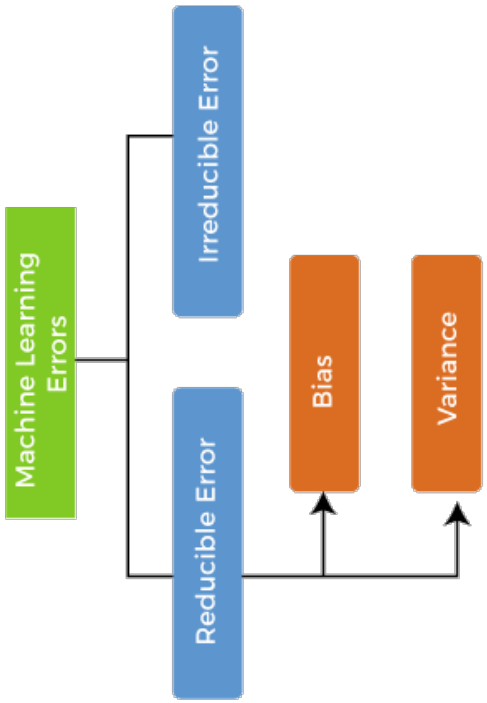
Podział algorytmów ML

Uczenie przez wzmacnianie (*ang. reinforcement learning*) – w przeciwieństwie do uczenia nadzorowanego i nienadzorowanego nie przygotowuje się zestawu danych uczących. Zamiast tego system uczy się metodą prób i błędów. Z tego powodu sekwencja właściwych decyzji spowoduje wzmocnienie procesu, ponieważ stanowi on najlepsze rozwiązanie danego problemu.



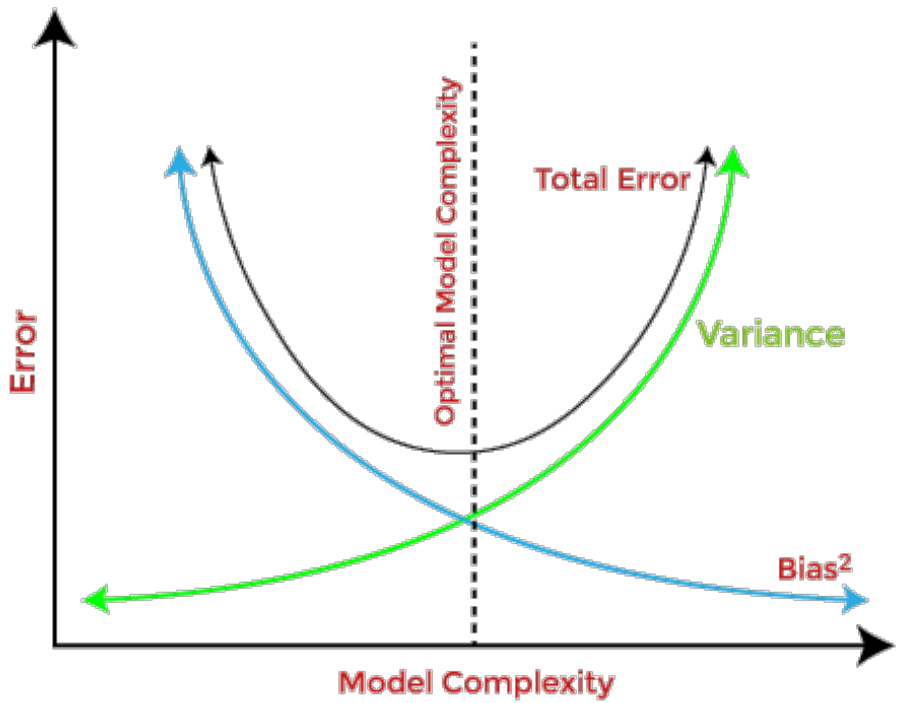
Źródła błędów w ML

1. Błąd nieredukowalny (*ang. irreducible error*) – błąd związany z danymi, który nie może być usunięty. Bywa również nazywany błędem bayesowskim (*ang. Bayes error*) – najmniejszy, możliwy do osiągnięcia poziom błędu. W wielu problemach za taki błąd możemy uznawać tzw. błąd ludzki (*ang. human-level error*), czyli taki jaki popełniają ludzie.
2. Obciążenie (*ang. bias*) – ogólne przesunięcie błędu względem najmniejszej możliwej wartości.
3. Wariancja (*ang. variance*) – zmienność wyników ze względu na nieco inne próbki.

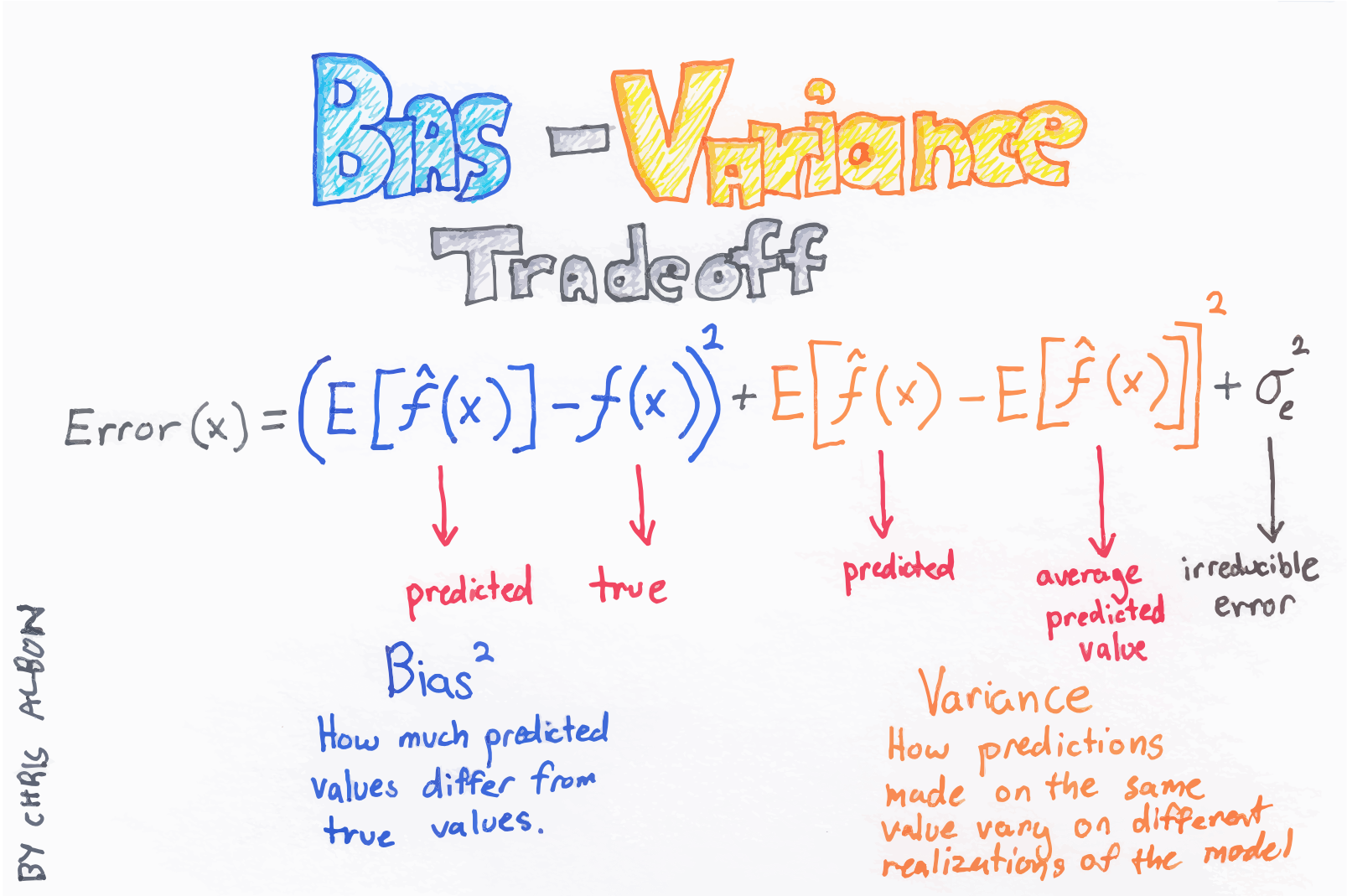


Źródła błędów w ML

Z obciążeniem mamy do czynienie w sytuacji gdy istnieje duża różnica pomiędzy błędem ludzkim (oczekiwanym), a uzyskanym przez model. Jeśli natomiast jest nieznaczna różnica pomiędzy tymi błędami, a duża różnica pomiędzy błędem na zbiorze uczącym i walidacyjnym, to mamy do czynienia z dużą wariancją.



Źródła błędów w ML



Źródła błędów w ML

Jak sobie radzić:

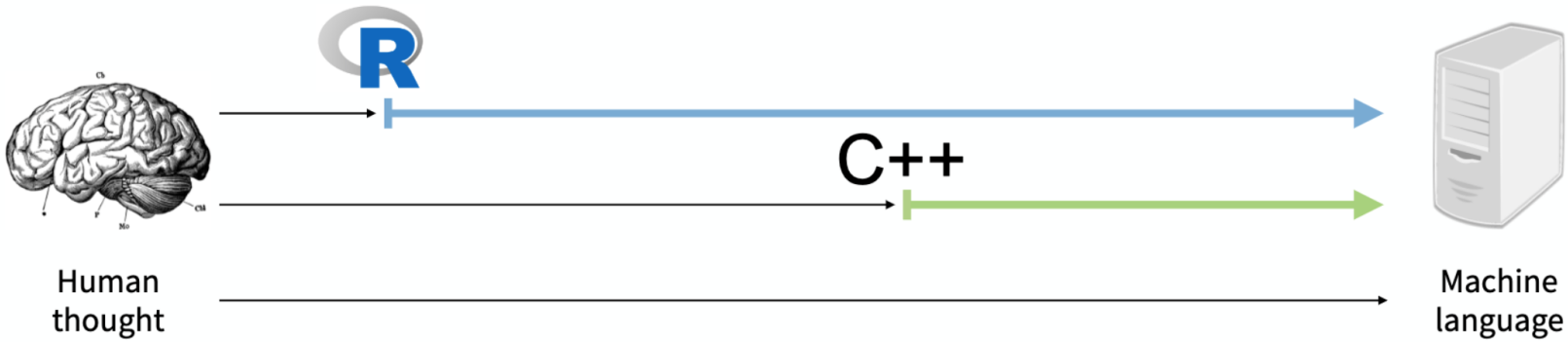
Obciążenie:

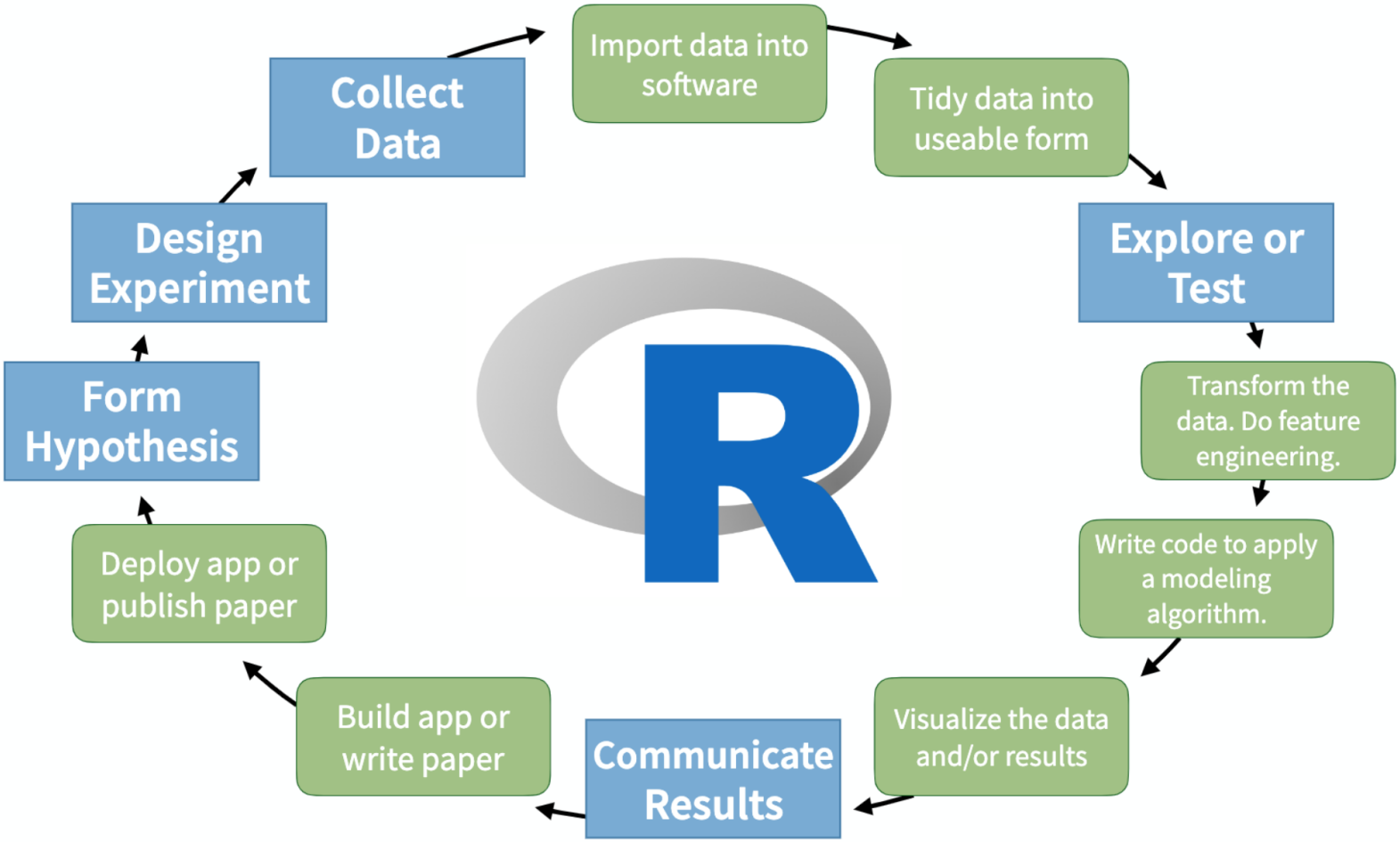
- Wyucz bardziej skomplikowany model.
- Ucz dłużej.
- Użyj lepszego algorytmu optymalizacji.
- Użyj innego modelu.
- Znajdź lepsze hiperparametry.

Wariancja:

- Użyj więcej danych.
- Użyj regularyzacji modelu.
- Zastosuj augmentację.
- Użyj innego modelu.
- Znajdź lepsze hiperparametry.

R - A computer language for scientists



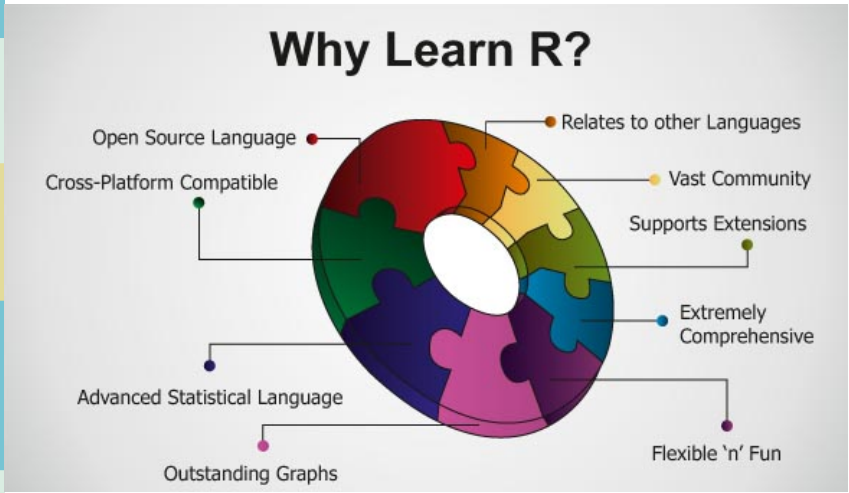

Collect Data
Import data into software
Tidy data into useable form
Explore or Test
Transform the data. Do feature engineering.
Write code to apply a modeling algorithm.
Visualize the data and/or results
Communicate Results
Build app or write paper
Deploy app or publish paper
Form Hypothesis
Design Experiment
R

WHAT IS R?

R is a scripting language for statistical data manipulation and analysis. It was inspired by, and is mostly compatible with, the statistical language S developed by AT&T. The name S, obviously standing for statistics, was an allusion to another programming language developed at AT&T with a one-letter name, C. S later was sold to a small firm, which added a GUI interface and named the result S-Plus.

WHY USE R FOR YOUR STATISTICAL WORK?

1. A public-domain implementation of the widely-regarded statistical language; R/S is the de facto standard among professional statisticians
2. comparable, and often superior, in power to commercial products in most senses
3. available for Windows, Macs, Linux
4. in addition to enabling statistical operations, it's a general programming language, so that you can automate your analyses and create new functions
5. object-oriented and functional programming structure
6. your data sets are saved between sessions, so you don't have to reload each time
7. open-software nature means it's easy to get help from the user community, and lots of new functions get contributed by users, many of which are prominent statisticians



R jest zaawansowanym pakietem statystycznym jak również językiem programowania istniejącym na platformy Windows, Linux oraz MacOS. Objęty jest licencją GNU GPL i oparty na języku S. Język R jest językiem interpretowanym, a nie kompilowanym. Z tego względu program w nim napisany nie będzie tak szybki jak np. program napisany w C++. Można jednak wykorzystać funkcje z bibliotek napisanych np. w C, C++ czy Fortran. Można również używać funkcji R w Javie, Pythonie czy C++, jak również w VB for Statistica.

Pakiet umożliwia również tworzenie zaawansowanych wykresów, które mogą zostać zapisane w formatach takich jak PDF i PNG czy JPG. O sile R stanowi ponad 20 000 bibliotek (główne repozytorium CRAN; istnieją również inne repozytoria np. Bioconductor zawierający blisko 2200 bibliotek do analizy danych genomicznych), przeznaczonych do najróżniejszych zastosowań. Całość wzbogacona jest kompleksową dokumentacją. Dodatkowo bardzo użyteczną cechą R jest dostępność zbiorów danych praktycznie dla każdego zagadnienia.

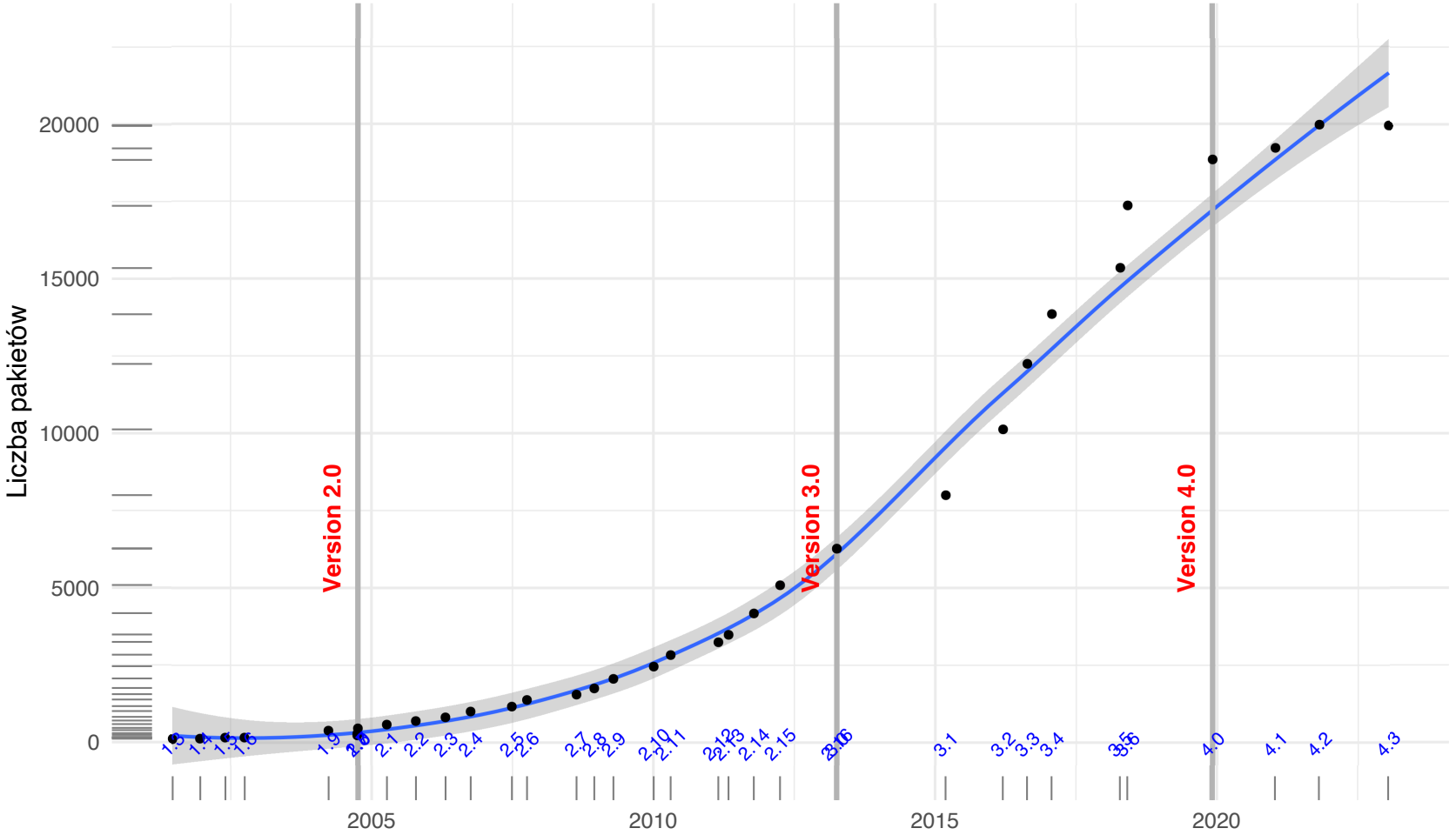
Wczesna historia R:


R. Gentleman (1959-) & R. Ihaka (1954-)

1994: Ross Ihaka i Robert Gentleman tworzą pierwszą wersję R (Robert and Ross) na Uniwersytecie w Auckland.

1997: Kurt Hornik i Fritz Leisch tworzą CRAN na Uniwersytecie Technicznym w Wiedniu.

29 lutego 2000: wersja 1.0.


Liczba pakietów
20000
15000
10000
5000
0
Version 2.0
Version 3.0
Version 4.0
1.5
1.6
1.7
1.8
1.9
2.0
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2.9
2.10
2.11
2.12
2.13
2.14
2.15
3.0
3.1
3.2
3.3
3.4
3.5
3.6
4.0
4.1
4.2
4.3
2005
2010
2015
2020

They use(d) R

European Environment Agency

http://www.revolutionanalytics.com/companies-using-r
http://www.rstudio.com/


FIRST TIME USING R
AFTER YEARS OF
EXPERIENCE WITH R
imgflip.com

RStudio jest zintegrowanym środowiskiem programowania (IDE) dla R. Dostępne jest w dwóch edycjach: RStudio Desktop oraz RStudio Server. Dostępne jest dla Windows, macOS oraz Linuxa. Występuje wersja darmowa (open source) oraz komercyjna. Zostało napisane głównie w C++. Pierwsza wersja beta powstała lutym 2011 roku. Wersja 1.0 została wydana 1 listopada 2016 roku.

BuzzR

RStudio anatomy

https://buzzrbeeline.blog/

Emma Rand

Script file

Write code here

To run code put your cursor on the line and click the run button

Edit to correct errors

⇒ record of commands that worked

Save scripts with the **.R** extension

⇒ syntax will be highlighted

⇒ good practice

`<-` is the assignment operator

⇒ puts what is on the right in to the object on the left

⇒ Assign results if you want to use them again

Console

When you click run, code is sent to the console and executed

`>` is the prompt

⇒ do not type it

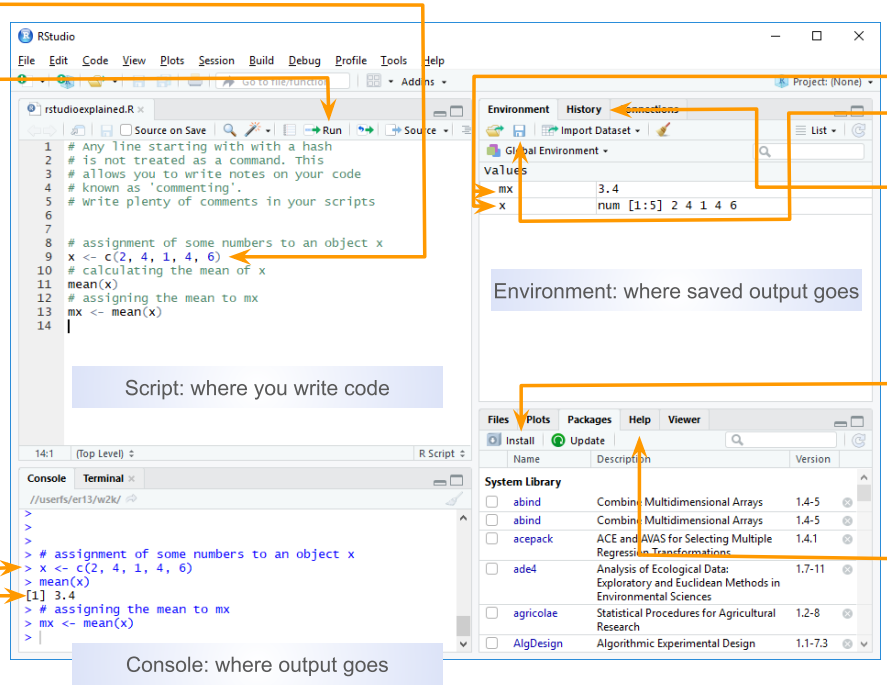
⇒ appears when R is ready for next command

Command output goes here by default

⇒ output is in a different colour

⇒ `[1]` indicates 3.4 is the first element of the output

⇒ many commands will not have output, the prompt just reappears



Environment

Name objects by assignment to use them again

All the objects you created in your session

Saving the environment saves all the objects, but not the code with a **.RData** extension

History

A history of every command you sent to the console, mistakes included.

File can be saved but usually you just need the script

Packages

Many functions come with R

A huge amount of extra functionality is available in packages

Packages can be installed by clicking the Install button

Help

Access to manual pages for all installed packages

Plots

Figure output appears here

RStudio IDE Cheat Sheet

learn more at www.rstudio.com

RStudio

The RStudio IDE is an Integrated Development Environment in R that comes in three versions

Desktop IDE
A local version of the IDE for your desktop

Open Source Server
for larger compute resources and remote access

Professional Server
for teams that share large compute resources, large data, and uniform environments for collaboration

Download all at **www.rstudio.com**. Each provides the same useful interface:

Documents and Apps

Open Shiny, R Markdown, knitr, Sweave, LaTeX, .Rd files and more in Source Pane

Check spelling | Render output | Choose output format | Choose output location | Insert code chunk

Jump to previous chunk | Jump to next chunk | Run selected lines | Publish to server | Show file outline

Access markdown guide at **Help > Markdown Quick Reference**

Jump to chunk | Set knitr chunk options | Run this and all previous code chunks | Run this code chunk

RStudio recognizes that files named **app.R**, **server.R**, **ui.R**, and **global.R** belong to a shiny app

Run app | Choose location to view app | Publish to shinyapps.io or server | Manage publish accounts

Write Code

Navigate tabs | Open in new window | Save | Find and replace | Compile as notebook | Run selected code

Cursors of shared users | Re-run previous code | Source with or without Echo | Show file outline

Multiple cursors/column selection with **Alt + mouse drag**.

Code diagnostics that appear in the margin. Hover over diagnostic symbols for details.

Syntax highlighting based on your file's extension

Tab completion to finish function names, file paths, arguments, and more.

Multi-language code snippets to quickly use common blocks of code.

Jump to function in file

Change file type

Working Directory

Press ↑ to see command history

Maximize, minimize panes

Drag pane boundaries

R Support

Import data file with wizard | History of past commands to run/add to source | Display .RPres slideshows **File > New File > R Presentation**

Load workspace | Save workspace | Delete all saved objects | Search inside environment

Choose environment to display from list of parent environments

Display objects as list or grid

Displays saved objects by type with short description

View in data viewer | View function source code

Create folder | Upload file | Delete file | Rename file | Change directory

Path to displayed directory

A File browser keyed to your working directory. Click on file or directory name to open.

RStudio Pro Features

Share Project with Collaborators | Active shared collaborators | Start **new R Session** in current project | Close R Session in project | **Select R Version**

Name of current project

Project System

File > New Project

RStudio saves the call history, workspace, and working directory associated with a project. It reloads each when you re-open a project.

RStudio opens plots in a dedicated Plots pane

Navigate recent plots | Open in window | **Export plot** | Delete plot | Delete all plots

GUI Package manager lists every installed package

Install Packages | Update Packages | Create reproducible package library for your project

Click to load package with **library()**. Unclick to detach package with **detach()**

Package version installed | Delete from library

RStudio opens documentation in a dedicated Help pane

Home page of helpful links | Search within help file | Search for help file

Viewer Pane displays HTML content, such as Shiny apps, RMarkdown reports, and interactive visualizations

Stop Shiny app | Publish to shinyapps.io, rpubs, RSConnect, … | Refresh

View(<data>) opens spreadsheet like view of data set

Filter rows by value or value range | Sort by values | Search for value

Debug Mode

Open with **debug()**, **browse()**, or a breakpoint. RStudio will open the debugger mode when it encounters a breakpoint while executing code.

Launch debugger mode from origin of error

Open traceback to examine the functions that R called before the error occurred

Click next to line number to add/remove a breakpoint.

Highlighted line shows where execution has paused

Show Traceback | Rerun with Debug

Next | Continue | Stop

Run commands in environment where execution has paused | Examine variables in executing environment | Select function in traceback to debug | Step through code one line at a time | Step into and out of functions to run | Resume execution | Quit debug mode

Version Control with Git or SVN

Turn on at **Tools > Project Options > Git/SVN**

Stage files: | Show file diff | Commit staged files | Push/Pull to remote | View History

Added | Deleted | Modified | Renamed | Untracked

Open shell to type commands

current branch

Package Writing

File > New Project > New Directory > R Package

Turn project into package, Enable roxygen documentation with **Tools > Project Options > Build Tools**

Roxygen guide at **Help > Roxygen Quick Reference**

More cheat sheets at **http://www.rstudio.com/resources/cheatsheets/**

Learn more at **support.rstudio.com** • RStudio IDE 0.99.832 • Updated: 01/16

1 LAYOUT	Windows/Linux	Mac
Move focus to Source Editor	Ctrl+1	Ctrl+1
Move focus to Console	Ctrl+2	Ctrl+2
Move focus to Help	Ctrl+3	Ctrl+3
Show History	Ctrl+4	Ctrl+4
Show Files	Ctrl+5	Ctrl+5
Show Plots	Ctrl+6	Ctrl+6
Show Packages	Ctrl+7	Ctrl+7
Show Environment	Ctrl+8	Ctrl+8
Show Git/SVN	Ctrl+9	Ctrl+9
Show Build	Ctrl+0	Ctrl+0

2 RUN CODE	Windows/Linux	Mac
Search command history	Ctrl+↑	Cmd+↑
Navigate command history	↑/↓	↑/↓
Move cursor to start of line	Home	Cmd+←
Move cursor to end of line	End	Cmd+→
Change working directory	Ctrl+Shift+H	Ctrl+Shift+H
Interrupt current command	Esc	Esc
Clear console	Ctrl+L	Ctrl+L
Quit Session (desktop only)	Ctrl+Q	Cmd+Q
Restart R Session	Ctrl+Shift+F10	Cmd+Shift+F10
Run current line/selection	Ctrl+Enter	Cmd+Enter
Run current (retain cursor)	Alt+Enter	Option+Enter
Run from current to end	Ctrl+Alt+E	Cmd+Option+E
Run the current function	Ctrl+Alt+F	Cmd+Option+F
Source a file	Ctrl+Shift+O	Cmd+Shift+O
Source the current file	Ctrl+Shift+S	Cmd+Shift+S
Source with echo	Ctrl+Shift+Enter	Cmd+Shift+Enter

3 NAVIGATE CODE	Windows /Linux	Mac
Goto File/Function	Ctrl+.	Ctrl+.
Fold Selected	Alt+L	Cmd+Option+L
Unfold Selected	Shift+Alt+L	Cmd+Shift+Option+L
Fold All	Alt+O	Cmd+Option+O
Unfold All	Shift+Alt+O	Cmd+Shift+Option+O
Go to line	Shift+Alt+G	Cmd+Shift+Option+G
Jump to	Shift+Alt+J	Cmd+Shift+Option+J
Switch to tab	Ctrl+Shift+.	Ctrl+Shift+.
Previous tab	Ctrl+F11	Ctrl+F11
Next tab	Ctrl+F12	Ctrl+F12
First tab	Ctrl+Shift+F11	Ctrl+Shift+F11
Last tab	Ctrl+Shift+F12	Ctrl+Shift+F12
Navigate back	Ctrl+F9	Cmd+F9
Navigate forward	Ctrl+F10	Cmd+F10
Jump to Brace	Ctrl+P	Ctrl+P
Select within Braces	Ctrl+Shift+Alt+E	Ctrl+Shift+Alt+E
Use Selection for Find	Ctrl+F3	Cmd+E
Find in Files	Ctrl+Shift+F	Cmd+Shift+F
Find Next	Win: F3, Linux: Ctrl+G	Cmd+G
Find Previous	W: Shift+F3, L: Ctrl+Shift	Cmd+Shift+G
Jump to Word	Ctrl+ ←/→	Option+ ←/→
Jump to Start/End	Ctrl+↑/↓	Cmd+↑/↓

4 WRITE CODE	Windows /Linux	Mac
Attempt completion	Tab or Ctrl+Space	Tab or Cmd+Space
Navigate candidates	↑/↓	↑/↓
Accept candidate	Enter, Tab, or→	Enter, Tab, or→
Dismiss candidates	Esc	Esc
Undo	Ctrl+Z	Cmd+Z
Redo	Ctrl+Shift+Z	Cmd+Shift+Z
Cut	Ctrl+X	Cmd+X
Copy	Ctrl+C	Cmd+C
Paste	Ctrl+V	Cmd+V
Select All	Ctrl+A	Cmd+A
Delete Line	Ctrl+D	Cmd+D
Select	Shift+[Arrow]	Shift+[Arrow]
Select Word	Ctrl+Shift+ ←/→	Option+Shift+ ←/→
Select to Line Start	Alt+Shift+←	Cmd+Shift+←
Select to Line End	Alt+Shift+→	Cmd+Shift+→
Select Page Up/Down	Shift+PageUp/Down	Shift+PageUp/Down
Select to Start/End	Shift+Alt+↑/↓	Cmd+Shift+↑/↓
Delete Word Left	Ctrl+Backspace	Ctrl+Opt+Backspace
Delete Word Right		Option+Delete
Delete to Line End		Ctrl+K
Delete to Line Start		Option+Backspace
Indent	Tab (at start of line)	Tab (at start of line)
Outdent	Shift+Tab	Shift+Tab
Yank line up to cursor	Ctrl+U	Ctrl+U
Yank line after cursor	Ctrl+K	Ctrl+K
Insert yanked text	Ctrl+Y	Ctrl+Y
Insert <-	Alt+-	Option+-
Insert %>%	Ctrl+Shift+M	Cmd+Shift+M
Show help for function	F1	F1
Show source code	F2	F2
New document	Ctrl+Shift+N	Cmd+Shift+N
New document (Chrome)	Ctrl+Alt+Shift+N	Cmd+Shift+Alt+N
Open document	Ctrl+O	Cmd+O
Save document	Ctrl+S	Cmd+S
Close document	Ctrl+W	Cmd+W
Close document (Chrome)	Ctrl+Alt+W	Cmd+Option+W
Close all documents	Ctrl+Shift+W	Cmd+Shift+W
Extract function	Ctrl+Alt+X	Cmd+Option+X
Extract variable	Ctrl+Alt+V	Cmd+Option+V
Reindent lines	Ctrl+I	Cmd+I
(Un)Comment lines	Ctrl+Shift+C	Cmd+Shift+C
Reflow Comment	Ctrl+Shift+/	Cmd+Shift+/
Reformat Selection	Ctrl+Shift+A	Cmd+Shift+A
Select within braces	Ctrl+Shift+E	Ctrl+Shift+E
Show Diagnostics	Ctrl+Shift+Alt+P	Cmd+Shift+Alt+P
Transpose Letters		Ctrl+T
Move Lines Up/Down	Alt+↑/↓	Option+↑/↓
Copy Lines Up/Down	Shift+Alt+↑/↓	Cmd+Option+↑/↓
Add New Cursor Above	Ctrl+Alt+Up	Ctrl+Alt+Up
Add New Cursor Below	Ctrl+Alt+Down	Ctrl+Alt+Down
Move Active Cursor Up	Ctrl+Alt+Shift+Up	Ctrl+Alt+Shift+Up
Move Active Cursor Down	Ctrl+Alt+Shift+Down	Ctrl+Alt+Shift+Down
Find and Replace	Ctrl+F	Cmd+F
Use Selection for Find	Ctrl+F3	Cmd+E
Replace and Find	Ctrl+Shift+J	Cmd+Shift+J

5 DEBUG CODE	Windows/Linux	Mac
Toggle Breakpoint	Shift+F9	Shift+F9
Execute Next Line	F10	F10
Step Into Function	Shift+F4	Shift+F4
Finish Function/Loop	Shift+F6	Shift+F6
Continue	Shift+F5	Shift+F5
Stop Debugging	Shift+F8	Shift+F8

6 VERSION CONTROL	Windows/Linux	Mac
Show diff	Ctrl+Alt+D	Ctrl+Option+D
Commit changes	Ctrl+Alt+M	Ctrl+Option+M
Scroll diff view	Ctrl+↑/↓	Ctrl+↑/↓
Stage/Unstage (Git)	Spacebar	Spacebar
Stage/Unstage and move to next	Enter	Enter

7 MAKE PACKAGES	Windows/Linux	Mac
Build and Reload	Ctrl+Shift+B	Cmd+Shift+B
Load All (devtools)	Ctrl+Shift+L	Cmd+Shift+L
Test Package (Desktop)	Ctrl+Shift+T	Cmd+Shift+T
Test Package (Web)	Ctrl+Alt+F7	Cmd+Alt+F7
Check Package	Ctrl+Shift+E	Cmd+Shift+E
Document Package	Ctrl+Shift+D	Cmd+Shift+D

8 DOCUMENTS AND APPS	Windows/Linux	Mac
Preview HTML (Markdown, etc.)	Ctrl+Shift+K	Cmd+Shift+K
Knit Document (knitr)	Ctrl+Shift+K	Cmd+Shift+K
Compile Notebook	Ctrl+Shift+K	Cmd+Shift+K
Compile PDF (TeX and Sweave)	Ctrl+Shift+K	Cmd+Shift+K
Insert chunk (Sweave and Knitr)	Ctrl+Alt+I	Cmd+Option+I
Insert code section	Ctrl+Shift+R	Cmd+Shift+R
Re-run previous region	Ctrl+Shift+P	Cmd+Shift+P
Run current document	Ctrl+Alt+R	Cmd+Option+R
Run from start to current line	Ctrl+Alt+B	Cmd+Option+B
Run the current code section	Ctrl+Alt+T	Cmd+Option+T
Run previous Sweave/Rmd code	Ctrl+Alt+P	Cmd+Option+P
Run the current chunk	Ctrl+Alt+C	Cmd+Option+C
Run the next chunk	Ctrl+Alt+N	Cmd+Option+N
Sync Editor & PDF Preview	Ctrl+F8	Cmd+F8

Previous plot	Ctrl+Alt+F11	Cmd+Option+F11
Next plot	Ctrl+Alt+F12	Cmd+Option+F12

Show Keyboard Shortcuts	Alt+Shift+K	Option+Shift+K

Why RStudio Server Pro?

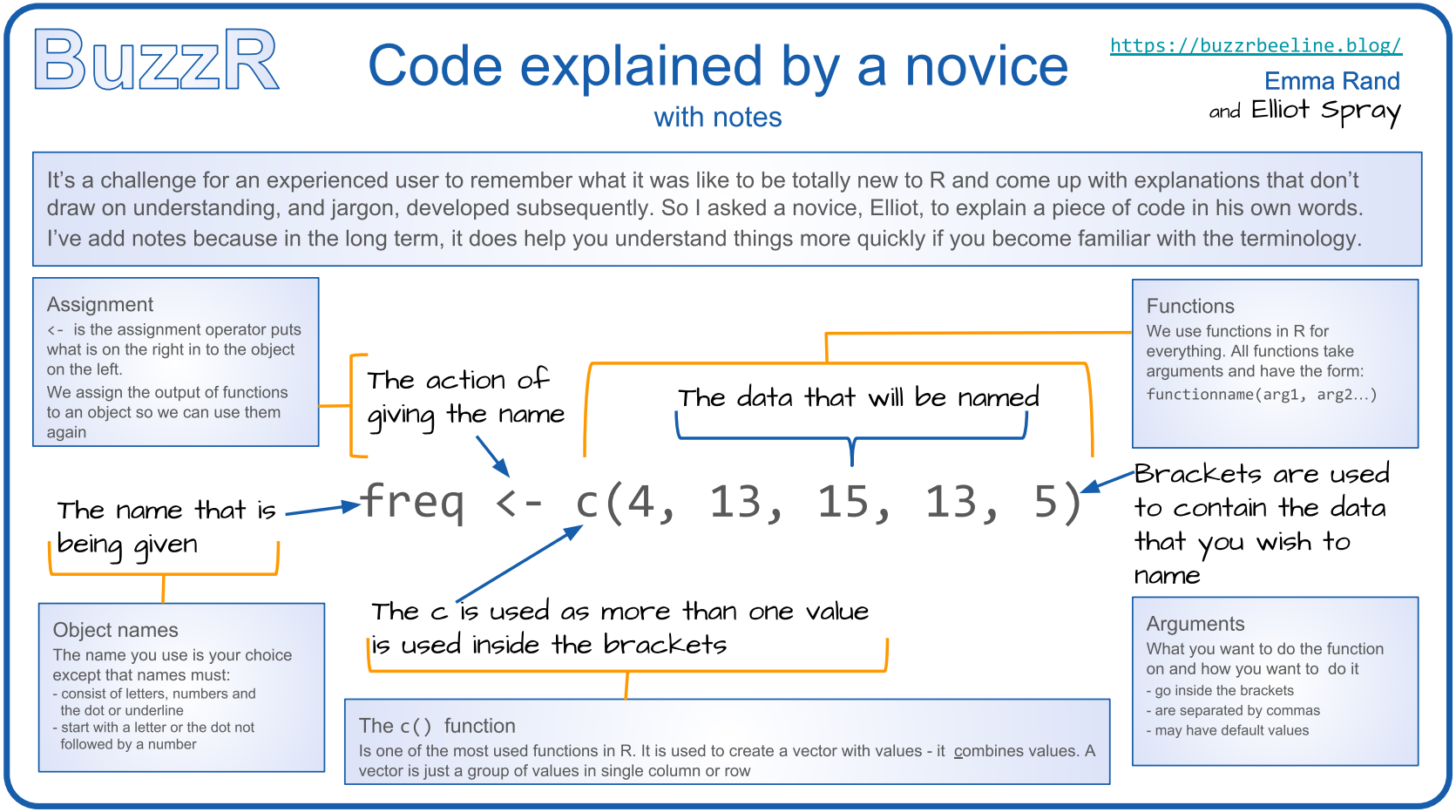
Do everything you would do with the open source server with a commercial license, support, and more.

- edit the same project at the same time as others
- switch easily from one version of R to a different version
- open and run multiple R sessions simultaneously
- see what you and others are doing on your server
- tune your resources to improve performance
- integrate with your authentication, authorization, and audit practices

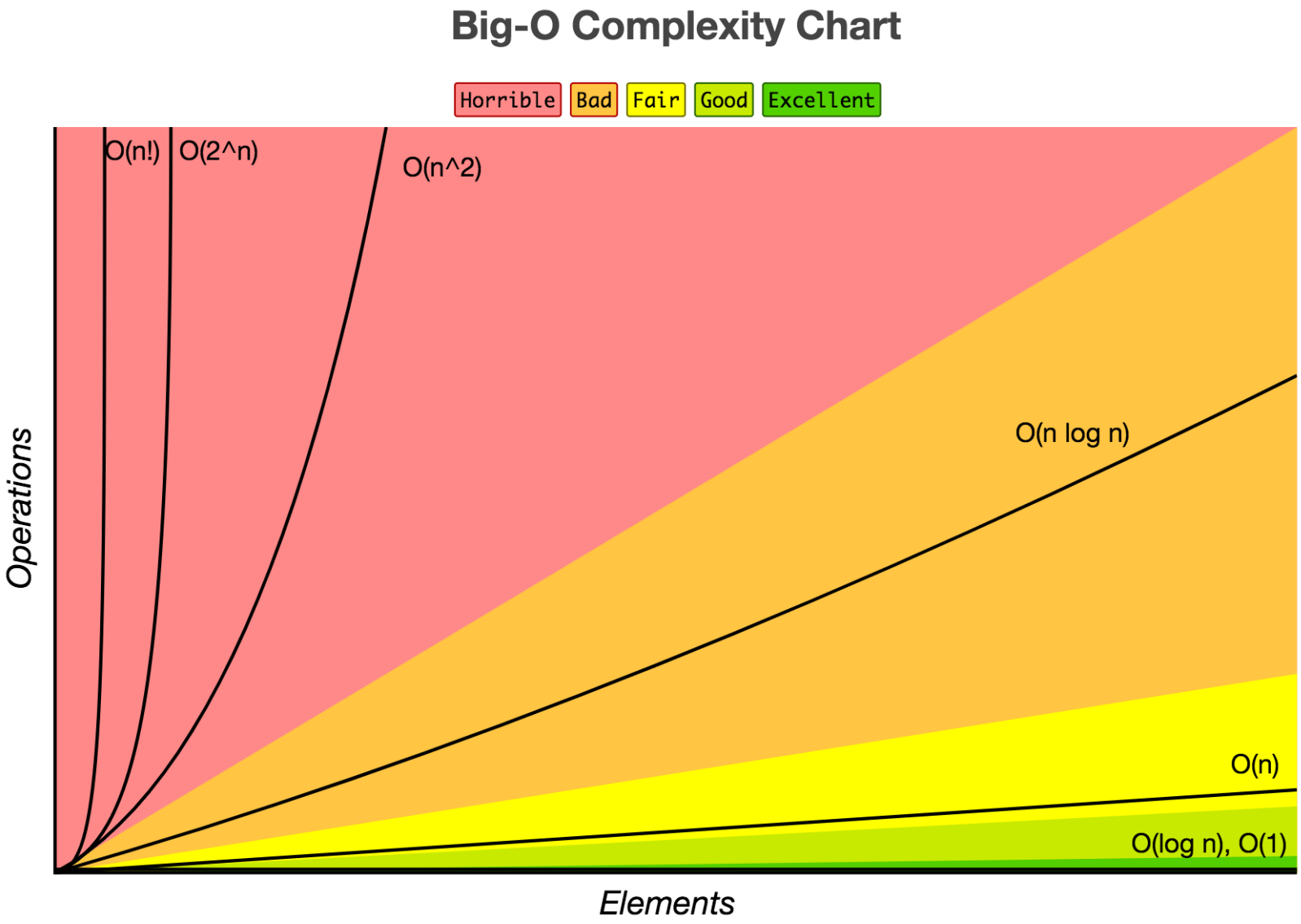
Download a free 45 day evaluation at
www.rstudio.com/products/rstudio-server-pro/

 More cheat sheets at **http://www.rstudio.com/resources/cheatsheets/**

Learn more at **support.rstudio.com** • RStudio IDE 0.99.832 • Updated: 01/16


BuzzR
Code explained by a novice
with notes
https://buzzrbeeline.blog/
Emma Rand
and Elliot Spray
It's a challenge for an experienced user to remember what it was like to be totally new to R and come up with explanations that don't draw on understanding, and jargon, developed subsequently. So I asked a novice, Elliot, to explain a piece of code in his own words. I've add notes because in the long term, it does help you understand things more quickly if you become familiar with the terminology.
Assignment
<- is the assignment operator puts what is on the right in to the object on the left.
We assign the output of functions to an object so we can use them again
The action of giving the name
The data that will be named
Functions
We use functions in R for everything. All functions take arguments and have the form:
functionname(arg1, arg2...)
The name that is being given
freq <- c(4, 13, 15, 13, 5)
Brackets are used to contain the data that you wish to name
Object names
The name you use is your choice except that names must:
- consist of letters, numbers and the dot or underline
- start with a letter or the dot not followed by a number
The c is used as more than one value is used inside the brackets
Arguments
What you want to do the function on and how you want to do it
- go inside the brackets
- are separated by commas
- may have default values
The c() function
Is one of the most used functions in R. It is used to create a vector with values - it combines values. A vector is just a group of values in single column or row

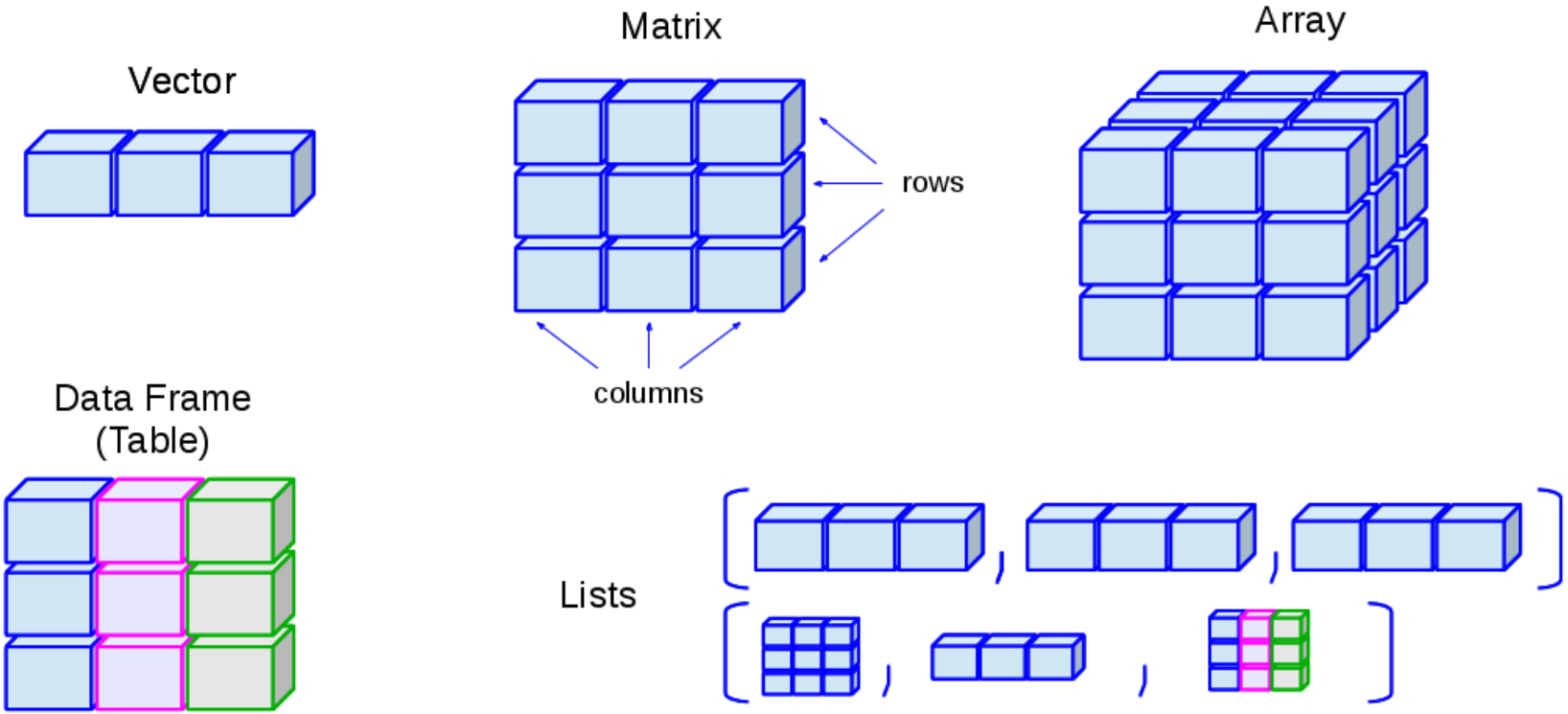
Złożoność obliczeniowa



Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Obiekty oraz typy



Eksploracyjna analiza danych (EDA)

EDA odpowiada za proces eksploracji danych w celu późniejszego wykonania analiz. Skupiona jest głównie na obejrzeniu danych, podsumowaniu ich, znalezieniu głównych charakterystyk i wizualizacji.

Eksploracyjna analiza danych (EDA)

`dplyr` jest pakietem zaprojektowanym do wydajnego (napisany w C++) i efektywnego manipulowania danymi. Dodatkowo wszystko co robimy na ramce danych możemy też zrobić na innych obiektach np. tabelach w bazie danych. Jest on obecnie częścią większej grupy pakietów zwanej `tidyverse`.

Data transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …**)** Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL**)** Count number of rows in each group defined by the variables in … Also **tally()**.
count(mtcars, cyl)

Group Cases

Use **group_by(**.data, …, .add = FALSE, .drop = TRUE**)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

```
mtcars %>%
  group_by(cyl) %>%
  summarise(avg = mean(mpg))
```

Use **rowwise(**.data, …**)** to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

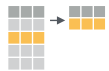
```
starwars %>%
  rowwise() %>%
  mutate(film_count = length(films))
```

ungroup(x, …**)** Returns ungrouped copy of table.
ungroup(g_mtcars)

Manipulate Cases

EXTRACT CASES

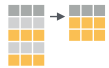
Row functions return a subset of rows as a new table.



filter(.data, …, .preserve = FALSE**)** Extract rows that meet logical criteria.
filter(mtcars, mpg > 20)

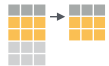
distinct(.data, …, .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(mtcars, gear)

slice(.data, …, .preserve = FALSE**)** Select rows by position.
slice(mtcars, 10:15)



slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE**)** Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
slice_sample(mtcars, n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE**)** and **slice_max()** Select rows with the lowest and highest values.
slice_min(mtcars, mpg, prop = 0.25)



slice_head(.data, …, n, prop**)** and **slice_tail()** Select the first or last rows.
slice_head(mtcars, n = 5)

Logical and boolean operators to use with filter()

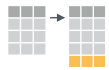
==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …, .by_group = FALSE**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES



add_row(.data, …, .before = NULL, .after = NULL**)** Add one or more rows to a table.
add_row(cars, speed = 1, dist = 1)

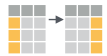
Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …**)** Extract column values as a vector, by name or index.
pull(mtcars, wt)

select(.data, …**)** Extract columns as a table.
select(mtcars, mpg, wt)



relocate(.data, …, .before = NULL, .after = NULL**)** Move columns to new position.
relocate(mtcars, mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. select(mtcars, mpg:cyl)

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g. mpg:cyl
ends_with(match**)**	**all_of(**x**)/any_of(**x, …, vars**)**	**-**, e.g. -gear
starts_with(match**)**	**matches(**match**)**	**everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE

across(.cols, .funs, …, .names = NULL**)** Summarise or mutate multiple columns in the same way.
summarise(mtcars, across(everything(), mean))

c_across(.cols**)** Compute across columns in row-wise data.
transmute(rowwise(UKgas), total = sum(c_across(1:2)))

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …, .keep = "all", .before = NULL, .after = NULL**)** Compute new column(s). Also **add_column(), add_count(),** and **add_tally().**
mutate(mtcars, gpm = 1 / mpg)

transmute(.data, …**)** Compute new column(s), drop others.
transmute(mtcars, gpm = 1 / mpg)

rename(.data, …**)** Rename columns. Use **rename_with()** to rename with a function.
rename(cars, distance = dist)

RStudio

dplyr – karta pomocy (2)

Vectorized Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSET

dplyr::**lag()** - offset elements by 1
dplyr::**lead()** - offset elements by -1

CUMULATIVE AGGREGATE

dplyr::**cumall()** - cumulative all()
dplyr::**cumany()** - cumulative any()
cummax() - cumulative max()
dplyr::**cummean()** - cumulative mean()
cummin() - cumulative min()
cumprod() - cumulative prod()
cumsum() - cumulative sum()

RANKING

dplyr::**cume_dist()** - proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISCELLANEOUS

dplyr::**case_when()** - multi-case if_else()

```
starwars %>%
  mutate(type = case_when(
    height > 200 | mass > 200 ~ "large",
    species == "Droid"        ~ "robot",
    TRUE                      ~ "other")
  )
```

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNT

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

POSITION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICAL

mean() - proportion of TRUE's
sum() - # of TRUE's

ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

tibble::**rownames_to_column()** Move row names into col.
a <- rownames_to_column(mtcars, var = "C")

tibble::**column_to_rownames()** Move col into row names.
column_to_rownames(a, var = "C")

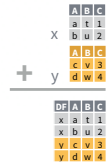
Also tibble::**has_rownames()** and tibble::**remove_rownames()**.

Combine Tables

COMBINE VARIABLES

bind_cols(…, .name_repair**)** Returns tables placed side by side as a single table. Column lengths must be equal. Columns will NOT be matched by id (to do that look at Relational Data below), so be sure to check that both tables are ordered the way you want before binding.

COMBINE CASES



bind_rows(…, .id = NULL**)** Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured).

RELATIONAL DATA

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matches = "na"**)** Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matches = "na"**)** Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matches = "na"**)** Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matches = "na"**)** Join data. Retain all values, all rows.

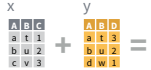
COLUMN MATCHING FOR JOINS

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

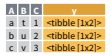
Use a "**Filtering Join**" to filter one table against the rows of another.



semi_join(x, y, by = NULL, copy = FALSE, …, na_matches = "na"**)** Return rows of x that have a match in y. Use to see what will be included in a join.

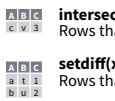
anti_join(x, y, by = NULL, copy = FALSE, …, na_matches = "na"**)** Return rows of x that do not have a match in y. Use to see what will not be included in a join.

Use a "**Nest Join**" to inner join one table to another into a nested data frame.



nest_join(x, y, by = NULL, copy = FALSE, keep = FALSE, name = NULL, …**)** Join data, nesting matches from y in a single new data frame column.

SET OPERATIONS



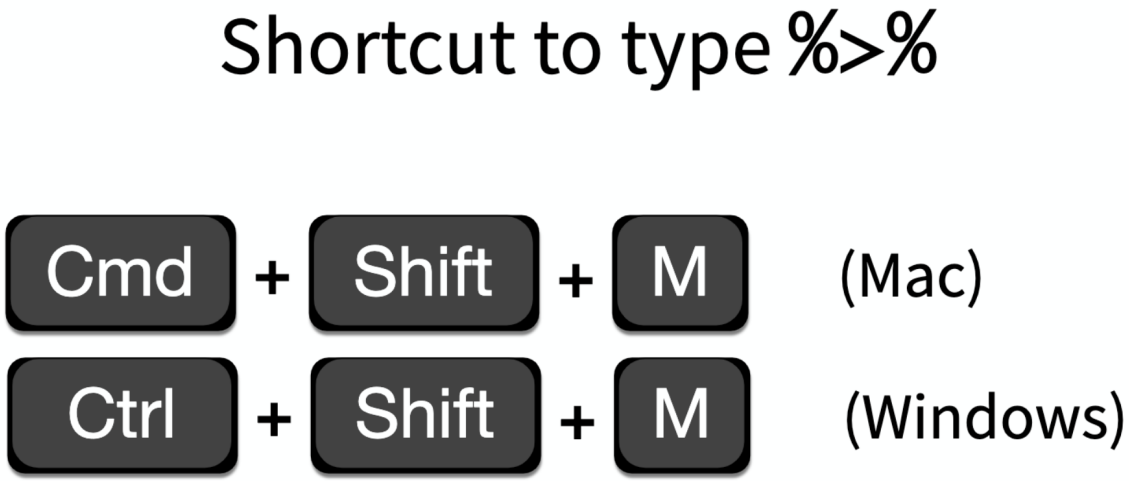
intersect(x, y, …) Rows that appear in both x and y.

setdiff(x, y, …) Rows that appear in x but not y.

union(x, y, …) Rows that appear in x or y. (Duplicates removed). **union_all()** retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

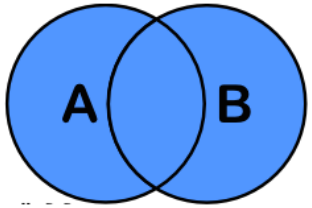
- filter,
- slice, top_n,
- arrange,
- %>% – strumienie.



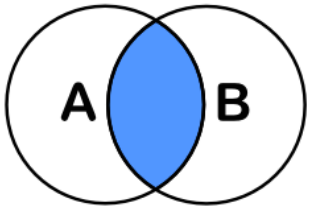
dplyr – wybieranie zmiennych, grupowanie i podsumowania

- select,
- mutate,
- group_by, count,
- summarise.

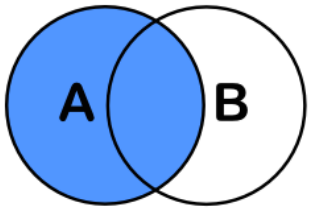
dplyr – łączenie ramek danych



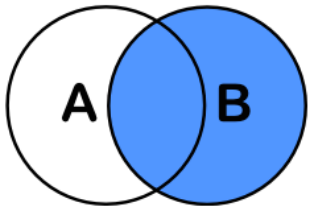
full_join(A,B)



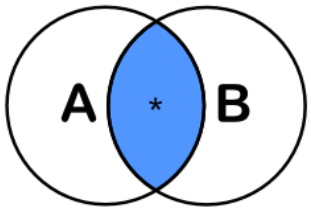
inner_join(A,B)



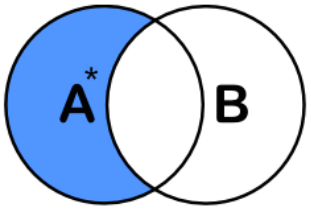
left_join(A,B)



right_join(A,B)

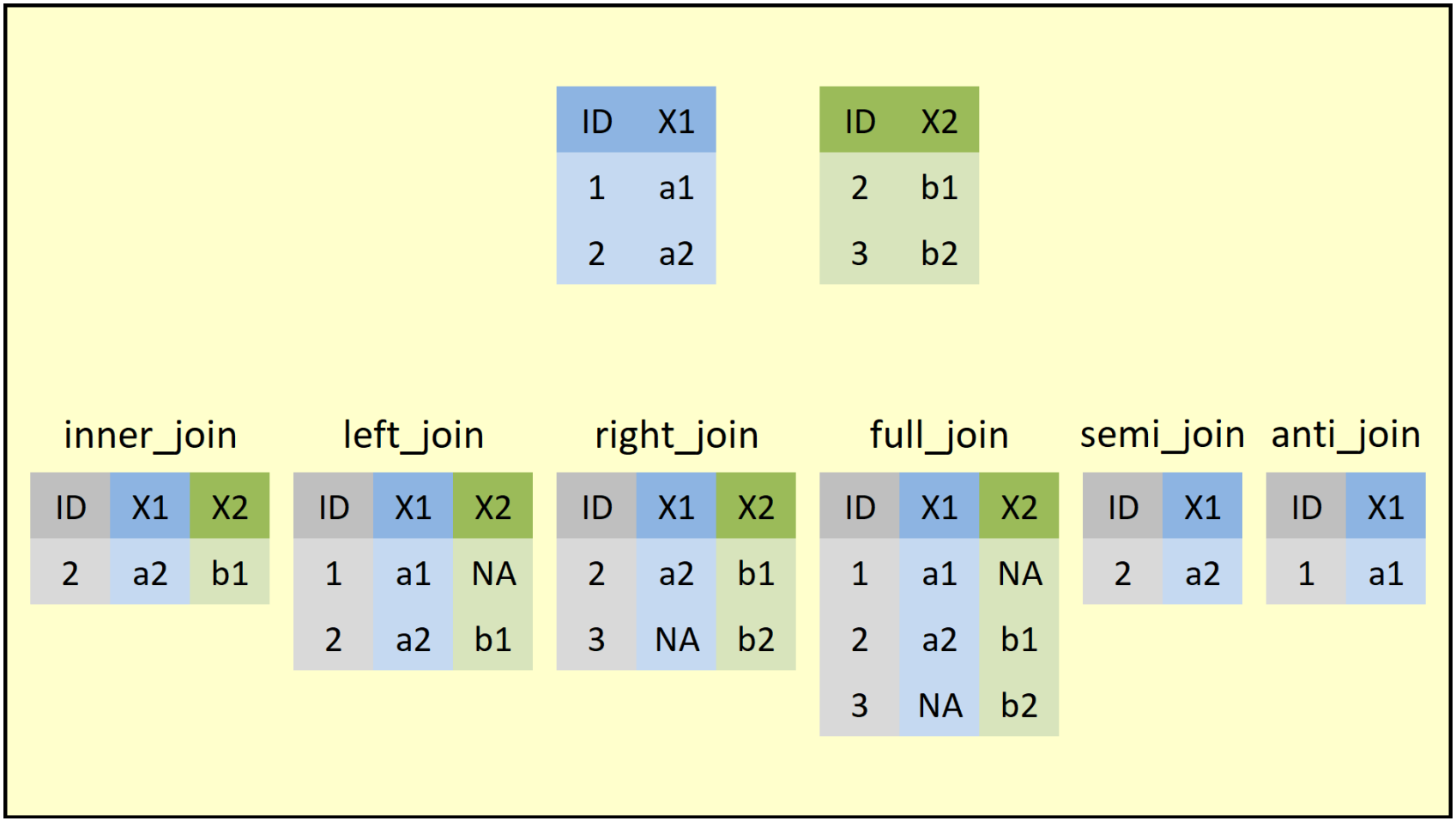


semi_join(A,B)



anti_join(A,B)

dplyr – łączenie ramek danych



ID	X1
1	a1
2	a2

ID	X2
2	b1
3	b2

inner_join

ID	X1	X2
2	a2	b1

left_join

ID	X1	X2
1	a1	NA
2	a2	b1

right_join

ID	X1	X2
2	a2	b1
3	NA	b2

full_join

ID	X1	X2
1	a1	NA
2	a2	b1
3	NA	b2

semi_join

ID	X1
2	a2

anti_join

ID	X1
1	a1

Tidy data

A data set is **tidy** iff:

1. Each **variable** is in its own **column**
2. Each **case** is in its own **row**
3. Each **value** is in its own **cell**

tame data

baker	cinnamon_1	cardamom_2	nutmeg_3
Emma	1	0	1
Harry	1	1	1
Ruby	1	0	1
Zainab	0	NA	0

trial	Emma	Harry	Ruby	Zainab
cinnamon_1	1	1	1	0
cardamom_2	0	1	0	NA
nutmeg_3	1	1	1	0

tidy data

baker	spice	order	correct
Emma	Cinnamon	1	1
Harry	Cinnamon	1	1
Ruby	Cinnamon	1	1
Zainab	Cinnamon	1	0
Emma	Cardamom	2	0
Harry	Cardamom	2	1
Ruby	Cardamom	2	0
Zainab	Cardamom	2	NA
Emma	Nutmeg	3	1
Harry	Nutmeg	3	1
Ruby	Nutmeg	3	1
Zainab	Nutmeg	3	0

Dane w postaci szerokiej i długiej

wide

id	x	y	z
1	a	c	e
2	b	d	f

long

id	key	val
1	x	a
2	x	b
1	y	c
2	y	d
1	z	e
2	z	f

Data tidying with tidyr : : CHEAT SHEET

Tidy data is a way to organize tabular data in a consistent data structure across packages.
A table is tidy if:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own row

Access **variables** as **vectors** — Preserve **cases** in vectorized operations

Tibbles

AN ENHANCED DATA FRAME

Tibbles are a table format provided by the **tibble** package. They inherit the data frame class, but have improved behaviors:

- **Subset** a new tibble with], a vector with [[and $.
- **No partial matching** when subsetting columns.
- **Display** concise views of the data on one screen.

options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)** Control default display settings.

View() or **glimpse()** View the entire data set.

CONSTRUCT A TIBBLE

tibble(…**)** Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…**)** Construct by rows.
tribble(~x, ~y,
1, "a",
2, "b",
3, "c")

Both make this tibble

```
A tibble: 3 x 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …**)** Convert a data frame to a tibble.

enframe(x, name = "name", value = "value"**)** Convert a named vector to a tibble. Also **deframe()**.

is_tibble(x**)** Test whether x is a tibble.

Reshape Data - Pivot data to reorganize values into a new layout.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

pivot_longer(data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE**)**

"Lengthen" data by collapsing several columns into two. Column names move to a new names_to column and values to a new values_to column.

pivot_longer(table4a, cols = 2:3, names_to ="year", values_to = "cases")

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

pivot_wider(data, names_from = "name", values_from = "value"**)**

The inverse of pivot_longer(). "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

pivot_wider(table2, names_from = type, values_from = count)

Split Cells - Use these functions to split or combine cells into individual, isolated values.

table5

country	century	year
A	19	99
A	20	00
B	19	99
B	20	00

→

country	year
A	1999
A	2000
B	1999
B	2000

unite(data, col, …, sep = "_", remove = TRUE, na.rm = FALSE**)** Collapse cells across several columns into a single column.

unite(table5, century, year, col = "year", sep = "")

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", …**)** Separate each cell in a column into several columns. Also **extract()**.

separate(table3, rate, sep = "/", into = c("cases", "pop"))

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M

separate_rows(data, …, sep = "[^[:alnum:].]+", convert = FALSE**)** Separate each cell in a column into several rows.

separate_rows(table3, rate, sep = "/")

Expand Tables

Create new combinations of variables or identify implicit missing values (combinations of variables not present in the data).

x

x1	x2	x3
A	1	3
B	1	4
B	2	3

→

x1	x2
A	1
A	2
B	1
B	2

expand(data, …**)** Create a new tibble with all possible combinations of the values of the variables listed in … Drop other variables.
expand(mtcars, cyl, gear, carb)

x

x1	x2	x3
A	1	3
B	1	4
B	2	3

→

x1	x2	x3
A	1	3
A	2	NA
B	1	4
B	2	3

complete(data, …, fill = list()**)** Add missing possible combinations of values of variables listed in … Fill remaining variables with NA.
complete(mtcars, cyl, gear, carb)

Handle Missing Values

Drop or replace explicit missing values (NA).

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(data, …**)** Drop rows containing NA's in … columns.
drop_na(x, x2)

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(data, …, .direction = "down"**)** Fill in NA's in … columns using the next or previous value.
fill(x, x2)

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(data, replace**)** Specify a value to replace NA in selected columns.
replace_na(x, list(x2 = 2))