Introduction to R – L7 – Tables

Nice tables in R

> This lecture will allow us to explore the world of nice tables in R.

We will first produce a table, then output it in a "raw", pure text format, and then make it fancier in an html-compliant nice format. Note that most of the formatting we'll do here will **not** be available in **pdf** or **docx** formats. To format nice tables in these other formats, you have to use other packages such as `kableExtra` or `huxtable` that we will not directly discuss here.

We will deal with **three** types of tables:

- tables that are the results of computations you run. We will use `gt`
- summary tables of statistical models. We will use `modelsummary`
- summary tables detailing stats of whole datasets. We will use `modelsummary`

So first let's **install the packages** and then **call the libraries**

```
## gt, for "great table", from the tidyverse developers
#install.packages("gt")

## modelsummary, the best tool to table regressions and models
#install.packages("modelsummary")

library(gt)
library(tidyverse)
```

Generic tables – i.e. results of computations

We start by creating a simple table of summary statistics as we have done earlier in the course, using our usual `tidyverse` tools – `filter`, `mutate`, `group_by`, `summarize` …

Exercice 1: mpg

> Using the `mpg` built-in dataset, make a table of the mean miles per gallon when driving within a city (`cty`) and on a highway (`hwy`) for each class, in 1999 and 2008. First use our usual tidyverse tools to create a simple text table.

```
table_exo1 <- mpg %>%
  group_by(class, year) %>%
  summarise(mean_cty = mean(cty, na.rm = T),
            mean_hwy = mean(hwy, na.rm = T))
```

```
`summarise()` has grouped output by 'class'. You can override using the
`.groups` argument.
```

```
table_exo1
```

```
# A tibble: 14 × 4
# Groups:   class [7]
   class       year mean_cty mean_hwy
   <chr>      <int>    <dbl>    <dbl>
 1 2seater     1999     15.5     24.5
 2 2seater     2008     15.3     25
 3 compact     1999     19.8     27.9
 4 compact     2008     20.5     28.7
 5 midsize     1999     18.2     26.5
 6 midsize     2008     19.3     28.0
 7 minivan     1999     16.2     22.5
 8 minivan     2008     15.4     22.2
 9 pickup      1999     13       16.8
10 pickup      2008     13       16.9
11 subcompact  1999     21.6     29
12 subcompact  2008     18.9     27.1
13 suv         1999     13.4     17.6
14 suv         2008     13.6     18.6
```

The output is text-only if you run this in a console; an html widget if you run it in the chunk; and again an ugly text-based copy of what you'd see in the console if you `Render` your `.qmd` file.

Please check all this.

Also, look at the object in the `Environment` pane. The object is a data.frame as we know it so far – just raw data.

Now try to make it less ugly by using `gt` – first with a simple call to `gt()`.

```
table_exo1_gt <- table_exo1 %>%
  gt()
table_exo1_gt
```

	year	mean_cty	mean_hwy
2seater			
	1999	15.50000	24.50000
	2008	15.33333	25.00000
compact			
	1999	19.76000	27.92000
	2008	20.54545	28.72727
midsize			
	1999	18.15000	26.50000
	2008	19.33333	28.04762
minivan			
	1999	16.16667	22.50000
	2008	15.40000	22.20000
pickup			
	1999	13.00000	16.81250
	2008	13.00000	16.94118
subcompact			
	1999	21.57895	29.00000
	2008	18.93750	27.12500
suv			
	1999	13.37931	17.55172
	2008	13.60606	18.63636

This is already better, both when looked at in the chunk and when rendered (try).

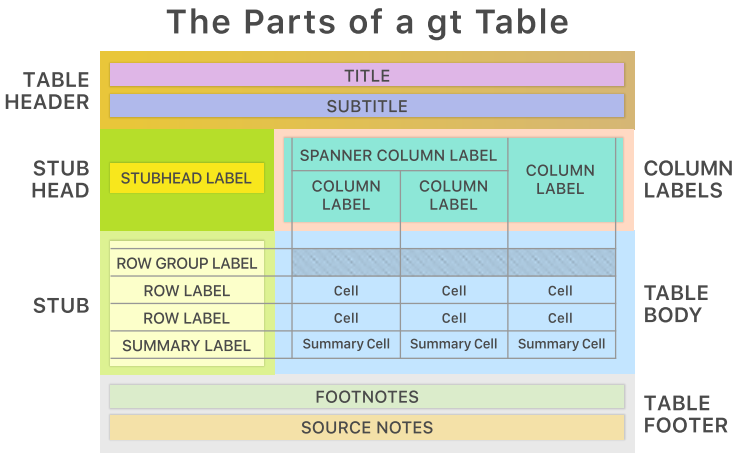
Note that `gt` automatically recognizes `groups` in your `data.frame`.

Look at the object in the `Environment` pane: it is a complex object (akin to the `ggplot` s we created along the course)

But what is `gt`? Let's dive deeper!

`gt` – `g`reat `t`ables

`gt` is a package to generate nice looking tables; it strives to be **as general as possible** and to allow customization of all elements of a table.



`gt` general structure and terminology

A `gt` table is created by simply calling `gt()` on a data.frame. So you can make all the changes / computations you want and then display those as a `gt` table by simply piping the object you created to `gt()`.

Exercise 2: `NYCflights` and `gt`

> Get our dear old `nycflights` dataset, and create a table of min, mean and max (on the columns) departure delays for each month (on the rows) for the 3 New York airports (on the columns). In other words, The table must look like this:

	JFK			LGA			EWR		
	min	mean	max	min	mean	max	min	mean	max
Jan									
Feb									
Mar									
...									

Then feed it to `gt()` to create a nice-looking HTML table. Check the result in pdf too. It doesn't matter right now if you don't get it perfectly right.

Important: remember to use `pivot_wider` to create the columns you need out of your originally `long` format dataset.

```
library(nycflights13)

# 1. generate the needed stats
exo2 <- flights %>%
  group_by(month, origin) %>%
  summarise(min_delay = min(dep_delay, na.rm = T),
            mean_delay = mean(dep_delay, na.rm = T),
            max_delay = max(dep_delay, na.rm = T))
```

```
`summarise()` has grouped output by 'month'. You can override using the
`.groups` argument.
```

```
exo2
```

```
# A tibble: 36 × 5
# Groups:   month [12]
   month origin min_delay mean_delay max_delay
   <int> <chr>      <dbl>      <dbl>     <dbl>
 1     1 EWR          -21      14.9       1126
 2     1 JFK          -17       8.62      1301
 3     1 LGA          -30       5.64       478
 4     2 EWR          -21      13.1        786
 5     2 JFK          -22      11.8        747
 6     2 LGA          -33       6.96       853
 7     3 EWR          -22      18.1        443
 8     3 JFK          -24      10.7        800
 9     3 LGA          -25      10.2        911
10     4 EWR          -21      17.4        545
# ℹ 26 more rows
```

```
# 2. realize it's in long format! use pivot_wider
exo2_wide <- exo2 %>%
  pivot_wider(names_from = origin, values_from = ends_with("delay"))
exo2_wide
```

```
# A tibble: 12 × 10
# Groups:   month [12]
   month min_delay_EWR min_delay_JFK min_delay_LGA mean_delay_EWR mean_delay_JFK
   <int>         <dbl>         <dbl>         <dbl>          <dbl>          <dbl>
 1     1           -21           -17           -30          14.9            8.62
 2     2           -21           -22           -33          13.1           11.8
 3     3           -22           -24           -25          18.1           10.7
 4     4           -21           -19           -21          17.4           12.2
 5     5           -20           -23           -24          15.4           12.5
 6     6           -19           -18           -21          22.5           20.5
 7     7           -18           -18           -22          22.0           23.8
 8     8           -17           -17           -26          13.5           12.9
 9     9           -23           -22           -24           7.29           6.64
10    10           -25           -18           -22           8.64           4.59
11    11           -22           -18           -32           6.72           4.68
12    12           -20           -43           -23          21.0           14.8
# ℹ 4 more variables: mean_delay_LGA <dbl>, max_delay_EWR <dbl>,
#   max_delay_JFK <dbl>, max_delay_LGA <dbl>
```

```
# 3. realize we have a problem as nested column titles are NOT a thing in R
#    but we'll deal with that later
#    let's just feed it to `gt` right now
exo2_wide %>%
  gt()
```

min_delay_EWR	min_delay_JFK	min_delay_LGA	mean_delay_EWR	mean_delay_JFK	mean_delay_LGA	max_d
1						
-21	-17	-30	14.905748	8.615826	5.641560	

min_delay_EWR	min_delay_JFK	min_delay_LGA	mean_delay_EWR	mean_delay_JFK	mean_delay_LGA	max_d
2						
-21	-22	-33	13.067263	11.791355	6.961582	
3						
-22	-24	-25	18.102457	10.721825	10.232041	
4						
-21	-19	-21	17.400058	12.249059	11.508915	
5						
-20	-23	-24	15.366141	12.519432	10.630508	
6						
-19	-18	-21	22.470810	20.499729	19.296698	
7						
-18	-18	-22	22.035112	23.769262	18.995163	
8						
-17	-17	-26	13.493714	12.914358	11.244042	
9						
-23	-22	-24	7.290954	6.635776	6.207439	
10						
-25	-18	-22	8.636436	4.592556	5.309137	
11						
-22	-18	-32	6.723769	4.678464	4.767144	
12						
-20	-43	-23	21.026575	14.788352	13.588830	

```
# 4. uh oh we've got a problem -- gt recognizes groups! that's bad (here)... so

exo2_gt <- exo2_wide %>%
  ungroup() %>%
  gt()
exo2_gt
```

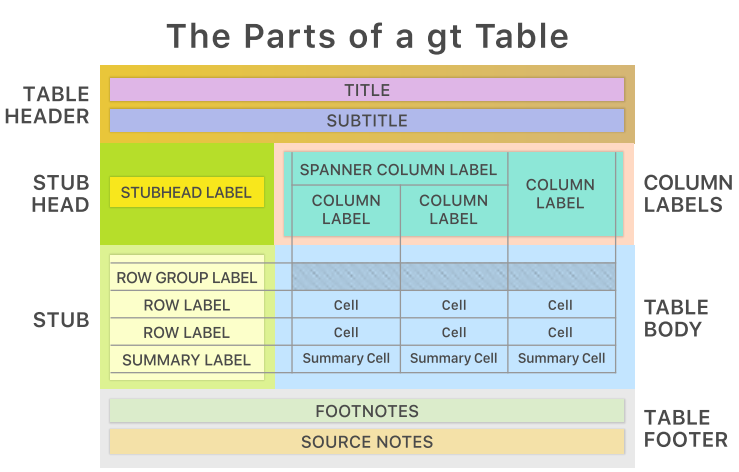
month	min_delay_EWR	min_delay_JFK	min_delay_LGA	mean_delay_EWR	mean_delay_JFK	mean_delay_LGA
1	-21	-17	-30	14.905748	8.615826	5.641560
2	-21	-22	-33	13.067263	11.791355	6.961582
3	-22	-24	-25	18.102457	10.721825	10.232041
4	-21	-19	-21	17.400058	12.249059	11.508915
5	-20	-23	-24	15.366141	12.519432	10.630508
6	-19	-18	-21	22.470810	20.499729	19.296698
7	-18	-18	-22	22.035112	23.769262	18.995163
8	-17	-17	-26	13.493714	12.914358	11.244042
9	-23	-22	-24	7.290954	6.635776	6.207439
10	-25	-18	-22	8.636436	4.592556	5.309137
11	-22	-18	-32	6.723769	4.678464	4.767144
12	-20	-43	-23	21.026575	14.788352	13.588830

This is ok, better than text, but it's still ugly and not what we want.

Luckily `gt` comes with a series of powerful functions to customize any aspect of a table!

Customizing a `gt` object

Once you have a `gt` object, you can use some simple families of functions to customize the table.



- `tab_header` `tab_caption` `tab_footnote` etc… deal with the different parts of the table
- to change the format of the data (eg round numbers, good formatting of dates, etc…) you can use the family of `fmt_...` functions. E.g. `fmt_date` to format dates, `fmt_number` to choose eg digit to round to, etc…
- `tab_spanner` creates the "spanners", i.e. the second- or third-level column labels.
- `tab_row_group` controls variables used for grouping (`gt` automatically uses the groups of your `df` if you don't tell it not to, as we have seen, but this can be changed).
- functions starting with `cols_...` allow you to change column behavior. For instance `cols_label` allows you to relabel columns (important: in standard R, data.frames cannot have two columns with the same name; but with `cols_label` this is no more a problem!), `cols_merge` merges data of two columns into one, Then you can change the alignment of the columns, their width, etc…
- You can add a summary across all rows with `summary_rows`
- `tab_style` allows you to change the style & appearance of any cell, row or column
- `tab_options` is similar to `theme` for `ggplot`, as it allows you to modify the look of single elements of a table.

Exercices 3: let's format nicely the table from Exo 1

> Let's call back the `gt` table we creatd in exercise 1. It has several problems: it's too long, the number of digits shown is not optimal; it has groups and we might not want that; the names of the columns are ugly, etc…

```
table_exo1_gt
```

year	mean_cty	mean_hwy
2seater		
1999	15.50000	24.50000
2008	15.33333	25.00000
compact		
1999	19.76000	27.92000
2008	20.54545	28.72727
midsize		
1999	18.15000	26.50000
2008	19.33333	28.04762
minivan		
1999	16.16667	22.50000
2008	15.40000	22.20000
pickup		
1999	13.00000	16.81250
2008	13.00000	16.94118
subcompact		
1999	21.57895	29.00000
2008	18.93750	27.12500
suv		

year	mean_cty	mean_hwy
1999	13.37931	17.55172
2008	13.60606	18.63636

Let's try to make it look better using the extra tools of `gt`. Let's:

- add a title and subtitle
- add a note telling the reader the source of the data
- format numbers with just 2 decimal places, but not for the "year" variable (that would be dumb)
- add a spanner on top of the cty and hwy columns
- rename those columns to some more decent names

```
table_exo1_gt %>%
  # header: title and subtitle
  tab_header(title = "Mean miles per gallon in city and highway",
             subtitle = "by class of car and year of immatriculation") %>%
  # source of the table
  tab_source_note("Source: {mpg} built-in dataset, R") %>%
  # formatting: 2 digits (but for year)
  fmt_number(columns = -year, decimals = 2) %>%
  # spanner
  tab_spanner(label = "Mean miles per gallon", columns = c(mean_cty, mean_hwy)) %>%
  # nice names
  cols_label(mean_cty = "Within cities", mean_hwy = "On highways")
```

Mean miles per gallon in city and highway

by class of car and year of immatriculation

	Mean miles per gallon	
year	Within cities	On highways
2seater		
1999	15.50	24.50
2008	15.33	25.00
compact		
1999	19.76	27.92
2008	20.55	28.73
midsize		
1999	18.15	26.50
2008	19.33	28.05
minivan		
1999	16.17	22.50
2008	15.40	22.20
pickup		
1999	13.00	16.81
2008	13.00	16.94
subcompact		
1999	21.58	29.00
2008	18.94	27.12
suv		
1999	13.38	17.55
2008	13.61	18.64

Source: {mpg} built-in dataset, R

This is already better, but it is also the chance to revise the good old `pivot_wider` and `pivot_longer` functions. Can we make the table wider, with year on the columns rather than rows? Let's restart from the raw object and make it wider, first

```
wider_exo1 <- table_exo1 %>%
  pivot_wider(names_from = year, values_from = c(mean_cty, mean_hwy)) %>%
  ungroup()
wider_exo1
```

```
# A tibble: 7 × 5
  class      mean_cty_1999 mean_cty_2008 mean_hwy_1999 mean_hwy_2008
  <chr>              <dbl>         <dbl>         <dbl>         <dbl>
1 2seater             15.5          15.3          24.5          25
2 compact             19.8          20.5          27.9          28.7
3 midsize             18.2          19.3          26.5          28.0
4 minivan             16.2          15.4          22.5          22.2
5 pickup              13            13            16.8          16.9
6 subcompact          21.6          18.9          29            27.1
7 suv                 13.4          13.6          17.6          18.6
```

Now let's make it into a `gt` object and then make it nicer using all the tricks above. In particular we want to:

- add a spanner for 1999 and one for 2008
- rename nicely cty and hwy
- 2 decimal places
- using `tab_style`, format the first column as bold

```
wider_exo1_cool <- wider_exo1 %>%
  gt() %>%
  # spanners
  tab_spanner(label = "Miles per gallon, 1999", columns = contains("1999")) %>%
  tab_spanner(label = "Miles per gallon, 2008", columns = contains("2008")) %>%
  # labels -- note that twice the same label is alright!
  cols_label(class = " ", mean_cty_1999 = "City",
             mean_hwy_1999 = "Highway", mean_cty_2008 = "City",
             mean_hwy_2008 = "Highway") %>%
  fmt_number(decimals = 2) %>%
  tab_style(style = list(cell_text(weight = "bold", transform = "capitalize")),
            locations = cells_body(columns = class))
wider_exo1_cool
```

	Miles per gallon, 1999		Miles per gallon, 2008	
	City	Highway	City	Highway
2seater	15.50	24.50	15.33	25.00
Compact	19.76	27.92	20.55	28.73
Midsize	18.15	26.50	19.33	28.05
Minivan	16.17	22.50	15.40	22.20
Pickup	13.00	16.81	13.00	16.94
Subcompact	21.58	29.00	18.94	27.12
Suv	13.38	17.55	13.61	18.64

Exercice 4: let's format nicely the table from Exo 2

Now let's go back to exercise 2. Remember what we wanted? This

	JFK			LGA			EWR		
	min	mean	max	min	mean	max	min	mean	max
Jan									
Feb									
Mar									
...									

Also, remember what we've got? this

```
exo2_gt <- exo2_wide %>% ungroup %>% gt()
exo2_gt
```

month	min_delay_EWR	min_delay_JFK	min_delay_LGA	mean_delay_EWR	mean_delay_JFK	mean_delay_LGA
1	-21	-17	-30	14.905748	8.615826	5.641560
2	-21	-22	-33	13.067263	11.791355	6.961582
3	-22	-24	-25	18.102457	10.721825	10.232041
4	-21	-19	-21	17.400058	12.249059	11.508915
5	-20	-23	-24	15.366141	12.519432	10.630508

month	min_delay_EWR	min_delay_JFK	min_delay_LGA	mean_delay_EWR	mean_delay_JFK	mean_delay_LGA
6	-19	-18	-21	22.470810	20.499729	19.296698
7	-18	-18	-22	22.035112	23.769262	18.995163
8	-17	-17	-26	13.493714	12.914358	11.244042
9	-23	-22	-24	7.290954	6.635776	6.207439
10	-25	-18	-22	8.636436	4.592556	5.309137
11	-22	-18	-32	6.723769	4.678464	4.767144
12	-20	-43	-23	21.026575	14.788352	13.588830

> Now you know how to turn that ugly table into the beautifully organized table we want! So as Nike once said, **just do it**.

```
nice_exo2 <- exo2_gt %>%
  # spanners
  tab_spanner(label = "Newark", columns = ends_with("EWR")) %>%
  tab_spanner(label = "JFK", columns = ends_with("JFK")) %>%
  tab_spanner(label = "La Guardia", columns = ends_with("LGA")) %>%
  # new names
  cols_label(month = "Month",
             min_delay_EWR = "min", mean_delay_EWR = "mean", max_delay_EWR = "max",
             min_delay_JFK = "min", mean_delay_JFK = "mean", max_delay_JFK = "max",
             min_delay_LGA = "min", mean_delay_LGA = "mean", max_delay_LGA = "max") %>%
  # 2 digits for mean
  fmt_number(columns = contains("mean"), decimals = 2) %>%
  # titles
  tab_header(title = "Delay at departure for the 3 NYC airports") %>%
  # month in bold, left aligned
  tab_style(style = cell_text(weight = "bold", align = "left"),
            locations = cells_body(columns = "month")) %>%
  # title centered and bold
  tab_options(heading.align = "center", heading.title.font.weight = "bold")
nice_exo2
```

Delay at departure for the 3 NYC airports									
	Newark			JFK			La Guardia		
Month	min	mean	max	min	mean	max	min	mean	max
1	-21	14.91	1126	-17	8.62	1301	-30	5.64	478
2	-21	13.07	786	-22	11.79	747	-33	6.96	853
3	-22	18.10	443	-24	10.72	800	-25	10.23	911
4	-21	17.40	545	-19	12.25	960	-21	11.51	812
5	-20	15.37	878	-23	12.52	853	-24	10.63	533
6	-19	22.47	502	-18	20.50	1137	-21	19.30	803
7	-18	22.04	653	-18	23.77	1005	-22	19.00	898
8	-17	13.49	424	-17	12.91	508	-26	11.24	520
9	-23	7.29	486	-22	6.64	1014	-24	6.21	696
10	-25	8.64	702	-18	4.59	342	-22	5.31	390
11	-22	6.72	798	-18	4.68	636	-32	4.77	413
12	-20	21.03	896	-43	14.79	825	-23	13.59	660

More customization of `gt`: `gtExtras`

`gt` can be also customized more easily using pre-packaged themes – akin to what we need in `ggplot` with `ggthemes` or with `hrbrthemes`. Themes are collected in the `gtExtras` library – that allows also some nice other features that we'll see below. Bu let's start by installing and importing the library

```
# install.packages("gtExtras")
library(gtExtras)
```

Now, we can apply some nice themes in one go. Some examples:

```
wider_exo1_cool %>%
  gt_theme_538()
```

```
Table has no assigned ID, using random ID 'yvzhohcxkv' to apply `gt::opt_css()`
Avoid this message by assigning an ID: `gt(id = '')` or `gt_theme_538(quiet = TRUE)`
```

	Miles per gallon, 1999		Miles per gallon, 2008	
	CITY	HIGHWAY	CITY	HIGHWAY
2seater	15.50	24.50	15.33	25.00
Compact	19.76	27.92	20.55	28.73
Midsize	18.15	26.50	19.33	28.05
Minivan	16.17	22.50	15.40	22.20
Pickup	13.00	16.81	13.00	16.94
Subcompact	21.58	29.00	18.94	27.12
Suv	13.38	17.55	13.61	18.64

```
wider_exo1_cool %>%
  gt_theme_nytimes()
```

	Miles per gallon, 1999		Miles per gallon, 2008	
	CITY	HIGHWAY	CITY	HIGHWAY
2seater	15.50	24.50	15.33	25.00
Compact	19.76	27.92	20.55	28.73
Midsize	18.15	26.50	19.33	28.05
Minivan	16.17	22.50	15.40	22.20
Pickup	13.00	16.81	13.00	16.94
Subcompact	21.58	29.00	18.94	27.12
Suv	13.38	17.55	13.61	18.64

```
wider_exo1_cool %>%
  gt_theme_guardian()
```

	Miles per gallon, 1999		Miles per gallon, 2008	
	City	Highway	City	Highway
2seater	15.50	24.50	15.33	25.00
Compact	19.76	27.92	20.55	28.73
Midsize	18.15	26.50	19.33	28.05
Minivan	16.17	22.50	15.40	22.20
Pickup	13.00	16.81	13.00	16.94
Subcompact	21.58	29.00	18.94	27.12
Suv	13.38	17.55	13.61	18.64

```
wider_exo1_cool %>%
  gt_theme_dot_matrix()
```

```
Table has no assigned ID, using random ID 'mlddkxpqsm' to apply `gt::opt_css()`
Avoid this message by assigning an ID: `gt(id = '')` or `gt_theme_538(quiet = TRUE)`
```

	miles per gallon, 1999		miles per gallon, 2008	
	city	highway	city	highway
2seater	15.50	24.50	15.33	25.00
Compact	19.76	27.92	20.55	28.73
Midsize	18.15	26.50	19.33	28.05
Minivan	16.17	22.50	15.40	22.20
Pickup	13.00	16.81	13.00	16.94
Subcompact	21.58	29.00	18.94	27.12
Suv	13.38	17.55	13.61	18.64

```
wider_exo1_cool %>%
  gt_theme_dark()
```

	Miles per gallon, 1999		Miles per gallon, 2008	
	CITY	HIGHWAY	CITY	HIGHWAY
2seater	15.50	24.50	15.33	25.00
Compact	19.76	27.92	20.55	28.73
Midsize	18.15	26.50	19.33	28.05
Minivan	16.17	22.50	15.40	22.20
Pickup	13.00	16.81	13.00	16.94
Subcompact	21.58	29.00	18.94	27.12
Suv	13.38	17.55	13.61	18.64

```
wider_exo1_cool %>%
  gt_theme_excel()
```

	Miles per gallon, 1999		Miles per gallon, 2008	
	City	Highway	City	Highway
2seater	15.50	24.50	15.33	25.00
Compact	19.76	27.92	20.55	28.73
Midsize	18.15	26.50	19.33	28.05
Minivan	16.17	22.50	15.40	22.20
Pickup	13.00	16.81	13.00	16.94
Subcompact	21.58	29.00	18.94	27.12
Suv	13.38	17.55	13.61	18.64

Still: they work only in HTML. As I said above, if you want nice pdf tables you have to use other packages as `huxtable` or `kableExtra`. In this course we focus on html output (slides, notebooks, etc) but pdf output is very relevant for publication. The good news is that all the different packages work in a very similar way, that is

- you work on the data to get to the point where yo have a raw table
- you pivot your data to get a table that has the needed shape
- you apply a table function (`gt`, but also `kable` or `hux`, depending on the package)
- you add layers as you did with `ggplot`, piping (`%>%`) each new layer on the table object

Exercise 5: check that the above themes work only in html by compiling to pdf

`gtExtras` also allow us to do some nice things like coloring cells according to their value, as in a heatmap. For instance, we can color the mean delay according to severity, like this:

```
nice_exo2 %>%
  gt_color_rows(contains("mean"))
```

```
Warning: Domain not specified, defaulting to observed range within each
specified column.
```

Delay at departure for the 3 NYC airports									
	Newark			JFK			La Guardia		
Month	min	mean	max	min	mean	max	min	mean	max
1	-21	14.91	1126	-17	8.62	1301	-30	5.64	478
2	-21	13.07	786	-22	11.79	747	-33	6.96	853
3	-22	18.10	443	-24	10.72	800	-25	10.23	911
4	-21	17.40	545	-19	12.25	960	-21	11.51	812
5	-20	15.37	878	-23	12.52	853	-24	10.63	533
6	-19	22.47	502	-18	20.50	1137	-21	19.30	803
7	-18	22.04	653	-18	23.77	1005	-22	19.00	898
8	-17	13.49	424	-17	12.91	508	-26	11.24	520
9	-23	7.29	486	-22	6.64	1014	-24	6.21	696
10	-25	8.64	702	-18	4.59	342	-22	5.31	390
11	-22	6.72	798	-18	4.68	636	-32	4.77	413
12	-20	21.03	896	-43	14.79	825	-23	13.59	660

We can also easily highlight a row we particularly care about, for instance in our exo1 dataset about cars, we can highlight SUVs, like this

```
wider_exo1_cool %>%
  gt_highlight_rows(rows = 7, fill = "chartreuse4", font_color = "white")
```

	Miles per gallon, 1999		Miles per gallon, 2008	
	City	Highway	City	Highway
2seater	15.50	24.50	15.33	25.00
Compact	19.76	27.92	20.55	28.73
Midsize	18.15	26.50	19.33	28.05
Minivan	16.17	22.50	15.40	22.20
Pickup	13.00	16.81	13.00	16.94
Subcompact	21.58	29.00	18.94	27.12
Suv	13.38	17.55	13.61	18.64

Finally, some more extra fancy stuff – you can insert miniplots right into tables. For instance, the distribution of a variable as a little distribution plot.

To do so, we need to pass the *whole data for each variable or group* as an extra column, and then call some specific functions. It's a bit advanced stuff, but it's not that difficult and the web is your friend!

Let's try to plot the distribution of departure time for each airport within a table:

```
flights %>%
  group_by(origin) %>%
  summarise(distro_dep_time = list(dep_time)) %>%
  gt() %>%
  gt_plt_dist(distro_dep_time)
```

origin	distro_dep_time
EWR	
JFK	
LGA	

Nice right?

Summaries of datasets with `modelsummary`

The package `modelsummary` allows you to do two things really well:

- to create summary tables of entire datasets, akin to what `summary()` or `skim()` do;
- to output nice tables of regression output

In both cases, the package does so with an approach that is in some ways the opposite of the one adopted by `gt`: where `gt` gives you fine control of everything adopting a *micro* approach and providing functions for each little element, `modelsummary` takes a *macro* approach and gives you big, customizable, single functions that do all the hard job fro you, with sensible defaults; but that you can tweak if you want to.

Summaries of datasets

These are known within `modelsummary` as "*data summaries*", and as such the function you need to call is `datasummary`.

Exercise 6: a simple data summary of the `mpg` dataset

> just create a nice-looking alternative to the `skim` command for our old friend the `mpg` dataset. so go from this:

```
library(skimr)
skim(mpg)
```

Data summary

Name	mpg
Number of rows	234
Number of columns	11
_____	
Column type frequency:	
character	6
numeric	5
_____	

Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
manufacturer	0	1	4	10	0	15	0
model	0	1	2	22	0	38	0
trans	0	1	8	10	0	10	0
drv	0	1	1	1	0	3	0
fl	0	1	1	1	0	5	0
class	0	1	3	10	0	7	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
displ	0	1	3.47	1.29	1.6	2.4	3.3	4.6	7	
year	0	1	2003.50	4.51	1999.0	1999.0	2003.5	2008.0	2008	
cyl	0	1	5.89	1.61	4.0	4.0	6.0	8.0	8	
cty	0	1	16.86	4.26	9.0	14.0	17.0	19.0	35	
hwy	0	1	23.44	5.95	12.0	18.0	24.0	27.0	44	

To something better looking and nicely formatted, using `datasummary_skim` like this:

```
library(modelsummary)
datasummary_skim(mpg)
```

	Unique	Missing Pct.	Mean	SD	Min	Median	Max	Histogram
displ	35	0	3.5	1.3	1.6	3.3	7.0	
year	2	0	2003.5	4.5	1999.0	2003.5	2008.0	
cyl	4	0	5.9	1.6	4.0	6.0	8.0	
cty	21	0	16.9	4.3	9.0	17.0	35.0	
hwy	27	0	23.4	6.0	12.0	24.0	44.0	
		N	%					
manufacturer	audi	18	7.7					
	chevrolet	19	8.1					
	dodge	37	15.8					
	ford	25	10.7					
	honda	9	3.8					
	hyundai	14	6.0					
	jeep	8	3.4					
	land rover	4	1.7					
	lincoln	3	1.3					
	mercury	4	1.7					
	nissan	13	5.6					
	pontiac	5	2.1					
	subaru	14	6.0					
	toyota	34	14.5					
	volkswagen	27	11.5					
model	4runner 4wd	6	2.6					
	a4	7	3.0					
	a4 quattro	8	3.4					
	a6 quattro	3	1.3					
	altima	6	2.6					
	c1500 suburban 2wd	5	2.1					
	camry	7	3.0					
	camry solara	7	3.0					

	Unique	Missing Pct.	Mean	SD	Min	Median	Max	Histogram
	caravan 2wd	11	4.7					
	civic	9	3.8					
	corolla	5	2.1					
	corvette	5	2.1					
	dakota pickup 4wd	9	3.8					
	durango 4wd	7	3.0					
	expedition 2wd	3	1.3					
	explorer 4wd	6	2.6					
	f150 pickup 4wd	7	3.0					
	forester awd	6	2.6					
	grand cherokee 4wd	8	3.4					
	grand prix	5	2.1					
	gti	5	2.1					
	impreza awd	8	3.4					
	jetta	9	3.8					
	k1500 tahoe 4wd	4	1.7					
	land cruiser wagon 4wd	2	0.9					
	malibu	5	2.1					
	maxima	3	1.3					
	mountaineer 4wd	4	1.7					
	mustang	9	3.8					
	navigator 2wd	3	1.3					
	new beetle	6	2.6					
	passat	7	3.0					
	pathfinder 4wd	4	1.7					
	ram 1500 pickup 4wd	10	4.3					
	range rover	4	1.7					
	sonata	7	3.0					
	tiburon	7	3.0					
	toyota tacoma 4wd	7	3.0					
trans	auto(av)	5	2.1					
	auto(l3)	2	0.9					
	auto(l4)	83	35.5					
	auto(l5)	39	16.7					
	auto(l6)	6	2.6					
	auto(s4)	3	1.3					
	auto(s5)	3	1.3					
	auto(s6)	16	6.8					
	manual(m5)	58	24.8					
	manual(m6)	19	8.1					
drv	4	103	44.0					
	f	106	45.3					
	r	25	10.7					
fl	c	1	0.4					
	d	5	2.1					
	e	8	3.4					

	Unique	Missing Pct.	Mean	SD	Min	Median	Max	Histogram
	p	52	22.2					
	r	168	71.8					
class	2seater	5	2.1					
	compact	47	20.1					
	midsize	41	17.5					
	minivan	11	4.7					
	pickup	33	14.1					
	subcompact	35	15.0					
	suv	62	26.5					

`datasummary` also allows you to create correlation matrices, using `datasummary_correlation` like this

```
datasummary_correlation(mpg)
```

	displ	year	cyl	cty	hwy
displ	1	.	.	.	.
year	.15	1	.	.	.
cyl	.93	.12	1	.	.
cty	-.80	-.04	-.81	1	.
hwy	-.77	.00	-.76	.96	1

and, handily, cross-tabulations, for instance to count how many observations you have for each carmaker and class. That is, you can have an ugly count like this, as we know:

```
mpg %>%
  group_by(manufacturer, class) %>%
  count()
```

```
# A tibble: 32 × 3
# Groups:   manufacturer, class [32]
   manufacturer class          n
   <chr>        <chr>      <int>
 1 audi         compact       15
 2 audi         midsize        3
 3 chevrolet    2seater        5
 4 chevrolet    midsize        5
 5 chevrolet    suv            9
 6 dodge        minivan       11
 7 dodge        pickup        19
 8 dodge        suv            7
 9 ford         pickup         7
10 ford         subcompact     9
# i 22 more rows
```

Or a nice looking table of counts including percentages using `datasummary_crosstab` like this:

```
datasummary_crosstab(manufacturer~class, data = mpg)
```

manufacturer		2seater	compact	midsize	minivan	pickup	subcompact	suv	All
audi	N	0	15	3	0	0	0	0	18
	% row	0.0	83.3	16.7	0.0	0.0	0.0	0.0	100.0
chevrolet	N	5	0	5	0	0	0	9	19
	% row	26.3	0.0	26.3	0.0	0.0	0.0	47.4	100.0
dodge	N	0	0	0	11	19	0	7	37
	% row	0.0	0.0	0.0	29.7	51.4	0.0	18.9	100.0
ford	N	0	0	0	0	7	9	9	25
	% row	0.0	0.0	0.0	0.0	28.0	36.0	36.0	100.0
honda	N	0	0	0	0	0	9	0	9
	% row	0.0	0.0	0.0	0.0	0.0	100.0	0.0	100.0
hyundai	N	0	0	7	0	0	7	0	14
	% row	0.0	0.0	50.0	0.0	0.0	50.0	0.0	100.0

manufacturer		2seater	compact	midsize	minivan	pickup	subcompact	suv	All
jeep	N	0	0	0	0	0	0	8	8
	% row	0.0	0.0	0.0	0.0	0.0	0.0	100.0	100.0
land rover	N	0	0	0	0	0	0	4	4
	% row	0.0	0.0	0.0	0.0	0.0	0.0	100.0	100.0
lincoln	N	0	0	0	0	0	0	3	3
	% row	0.0	0.0	0.0	0.0	0.0	0.0	100.0	100.0
mercury	N	0	0	0	0	0	0	4	4
	% row	0.0	0.0	0.0	0.0	0.0	0.0	100.0	100.0
nissan	N	0	2	7	0	0	0	4	13
	% row	0.0	15.4	53.8	0.0	0.0	0.0	30.8	100.0
pontiac	N	0	0	5	0	0	0	0	5
	% row	0.0	0.0	100.0	0.0	0.0	0.0	0.0	100.0
subaru	N	0	4	0	0	0	4	6	14
	% row	0.0	28.6	0.0	0.0	0.0	28.6	42.9	100.0
toyota	N	0	12	7	0	7	0	8	34
	% row	0.0	35.3	20.6	0.0	20.6	0.0	23.5	100.0
volkswagen	N	0	14	7	0	0	6	0	27
	% row	0.0	51.9	25.9	0.0	0.0	22.2	0.0	100.0
All	N	5	47	41	11	33	35	62	234
	% row	2.1	20.1	17.5	4.7	14.1	15.0	26.5	100.0

You will see that the info is the same, just better formatted (and not accessible: that is the drawback, if you plan on using your computed values further along the way, the standard way is ugly but gives you usable data; `datasummary` is nice looking but it is helpful only for presentation purposes, as the data is not easily accessible – if at all. It's for displaying.

Regression tables with `modelsummary`

Another nice tool under our belt is to be able to report *regression output* nicely. Standard `R` leaves a lot to be desired. Let's look this up!

Exercise 7: regression output

> Regress `Ozone` on `Temp`, `Solar.R` and `Wind` using `airquality`, our old friend from last course. Then show its `summary()`. It's ugly as hell

```
reg <- lm(Ozone ~ Temp + Solar.R + Wind, data = airquality)
summary(reg)
```

```
Call:
lm(formula = Ozone ~ Temp + Solar.R + Wind, data = airquality)

Residuals:
    Min      1Q  Median      3Q     Max 
-40.485 -14.219  -3.551  10.097  95.619 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) -64.34208   23.05472  -2.791  0.00623 ** 
Temp          1.65209    0.25353   6.516 2.42e-09 ***
Solar.R       0.05982    0.02319   2.580  0.01124 *  
Wind         -3.33359    0.65441  -5.094 1.52e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.18 on 107 degrees of freedom
  (42 observations deleted due to missingness)
Multiple R-squared:  0.6059,    Adjusted R-squared:  0.5948 
F-statistic: 54.83 on 3 and 107 DF,  p-value: < 2.2e-16
```

> we know we can `tidy()` it to generate a nice `data.frame` that we can use in further analyeses. We did this last time; to revise, just do it (remember to import the `broom` library)

```
library(broom)
tidy(reg)
```

```
# A tibble: 4 × 5
  term        estimate std.error statistic       p.value
  <chr>          <dbl>     <dbl>     <dbl>         <dbl>
1 (Intercept) -64.3      23.1        -2.79 0.00623
2 Temp          1.65      0.254       6.52 0.00000000242
3 Solar.R       0.0598    0.0232      2.58 0.0112
4 Wind         -3.33      0.654      -5.09 0.00000152
```

This is nice to be used further, but it's not nice to display. What if our work is done and we need to show the regression? Let's try to do this with `modelsummary`

```
modelsummary(reg)
```

	(1)
(Intercept)	-64.342
	(23.055)
Temp	1.652
	(0.254)
Solar.R	0.060
	(0.023)
Wind	-3.334
	(0.654)
Num.Obs.	111
R2	0.606
R2 Adj.	0.595
AIC	998.7
BIC	1012.3
Log.Lik.	-494.359
F	54.834
RMSE	20.80

Nice, right? But not perfect. Can it be customized?

Customizing `modelsummary` objects

Yes it can. `modelsummary` offers two ways of customizing a regression output table:

1. using dedicated parameters of the `modelsummary()` function, that pertain to regression-relevant stuff (coefficient names, stars, statistics, and the like);
2. by saving the `modelsummary` object to a `gt` version of itself, and then just applying all we know from `gt` (handy!)

Among the functions that we can use to custmize appearance, we single out

- `fmt` that allows to give the formatting of numbers, and in particular the number of digits
- `estimate` that allows to tell `modelsummary` which estimates we want to show in which way (eg std. errors, conf int, stars…)
- `statistic` that displays one or more statistics under the estimate
- `coef_omit` that allows you to omit some coefficients (eg intercept)
- `coef_rename` that allows you to rename some coefficients

Exercise 8: customize the `reg` object to look better

> let's take the regression we made and
>
> - display in the estimate row the estimate and its conf interval
> - display in the statistic row the p-value and the stars
> - omit the intercept
> - format numbers with 2 digits
> - rename the coefficients to be human-readable (eg Temperature instead of Temp)
> - and rename the whole model by giving it the predicted variable name (`Ozone`)

```
modelsummary(list("Ozone" = reg),
             estimate = "{estimate}  [{conf.low};{conf.high}]",
             statistic = "{p.value}{stars}",
             coef_omit = "(Intercept)",
             coef_rename = c("Temp" = "Temperature",
                             "Solar.R" = "Solar Radiation"),
```

```
            fmt = 2
)
```

	Ozone
Temperature	1.65 [1.15;2.15]
	<0.01***
Solar Radiation	0.06 [0.01;0.11]
	0.01*
Wind	-3.33 [-4.63;-2.04]
	<0.01***
Num.Obs.	111
R2	0.606
R2 Adj.	0.595
AIC	998.7
BIC	1012.3
Log.Lik.	-494.359
F	54.834
RMSE	20.80

> Also, `modelsummary` can accommodate several models side-by-side. Let's run a couple more regressions and add them

```
# regression of Ozone on Temp  and Solar.R alone
reg2 <- lm(Ozone ~ Temp + Solar.R , data = airquality)

# regression of Ozone an Temp alone
reg3 <- lm(Ozone ~ Temp  , data = airquality)

# modelsummary of the three models
modelsummary(list(reg, reg2, reg3))
```

	(1)	(2)	(3)
(Intercept)	-64.342	-145.703	-146.995
	(23.055)	(18.447)	(18.287)
Temp	1.652	2.278	2.429
	(0.254)	(0.246)	(0.233)
Solar.R	0.060	0.057	
	(0.023)	(0.026)	
Wind	-3.334		
	(0.654)		
Num.Obs.	111	111	116
R2	0.606	0.510	0.488
R2 Adj.	0.595	0.501	0.483
AIC	998.7	1020.8	1067.7
BIC	1012.3	1031.7	1076.0
Log.Lik.	-494.359	-506.410	-530.853
F	54.834	56.275	108.529
RMSE	20.80	23.18	23.51

> Finally, let's transform the `modelsummary` of the three models into a `gt` object – this you do using `output = 'gt'` – and then add a title, a spanner, and some other fancy stuff, then use a theme of your choice:

```
modelsummary(list(reg, reg2, reg3),
             output = "gt") %>%
  tab_header(title = "Three regressions using Airquality") %>%
  tab_source_note("from the airquality built-in dataset") %>%
```

```
tab_spanner(columns = contains("("), label = "Ozone") %>%
gt_theme_nytimes()
```

Three regressions using Airquality

	Ozone		
	(1)	(2)	(3)
(Intercept)	-64.342	-145.703	-146.995
	(23.055)	(18.447)	(18.287)
Temp	1.652	2.278	2.429
	(0.254)	(0.246)	(0.233)
Solar.R	0.060	0.057	
	(0.023)	(0.026)	
Wind	-3.334		
	(0.654)		
Num.Obs.	111	111	116
R2	0.606	0.510	0.488
R2 Adj.	0.595	0.501	0.483
AIC	998.7	1020.8	1067.7
BIC	1012.3	1031.7	1076.0
Log.Lik.	-494.359	-506.410	-530.853
F	54.834	56.275	108.529
RMSE	20.80	23.18	23.51
from the airquality built-in dataset			

To know more

That's it for today. To know more about tables, internet is your friend.

- `gt` page: https://gt.rstudio.com/index.html
- `modelsummary` page: https://modelsummary.com
- `gtExtras` page: https://jthomasmock.github.io/gtExtras/
- More table packages that you might like and/or want to check out (better pdf tables, fancier html widgets…):
  - `huxtable`: https://hughjonesd.github.io/huxtable/
  - `kableExtra`:
    - html: https://inbo.r-universe.dev/kableExtra/doc/awesome_table_in_html.html
    - pdf: https://haozhu233.github.io/kableExtra/awesome_table_in_pdf.pdf
  - `tinytable`: https://vincentarelbundock.github.io/tinytable/