

DataCamp EDA in R

DataCamp EDA in R

Table of Contents

- [README](#)
- [Emacs setup \(optional\)](#)
- [Exploring categorical data](#)

README

- This is an example of how I work through a [DataCamp lesson](#) - no matter what the subject. It's a little more work when there are videos involved. Time to completion about 2 hours.
- I am pretty sure that you won't really understand what's going on without doing it at this level of granularity and with the original data set.
- This is also a way of keeping the DataCamp lessons interesting because otherwise I'd sometimes get very bored with the pace and/or the subject matter. Gotta keep things buzzing!
- As a classroom assignment, this file has been sanitized for use as a practice exercise. The complete file is available as a PDF in the `ds205/pdf/` directory in GitHub.

Emacs setup (optional)

Hide emphatic characters like ~, *

To **not** see the emphatic characters like ~ or * or / in the Org file text, run the following code chunk (or put the code in your `/.emacs` file): if successful, you should see "t" in the minibuffer.

```
(setq-default org-hide-emphasis-markers t)
```

This will only work for new buffers. If you don't put it in your `/.emacs` file, the command will only work for the current Emacs session.

Close and reopen this file to see an effect.

Change your theme

- In Emacs, type `M-x custom-themes`
- In the buffer that appears, select Leuven
- Select Apply and Save Setting
- This will work immediately

Exploring categorical data

Import and view the data

- [X] Download the raw data `Comics` data [from DataCamp](#) to your PC into a directory `./data/`

- [X]

Read the data set into an R data frame named `comics` and check the structure of the data frame.

```
comics <- read.csv(file="data/comics.csv", header=TRUE)
str(comics)
```

```
'data.frame': 23272 obs. of 11 variables:
 $ name      : chr "Spider-Man (Peter Parker)" "Captain America (Steven Rogers)" "Wolverine (James \\"Logan\\" Howlett)" ...
 $ id        : chr "Secret" "Public" "Public" "Public" ...
 $ align     : chr "Good" "Good" "Neutral" "Good" ...
 $ eye       : chr "Hazel Eyes" "Blue Eyes" "Blue Eyes" "Blue Eyes" ...
 $ hair      : chr "Brown Hair" "White Hair" "Black Hair" "Black Hair" ...
 $ gender    : chr "Male" "Male" "Male" "Male" ...
 $ gsm       : chr NA NA NA NA ...
 $ alive     : chr "Living Characters" "Living Characters" "Living Characters" "Living Characters" ...
 $ appearances : int 4043 3360 3061 2961 2258 2255 2072 2017 1955 1934 ...
 $ first_appear: chr "Aug-62" "Mar-41" "Oct-74" "Mar-63" ...
 $ publisher  : chr "marvel" "marvel" "marvel" "marvel" ...
```

- []

Print the first 10 rows of the dataset.

```
head(comics, 10)
```

	name	id	align	eye	hair	
1	Spider-Man (Peter Parker)	Secret	Good	Hazel Eyes	Brown Hair	
2	Captain America (Steven Rogers)	Public	Good	Blue Eyes	White Hair	
3	Wolverine (James \\"Logan\\" Howlett)	Public	Neutral	Blue Eyes	Black Hair	
4	Iron Man (Anthony \\"Tony\\" Stark)	Public	Good	Blue Eyes	Black Hair	
5	Thor (Thor Odinson)	No Dual	Good	Blue Eyes	Blond Hair	
6	Benjamin Grimm (Earth-616)	Public	Good	Blue Eyes	No Hair	
7	Reed Richards (Earth-616)	Public	Good	Brown Eyes	Brown Hair	
8	Hulk (Robert Bruce Banner)	Public	Good	Brown Eyes	Brown Hair	
9	Scott Summers (Earth-616)	Public	Neutral	Brown Eyes	Brown Hair	
10	Jonathan Storm (Earth-616)	Public	Good	Blue Eyes	Blond Hair	
	gender	gsm	alive	appearances	first_appear	publisher
1	Male <					
	Living Characters	4043	Aug-62			marvel
2	Male <					
	Living Characters	3360	Mar-41			marvel
3	Male <					
	Living Characters	3061	Oct-74			marvel
4	Male <					
	Living Characters	2961	Mar-63			marvel
5	Male <					
	Living Characters	2258	Nov-50			marvel
6	Male <					
	Living Characters	2255	Nov-61			marvel
7	Male <					
	Living Characters	2072	Nov-61			marvel
8	Male <					
	Living Characters	2017	May-62			marvel
9	Male <					
	Living Characters	1955	Sep-63			marvel

```
10  Male <
Living Characters      1934      Nov-61      marvel
```

- []

What happens when you only enter the name of the data frame *in the console*? What is the default maximum value of rows displayed? Enter the command below to print this value.

```
getOption("max.print")
```

```
[1] 99999
```

- []

Unlike the data shown in the video, the data frame you are currently working with is not a `data.frame` but a "tibble".

More importantly, the tibble contains `factor` variables where the `comics` data frame has character variables.

Re-import `comics.csv` into `comics` so that the characters in the data frame become `factor` variables. *Tip: check the `read.csv` help.*

```
comics <- read.csv(file="data/comics.csv",
                    header=TRUE,
                    stringsAsFactors=TRUE )
str(comics)
```

```
'data.frame': 23272 obs. of 11 variables:
 $ name     : Factor w/ 23272 levels "'Spinner (Earth-616)",...: 19830 3335 22769 9647 2...
 $ id       : Factor w/ 4 levels "No Dual","Public",...: 3 2 2 2 1 2 2 2 2 2 ...
 $ align    : Factor w/ 4 levels "Bad","Good","Neutral",...: 2 2 3 2 2 2 2 2 3 2 ...
 $ eye      : Factor w/ 26 levels "Amber Eyes","Auburn Hair",...: 11 5 5 5 5 5 6 6 6 5 ...
 $ hair     : Factor w/ 28 levels "Auburn Hair",...: 7 27 3 3 4 14 7 7 7 4 ...
 $ gender   : Factor w/ 3 levels "Female","Male",...: 2 2 2 2 2 2 2 2 2 2 ...
 $ gsm      : Factor w/ 6 levels "Bisexual Characters",...: NA NA NA NA NA NA NA NA ...
 $ alive    : Factor w/ 2 levels "Deceased Characters",...: 2 2 2 2 2 2 2 2 2 2 ...
 $ appearances : int 4043 3360 3061 2961 2258 2255 2072 2017 1955 1934 ...
 $ first_appear: Factor w/ 1606 levels "1935, October",...: 874 1278 1513 1296 1424 1432 14...
 $ publisher : Factor w/ 2 levels "dc","marvel": 2 2 2 2 2 2 2 2 2 2 ...
```

Working with factors - levels, NAs, contingency

- []

Let's look at factor levels.

- Store the variables `align` and `id` of the data frame in vectors of that name.
- Print the `levels` of the `align` and `id` columns that indicate how good and hidden a superhero is.
- Can you do this in 3 lines (instead of 4) using vectorization?

```
align <- comics$align
id <- comics$id
levels(c(align,id)) ## using vectorization
```

```
[1] "Bad"           "Good"          "Neutral"
[4] "Reformed Criminals" "No Dual"       "Public"
[7] "Secret"        "Unknown"
```

- []

In the video, you're told that `levels` has ignored `NA` values.

How many `NA` values does `align` and `id` have?

Tip: use the `is.na` function to find out

```
sum(is.na(align))
sum(is.na(id))
```

```
[1] 3413
[1] 5783
```

```
sum(is.na(c(align,id)))
sum(is.na(c(comics$align,comics$id)))
c(align,id) |> is.na() |> sum()
sum(is.na(align) + is.na(id))
```

```
[1] 9196
[1] 9196
[1] 9196
[1] 9196
```

- []

Print the contingency table for `align` and `id`, which shows how these two categorical variables are connected.¹

```
tbl <- table(align, id)
tbl
```

		id			
		No Dual	Public	Secret	Unknown
align	Bad	474	2172	4493	7
	Good	647	2930	2475	0
Neutral	390	965	959	2	
	Reformed Criminals	0	1	1	0

- []

How many "good" superheroes are there, who are also "secret"? The answer, from the table, is 2475. Which command gets you this number, too?

```
names(table(align == "Good" & id == "Secret")) ## find the name  
table(align == "Good" & id == "Secret")[2] ## extract element
```

```
[1] "FALSE" "TRUE"  
TRUE  
2475
```

Good and Public:

```
table(align == "Good" & id == "Public")["TRUE"] ## extract element
```

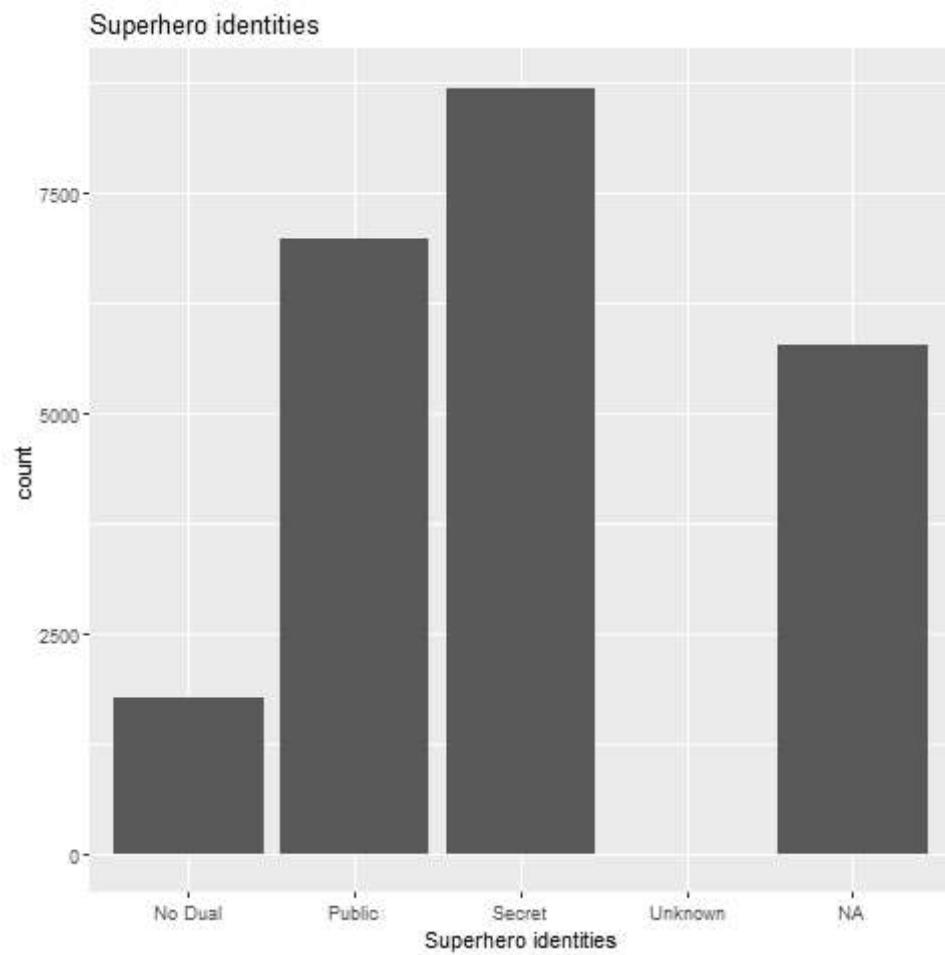
```
TRUE  
2930
```

Barplots with `comics` and `ggplot`

- [] Create a directory `./img/` for the plots.
- []

Make a barplot of the superhero identities (`x = comics$id`).

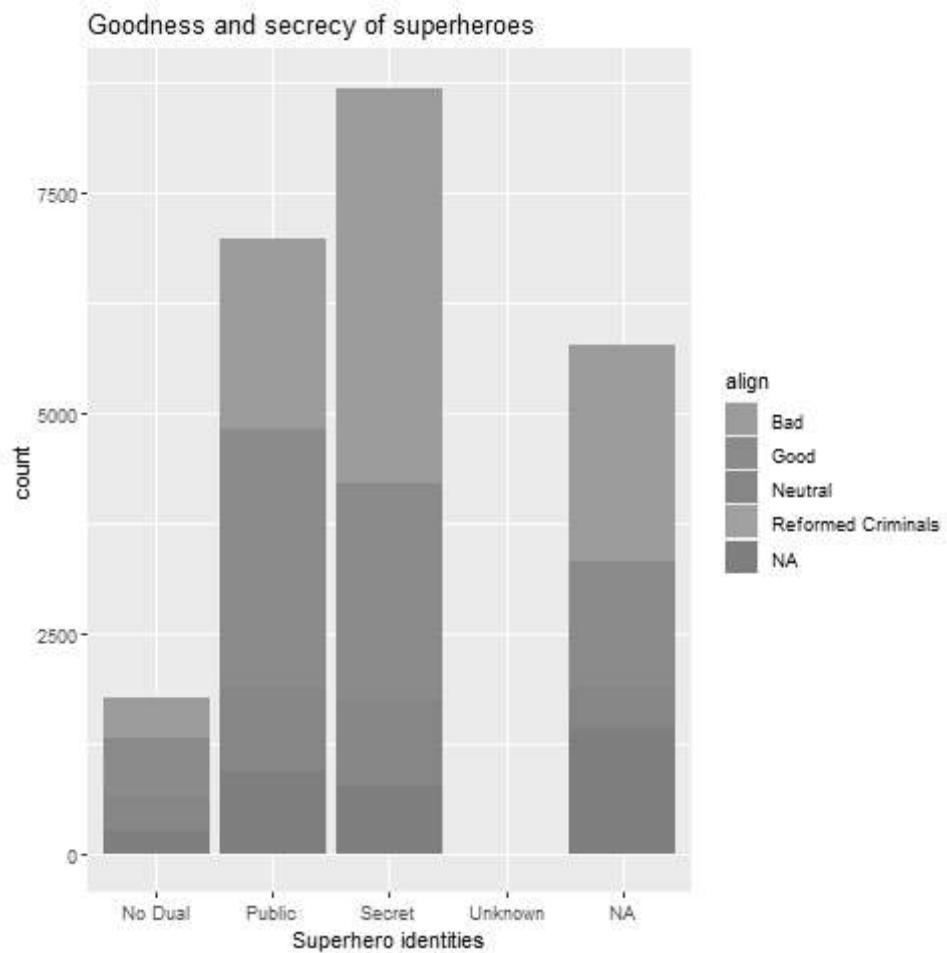
```
library(ggplot2)  
ggplot(data = comics, aes(x = id)) +  
  geom_bar() +  
  xlab("Superhero identities") +  
  ggtitle("Superhero identities")
```



- []

Make a *stacked* barplot that shows superhero identities (`x = comics$id`) and superhero goodness (`fill = comics$align`).

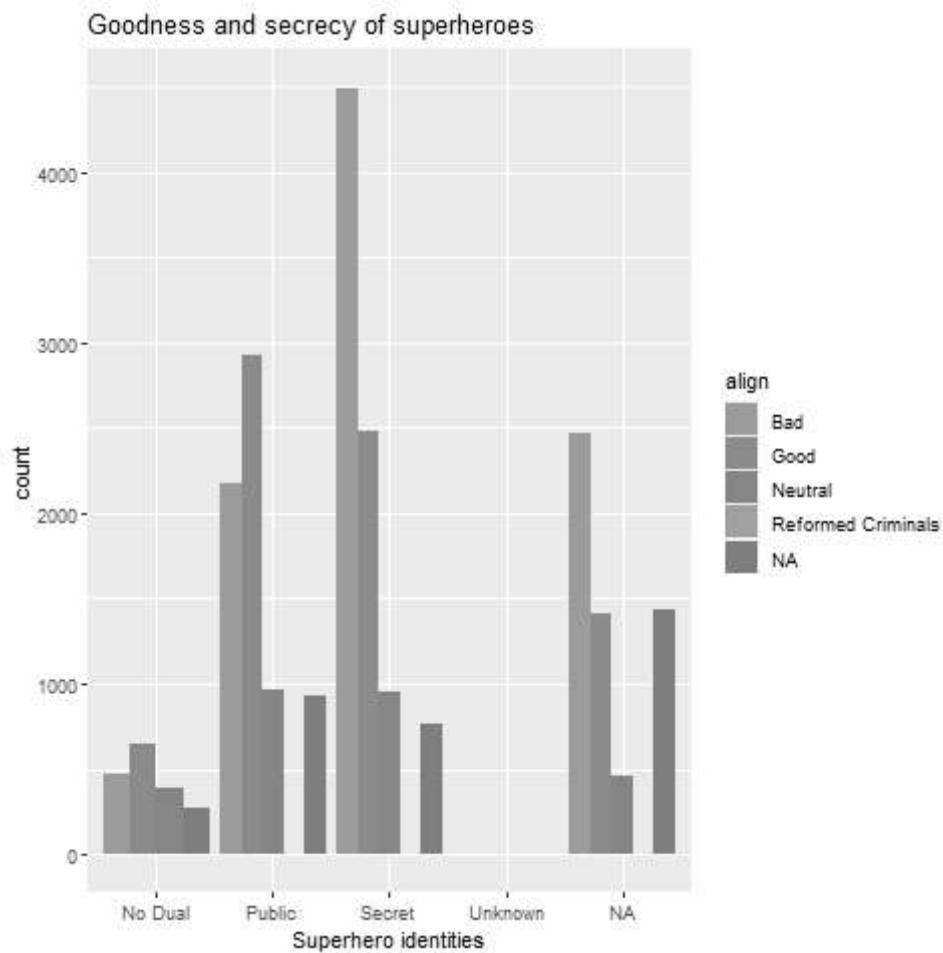
```
library(ggplot2)
ggplot(data = comics, aes(x = id, fill = align)) +
  geom_bar() +
  xlab("Superhero identities") +
  ggtitle("Goodness and secrecy of superheroes")
```



- []

Make a *side-by-side* barplot that shows superhero identities (`x = comics$id`) and superhero goodness (`fill = comics$align`).

```
library(ggplot2)
ggplot(data = comics, aes(x = id, fill = align)) +
  geom_bar(position = "dodge") +
  xlab("Superhero identities") +
  ggtitle("Goodness and secrecy of superheroes")
```



- [] Go back over the last plots and add text:
 - x labels
 - y labels
 - plot titles

Exploring proportions

- []

Simplify the display format using the `options` command. Set the `scipen` attribute to 999 and the `digits` to 3. Check the doc if you're curious what `scipen` means.

```
options(scipen=999, digits=3)
```

- []

The `options` settings are only valid for the current R session. How can you make them permanent?

ANSWER: by adding the command to the `.Rprofile` file in `~/`

- [] Store the contingency table for `comics$align` and `comics$id` in the variable `tab_cnt`.

- Find out what kind of R object `tab_cnt` is. Use `class` and the `is.X` family of functions (e.g. `is.data.frame`).

```
tab_cnt <- table(comics$align, comics$id)
class(tab_cnt)
is.matrix(tab_cnt)
```

```
[1] "table"
[1] TRUE
```

- []

Check that the proportions (counts as parts of a whole, computed with `prop.table`) of all entries in the table `tab_cnt` add up to 1.

```
pt <- prop.table(tab_cnt)
sum(pt)
```

```
[1] 1
```

- []

What is the default `margin` of `prop.table`? Can you define `margin`?

ANSWER: `margin` is a vector giving the margins to split a data structure by. The default is `margin=NULL`. For a matrix or table, `margin=1` indicates rows, `margin=2` indicates columns, `c(1,2)` indicates rows and columns.

- []

Print the table of proportions with the default `margin`.

```
prop.table(tab_cnt, margin=NULL)
```

	No	Dual	Public	Secret	Unknown
Bad	0.0305491	0.1399845	0.2895721	0.0004511	
Good	0.0416989	0.1888373	0.1595128	0.0000000	
Neutral	0.0251353	0.0621939	0.0618072	0.0001289	
Reformed	0.0000000	0.0000644	0.0000644	0.0000000	
Criminals					

- []

Condition on a row: print only the proportions of Bad superheroes.

```
row_props <- prop.table(tab_cnt, 1)
paste("baddies:")
row_props[1,]
```

```
[1] "baddies:"
No Dual Public Secret Unknown
0.06633 0.30395 0.62874 0.00098
```

- []

Condition on a column: print only the proportions of `Public` superheroes.

```
col_props <- prop.table(tab_cnt, 2)
paste("public:")
col_props[,2]
```

	Bad	Good	Neutral	Reformed	Criminals
[1] "public:"	0.357943	0.482861	0.159031	0.000165	

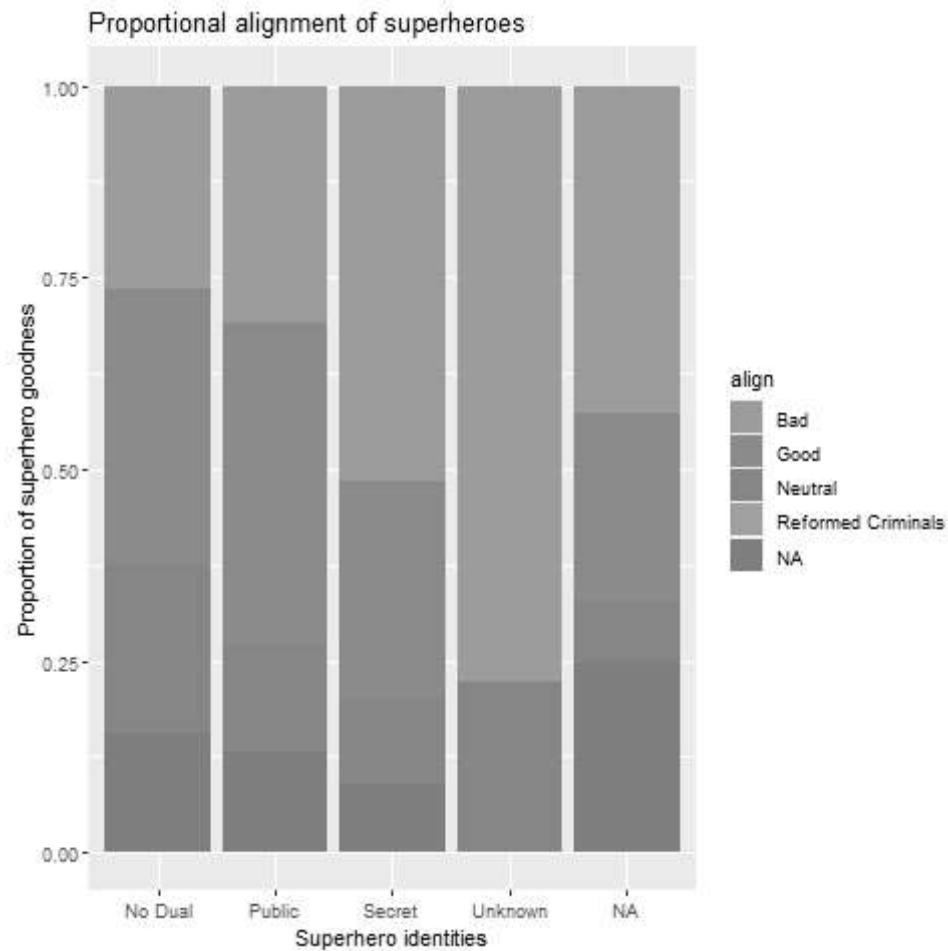
- []

Make a *stacked* barplot that shows the *proportion* (not the *count*) of superhero identities (`comics$id`) and superhero goodness (`comics$align`), conditioned on identities.

Don't forget to label and title the graph appropriately.

Tip: to do this in `ggplot2` remember that you need to specify the `position=fill` in the `geom_bar` geometry.

```
library(ggplot2)
ggplot(data = comics, aes(x = id, fill = align)) +
  geom_bar(position = "fill") +
  xlab("Superhero identities") +
  ylab("Proportion of superhero goodness") +
  ggtitle("Proportional alignment of superheroes")
```



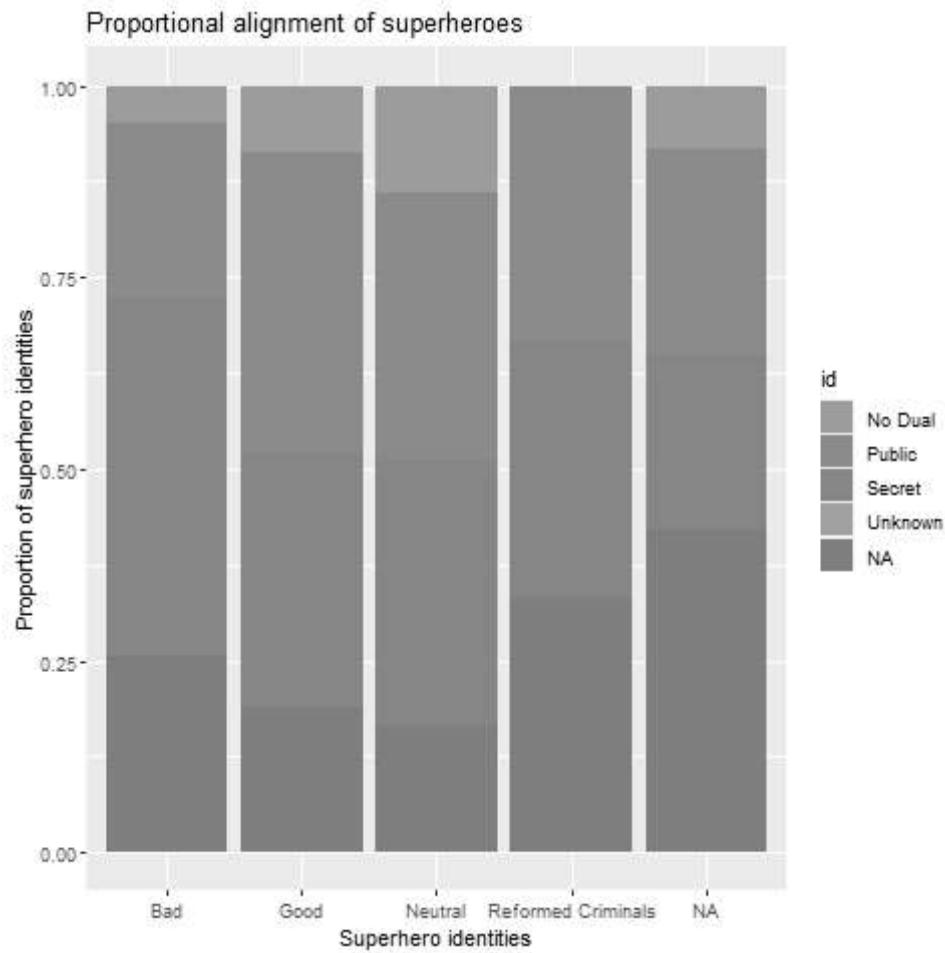
- []

Make a *stacked* barplot that shows the *proportion* (not the *count*) of superhero identities (`comics$id`) and superhero goodness (`comics$align`), conditioned on goodness.

Don't forget to label and title the graph appropriately.

Tip: to do this in `ggplot2` remember that you need to specify the `position=fill` in the `geom_bar` geometry.

```
library(ggplot2)
ggplot(data = comics, aes(x = align, fill = id)) +
  geom_bar(position = "fill") +
  xlab("Superhero identities") +
  ylab("Proportion of superhero identities") +
  ggtitle("Proportional alignment of superheroes")
```



- [] Create a contingency table for the variables `id` and `alive` showing how these are related.
 - Store the result in `tab` and print it with the names `[id]` and `[alive]` for rows and columns, respectively.

```
id <- comics$id ## superhero identity
al <- comics$alive ## superhero living or deceased
tab <- table("[id]"=id,"[alive]"=al)
tab
```

[id]	[alive]		Living Characters
	Deceased Characters	Living Characters	
No Dual	443	1345	
Public	1683	5309	
Secret	2249	6449	
Unknown	2	7	

- []

Add a column `Total` to the table with the row totals for each `id`.

Tip: to run sums over columns, you can use `rowSums`, to bind a column to a data frame or matrix/table, use `cbind`.

```
tab <- cbind(tab, "Total"=rowSums(tab))
tab
```

	Deceased Characters	Living Characters	Total
No Dual	443	1345	1788
Public	1683	5309	6992
Secret	2249	6449	8698
Unknown	2	7	9

- []

Print only the proportions of Secret superheroes that are alive.

Tip: condition on columns (margin=2).

```
prop.table(tab,margin=2)
paste("Secret & alive:")
prop.table(tab,margin=2)[["Secret","Living Characters"]]
prop.table(tab,margin=2)[3,2]
```

```
Deceased Characters Living Characters Total
No Dual          0.10211      0.102593 0.102247
Public           0.384510     0.404958 0.399840
Secret           0.513822     0.491915 0.497398
Unknown          0.000457     0.000534 0.000515
[1] "Secret & alive:"
[1] 0.492
[1] 0.492
```

Distribution of one variable

- []

Print the levels in `comics$align`, then reorder them and print them again. Reorder the levels so that the order is Bad, Neutral, Good.

```
levels(comics$align)
comics$align <- factor(comics$align,
                      levels=c("Bad","Neutral","Good"))
levels(comics$align)
```

```
[1] "Bad"           "Good"          "Neutral"
[4] "Reformed Criminals"
[1] "Bad"           "Neutral"       "Good"
```

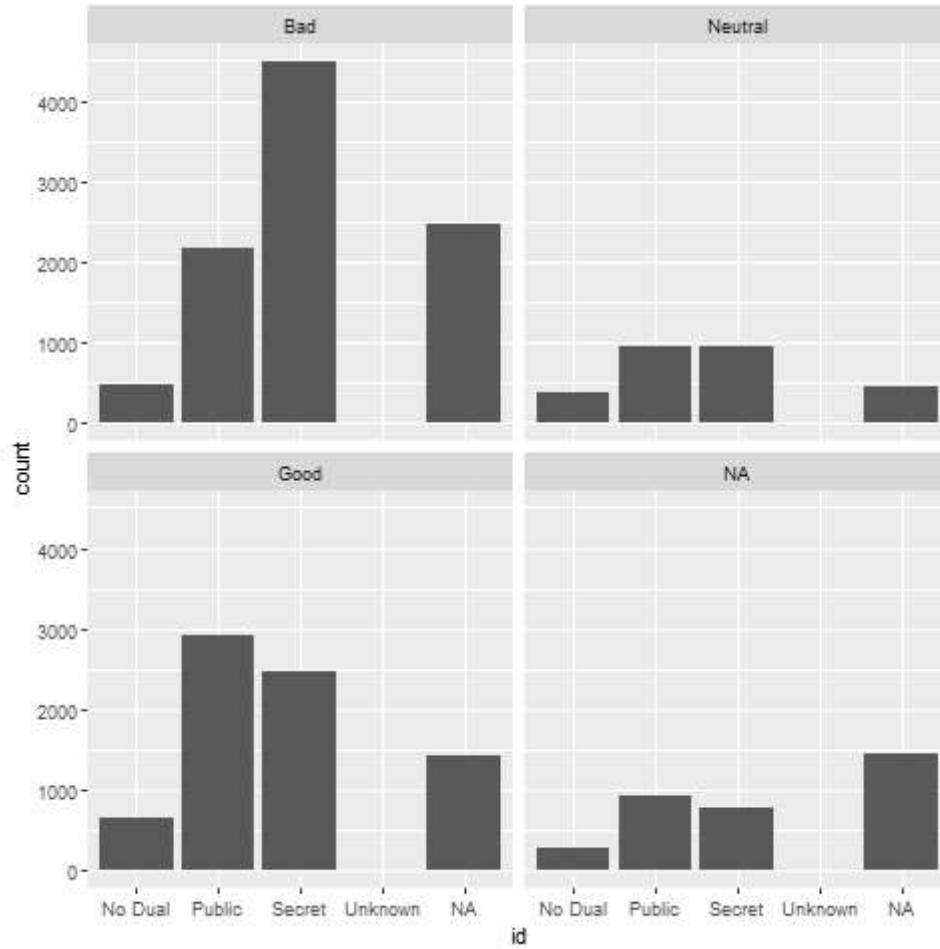
- []

Make a *faceted* bar chart of `comics$id` and `comics$align` (with the reordered, limited number of levels).

- Plot using only the `id` data
- Add a `facet_wrap` layer with `align`

Tip: the facet_wrap function uses the ~ operator where y~x.

```
ggplot(comics, aes(x=id)) +
  geom_bar() +
  facet_wrap(~ align)
```



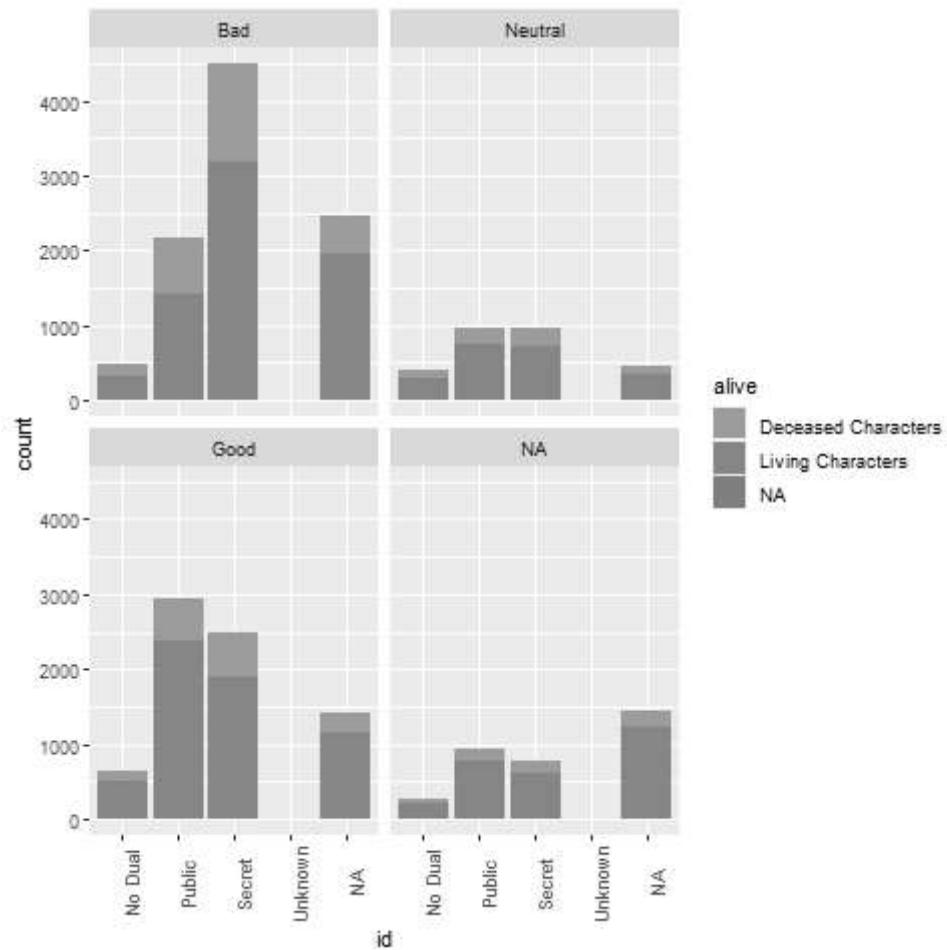
- []

Make a *faceted* bar chart of `comics$id` and `comics$align` (with the reordered, limited number of levels).

- Plot using only the `id` data
- Add `alive` to the `aes` arguments so that the bars are filled (and colored) according to its values
- Add a `facet_wrap` layer with `align`

Tip: the facet_wrap function uses the ~ operator where y~x.

```
ggplot(comics, aes(x=id, fill=alive)) +
  geom_bar() +
  facet_wrap(~ align) +
  theme(axis.text.x =
        element_text(angle = 90))
```



Footnotes:

¹ German lesson! The German word for contingency, "zusammenhängen", means "hang together".

Author: DataCamp / Andrew Bray & Lyon College / Marcus Birkenkrahe

Created: 2022-04-23 Sat 17:33