Environmental Health Big Data Analysis – R4ds (2) Data Wrangling and Programing

Il-Youp Kwak

Chung-Ang University

What we will learn

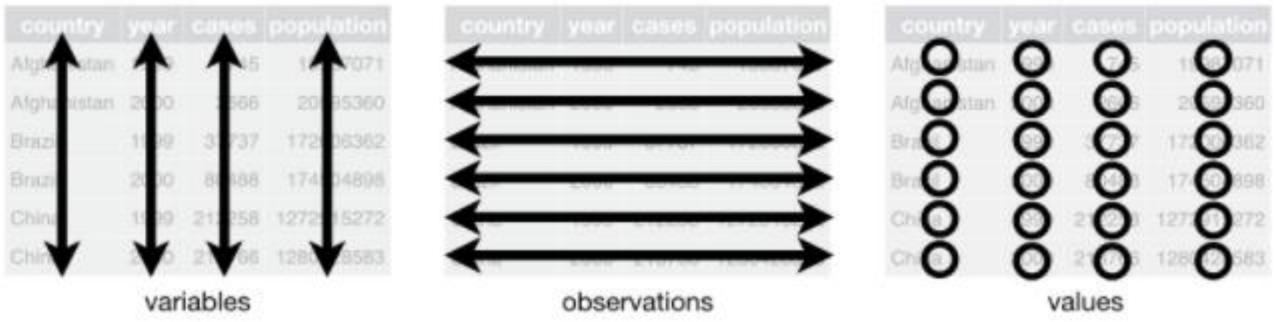
1. Data transformation
2. Data visualization
3. Data wrangling
4. Functional programming with R
5. Something useful

Data Wrangling - tidyr

"Tidy datasets are all alike, but every messy dataset is messy in its own way." — Hadley Wickham

Tidy data:

- **Every column is variable**
- **Every row is an observation.**
- **Every cell is a single value.**



https://r4ds.had.co.nz/tidy-data.html

Why tidyr?

- If you have a consistent data structure, it's easier to learn the tools that work with it.

- Most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

- dplyr, ggplot2, and all the other packages in the tidyverse are designed to work with tidy data.

https://r4ds.had.co.nz/tidy-data.html

Example Data

Representation of the same data in multiple ways

```
table1
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

```
table2
#> # A tibble: 12 x 4
#>   country      year type            count
#>   <chr>       <int> <chr>           <int>
#> 1 Afghanistan  1999 cases             745
#> 2 Afghanistan  1999 population   19987071
#> 3 Afghanistan  2000 cases            2666
#> 4 Afghanistan  2000 population   20595360
#> 5 Brazil       1999 cases           37737
#> 6 Brazil       1999 population  172006362
#> # … with 6 more rows
```

```
table3
#> # A tibble: 6 x 3
#>   country      year rate
#> * <chr>       <int> <chr>
#> 1 Afghanistan  1999 745/19987071
#> 2 Afghanistan  2000 2666/20595360
#> 3 Brazil       1999 37737/172006362
#> 4 Brazil       2000 80488/174504898
#> 5 China        1999 212258/1272915272
#> 6 China        2000 213766/1280428583
```

```
table4a  # cases
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766
```

```
table4b  # population
#> # A tibble: 3 x 3
#>   country         `1999`     `2000`
#> * <chr>            <int>      <int>
#> 1 Afghanistan   19987071   20595360
#> 2 Brazil       172006362  174504898
#> 3 China       1272915272 1280428583
```

Pivoting

- **One variable might be spread across multiple columns. (Use Pivot_longer())**

```
table4a  # cases
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766
```

```
table4b  # population
#> # A tibble: 3 x 3
#>   country         `1999`     `2000`
#> * <chr>            <int>      <int>
#> 1 Afghanistan   19987071   20595360
#> 2 Brazil       172006362  174504898
#> 3 China       1272915272 1280428583
```

- **One observation might be scattered across multiple rows. (Use Pivot_wider())**

https://r4ds.had.co.nz/tidy-data.html

Apply Pivot_longer() to table4a

```
table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")
#> # A tibble: 6 x 3
#>   country     year   cases
#>   <chr>       <chr>  <int>
#> 1 Afghanistan 1999     745
#> 2 Afghanistan 2000    2666
#> 3 Brazil      1999   37737
#> 4 Brazil      2000   80488
#> 5 China       1999  212258
#> 6 China       2000  213766
```

https://r4ds.had.co.nz/tidy-data.html

Apply Pivot_longer() and combine data

```
tidy4a <- table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")
tidy4b <- table4b %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "population")
left_join(tidy4a, tidy4b)
#> Joining, by = c("country", "year")
#> # A tibble: 6 x 4
#>   country     year   cases population
#>   <chr>       <chr>  <int>      <int>
#> 1 Afghanistan 1999     745   19987071
#> 2 Afghanistan 2000    2666   20595360
#> 3 Brazil      1999   37737  172006362
#> 4 Brazil      2000   80488  174504898
#> 5 China       1999  212258 1272915272
#> 6 China       2000  213766 1280428583
```

```
table4a  # cases
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766
```

```
table4b  # population
#> # A tibble: 3 x 3
#>   country         `1999`     `2000`
#> * <chr>            <int>      <int>
#> 1 Afghanistan   19987071   20595360
#> 2 Brazil       172006362  174504898
#> 3 China       1272915272 1280428583
```

https://r4ds.had.co.nz/tidy-data.html

R4ds_2.ipynb

File Edit View Insert Runtime Tools Help Unsaved changes since 8:58 PM

+ Code + Text

RAM Disk Editing

R for Data Science 실습 2 - Data wrangling and programing

Data Wrangling (tidyr)

[1]
```
library(tidyverse)
```

```
Warning message in system("timedatectl", intern = TRUE):
 "running command 'timedatectl' had status 1"
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──

✓ ggplot2 3.3.5     ✓ purrr   0.3.4
✓ tibble  3.1.4     ✓ dplyr   1.0.7
✓ tidyr   1.1.3     ✓ stringr 1.4.0
✓ readr   2.0.1     ✓ forcats 0.5.1

── Conflicts ──────────────────────────────────────────── tidyverse_conflicts() ──
✕ dplyr::filter() masks stats::filter()
✕ dplyr::lag()    masks stats::lag()
```

Example datasets

[ ]
```
table1
```

A tibble: 6 × 4

✓ 1s completed at 8:59 PM

Pivoting

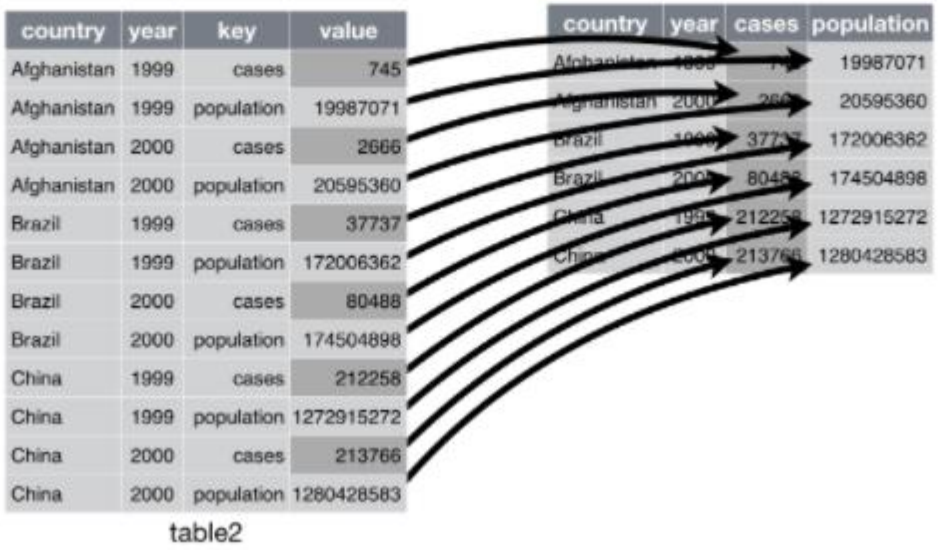
- One variable might be spread across multiple columns. (Use Pivot_longer())
- **One observation might be scattered across multiple rows. (Use Pivot_wider())**

```
table2
#> # A tibble: 12 x 4
#>   country      year type          count
#>   <chr>       <int> <chr>         <int>
#> 1 Afghanistan  1999 cases           745
#> 2 Afghanistan  1999 population  19987071
#> 3 Afghanistan  2000 cases          2666
#> 4 Afghanistan  2000 population  20595360
#> 5 Brazil       1999 cases         37737
#> 6 Brazil       1999 population 172006362
#> # … with 6 more rows
```

https://r4ds.had.co.nz/tidy-data.html

Apply Pivot_wider()

```
table2 %>%
   pivot_wider(names_from = type, values_from = count)
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```



https://r4ds.had.co.nz/tidy-data.html

A tibble: 6 × 4

country	year	cases	population
<chr>	<chr>	<int>	<int>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Q: What is right_join? try ?right_join and study join methods

Pivot_wider()

One observation might be scattered across multiple rows

```
[ ] table2
```

A tibble: 12 × 4

country	year	type	count

Separating and uniting

- **One column contains two variables (Use separate())**

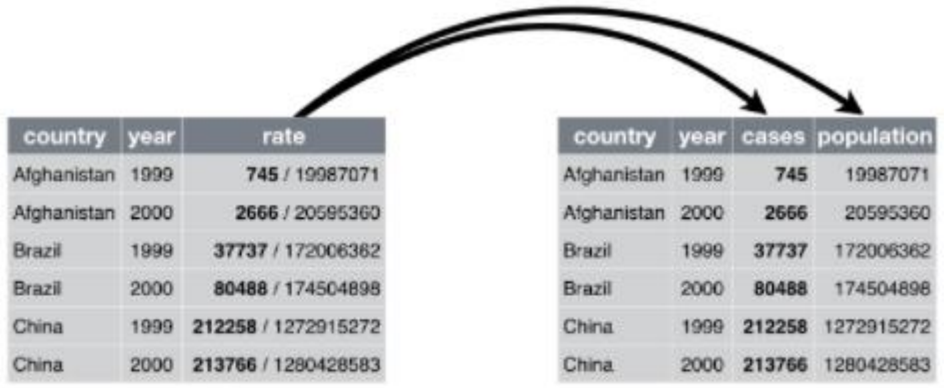
```
table3
#> # A tibble: 6 x 3
#>   country      year rate
#> * <chr>       <int> <chr>
#> 1 Afghanistan  1999 745/19987071
#> 2 Afghanistan  2000 2666/20595360
#> 3 Brazil       1999 37737/172006362
#> 4 Brazil       2000 80488/174504898
#> 5 China        1999 212258/1272915272
#> 6 China        2000 213766/1280428583
```

- **Single variable is spread across multiple columns. (Use unite())**

https://r4ds.had.co.nz/tidy-data.html

Apply separate()

```
table3 %>%
  separate(rate, into = c("cases", "population"))
#> # A tibble: 6 x 4
#>   country      year cases  population
#>   <chr>       <int> <chr>  <chr>
#> 1 Afghanistan  1999 745    19987071
#> 2 Afghanistan  2000 2666   20595360
#> 3 Brazil       1999 37737  172006362
#> 4 Brazil       2000 80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```



country	year	rate
Afghanistan	1999	**745** / 19987071
Afghanistan	2000	**2666** / 20595360
Brazil	1999	**37737** / 172006362
Brazil	2000	**80488** / 174504898
China	1999	**212258** / 1272915272
China	2000	**213766** / 1280428583

table3

country	year	cases	population
Afghanistan	1999	**745**	19987071
Afghanistan	2000	**2666**	20595360
Brazil	1999	**37737**	172006362
Brazil	2000	**80488**	174504898
China	1999	**212258**	1272915272
China	2000	**213766**	1280428583

Separating and uniting

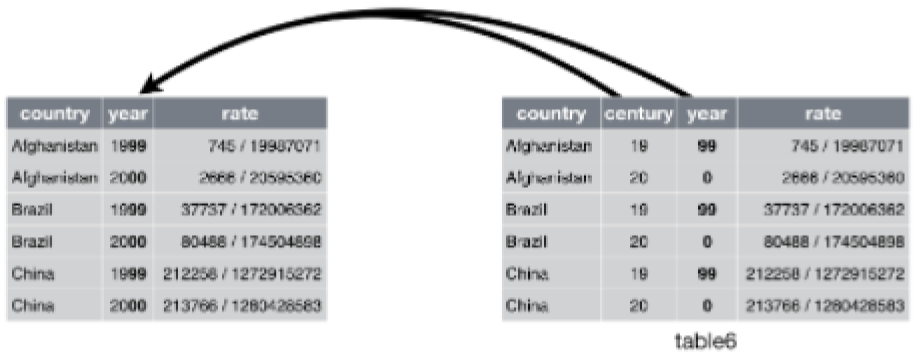
- One column contains two variables (Use separate())
- **Single variable is spread across multiple columns. (Use unite())**

	country	century	year	rate
	<chr>	<chr>	<chr>	<chr>
1	Afghanistan	19	99	745/19987071
2	Afghanistan	20	00	2666/20595360
3	Brazil	19	99	37737/172006362
4	Brazil	20	00	80488/174504898
5	China	19	99	212258/1272915272
6	China	20	00	213766/1280428583

https://r4ds.had.co.nz/tidy-data.html

Apply unite()

```
table5 %>%
  unite(new, century, year, sep = "")
#> # A tibble: 6 x 3
#>   country     new   rate
#>   <chr>       <chr> <chr>
#> 1 Afghanistan 1999  745/19987071
#> 2 Afghanistan 2000  2666/20595360
#> 3 Brazil      1999  37737/172006362
#> 4 Brazil      2000  80488/174504898
#> 5 China       1999  212258/1272915272
#> 6 China       2000  213766/1280428583
```



https://r4ds.had.co.nz/tidy-data.html

Separating and Uniting

separate()

One column contains two variables

```
table3
```

A tibble: 6 × 3

	country	year	rate
	<chr>	<int>	<chr>
1	Afghanistan	1999	745/19987071
2	Afghanistan	2000	2666/20595360
3	Brazil	1999	37737/172006362
4	Brazil	2000	80488/174504898
5	China	1999	212258/1272915272
6	China	2000	213766/1280428583

```
table3 %>% separate(rate, into = c("cases", "population"))
```

Functional Programming

- For loops are quite verbose, and require quite a bit of bookkeeping code that is duplicated for every for loop.

- Functional programming offers tools to extract out this duplicated code, so each common for loop pattern gets its own function.

https://r4ds.had.co.nz/iteration.html

For loops

Random data

```
df <- tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)
```

To calculate median

```
median(df$a)
#> [1] -0.2457625
median(df$b)
#> [1] -0.2873072
median(df$c)
#> [1] -0.05669771
median(df$d)
#> [1] 0.1442633
```

With for loops

```
output <- vector("double", ncol(df))  # 1. output
for (i in seq_along(df)) {            # 2. sequence
  output[[i]] <- median(df[[i]])      # 3. body
}
output
#> [1] -0.24576245 -0.28730721 -0.05669771  0.14426335
```

https://r4ds.had.co.nz/iteration.html

Three components

The output: output <- vector("double", length(x)).
Allocate sufficient space for the output.
(If you grow the for loop at each iteration using c() (for example), your for loop will be very slow)

The sequence: i in seq_along(df).
what to loop over:
each run of the for loop will assign i to a different value from seq_along(df)

The body: output[[i]] <- median(df[[i]]).
This is the code that does the work.

```
output <- vector("double", ncol(df))  # 1. output
for (i in seq_along(df)) {            # 2. sequence
  output[[i]] <- median(df[[i]])      # 3. body
}
output
#> [1] -0.24576245 -0.28730721 -0.05669771  0.14426335
```

https://r4ds.had.co.nz/iteration.html

For loops vs. functionals

Possible to wrap up for loops in a function

```
col_mean <- function(df) {
	output <- vector("double", ncol(df))
	for (i in seq_along(df)) {
		output[[i]] <- mean(df[[i]])
	}
	output
}
```

```
col_median <- function(df) {
	output <- vector("double", ncol(df))
	for (i in seq_along(df)) {
		output[[i]] <- median(df[[i]])
	}
	output
}
```

```
col_sd <- function(df) {
	output <- vector("double", ncol(df))
	for (i in seq_along(df)) {
		output[[i]] <- sd(df[[i]])
	}
	output
}
```

```
col_summary <- function(df, fun) {
  out <- vector("double", length(df))
  for (i in seq_along(df)) {
    out[i] <- fun(df[[i]])
  }
  out
}
col_summary(df, median)
#> [1] -0.51850298  0.02779864  0.17295591 -0.61163819
col_summary(df, mean)
#> [1] -0.3260369  0.1356639  0.4291403 -0.2498034
```

https://r4ds.had.co.nz/iteration.html

R4ds_2.ipynb

File Edit View Insert Runtime Tools Help Unsaved changes since 8:58 PM

+ Code + Text

[25]

Brazil	1999	37737/172006362
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

Functional Programming

```
df <- tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)
```

```
median(df$a)
#> [1] -0.2457625
median(df$b)
#> [1] -0.2873072
median(df$c)
#> [1] -0.05669771
median(df$d)
#> [1] 0.1442633
```

0.423909950150083
0.0269074082896925
-0.0806930728782541

0s completed at 9:34 PM

The map function (purrr)

the purrr package provides a family of functions for looping patterns over a vector

- **map() makes a list**
- **map_lgl() makes a logical vector**
- **map_int() makes an integer vector**
- **map_dbl() makes a double vector**
- **map_chr() makes a character vector**

Alternatives: apply, lapply, etc

https://r4ds.had.co.nz/iteration.html

The map function (purrr)

```
df %>% map_dbl(mean)
#>          a          b          c          d
#> -0.3260369  0.1356639  0.4291403 -0.2498034
df %>% map_dbl(median)
#>           a           b           c           d
#> -0.51850298  0.02779864  0.17295591 -0.61163819
df %>% map_dbl(sd)
#>         a         b         c         d
#> 0.9214834 0.4848945 0.9816016 1.1563324
```

You can define a function in a map function

```
models <- mtcars %>%
  split(.$cyl) %>%
  map(function(df) lm(mpg ~ wt, data = df))
```

```
models <- mtcars %>%
  split(.$cyl) %>%
  map(~lm(mpg ~ wt, data = .))
```

https://r4ds.had.co.nz/iteration.html

To extract a component

```
models %>%
  map(summary) %>%
  map_dbl(~.$r.squared)
#>         4         6         8
#> 0.5086326 0.4645102 0.4229655
```

```
models %>%
  map(summary) %>%
  map_dbl("r.squared")
#>         4         6         8
#> 0.5086326 0.4645102 0.4229655
```

You can also use an integer to select elements by position

```
x <- list(list(1, 2, 3), list(4, 5, 6), list(7, 8, 9))
x %>% map_dbl(2)
#> [1] 2 5 8
```

https://r4ds.had.co.nz/iteration.html

R4ds_2.ipynb

File Edit View Insert Runtime Tools Help Unsaved changes since 8:58 PM

[43]

0.81344836910122 · 0.405377190952006 · 0.520463400489272 · -0.242193831844869

The map function (purrr)

the purrr package provides a family of functions for looping patterns over a vector

remind apply()

```
str(df)
```

```
tibble [10 × 4] (S3: tbl_df/tbl/data.frame)
 $ a: num [1:10] -0.97 0.586 -2.17 0.906 -0.997 ...
 $ b: num [1:10] 1.064 -1.168 -0.623 0.239 -1.17 ...
 $ c: num [1:10] -0.0722 -0.0892 0.4624 0.7497 0.983 ...
 $ d: num [1:10] 0.516 -0.508 0.301 -0.813 0.284 ...
```

```
df %>% map_dbl(mean)
```

a: -0.0846391036597818 b: -0.00240030324052032 c: -0.0309176854543395 d: -0.0832475108396539

```
df %>% map_dbl(median)
```

a: 0.423909950150083 b: 0.0269074082896925 c: -0.0806930728782541 d: 0.0468444173239768

```
mtcars %>%
  split(.$cyl)
```

0s completed at 9:56 PM

Something useful

- R markdown
- Boxplot vs Violinplot
- Do not use pie chart?
- Effective 2D visualization with t-SNE and UMAP

https://r4ds.had.co.nz/tidy-data.html

R for Data Science

Search

Table of contents
Welcome
1 Introduction
Explore
2 Introduction
3 Data visualisation
4 Workflow: basics
5 Data transformation
6 Workflow: scripts
7 Exploratory Data Analysis
8 Workflow: projects
Wrangle
9 Introduction
10 Tibbles
11 Data import
12 Tidy data
13 Relational data
14 Strings
15 Factors
16 Dates and times

27 R Markdown

27.1 Introduction

R Markdown provides an unified authoring framework for data science, combining your code, its results, and your prose commentary. R Markdown documents are fully reproducible and support dozens of output formats, like PDFs, Word files, slideshows, and more.

R Markdown files are designed to be used in three ways:

1. For communicating to decision makers, who want to focus on the conclusions, not the code behind the analysis.
2. For collaborating with other data scientists (including future you!), who are interested in both your conclusions, and how you reached them (i.e. the code).
3. As an environment in which to *do* data science, as a modern day lab notebook where you can capture not only what you did, but also what you were thinking.

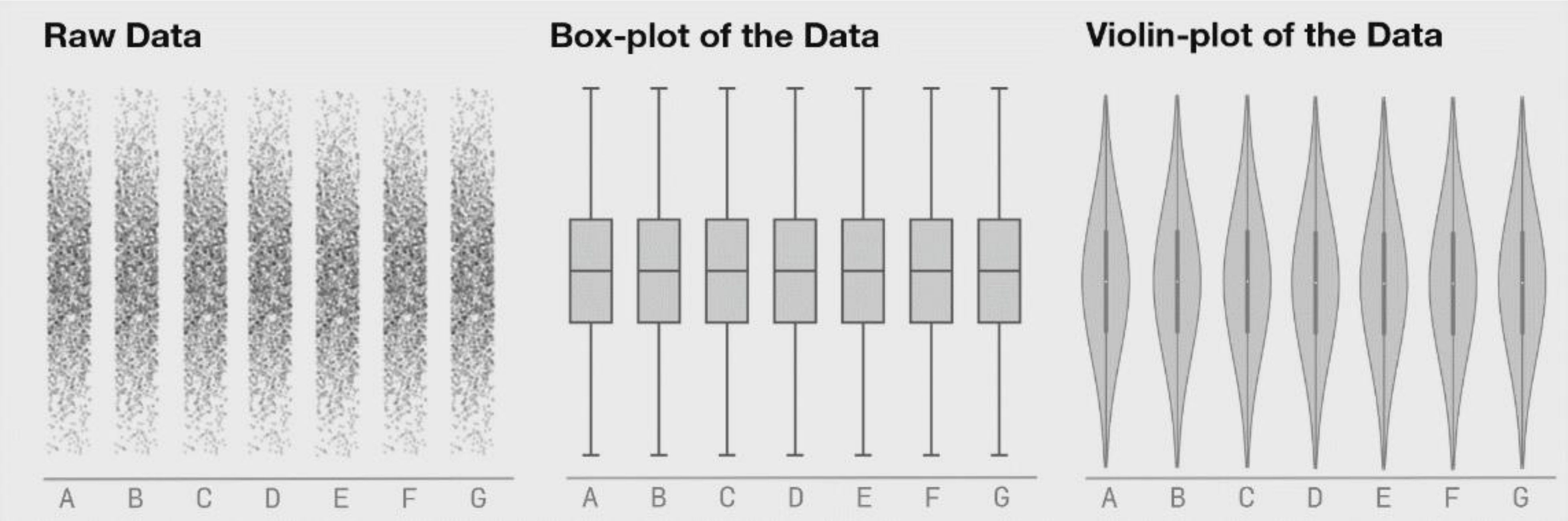
R Markdown integrates a number of R packages and external tools. This means that help is, by-and-large, not available through `?`. Instead, as you work through this chapter, and use R Markdown in the future, keep these resources close to hand:

- R Markdown Cheat Sheet: *Help > Cheatsheets > R Markdown Cheat Sheet*,

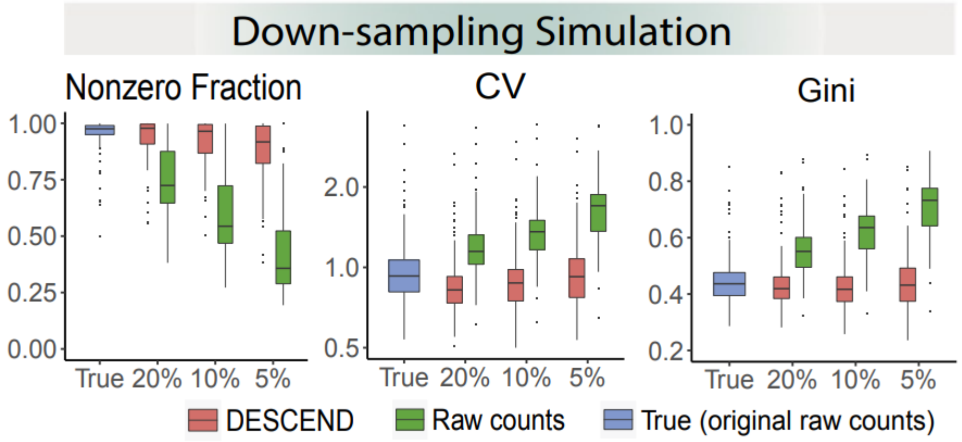
On this page
27 R Markdown
27.1 Introduction
27.2 R Markdown basics
27.3 Text formatting with Markdown
27.4 Code chunks
27.5 Troubleshooting
27.6 YAML header
27.7 Learning more

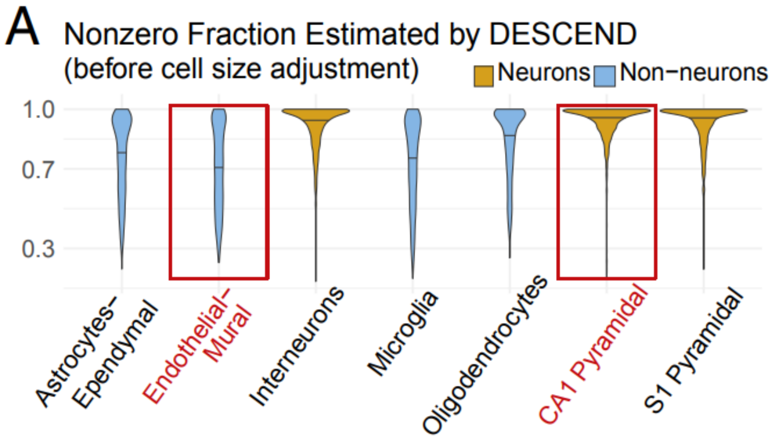
View source
Edit this page

Violinplot shows the data density



Example





https://blog.bioturing.com/2018/05/16/5-reasons-you-should-use-a-violin-graph/

R4ds_2.ipynb

File Edit View Insert Runtime Tools Help Unsaved changes since 8:58 PM

Share

+ Code + Text

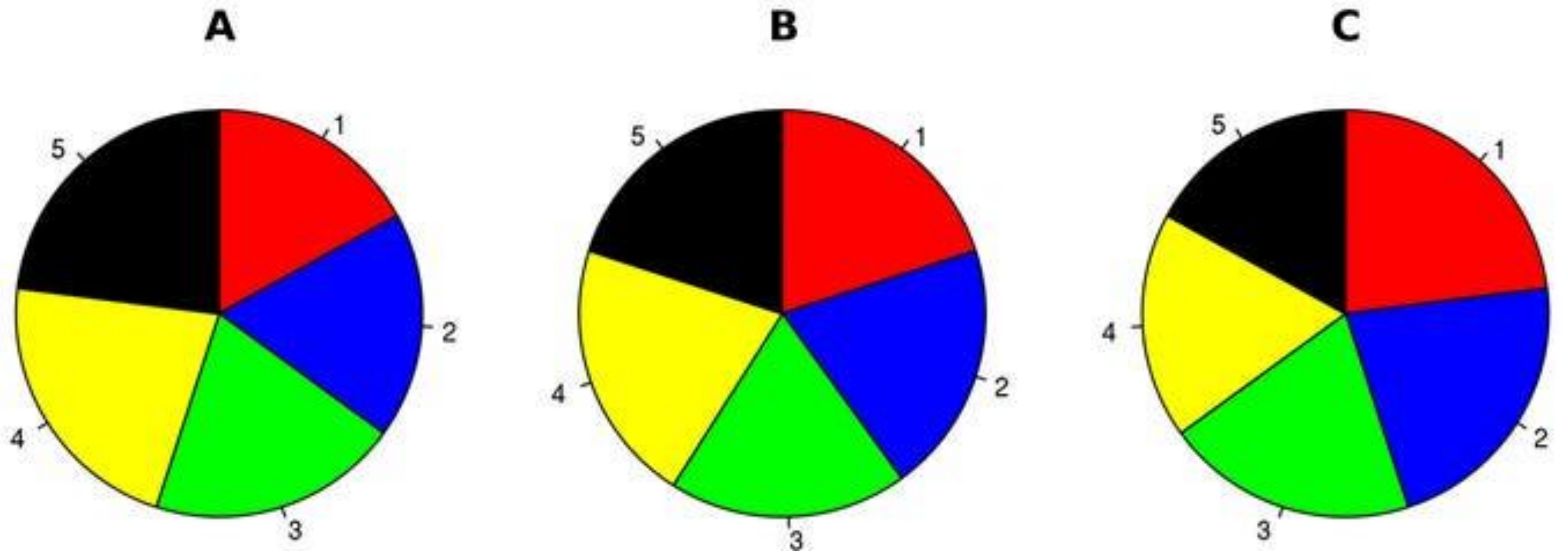
RAM
Disk

Editing

0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48
0.24	Very Good	I	VVS1	62.3	57	336	3.95	3.98	2.47
0.26	Very Good	H	SI1	61.9	55	337	4.07	4.11	2.53
0.22	Fair	E	VS2	65.1	61	337	3.87	3.78	2.49
0.23	Very Good	H	VS1	59.4	61	338	4.00	4.05	2.39
0.30	Good	J	SI1	64.0	55	339	4.25	4.28	2.73
0.23	Ideal	J	VS1	62.8	56	340	3.93	3.90	2.46
0.22	Premium	F	SI1	60.4	61	342	3.88	3.84	2.33
0.31	Ideal	J	SI2	62.2	54	344	4.35	4.37	2.71
0.20	Premium	E	SI2	60.2	62	345	3.79	3.75	2.27
0.32	Premium	E	I1	60.9	58	345	4.38	4.42	2.68
0.30	Ideal	I	SI2	62.0	54	348	4.31	4.34	2.68
0.30	Good	J	SI1	63.4	54	351	4.23	4.29	2.70
0.30	Good	J	SI1	63.8	56	351	4.23	4.26	2.71
0.30	Very Good	J	SI1	62.7	59	351	4.21	4.27	2.66
0.30	Good	I	SI2	63.3	56	351	4.26	4.30	2.71
0.23	Very Good	E	VS2	63.8	55	352	3.85	3.92	2.48

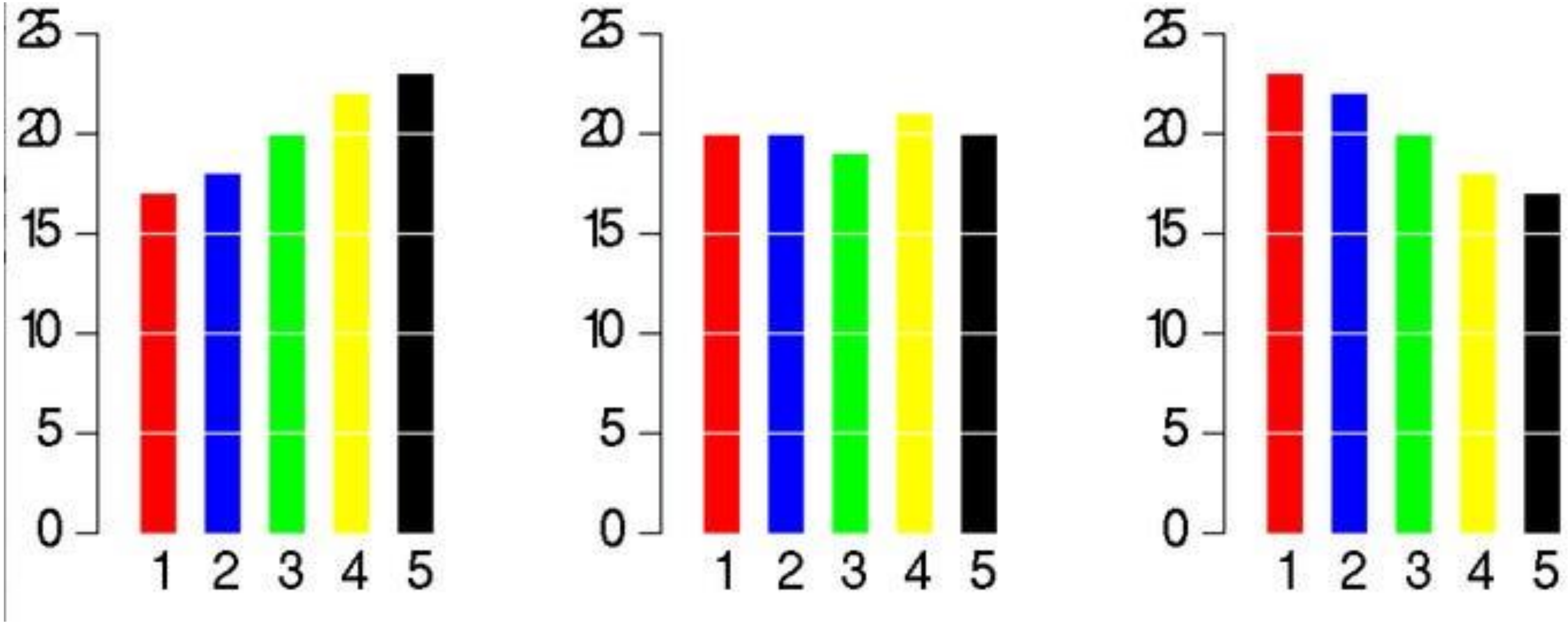
✓ 1s completed at 10:29 PM

Do not use pie chart?



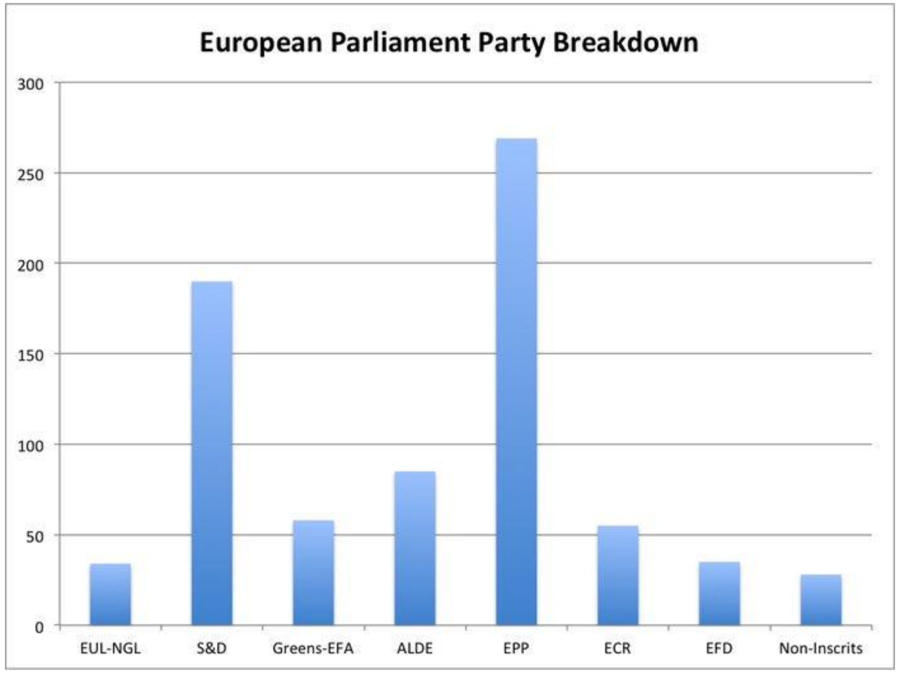
https://www.businessinsider.com/pie-charts-are-the-worst-2013-6

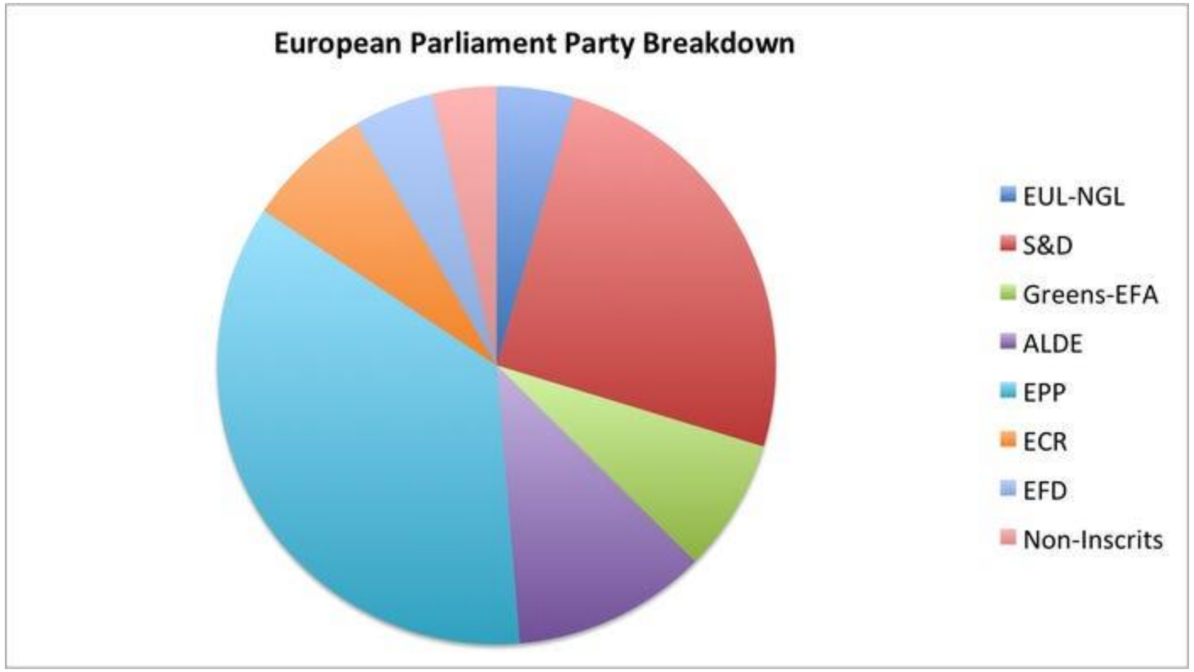
Same Information from Bar Plot



https://www.businessinsider.com/pie-charts-are-the-worst-2013-6

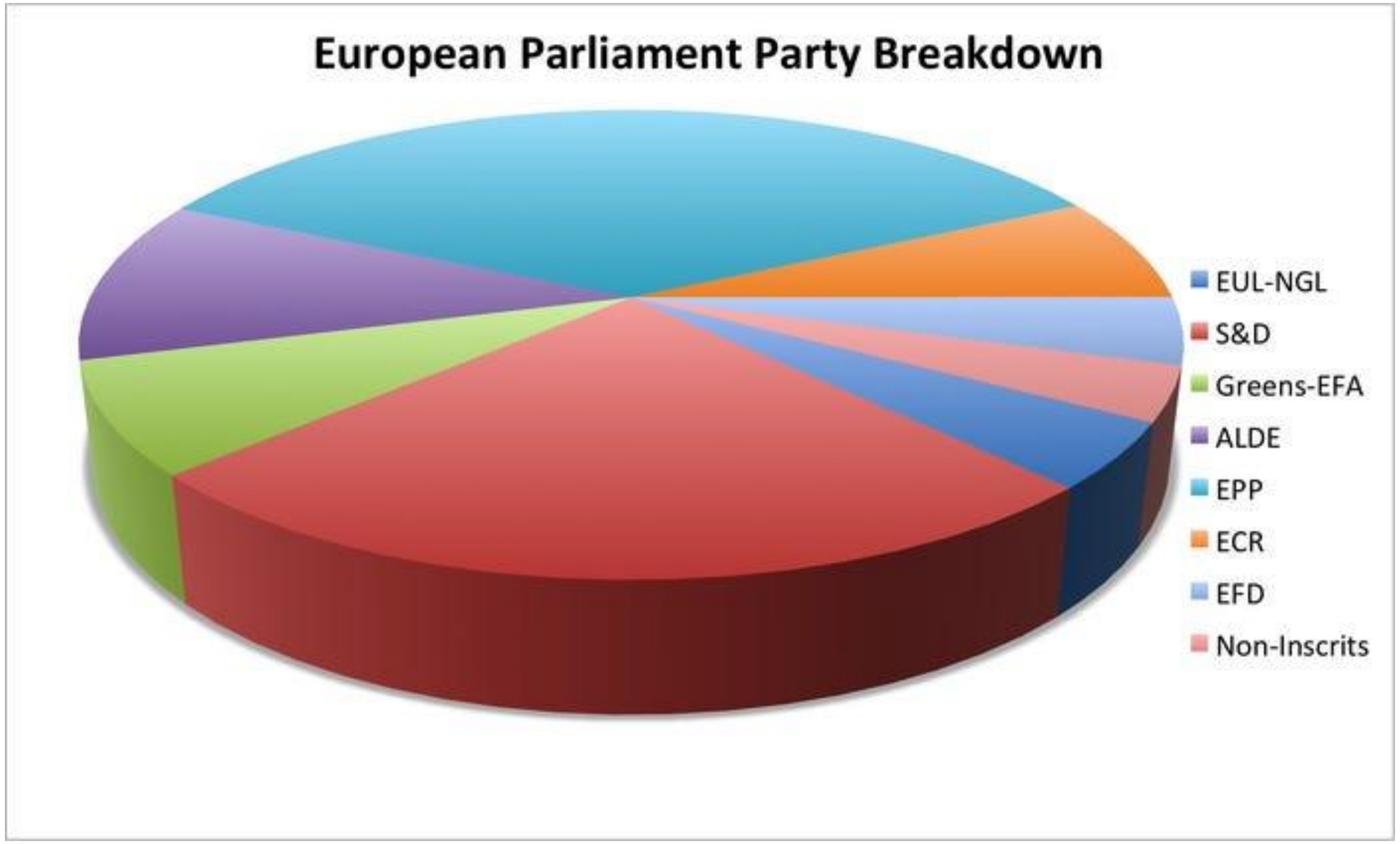
Human Eyes are not Good at Evaluating Sizes





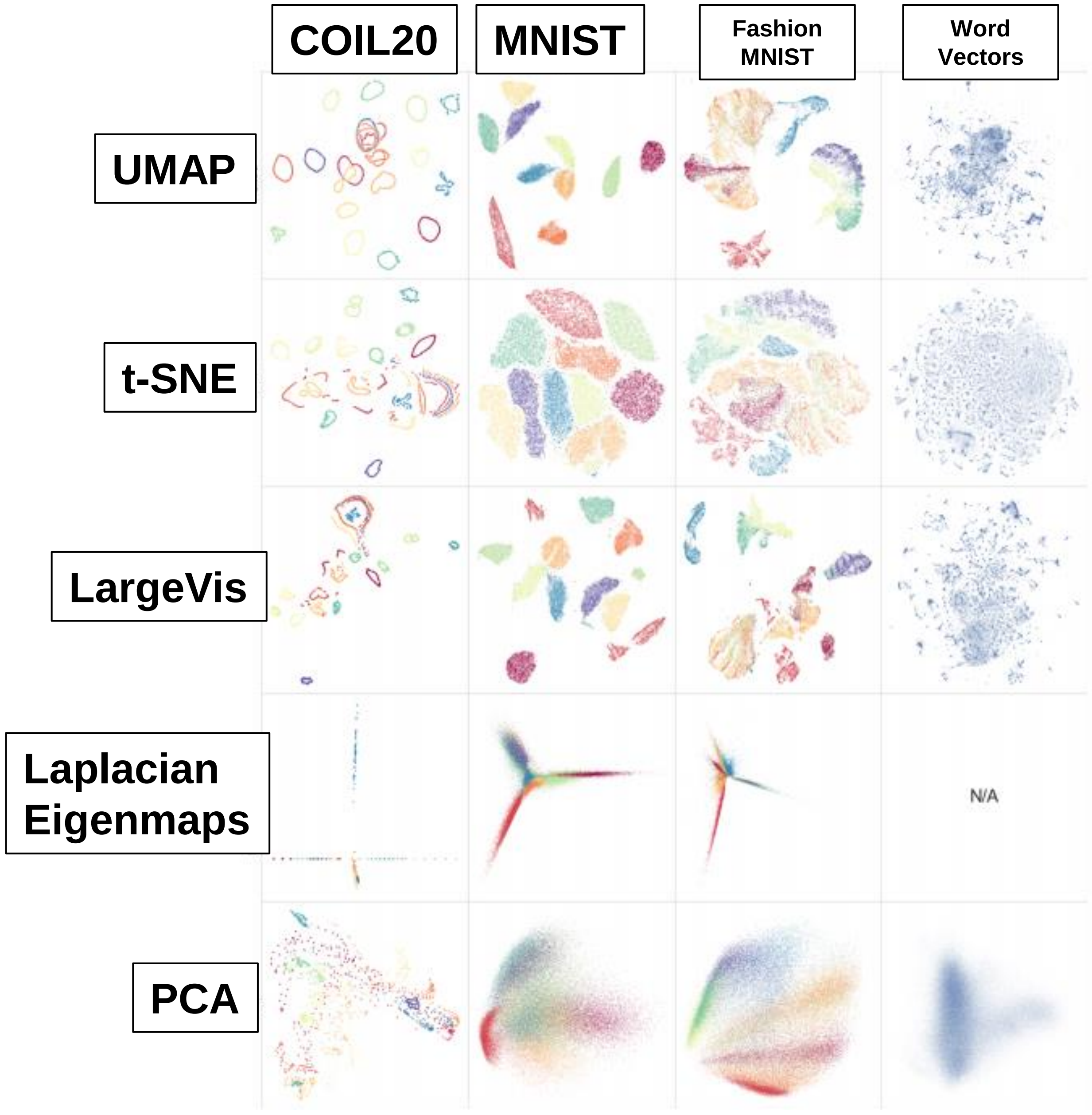
https://www.businessinsider.com/pie-charts-are-the-worst-2013-6

3d Plot even Worse



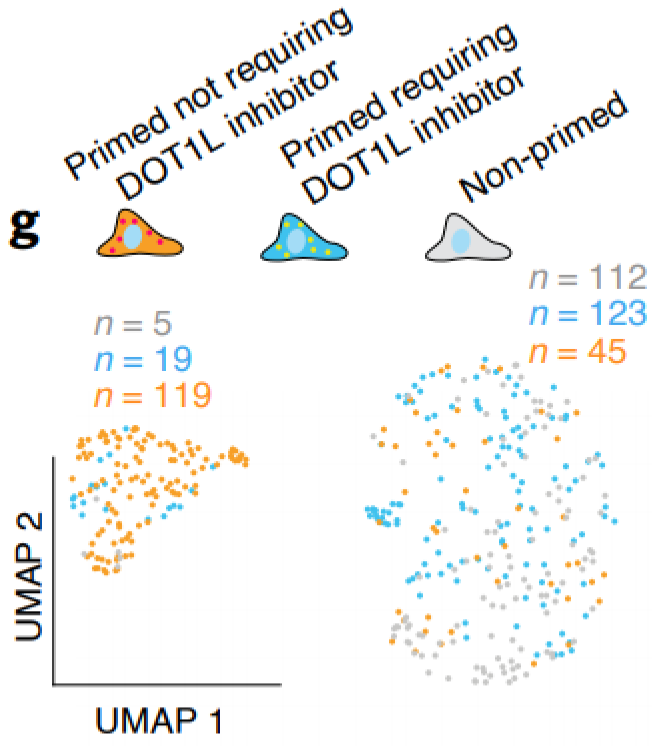
https://www.businessinsider.com/pie-charts-are-the-worst-2013-6

Effective 2D visualization with t-SNE and UMAP



When visualizing multidimensional data in 2D, t-SNE and UMAP works well

Ex.



Journal of Machine Learning Research 9 (2008) 2579-2605 Submitted 5/08; Revised 9/08; Published 11/08

Visualizing Data using t-SNE

Laurens van der Maaten LVDMAATEN@GMAIL.COM
TiCC
Tilburg University
P.O. Box 90153, 5000 LE Tilburg, The Netherlands

Geoffrey Hinton HINTON@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
6 King's College Road, M5S 3G4 Toronto, ON, Canada

Editor: Yoshua Bengio

Abstract

We present a new technique called "t-SNE" that visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map. The technique is a variation of Stochastic Neighbor Embedding (Hinton and Roweis, 2002) that is much easier to optimize, and produces significantly better visualizations by reducing the tendency to crowd points together in the center of the map. t-SNE is better than existing techniques at creating a single map that reveals structure at many different scales. This is particularly important for high-dimensional data that lie on several different, but related, low-dimensional manifolds, such as images of objects from multiple classes seen from multiple viewpoints. For visualizing the structure of very large data sets, we show how t-SNE can use random walks on neighborhood graphs to allow the implicit structure of all of the data to influence the way in which a subset of the data is displayed. We illustrate the performance of t-SNE on a wide variety of data sets and compare it with many other non-parametric visualization techniques, including Sammon mapping, Isomap, and Locally Linear Embedding. The visualizations produced by t-SNE are significantly better than those produced by the other techniques on almost all of the data sets.

Thank you! ☺