Introduction to Tidyverse

Mustafa Anil Kocak March 6, 2024

Course Content

First Encounter with R Programming!

- [ ] R studio, scripts, markdowns
- [ ] Functions, packages, help pages
- [ ] R Objects, notation & modifying values
- [ ] Data structures, programming

Introduction to Tidyverse

- [ ] Visualization
- [ ] Wrangling
- [ ] Exploratory Data Analysis

Miscellaneous

- [ ] Practice and case studies
- [ ] Advanced data visualization
- [ ] Machine learning examples

Course Content

First Encounter with R Programming!

- [ ] ~~R studio, scripts, markdowns~~
- [ ] ~~Functions, packages, help pages~~
- [ ] ~~R Objects, notation & modifying values~~
- [ ] ~~Data structures, programming~~

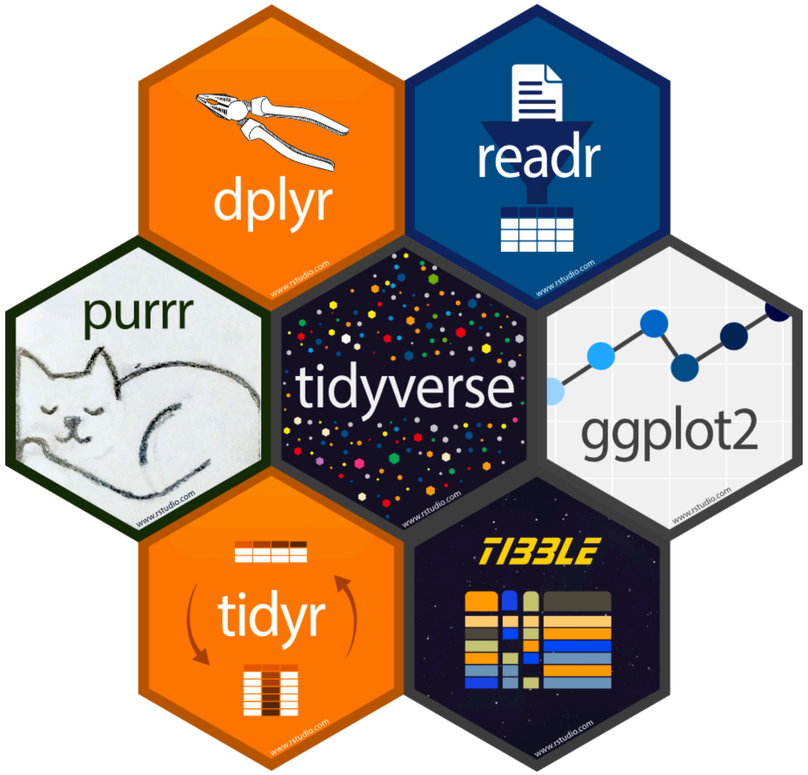
Introduction to Tidyverse

- [ ] **Visualization**
- [ ] **Wrangling**
- [ ] **Exploratory Data Analysis**

Miscellaneous

- [ ] Practice and case studies
- [ ] Advanced data visualization
- [ ] Machine learning examples

What is tidyverse?



"At a high level, the tidy verse is a language for solving data science challenges with R code. Its primary goal is to facilitate a conversation between a human and a computer about data. Less abstractly, the tidyverse is a collection of R packages that share a high-level design philosophy and low-level grammar and data structures, so that learning one package makes it easier to learn the next."

Wickham et al., 2019

Tidyverse Design Principles

- It is **human centered**, i.e. the tidyverse is designed specifically to support the activities of a human data analyst.
- It is **consistent**, so that what you learn about one function or package can be applied to another, and the number of special cases that you need to remember is as small as possible.
- It is **composable**, allowing you to solve complex problems by breaking them down into small pieces, supporting a rapid cycle of exploratory iteration to find the best solution.
- It is **inclusive**, because the tidyverse is not just the collection of packages, but it is also the community of people who use them.

design.tidyverse.org

Introduction to Tidyverse

Tibbles vs Data Frames

Visualization: ggplot

Transformations: dplyr

Tidy data: tidyr

Exploratory Data Analysis

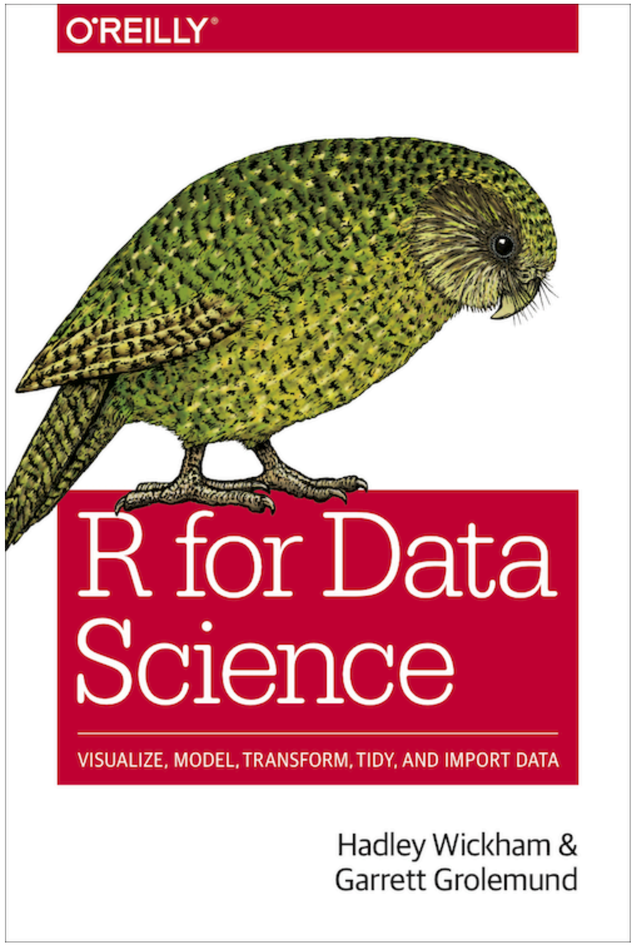
Tidyverse for Data Analysis

Freely available online:

- [ ] https://r4ds.had.co.nz/

We will cover a selection of chapters in detail

- [ ] Chapters 9, 3 , 5 , 10, 19
- [ ] Please check out the rest in your own time
- [ ] Definitely bookmark it to revisit in the future.



Hadley Wickham & Garrett Grolemund

Introduction

Visualisation

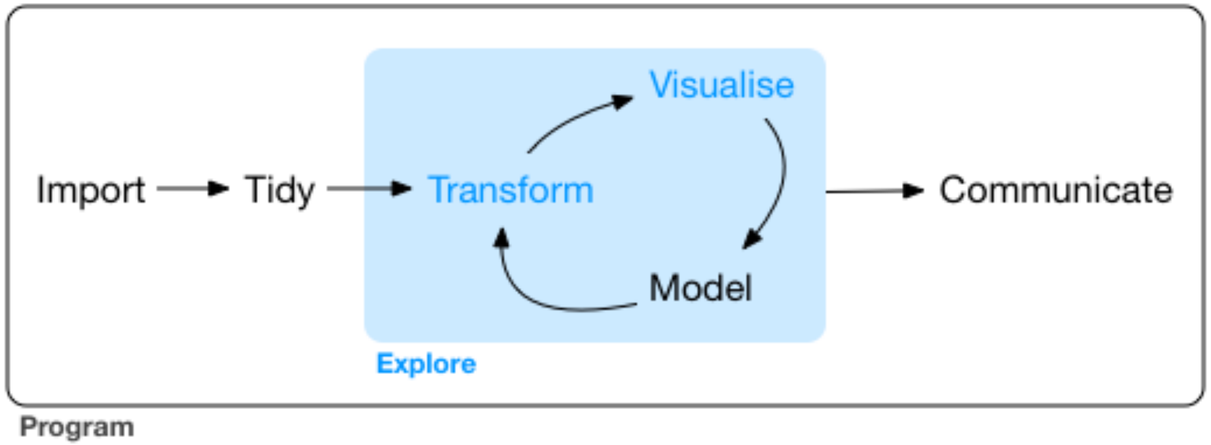
- ☐ Basic structure of a ggplot2 plot

Transformation

- ☐ Select/filter/arrange/group/ summarize/pivot data frames

Exploratory data analysis

- ☐ Visualization + transformation to ask questions to the data



We will re-visit modeling in following lectures, but for a deeper coverage please see: *https://www.tmwr.org/*

Tibbles vs Data Frames

Tibbles are opinionated data frames!

- See vignette("tibble")

Creating a tibble: as_tibble(), tibble(), tribble()

Non-syntactic column names and no row names

Main differences: Printing and subsetting

- Main difference is [ ]
- as.data.frame()

Data visualisation

Aesthetics (mappings)

Faceting

Geometric Objects

Statistical Transformations

Positional Arguments

Coordinate Systems

Grammar of Graphics

Introduction

We will work with ggplot2 package, which implements grammar of graphics.

See http://vita.had.co.nz/papers/layered-grammar.pdf

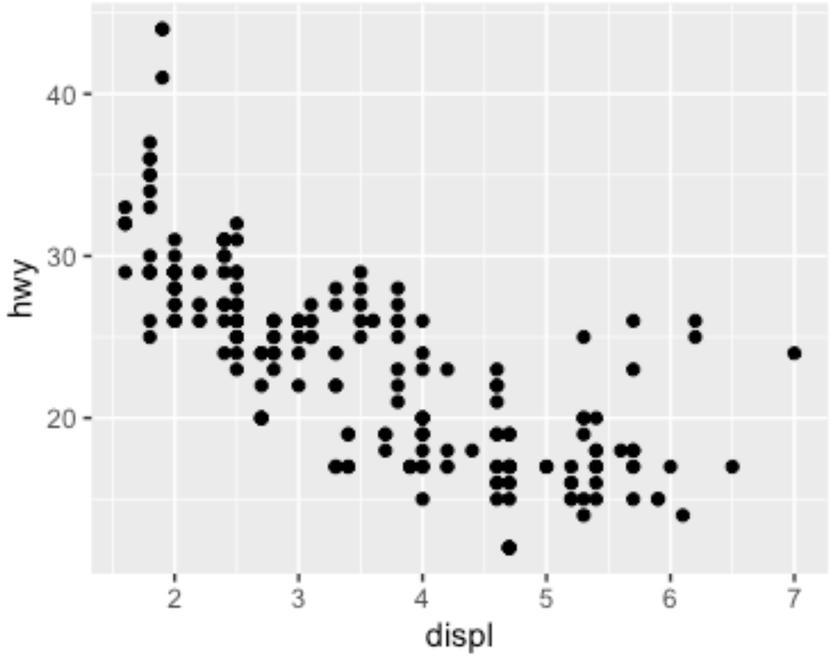
```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

"The simple graph has brought more information to the data analyst's mind than any other device."

— John Tukey

First Steps

```
ggplot(data = mpg) +
    geom_point(aes(x = displ,
                   y = hwy))
```



mpg {ggplot2} R Documentation

Fuel economy data from 1999 to 2008 for 38 popular models of cars

Description

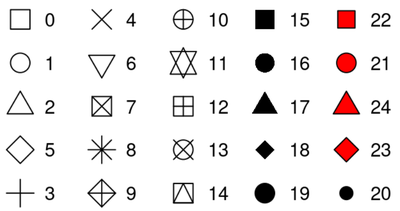
This dataset contains a subset of the fuel economy data that the EPA makes available on https://fueleconomy.gov/. It contains only models which had a new release every year between 1999 and 2008 - this was used as a proxy for the popularity of the car.

```
> mpg
# A tibble: 234 × 11
   manufacturer model      displ  year   cyl trans      drv     cty   hwy fl    class
   <chr>        <chr>      <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 audi         a4           1.8  1999     4 auto(l5)   f        18    29 p     compact
 2 audi         a4           1.8  1999     4 manual(m5) f        21    29 p     compact
 3 audi         a4           2    2008     4 manual(m6) f        20    31 p     compact
 4 audi         a4           2    2008     4 auto(av)   f        21    30 p     compact
 5 audi         a4           2.8  1999     6 auto(l5)   f        16    26 p     compact
 6 audi         a4           2.8  1999     6 manual(m5) f        18    26 p     compact
 7 audi         a4           3.1  2008     6 auto(av)   f        18    27 p     compact
 8 audi         a4 quattro   1.8  1999     4 manual(m5) 4        18    26 p     compact
 9 audi         a4 quattro   1.8  1999     4 auto(l5)   4        16    25 p     compact
10 audi         a4 quattro   2    2008     4 manual(m6) 4        20    28 p     compact
# … with 224 more rows
# ℹ Use `print(n = ...)` to see more rows
```

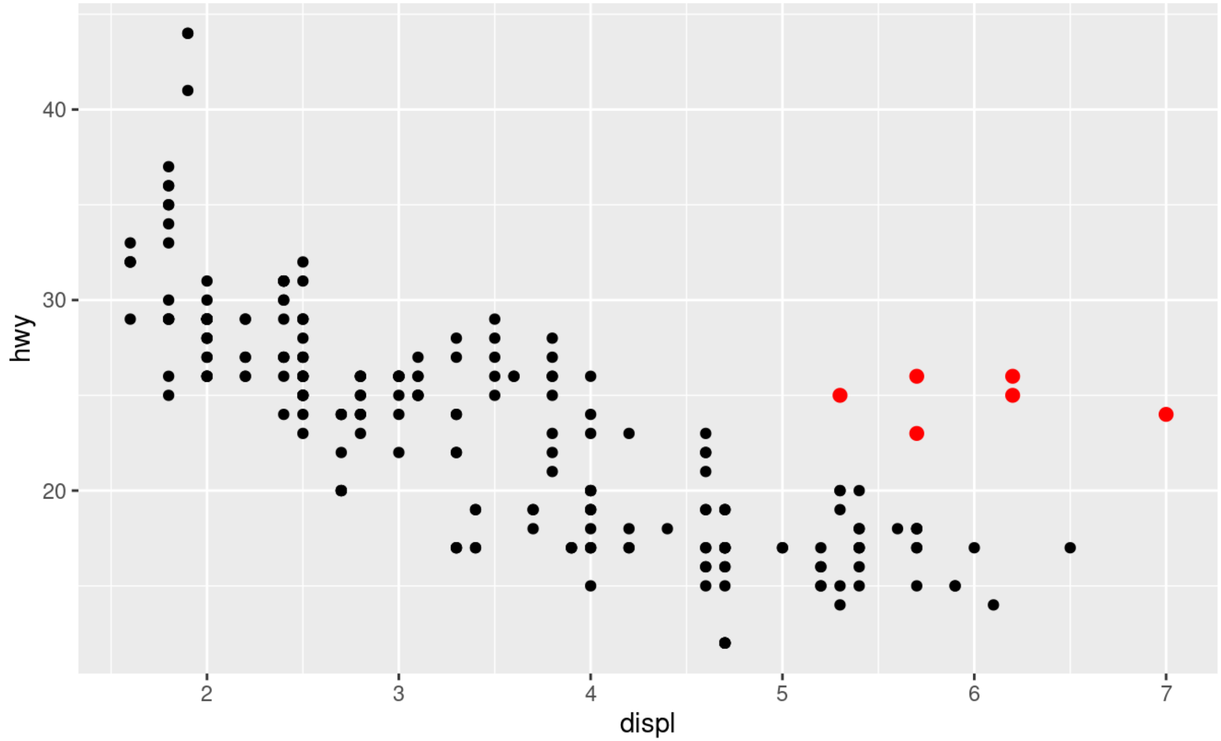
Aesthetic mappings : aes()

Aesthetics: visuals properties of the objects in your plot.

- Size, shape, alpha, fill, color, …



- Note x and y values (position) is also part of the aesthetics!



"The greatest value of a picture is when it forces us to notice what we never expected to see."

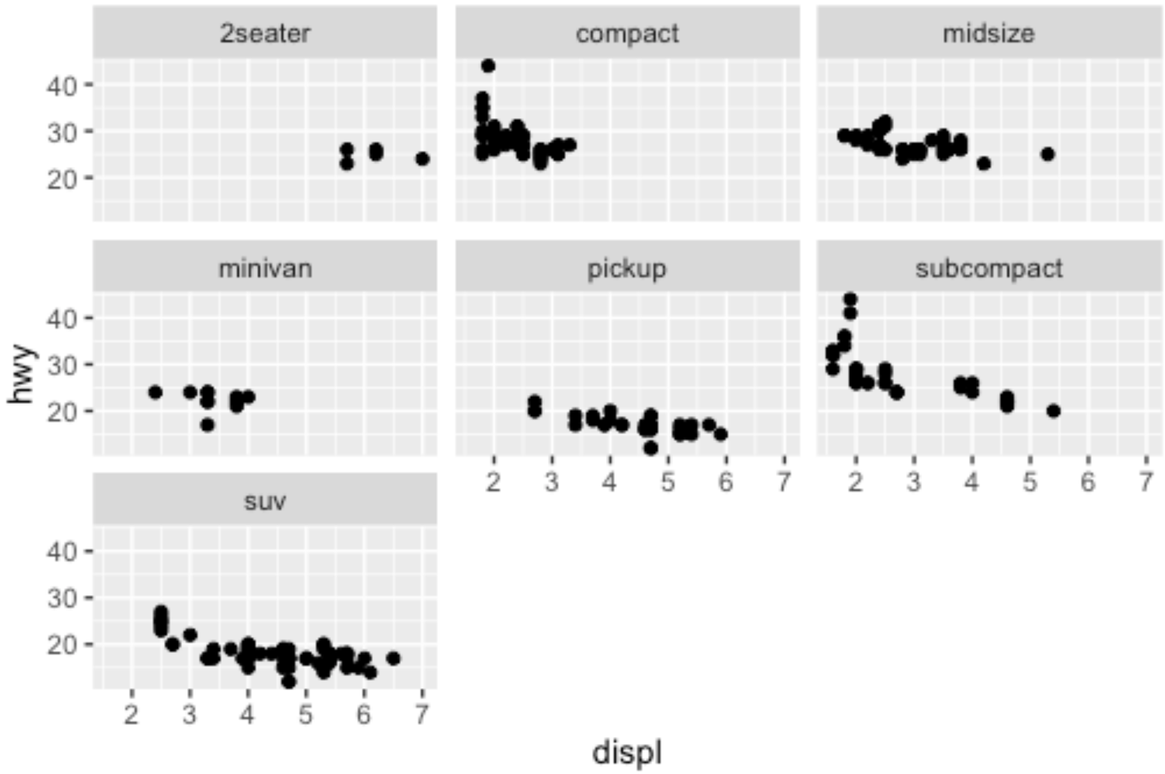
— John Tukey

Facets

Alternative way to encode new information

- Better suited for qualitative comparisons
- facet_wrap() and facet_grid()
- From ggforce package:

  facet_wrap_paginate() and facet_zoom()

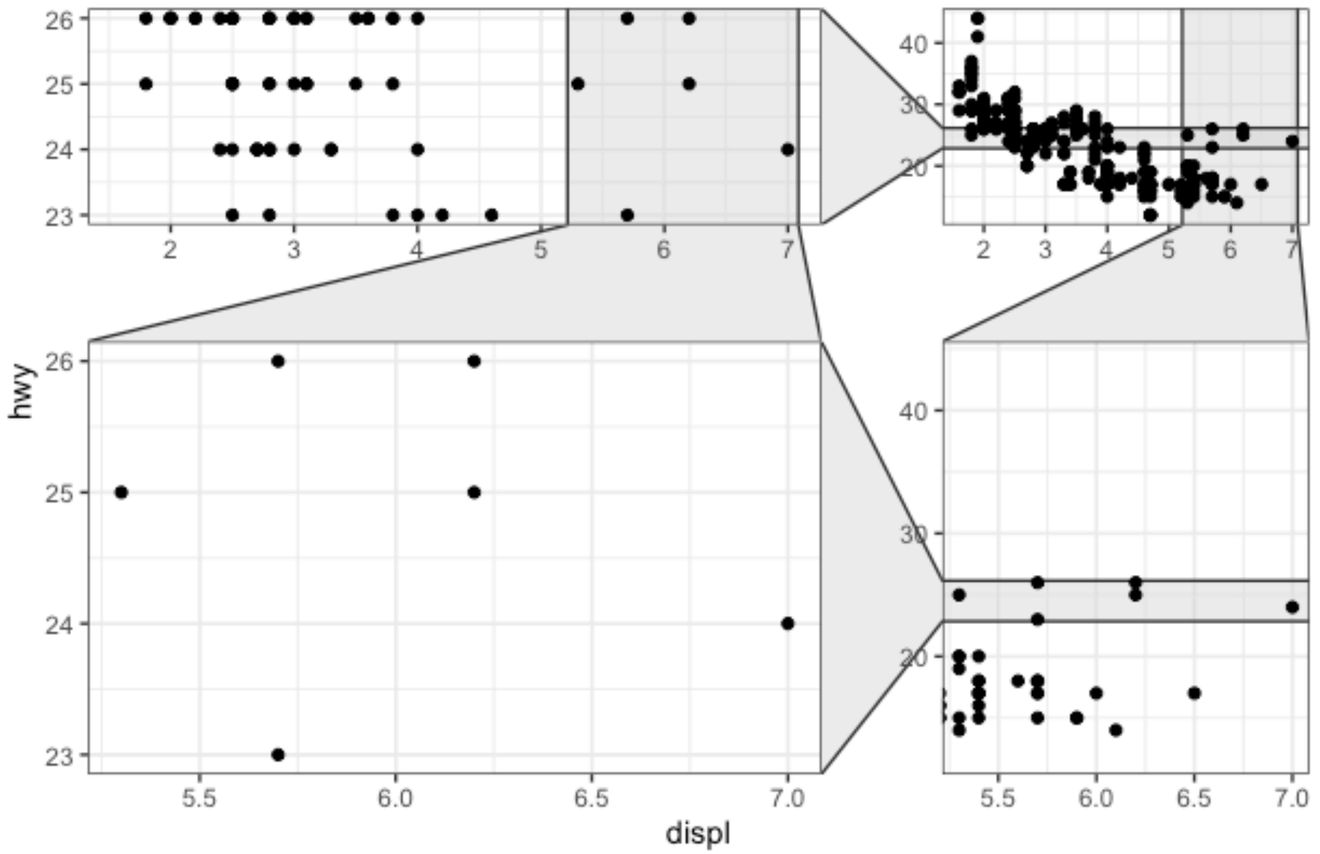


Facets

Alternative way to encode new information

- Better suited for qualitative comparisons
- facet_wrap() and facet_grid()
- From ggforce package:

  facet_wrap_paginate() and facet_zoom()



Geometric objects

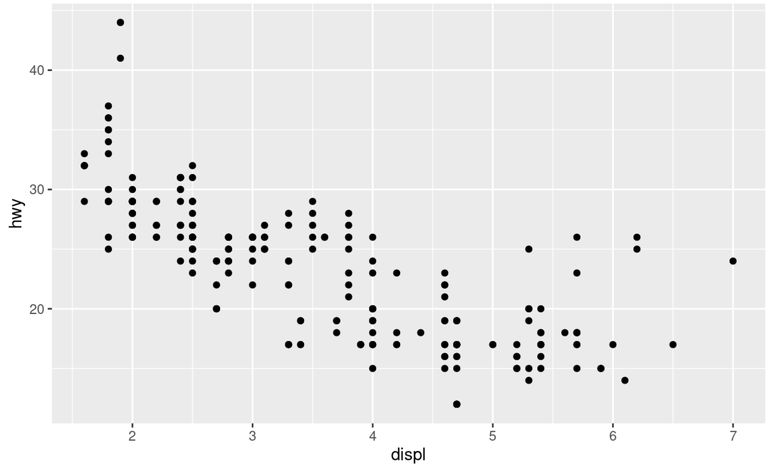
Geoms are geometrical objects that a plot uses to represent data

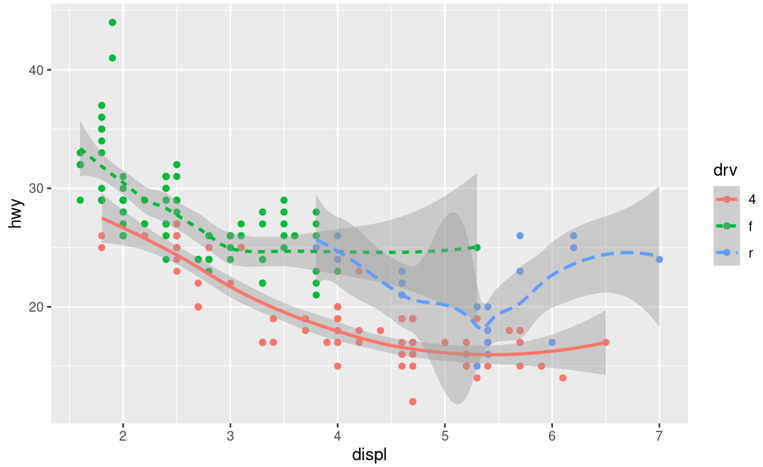
- e.g. points, bars, lines, box plots, smooth, path, violin, histogram, density, contours, tiles, rugs, …

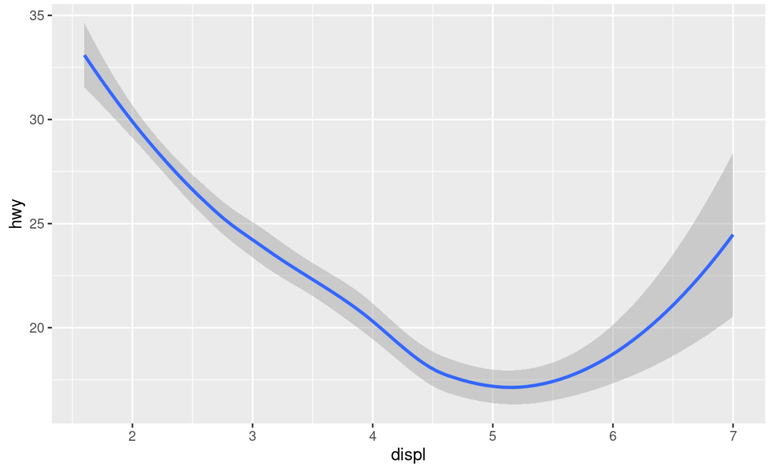
Geoms, mappings, aesthetics…

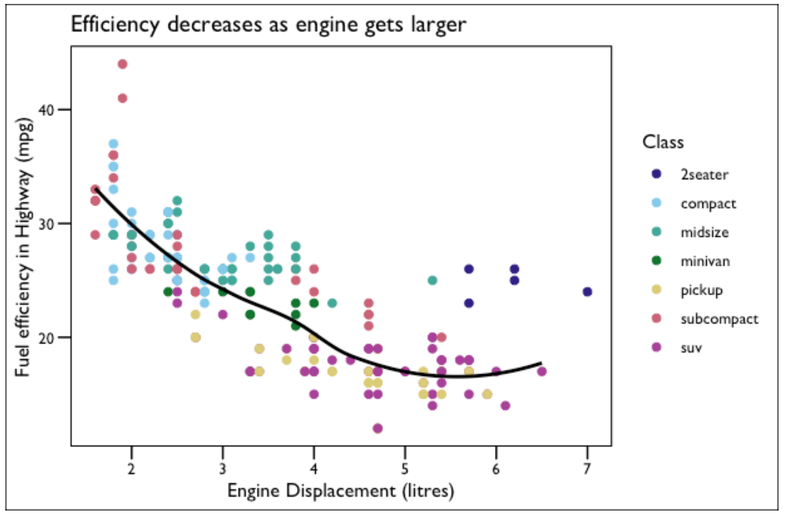
Useful resources:

- https://posit.co/resources/cheatsheets/
- https://exts.ggplot2.tidyverse.org/gallery/



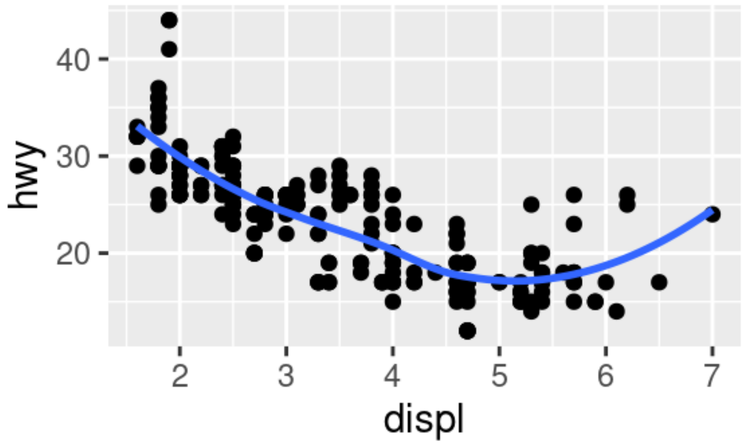


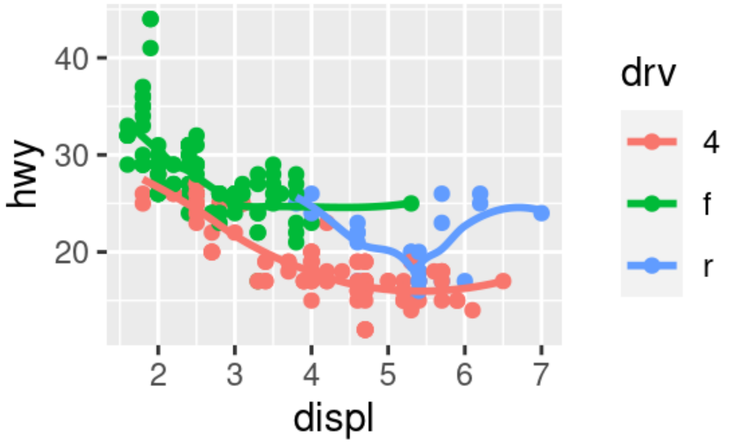


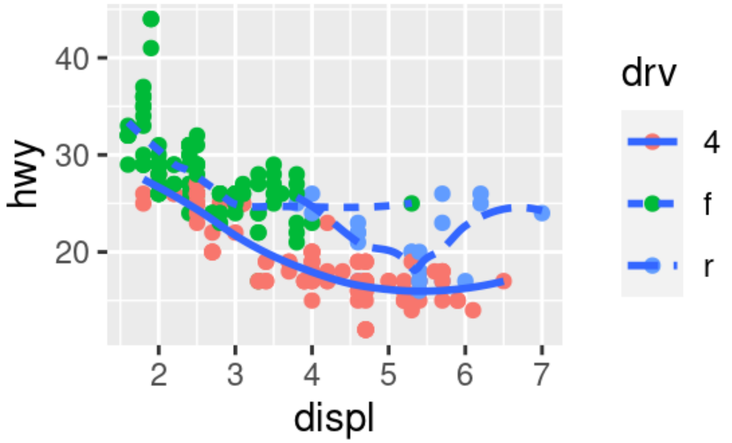


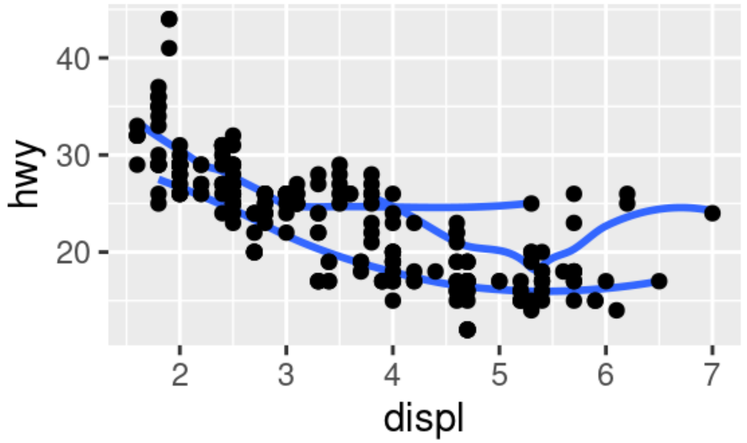
Geometric objects - exercises

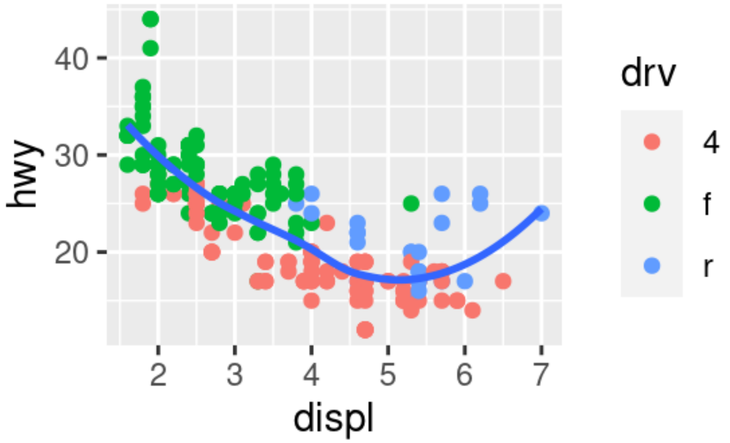
Recreate the R code necessary to generate the following graphs

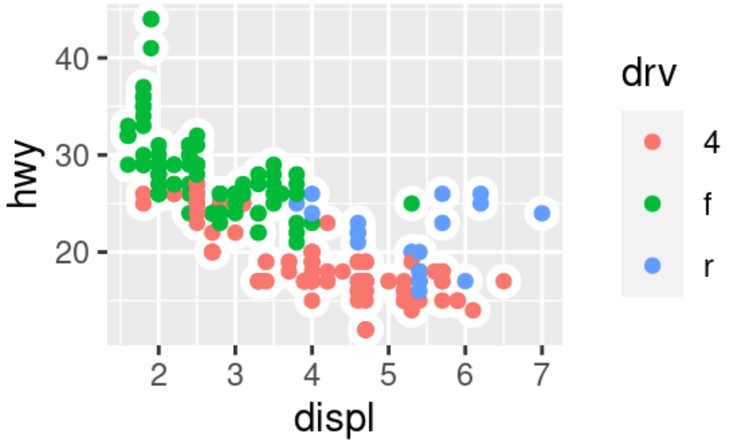












Statistical transformations

Geoms create some statistical summaries under the hood:

- [ ] Bar charts, histograms, and frequency polygons bin your data and then plot bin counts.
- [ ] Smoothers fit a model to your data and then plot predictions from the model.
- [ ] Boxplots compute a robust summary of the distribution and then display a specially formatted box.

1. **`geom_bar()`** begins with the **`diamonds`** data set

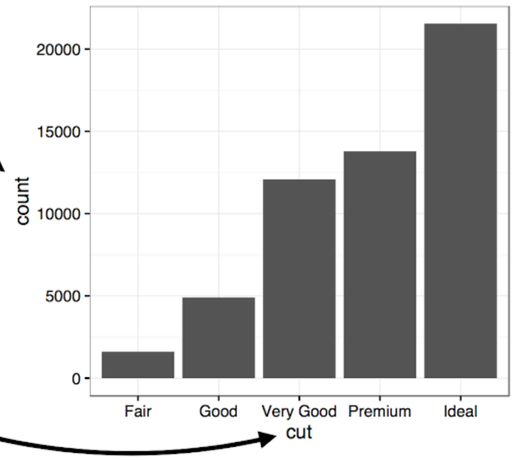
carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...

2. **`geom_bar()`** transforms the data with the "count" stat, which returns a data set of cut values and counts.

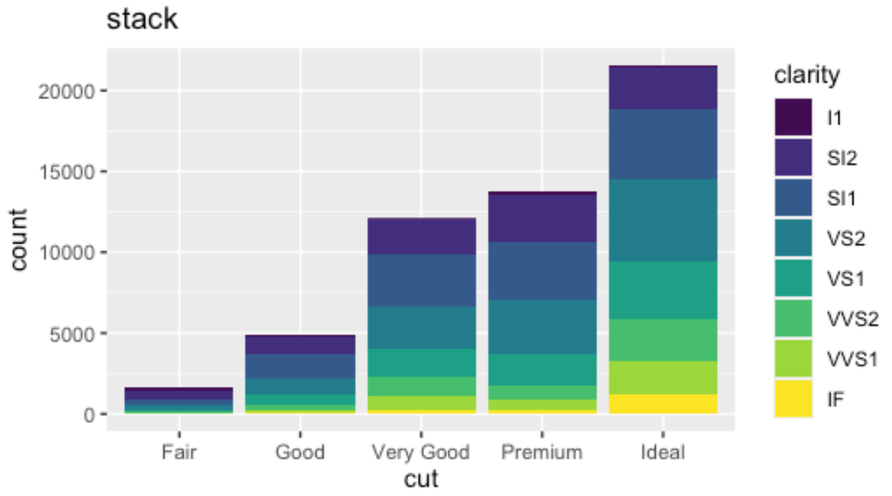
stat_count()

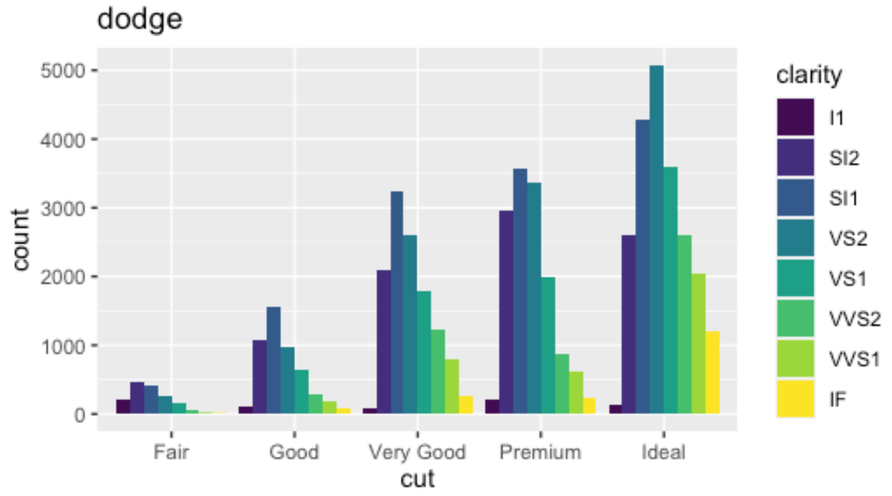
cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1

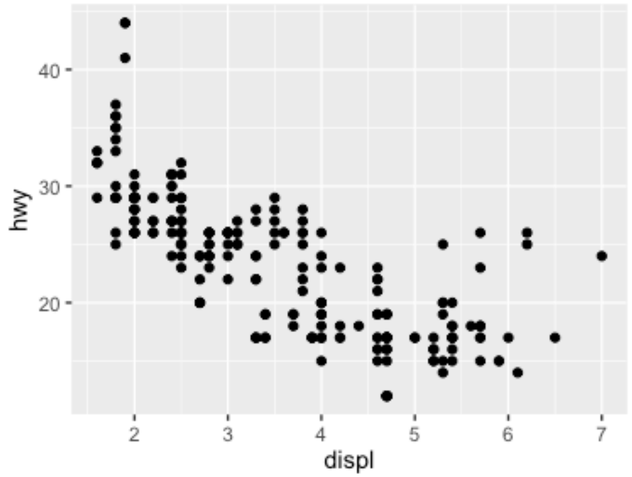
3. **`geom_bar()`** uses the transformed data to build the plot. cut is mapped to the x axis, count is mapped to the y axis.

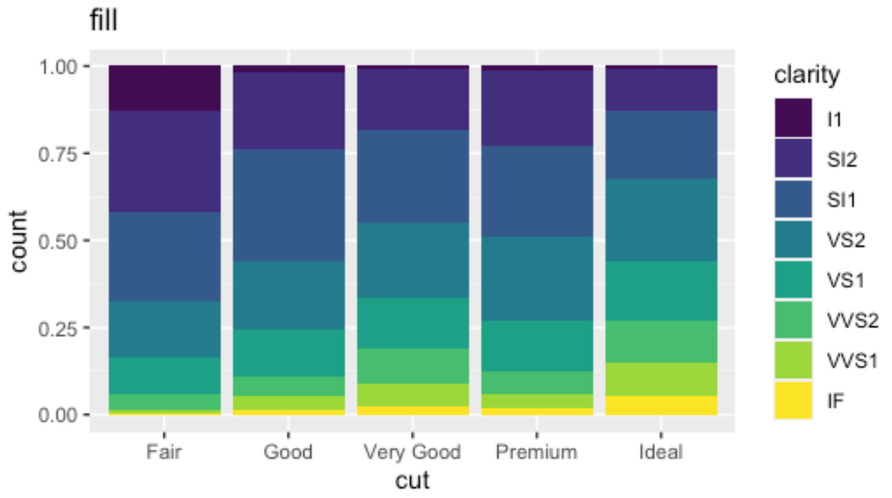


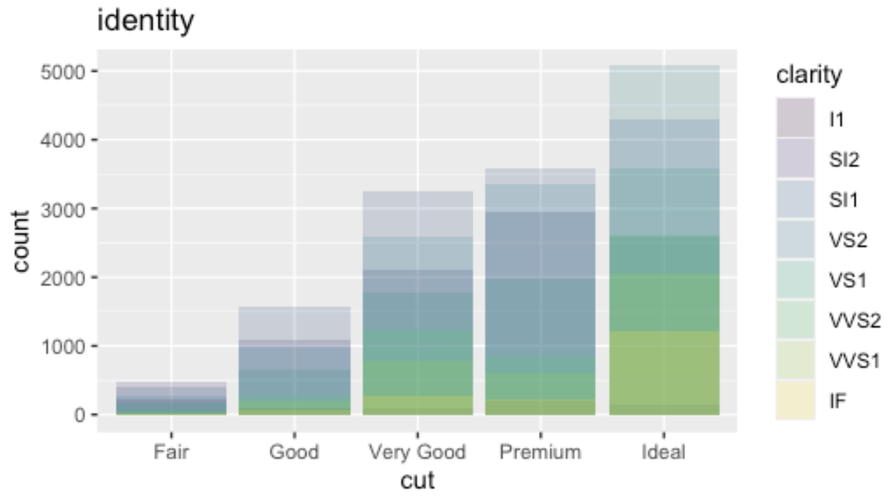
Position adjustments

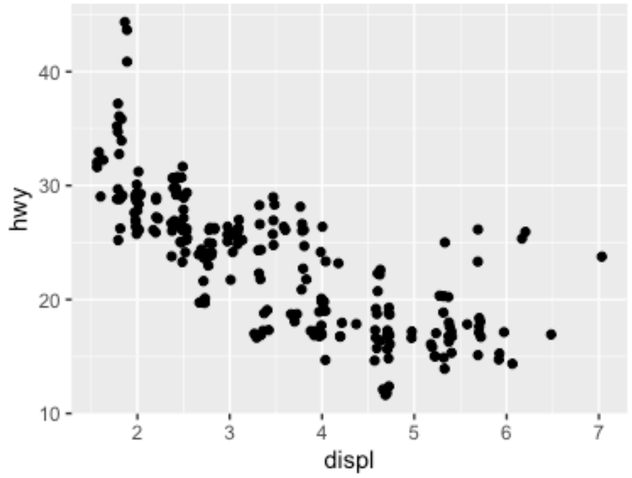












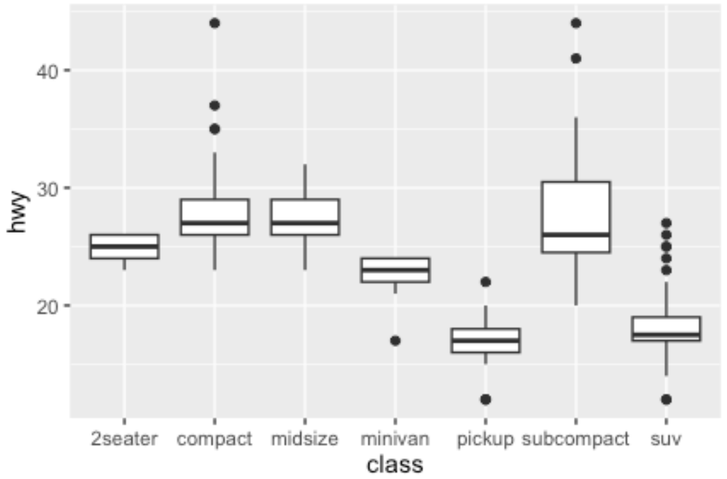
Coordinate systems

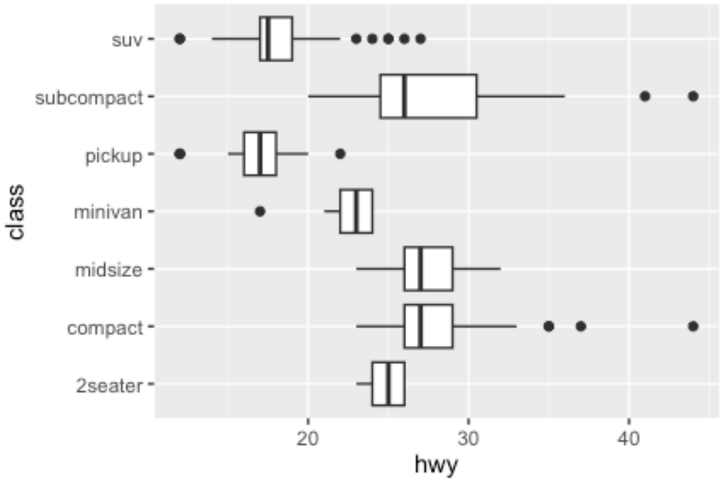
Some handy coordinate transformations:

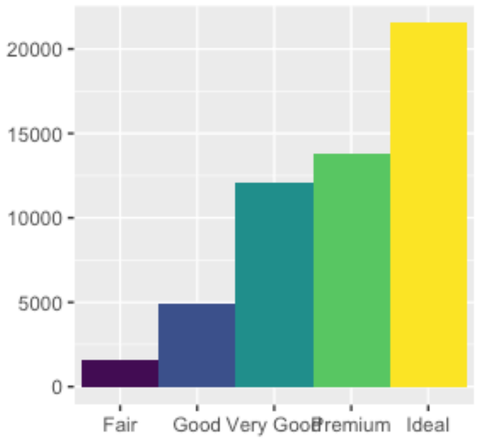
- coord_flip/polar/trans/ cartesian/fixed

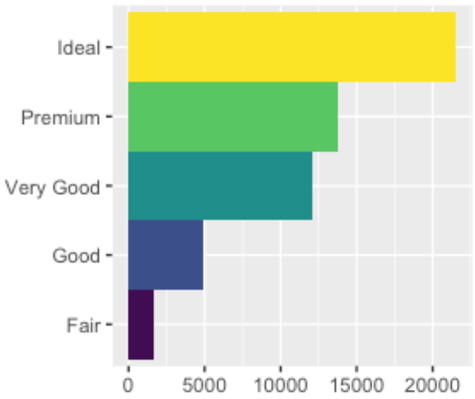
Misc.

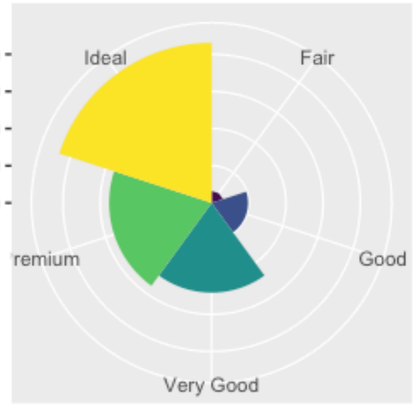
- labs(), geom_abline(), reorder()





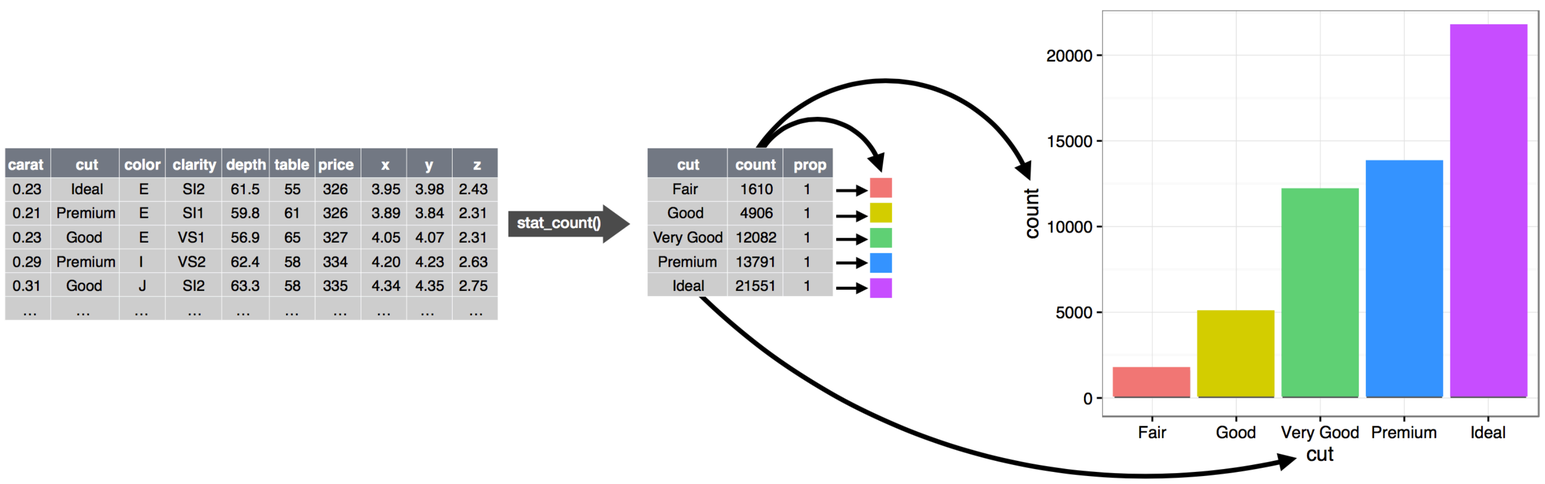






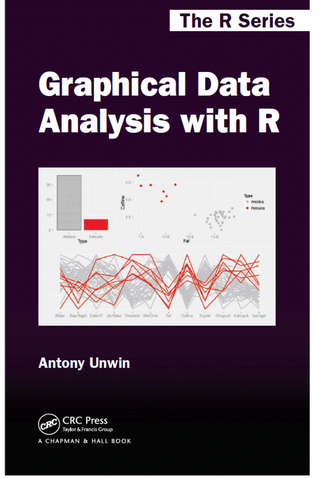
The layered grammar of graphics - recap

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```



1. Begin with the **`diamonds`** data set
2. Compute counts for each cut value with **`stat_count()`**.
3. Represent each observation with a bar.
4. Map the **`fill`** of each bar to the **`..count..`** variable.
5. Place geoms in a cartesian coordinate system.
6. Map the y values to **`..count..`** and the x values to **`cut`**.

Some general advice from GDAR



1. In a scatterplot of causally related variables the dependent variable is drawn on the vertical axis and the explanatory variable on the horizontal axis.
2. Numbers increase to the right and up.
3. The x-axis crosses the y-axis at $y = 0$. When it doesn't, this should be made clear.
4. Scales are linear, and when they are not this should be emphasised.
5. Time is usually drawn on the horizontal axis, progressing from left to right.
6. Graphics are always drawn to show all the data. If some cases are out of range, this should be explicitly stated.
7. Aspect ratios (the relationship of the height of a graphic to its width) should be chosen to make the slope of lines of interest about 45°, an idea first discussed thoroughly in [Cleveland et al., 1988].
8. Points usually represent individual cases and areas represent counts or weights.
9. Vertical bars represent frequencies of continuous variables when there are no gaps between them, and frequencies of categorical variables when there are.
10. Distinct colours are used to represent groups, while shading or continuous spectra are used to represent scales.

A number of recommendations recur in this book:

- Use graphics to discover information that is difficult to investigate statistically.
- Draw many graphics and vary the graphics options.
- Gain experience in interpreting graphics and use the graphics types you know well.
- Consider reformatting datasets before drawing graphics.
- Make appropriate comparisons and choose comparable scales.
- Check any graphical result you find with statistical models, where possible. Statistics and graphics complement one another.
- Always remember how important context is in interpreting results.

The same statistics and graphics might be interpreted quite differently for different applications. That is why this book has concentrated on using 'real' datasets. Finally, interactive graphics have occasionally been mentioned in this book as a tool with potential for GDA and that is an important topic for the future.

Practice Time

https://simplystatistics.org/posts/2019-08-28-you-can-replicate-almost-any-plot-with-ggplot2/

Homework

We covered Chapter 9

Self-study for reinforcement: Chapter 1 & 2

We learned how to make plots and the grammar of graphics.

Chapter 10 & 11 provide guidance for which graphs to generate and how to present.

Data Transformations

Filtering rows

Arranging rows

Selecting columns

Adding (mutating) columns

Grouping and summaries

Introduction

Pick observations by their values (filter()).

Reorder the rows (arrange()).

Pick variables by their names (select()).

Create new variables from existing variables (mutate()).

Collapse many values down to a single summary (summarise()).

All verbs work similarly:

1. The first argument is a data frame.
2. The subsequent arguments describe what to do with the data frame, using the variable names (without quotes).
3. The result is a new data frame.

```
> head(flights) # note the variable types at the top
# A tibble: 6 x 19
   year month   day dep_time sched_de…¹ dep_d…² arr_t…³ sched…⁴ arr_d…⁵ carrier flight tailnum origin dest  air_t…⁶ dista…⁷  hour minute
  <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>    <int> <chr>   <chr>  <chr>   <dbl>   <dbl> <dbl>  <dbl>
1  2013     1     1      517        515       2     830     819      11 UA        1545 N14228  EWR    IAH       227    1400     5     15
2  2013     1     1      533        529       4     850     830      20 UA        1714 N24211  LGA    IAH       227    1416     5     29
3  2013     1     1      542        540       2     923     850      33 AA        1141 N619AA  JFK    MIA       160    1089     5     40
4  2013     1     1      544        545      -1    1004    1022     -18 B6         725 N804JB  JFK    BQN       183    1576     5     45
5  2013     1     1      554        600      -6     812     837     -25 DL         461 N668DN  LGA    ATL       116     762     6      0
6  2013     1     1      554        558      -4     740     728      12 UA        1696 N39463  EWR    ORD       150     719     5     58
```

Filtering rows: filter()

data.filtered = filter(data, cond1, cond2, …)

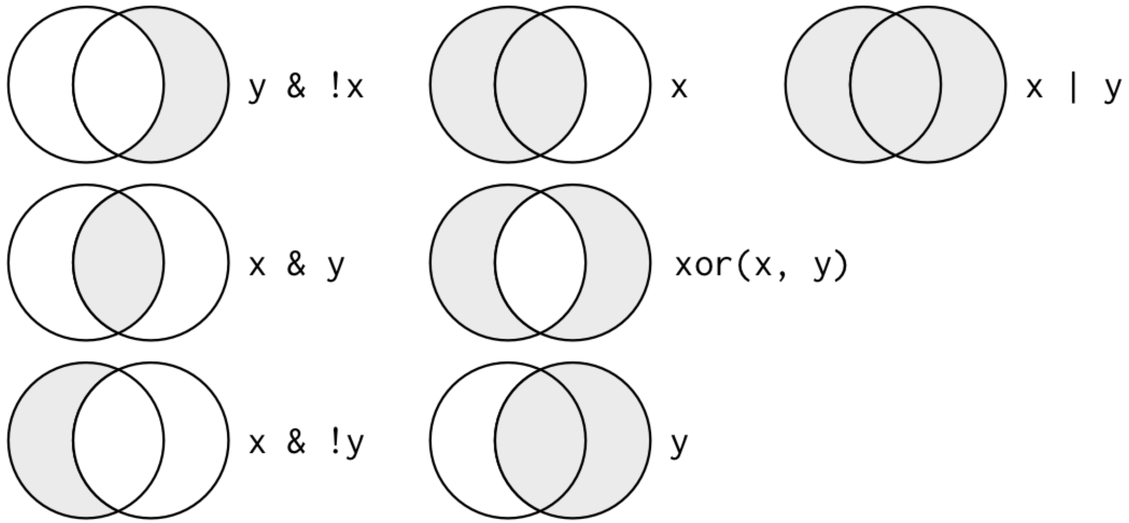
- data.filtered = data %>%
  filter(cond1, cond2) %>%
  ….

De Morgan's rule

- P & Q === !(!P | !Q)

Misc:

- near(), NA 's, assignment with printing ()
- NA's are not treated as TRUE



Arranging Rows: arrange()

data.sorted = arrange(data, cond1, cond2, …)

data.sorted.descendent = arrange(data, desc(cond1))

NA's are listed as last

Commonly used with group_by and head

Selecting columns: select()

Select a subset of columns and drop the rest:

select(data, col1, col2, ….)
select(data, 1,2, …)
select(data, col1:col2)
select(data, 2:5)

There are a number of helper functions you can use within select():

starts_with(“abc”), ends_with(“xyz”), contains(“ijk”),
matches(“(.)\\1”): This one matches any variables that contain repeated characters. num_range("x", 1:3):
matches x1, x2 and x3.

Misc:

distinct(), tolower(), toupper(), word(), paste0(), paste()

Adding columns: mutate()

data.new = mutate(data , new_variable = some_function(old_variables))

data.new = transmute(data , updated_variable = some_function(old_variables))

The function must be vectorised

- It must take a vector of values as input, return a vector with the same number of values as output.

Misc: *%/%, %%, lag(), lead(), diff(), cumsum(), cummean(), cumprod(), cummin(), cummax(), min_rank(), cume_dist(), rank(), sort(), order(), ntile(), log(), log2(), log10()*

Grouped summaries: summary() and group_by()

```
data %>%
    group_by(grouping1, grouping2) %>%
    summarize(summary1 = f(…),
                summary2 = g(…))
```

Grouped datasets may become hard to track

Don't forget to ungroup() your datasets

Grouping can also be used with filter and mutates

Misc:

n(), sd(), IQR(), mad(), first(), last(), nth(), sum(!is.na(x)), n_distinct(), count(),

http://varianceexplained.org/r/empirical_bayes_baseball/

https://ggvis.rstudio.com/

vignette("window-functions")

"A good way to pronounce %>% when reading code is 'then'"

Tidy Data

Tidy Data

Pivoting

Separating and Uniting

Missing Values

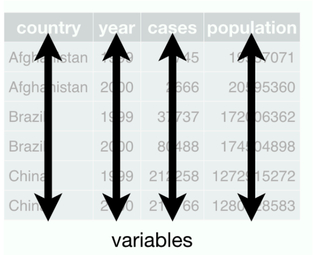
Case Study (Homework)

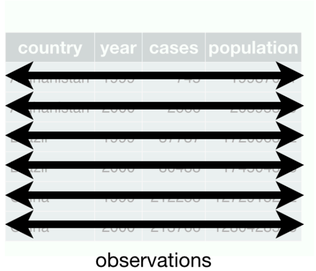
Non-tidy Data

Tidy Data

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

- Hadley Wickham





- [ ] Each dataset is a tibble
- [ ] Each variable is a column

http://www.jstatsoft.org/v59/i10/paper

```
> table1
# A tibble: 6 × 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
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
# A tibble: 6 × 3
  country      year rate
  <chr>       <dbl> <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
```

```
> table2
# A tibble: 12 × 4
   country      year type            count
   <chr>       <dbl> <chr>           <dbl>
 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
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```

```
> table4a
# A tibble: 3 × 3
  country     `1999` `2000`
  <chr>        <dbl>  <dbl>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
> table4b
# A tibble: 3 × 3
  country         `1999`     `2000`
  <chr>            <dbl>      <dbl>
1 Afghanistan   19987071   20595360
2 Brazil       172006362  174504898
3 China       1272915272 1280428583
```

Pivoting

- [ ] What is an observation?
- [ ] What are the variables?

We often need to handle:

- [ ] If one variable is is spread across columns — pivot_longer()
- [ ] If an observation spread across rows — pivot_wider()

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

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

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

table2

Separating and Uniting

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

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

table3

separate(rate, into = c(“cases”, “population”))

country	year	rate
Afghanistan	19**99**	745 / 19987071
Afghanistan	20**00**	2666 / 20595360
Brazil	19**99**	37737 / 172006362
Brazil	20**00**	80488 / 174504898
China	19**99**	212258 / 1272915272
China	20**00**	213766 / 1280428583

country	century	year	rate
Afghanistan	19	**99**	745 / 19987071
Afghanistan	20	**0**	2666 / 20595360
Brazil	19	**99**	37737 / 172006362
Brazil	20	**0**	80488 / 174504898
China	19	**99**	212258 / 1272915272
China	20	**0**	213766 / 1280428583

table6

unite(year, century, year, sep = “”)

Missing values

"An explicit missing value is the presence of an absence; and implicit missing values is the absence of a presence"

- ☐ values_drop_na = TRUE for pivot_longer()
- ☐ complete()
- ☐ fill()

```
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66)
)
```

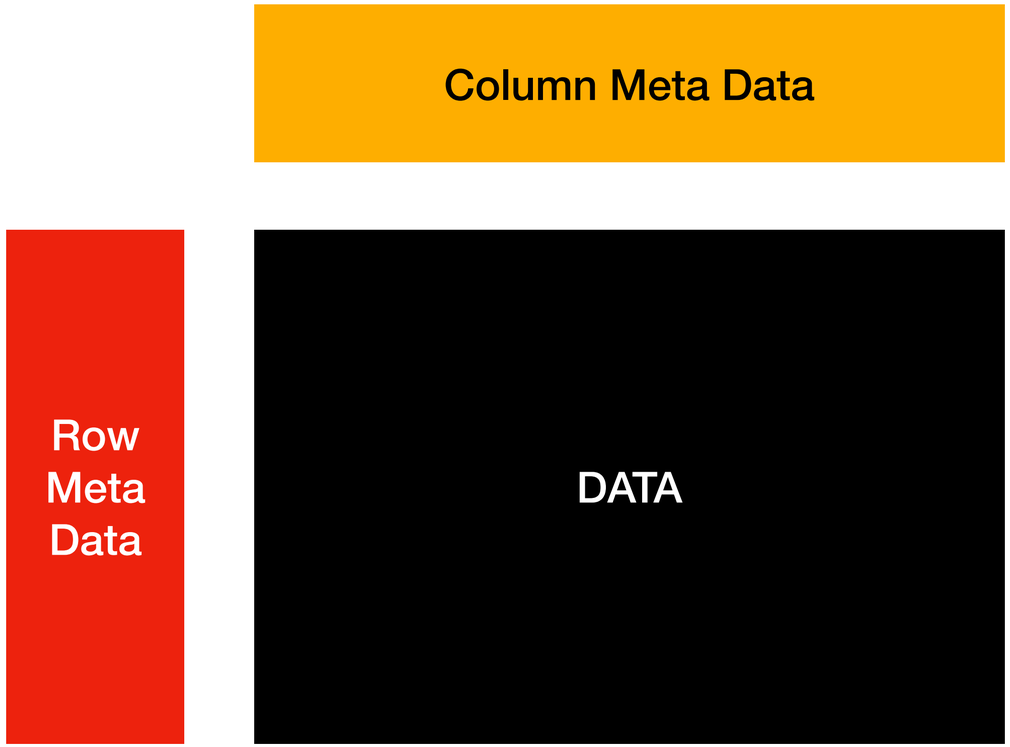
```
stocks %>%
  pivot_wider(names_from = year, values_from = return)
#> # A tibble: 4 × 3
#>     qtr `2015` `2016`
#>   <dbl>  <dbl>  <dbl>
#> 1     1   1.88  NA
#> 2     2   0.59   0.92
#> 3     3   0.35   0.17
#> 4     4  NA      2.66
```

Non-tidy Data

There are place and time for non-tidy data:

- Alternative representations may have substantial performance or space advantages.
- Specialised fields have evolved their own conventions for storing data that may be quite different to the conventions of tidy data.

https://simplystatistics.org/posts/2016-02-17-non-tidy-data/



Relational Data

nycflights13

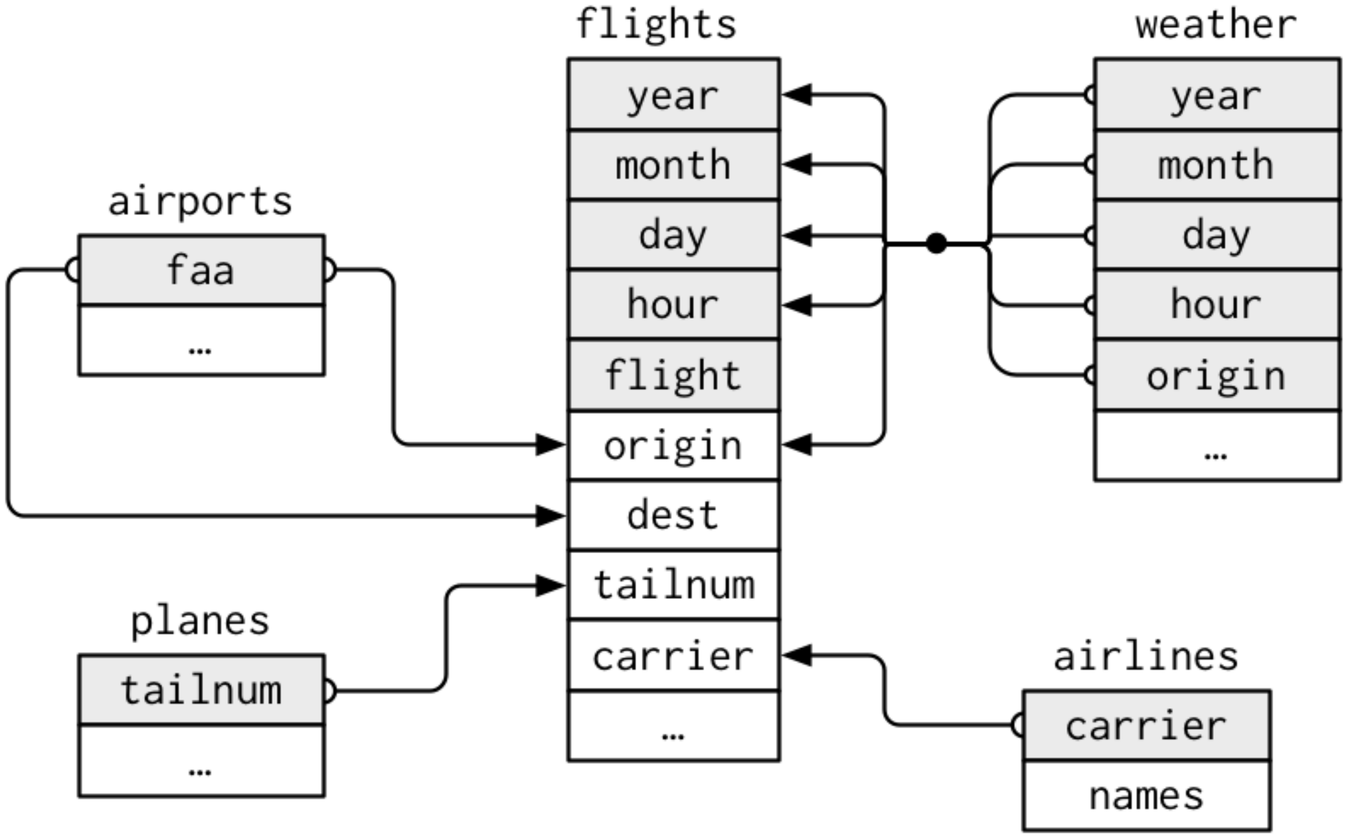
Keys

Mutating Joins

Filtering Joins

Join Problems

nycflights13



Keys

The variables used to connect each pair of tables are called keys.

A key is a variable (or set of variables) that uniquely identifies an observation.

- ☐ A primary key uniquely identifies an observation in its own table.
- ☐ A foreign key uniquely identifies an observation in another table.

It's good practice to verify that keys do indeed uniquely identify each observation.

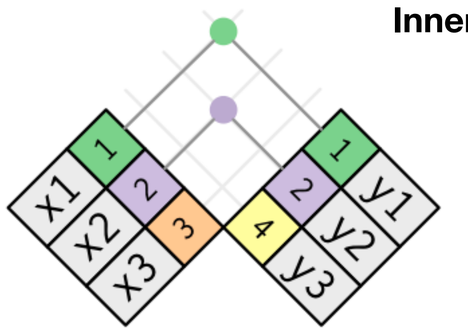
Some datasets don't have primary keys!

- ☐ It is a good idea to create a new one.

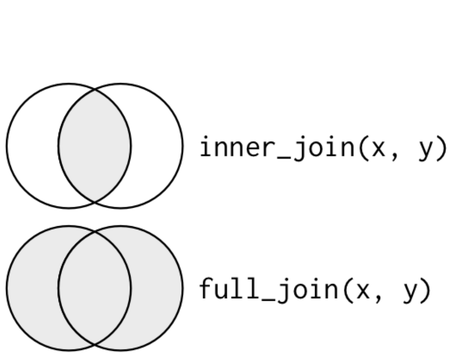
Mutating Joins - inner and outer joins

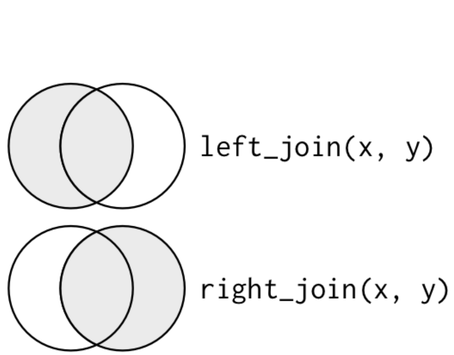
x	
1	x1
2	x2
3	x3

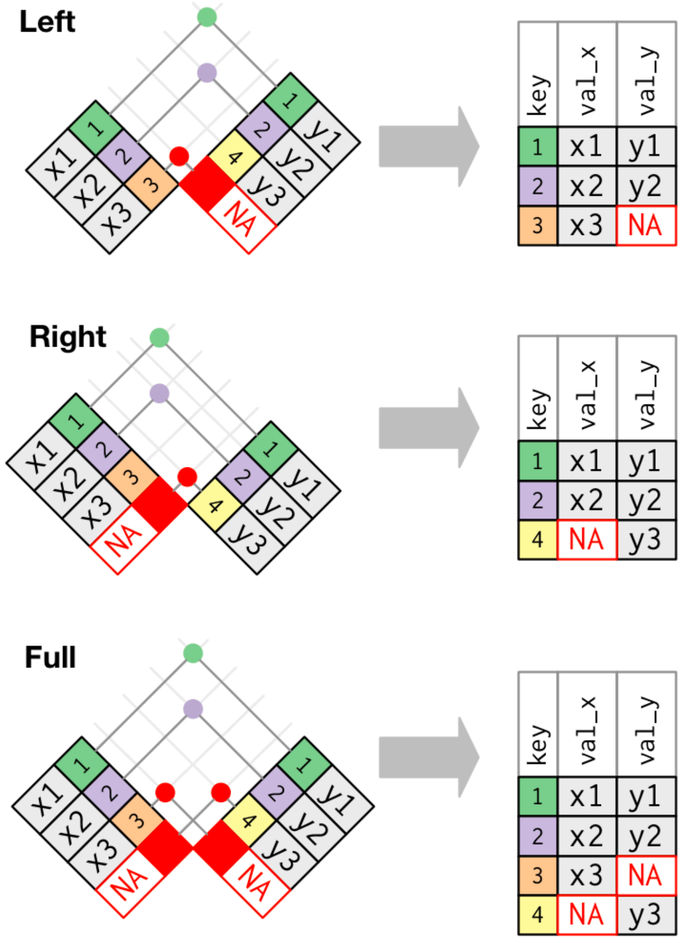
y	
1	y1
2	y2
4	y3



key	val_x	val_y
1	x1	y1
2	x2	y2





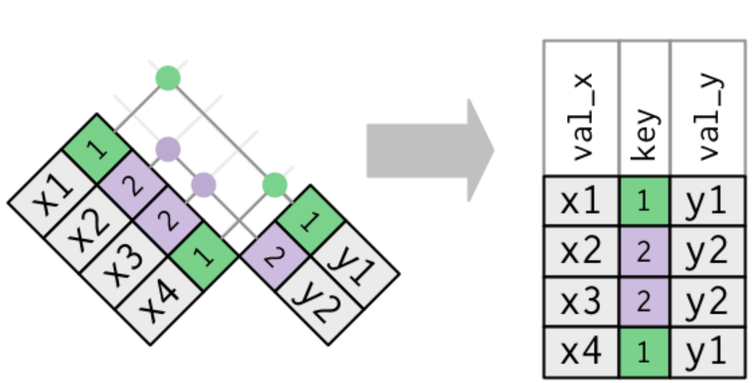


key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

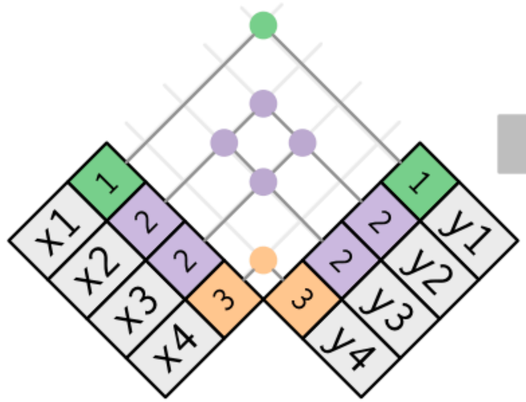
key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

Mutating Joins - duplicate keys



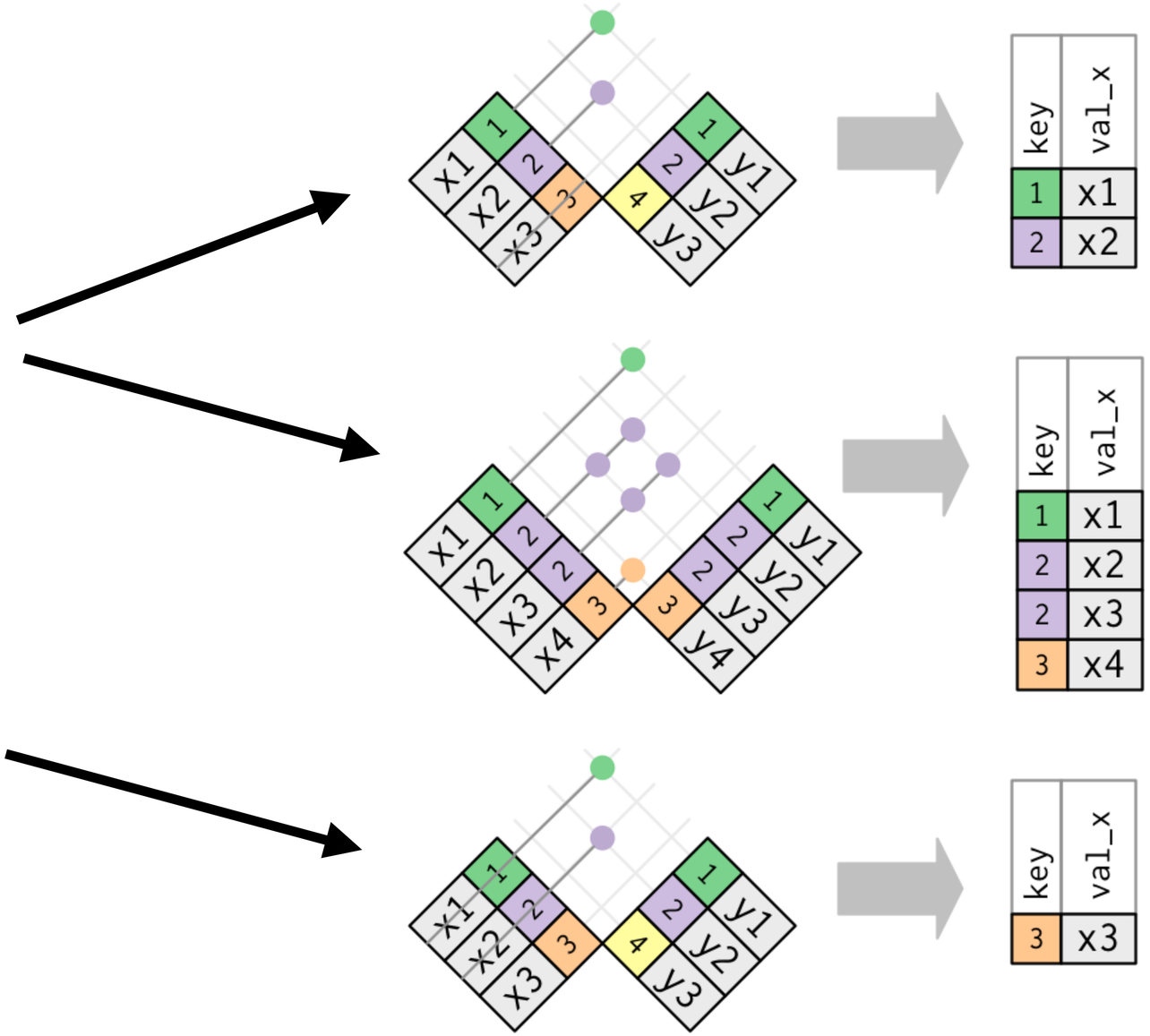
val_x	key	val_y
x1	1	y1
x2	2	y2
x3	2	y2
x4	1	y1



key	val_x	val_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

Filtering Joins

- *semi_join(x, y)* **keeps** all observations in x that have a match in y.
- *anti_join(x, y)* **drops** all observations in x that have a match in y.



Join Problems

1. Start by identifying the variables that form the primary key in each table.

2. Check that none of the variables in the primary key are missing. If a value is missing then it can't identify an observation!

3. Check that your foreign keys match primary keys in another table.

   The best way to do this is with an anti_join().

   If you do have missing keys, be thoughtful about your use of inner vs. outer joins.