Data Management With R: Data Transformation

Matthias Haber

18 September 2017

Last week's homework

Making sure everyone is set up

Data transformation with dplyr

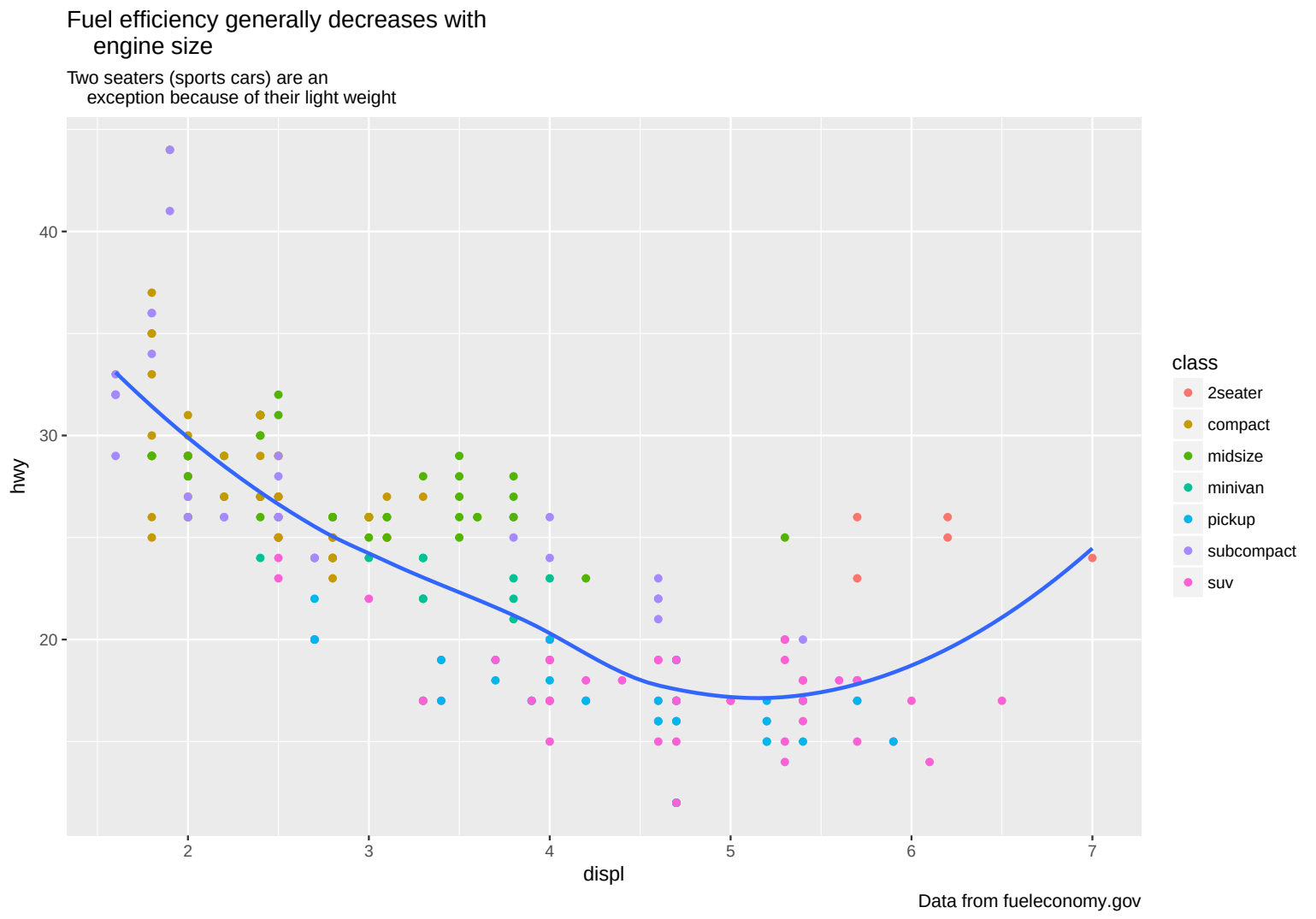
Homework Exercises

Last week's homework

Plot 1

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    title = "Fuel efficiency generally decreases with
    engine size",
    subtitle = "Two seaters (sports cars) are an
    exception because of their light weight",
    caption = "Data from fueleconomy.gov"
  )
```

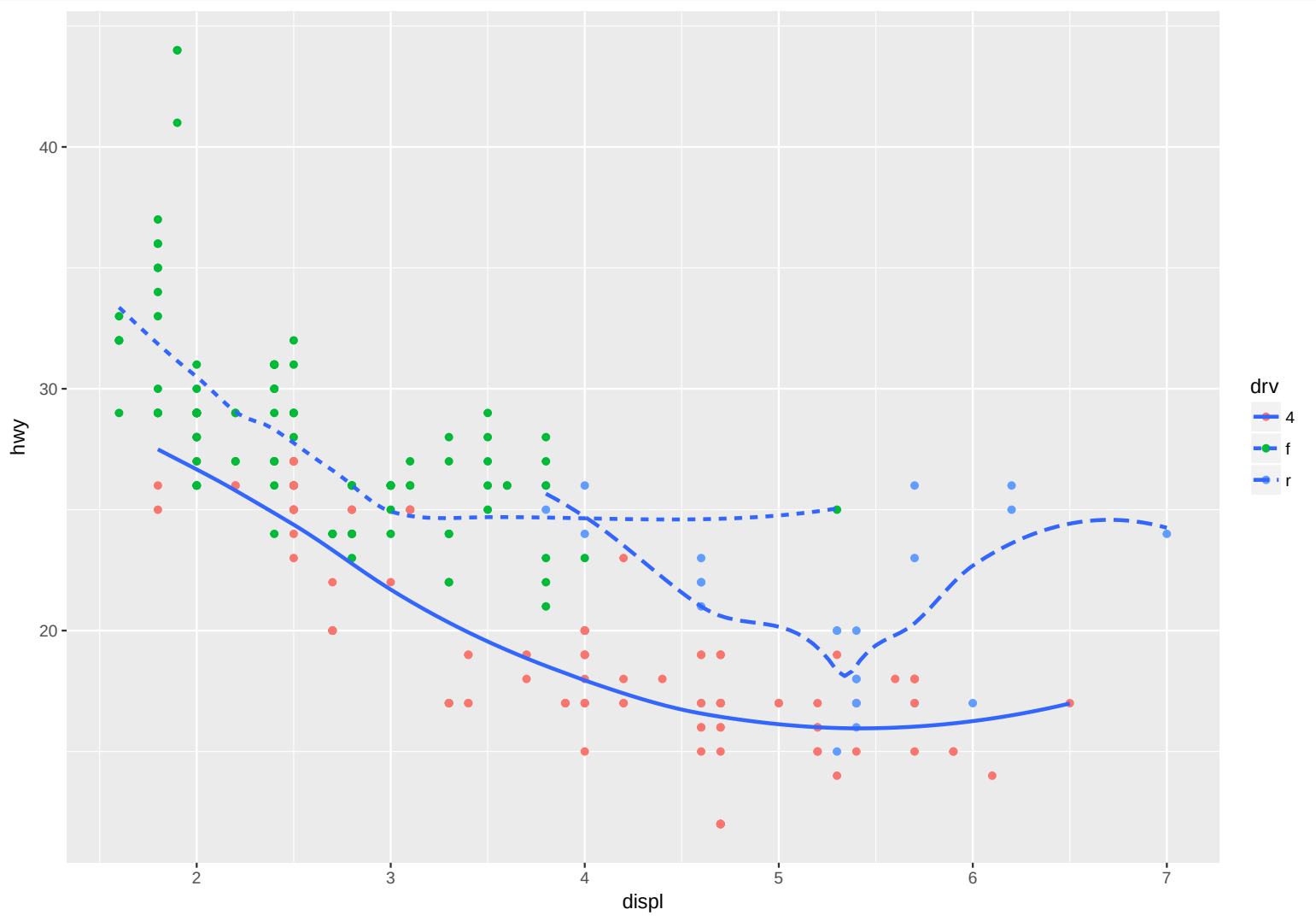
Plot 1


Fuel efficiency generally decreases with engine size
Two seaters (sports cars) are an exception because of their light weight
hwy
40
30
20
2
3
4
5
6
7
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv
Data from fueleconomy.gov

Plot 2

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = drv)) +
  geom_smooth(aes(linetype = drv), se = FALSE,
              method = "loess")
```

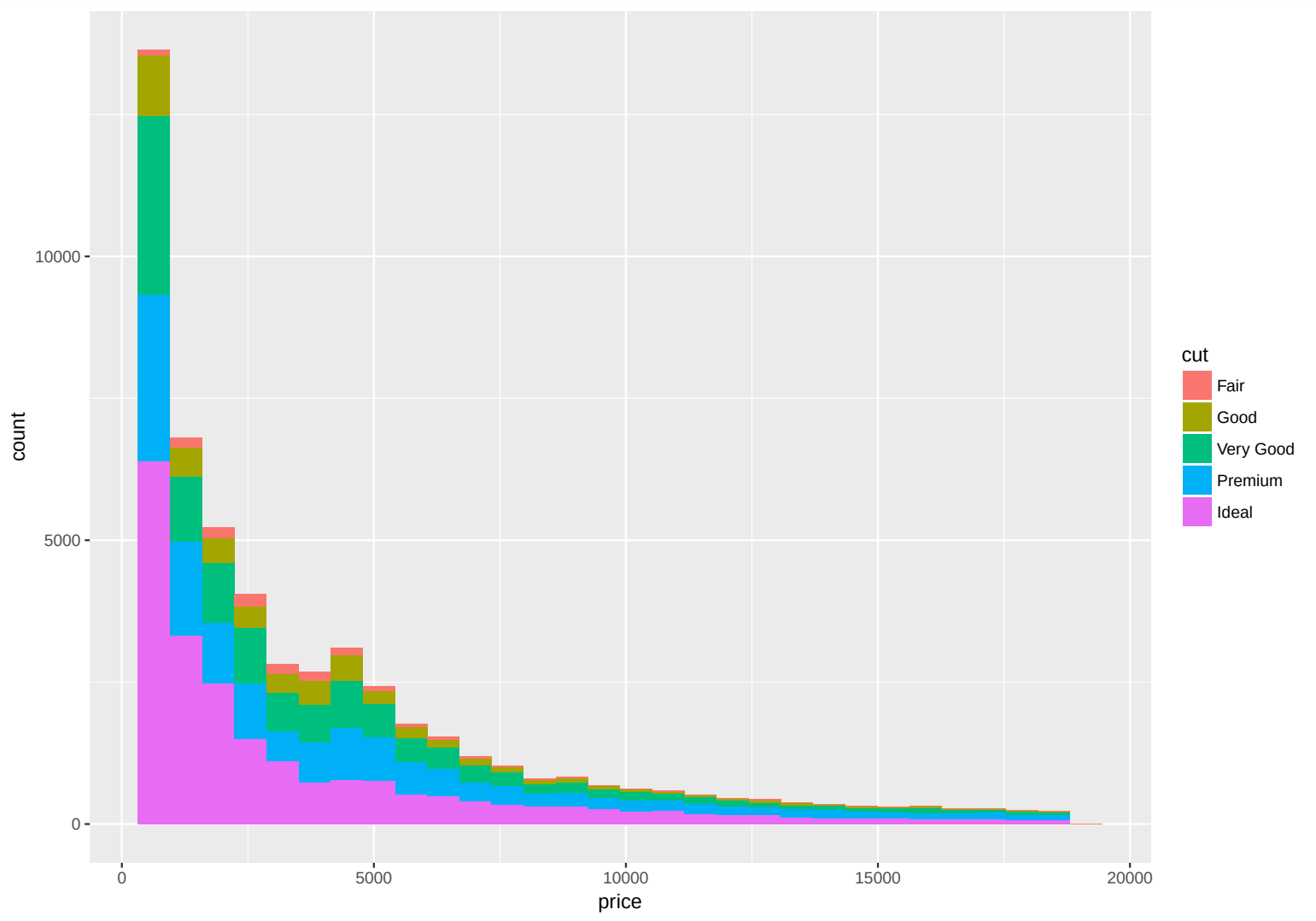
Plot 2



Plot 3

```
ggplot(data = diamonds,
       mapping = aes(x = price, fill = cut)) +
  geom_bar(stat = "bin")
```

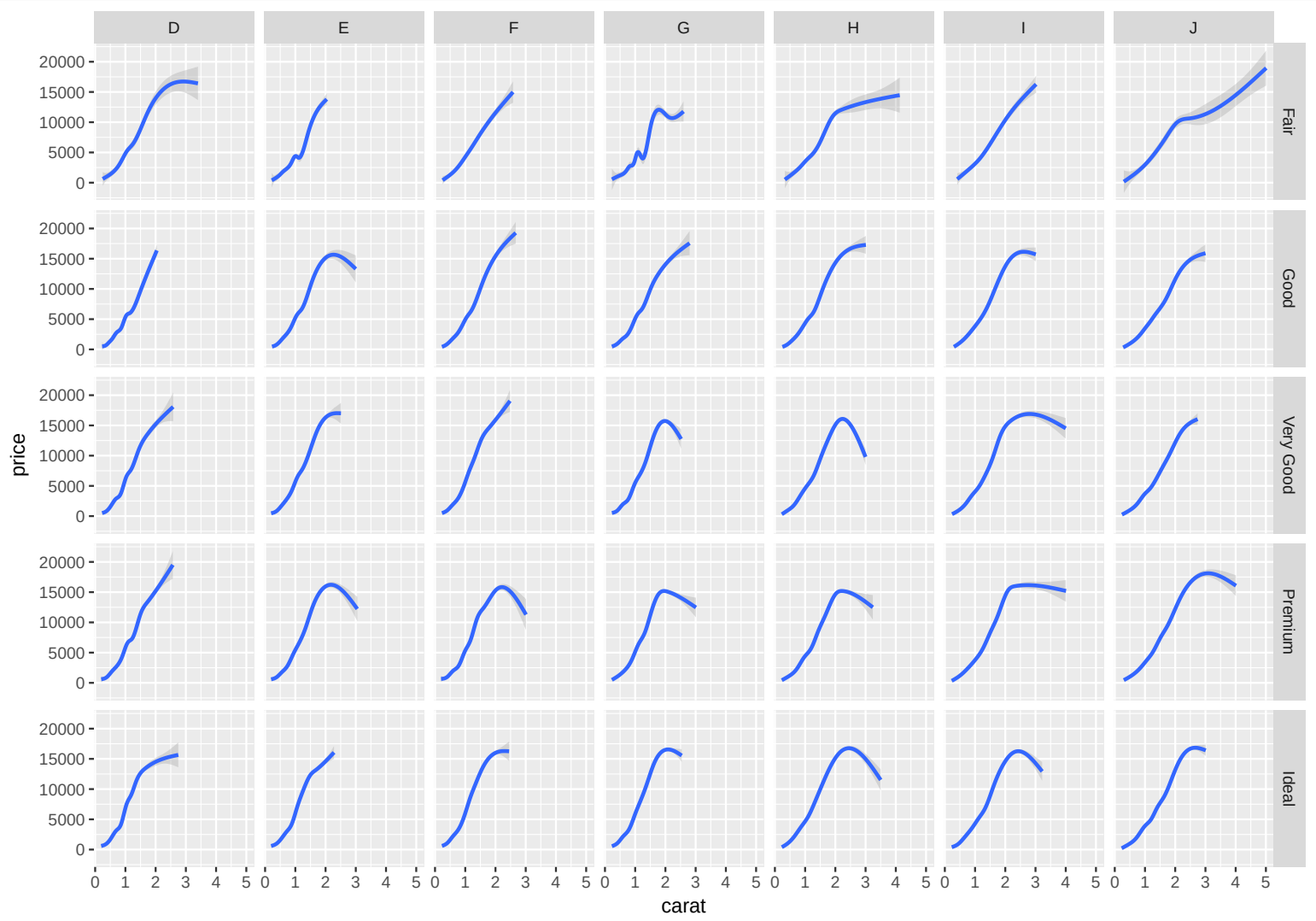
Plot 3


count
10000
5000
0
0
5000
10000
15000
20000
price
cut
Fair
Good
Very Good
Premium
Ideal

Plot 4

```
ggplot(data = diamonds,
       mapping = aes(x = carat, y = price)) +
  geom_smooth(alpha = 0.3) +
  facet_grid(cut ~ color)
```

Plot 4


D
E
F
G
H
I
J
Fair
Good
Very Good
Premium
Ideal
price
carat
20000
15000
10000
5000
0
0 1 2 3 4 5

Making sure everyone is set up

Packages

```
library(tidyverse)
```

Data

336,776 flights that departed from New York City in 2013

```
# install.packages("nycflights13")
library(nycflights13)
```

year	month	day	dep_time	sched_dep_time	dep_delay
2013	1	1	517	515	2
2013	1	1	533	529	4
2013	1	1	542	540	2
2013	1	1	544	545	-1

Data transformation with dplyr

Piping

The pipe operator %>% (Ctrl/Cmd+Shift+M) allows you to write code in sequences which has several benefits:

- serves the natural way of reading ("First this, then this, . . . ")
- avoids nested function calls
- minimizes the need for local variables and function definitions

Piping

`dplyr` is designed to work with the pipe, so this

```
df %>%
  select(x, y) %>%
  filter(year == 2017)
```

returns the sames as this

```
filter(select(df, x, y), year == 2017)
```

Variable types

- int: integers
- dbl: doubles, or real numbers
- chr: character vectors, or strings
- dttm: date-times (a date + a time)
- lgl: logical, vectors that contain only `TRUE` or `FALSE`
- fctr: factors
- date: dates

`dplyr` core functions

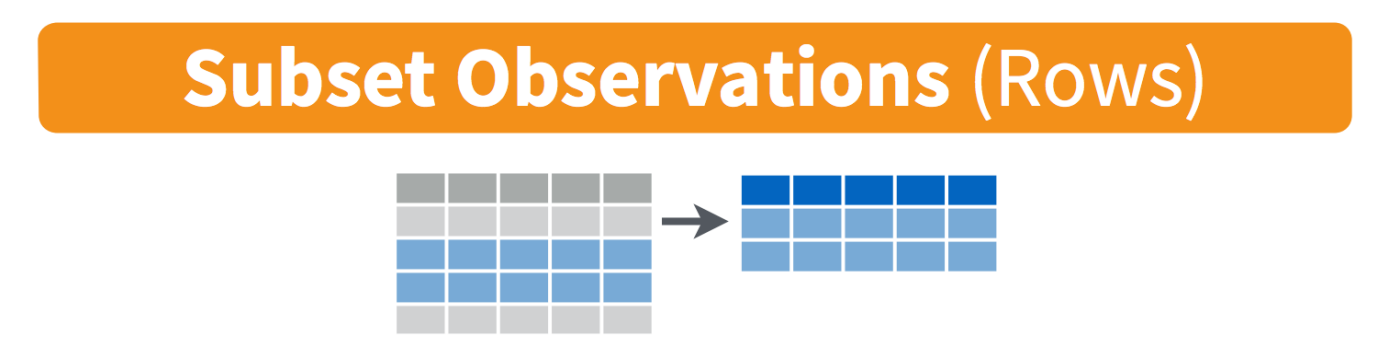
- `filter()`: select rows by their values
- `arrange()`: order rows
- `select()`: select columns by their names
- `mutate()`: create new variables
- `summarize()`: collapse many values down to a single summary
- `group_by()`: operate on it group-by-group
- `rename()`: rename columns
- `distinct()`: find distinct rows

Command structure (for all dplyr verbs):

- first argument is a data frame
- return value is a data frame
- nothing is modified in place

`filter()`

`filter()` allows to subset observations based on their values. The function takes logical expressions and returns the rows for which all are `TRUE`.



filter()

Let's select all flights on January 1st:

```
filter(flights, month == 1, day == 1)
```

year	month	day	dep_time	sched_dep_time	dep_delay
2013	1	1	517	515	2
2013	1	1	533	529	4
2013	1	1	542	540	2
2013	1	1	544	545	-1
2013	1	1	554	600	-6
2013	1	1	554	558	-4

filter()

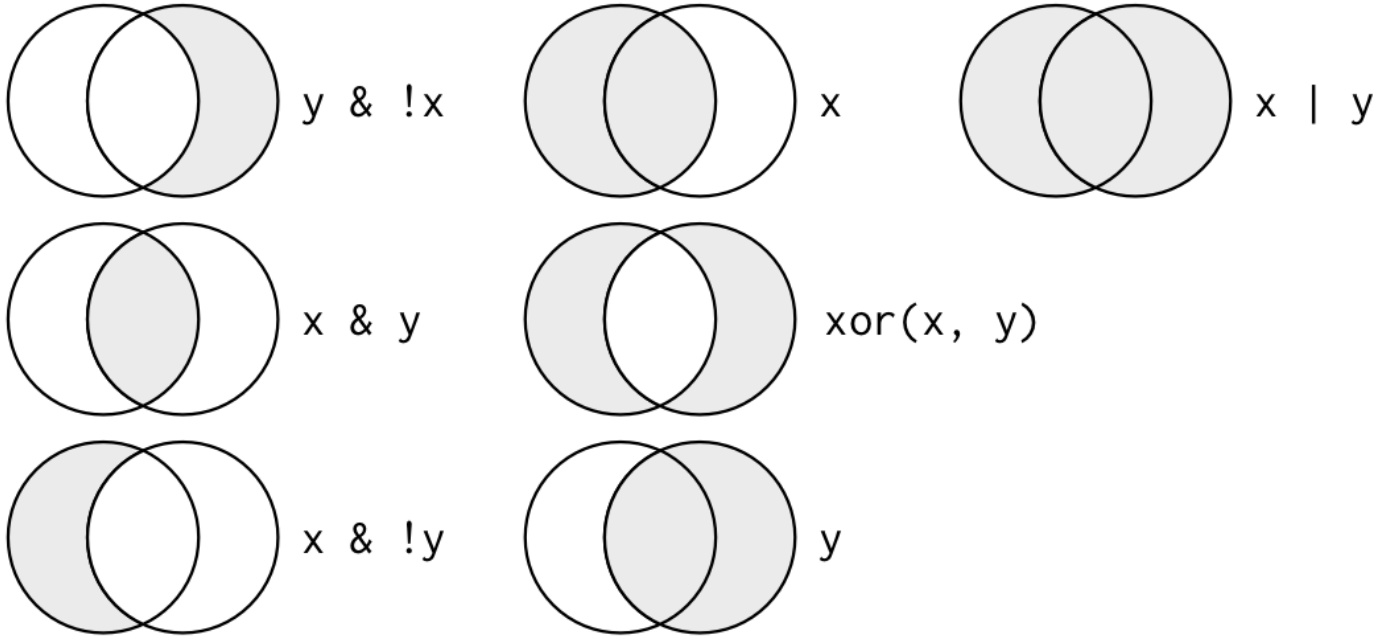
`filter()` revolves around using comparison operators: `>`, `>=`, `<`, `<=`, `!=` (not equal), and `==` (equal).

`dplyr` functions like `filter()` never modify inputs but instead return a new data frame that needs to be assigned to an object if you want to save the result.

```
jan1 <- filter(flights, month == 1, day == 1)
```

Boolean operators

`filter()` also supports the Boolean operators `&` ("and"), `|` ("or"), and `!` (is "not").



De Morgan's law: `!(x & y)` = `!x | !y` & `!(x | y)` = `!x & !y`

Boolean operators

Why does this not work?

```
filter(flights, month == 11 | 12)
```

Generally a good idea to use `x %in% y`, which will select every row where `x` is part of the values of `y`.

```
filter(flights, month %in% c(11, 12))
```

`between` condition

Another useful dplyr filtering helper is `between()`. `between(x, left, right)` is equivalent to `x >= left & x <= right`.

To `filter()` all flights that departed between midnight and 6am (inclusive):

```
filter(flights, between(dep_time, 0, 600))
```

`filter()` exclusion

`filter()` by default excludes `FALSE` and `NA` values.

```
df <- tibble(x = c(1, NA, 3))
filter(df, x > 1)
```

```
## # A tibble: 1 x 1
##       x
##   <dbl>
## 1     3
```

`filter()` exclusion

If you want to preserve missing values, you have to explicitly state it.

```
filter(df, is.na(x) | x > 1)
```

```
## # A tibble: 2 x 1
##       x
##   <dbl>
## 1    NA
## 2     3
```

`filter()` exercises

1. Find all flights that
    1.1 Had an arrival delay of two or more hours
    1.2 Arrived more than two hours late, but didn't leave late
    1.3 Flew to Houston (`IAH` or `HOU`)
    1.4 Were operated by United, American, or Delta
    1.5 Departed in summer (July, August, and September)

arrange()

`arrange()` takes a data frame and a set of column names to order the rows by. Multiple column names are evaluated subsequently.

```
arrange(flights, year, month, day)
```

year	month	day	dep_time	sched_dep_time	dep_delay
2013	1	1	517	515	2
2013	1	1	533	529	4
2013	1	1	542	540	2
2013	1	1	544	545	-1
2013	1	1	554	600	-6
2013	1	1	554	558	-4

`arrange()` in descending order

By dafault `arrange()` sorts values in ascending order. Use `desc()` to re-order by a column in descending order.

```
arrange(flights, desc(arr_delay))
```

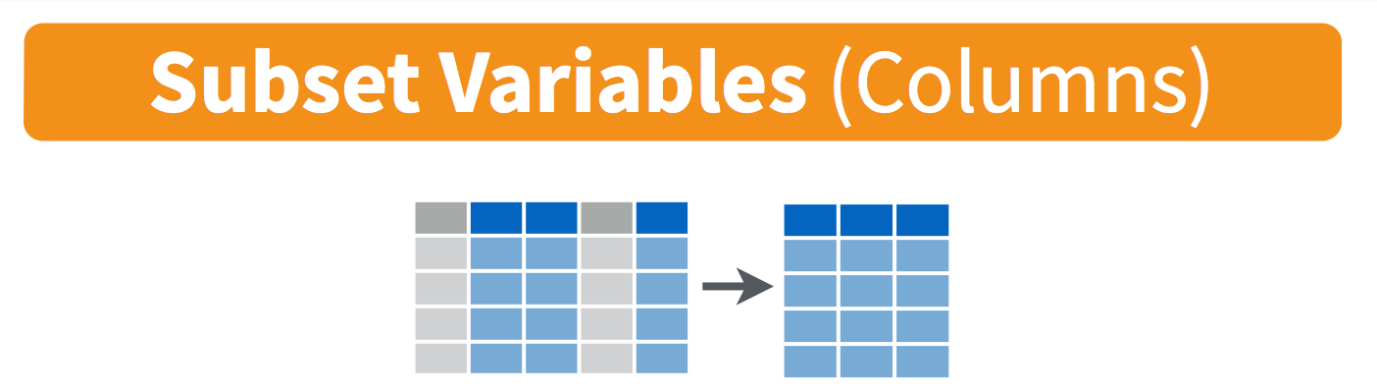
year	month	day	dep_time	sched_dep_time	dep_delay
2013	1	9	641	900	1301
2013	6	15	1432	1935	1137
2013	1	10	1121	1635	1126
2013	9	20	1139	1845	1014
2013	7	22	845	1600	1005
2013	4	10	1100	1900	960

`arrange()` exercises

2. Sort `flights` to
    2.1 find the flight that left earliest
    2.2 find the most delayed flight
    2.3 find the flight that travelled the longest and that travelled the shortest

select()

`select()` is used to select a subset of variables from a dataset.




```
select(flights, year, month, day)
```

year	month	day
2013	1	1
2013	1	1
2013	1	1
2013	1	1

`select()`

`select()` has various helper functions:

- `everything()`: selects all variables.
- `starts_with("abc")`: matches names that begin with “abc”.
- `ends_with("xyz")`: matches names that end with “xyz”.
- `contains("ijk")`: matches names that contain “ijk”.
- `matches("(.)\\1")`: selects variables that match a regular expression.
- `num_range("x", 1:3)` matches `x1`, `x2` and `x3`.

See `?select` for more details.

select()

You can use `select()` to rename variables

```
select(flights, tail_num = tailnum)
```

which will drop all of the variables not explicitly mentioned. Therefore it's better to use `rename()` instead:

```
rename(flights, tail_num = tailnum)
```

`select()` exercises

3.1 What are three distinct ways to select `dep_time`, `dep_delay`, `arr_time`, and `arr_delay` from `flights`.

3.2 What does the `one_of()` function do? Why might it be helpful in conjunction with this vector?

```
vars <- c("year", "month", "day", "dep_delay",
          "arr_delay")
```

`mutate()`

`mutate()` allows to add new columns to the end of your dataset that are functions of existing columns.

mutate()

```
flights %>%
  select(ends_with("delay"), distance, air_time) %>%
  mutate(gain = arr_delay - dep_delay,
         speed = distance / air_time * 60
)
```

dep_delay	arr_delay	distance	air_time	gain	speed
2	11	1400	227	9	370.0441
4	20	1416	227	16	374.2731
2	33	1089	160	31	408.3750
-1	-18	1576	183	-17	516.7213
-6	-25	762	116	-19	394.1379
-4	12	719	150	16	287.6000

transmute()

Use `transmute()` to only keep the new variables:

```
transmute(flights,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

Functions to use with `mutate()`

There are many functions for creating new variables with `mutate()`:

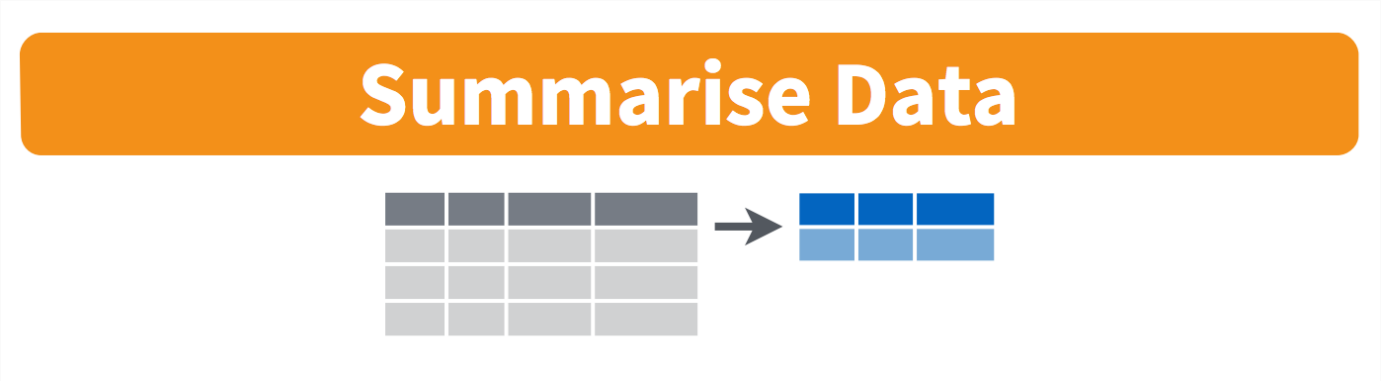
- Arithmetic operators: `+`, `-`, `*`, `/`, `^` (e.g. `air_time / 60`).
- Aggregate functions: `sum(x)` `mean(y)` (e.g. `mean(dep_delay)`).
- Modular arithmetic: `%/%` (integer division) and `%%` (remainder), where `x == y * (x %/% y) + (x %% y)`.
- Logs: `log()`, `log2()`, `log10()`.
- Offsets: `lead()` and `lag()` (e.g. `x - lag(x)`).
- Cumulative and rolling aggregates: `cumsum()`, `cumprod()`, `cummin()`, `cummax()`, `cummean()`.
- Logical comparisons, `<`, `<=`, `>`, `>=`, `!=`.
- Ranking: `min_rank()`, `row_number()`, `dense_rank()`, `percent_rank()`, `cume_dist()`, `ntile()`.

`mutate()` exercises

4. Use `mutate()` to
    - 4.1 Create new variables for `dep_time` and `sched_dep_time` that measure times in the number of minutes.
    - 4.2 Compare `air_time` with `arr_time - dep_time`. What do you see? What do you need to do to fix it?
    - 4.3 Find the 10 most delayed flights using a ranking function. How do you want to handle ties?

summarize()

`summarize()` collapses a data frame to a single row.

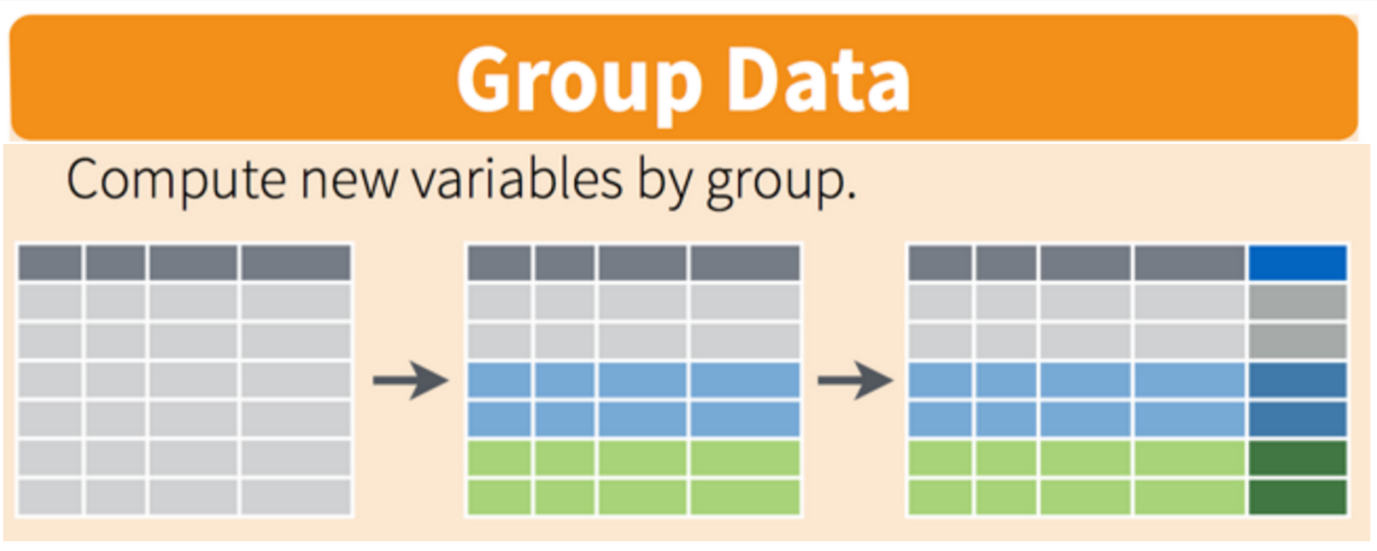



```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##       delay
##       <dbl>
## 1 12.63907
```

`summarize()` with `group_by()`

`summarize()` is most effectively used with `group_by()`, which changes the unit of analysis from the complete dataset to individual groups.



Grouping is most useful in conjunction with `summarise()`, but you can also do convenient operations with `mutate()` and `filter()`.

summarize() with group_by()

For example, to get the average delay per date

```
flights %>%
  group_by(year, month, day) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE))
```

`summarize()` count

For aggregations it is generally a good idea to include a count `n()`. For example, let's look at the (not cancelled) planes that have the highest average delays:

```
flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))
  group_by(tailnum) %>%
  summarise(delay = mean(arr_delay)) %>%
  arrange(delay)
```

`summarize()` useful functions

There are a number of useful summary functions:

- Measures of location: `mean(x)`, `sum(x)`, `median(x)`.
- Measures of spread: `sd(x)`, `IQR(x)`, `mad(x)`.
- Measures of rank: `min(x)`, `quantile(x, 0.25)`, `max(x)`.
- Measures of position: `first(x)`, `nth(x, 2)`, `last(x)`.
- Counts: `n()`, `sum(!is.na(x))`, `n_distinct(x)`.
- Counts and proportions of logical values: `sum(x > 10)`, `mean(y == 0)`.

`summarize()` exercises

5. Use `summarize()` to
    5.1 Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?
    5.2 Find the carrier with the worst delays.

Homework Exercises

Homework Exercises

For this week's homework exersises go to Moodle and answer the Quiz posted in the Data Transformation section. You will be asked a number of questions randomly selected from a question pool. If you work in pairs, then you might get two different sets of questions.

Deadline: Sunday, September 24 before midnight.

That's it for today. Questions?