Statistical Methods for Insurance: Compiling data for problem solving

Di Cook & Souhaib Ben Taieb, Econometrics and Business Statistics, Monash University

W8.C1

Overview of this class

- What is `tidy data`? Why do you want tidy data? Getting your data into tidy form using tidyr.
- Wrangling verbs: `filter`, `arrange`, `select`, `mutate`, `summarise`, with dplyr
- Date and time with lubridate

Terminology

1. `Cases, records, individuals, subjects, experimental units, example, instance`: things we are collecting information about
2. `Variables, attributes, fields, features`: what we are measuring on each record/case/…/instance

Generally we think of cases being on the rows, and variables being in the columns of a table. This is a basic data structure. BUT data often is given to us in many other shapes than this. Getting into a tidy shape will allow you to efficiently use it for modeling.

Example 1

Inst	AvNumPubs	AvNumCits	PctCompletion
ARIZONA STATE UNIVERSITY	0.90	1.57	31.7
AUBURN UNIVERSITY	0.79	0.64	44.4
BOSTON COLLEGE	0.51	1.03	46.8
BOSTON UNIVERSITY	0.49	2.66	34.2

- Cases: __________
- Variables: __________

Example 2

V1	V2	V3	V4	V5	V9	V13	V17	V21	V25	V29	V33	V37	V41	V45	V49	V53	V57
ASN00086282	1970	7	TMAX	141	124	113	123	148	149	139	153	123	108	119	112	126	112
ASN00086282	1970	7	TMIN	80	63	36	57	69	47	84	78	49	42	48	56	51	36
ASN00086282	1970	7	PRCP	3	30	0	0	36	3	0	0	10	23	3	0	5	0
ASN00086282	1970	8	TMAX	145	128	150	122	109	112	116	142	166	127	117	127	159	143

- Cases: __________
- Variables: __________

Example 3

Here are the column headers …

```
#>  [1] "iso2"   "year"   "m_04"   "m_514"  "m_014"  "m_1524" "m_2534"
#>  [8] "m_3544" "m_4554" "m_5564" "m_65"   "m_u"    "f_04"   "f_514"
#> [15] "f_014"  "f_1524" "f_2534" "f_3544" "f_4554" "f_5564" "f_65"
#> [22] "f_u"
```

- Cases: __________
- Variables: __________

Example 4

We'll commonly find these data on web sites:

religion	<$10k	$10-20k	$20-30k	$30-40k
Agnostic	27	34	60	81
Atheist	12	27	37	52
Buddhist	27	21	30	34
Catholic	418	617	732	670
Don't know/refused	15	14	15	11

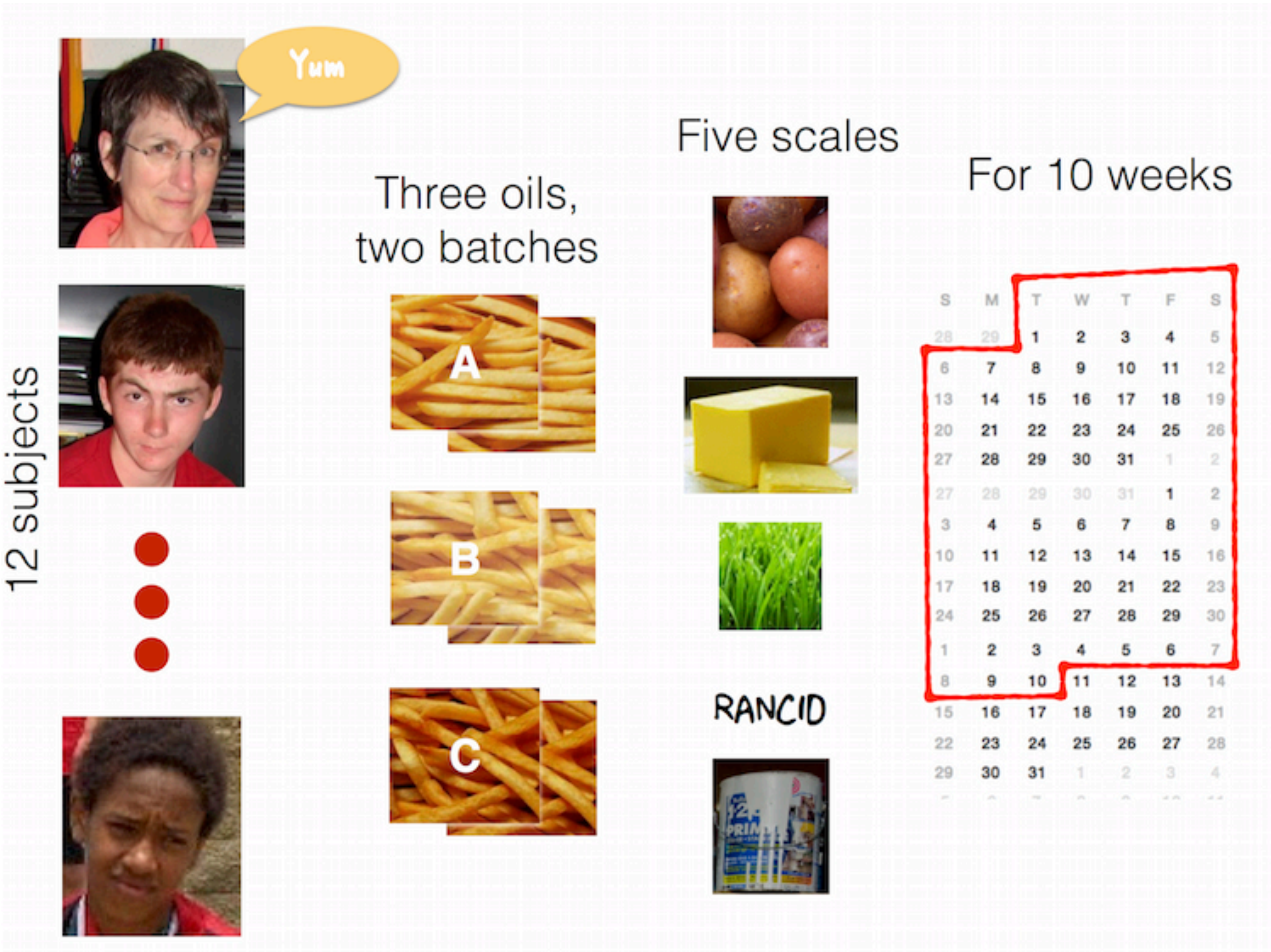
- Cases: __________
- Variables: __________

Example 5

10 week sensory experiment, 12 individuals assessed taste of french fries on several scales (how potato-y, buttery, grassy, rancid, paint-y do they taste?), fried in one of 3 different oils, replicated twice. First few rows:

time	treatment	subject	rep	potato	buttery	grassy	rancid	painty
1	1	3	1	2.9	0.0	0.0	0.0	5.5
1	1	3	2	14.0	0.0	0.0	1.1	0.0
1	1	10	1	11.0	6.4	0.0	0.0	0.0
1	1	10	2	9.9	5.9	2.9	2.2	0.0

What do you like to know?



Yum
12 subjects
Three oils,
two batches
A
B
C
Five scales
RANCID
For 10 weeks

Messy Data Patterns

There are various features of messy data that one can observe in practice. Here are some of the more commonly observed patterns.

- Column headers are values, not variable names
- Variables are stored in both rows and columns, contingency table format
- Information stored in multiple tables
- Dates in many different formats
- Not easy to analyse

What is Tidy Data?

- Each observation forms a row
- Each variable forms a column
- Contained in a single table
- Long form makes it easier to reshape in many different ways
- Wide form is common for analysis/modeling


Description by Hadley Wickham
Tidy data = lego
http://www.flickr.com/photos/wwworks/2473052504


Description by Hadley Wickham
Messy data = play mobile
https://www.flickr.com/photos/kafka4prez/57282282

Tidy vs Messy

- Tidy data facilitates analysis in many different ways, answering multiple questions, applying methods to new data or other problems
- Messy data may work for one particular problem but is not generalisable

Tidy Verbs

- `gather`: specify the `keys` (identifiers) and the `values` (measures) to make long form (used to be called melting)
- `spread`: variables in columns (used to be called casting)
- nest/unnest: working with lists
- separate/unite: split and combine columns

French fries example

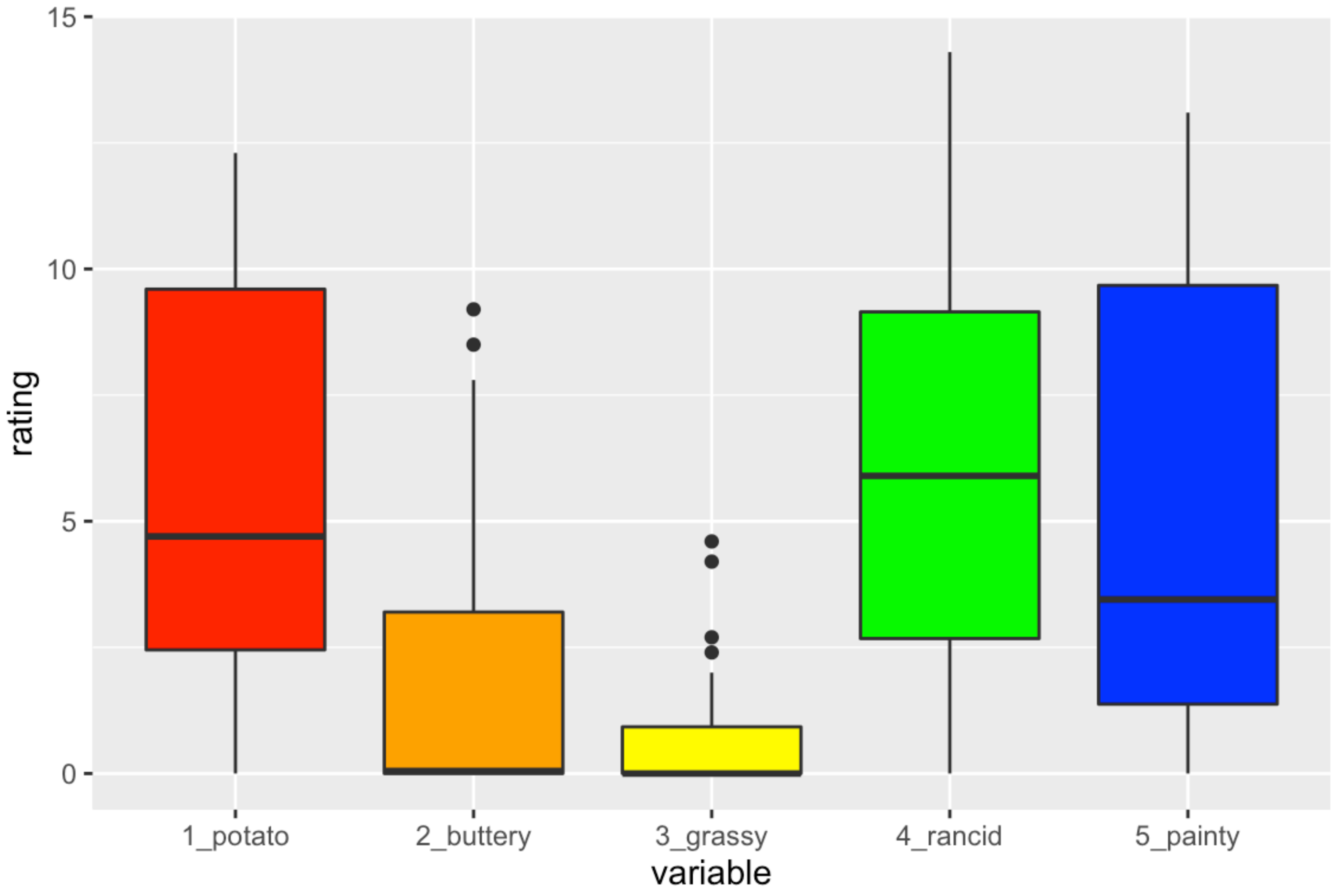
	time	treatment	subject	rep	potato	buttery	grassy	rancid	painty
61	1	1	3	1	2.9	0.0	0.0	0.0	5.5
25	1	1	3	2	14.0	0.0	0.0	1.1	0.0
62	1	1	10	1	11.0	6.4	0.0	0.0	0.0
26	1	1	10	2	9.9	5.9	2.9	2.2	0.0
63	1	1	15	1	1.2	0.1	0.0	1.1	5.1
27	1	1	15	2	8.8	3.0	3.6	1.5	2.3

This format is not ideal for data analysis

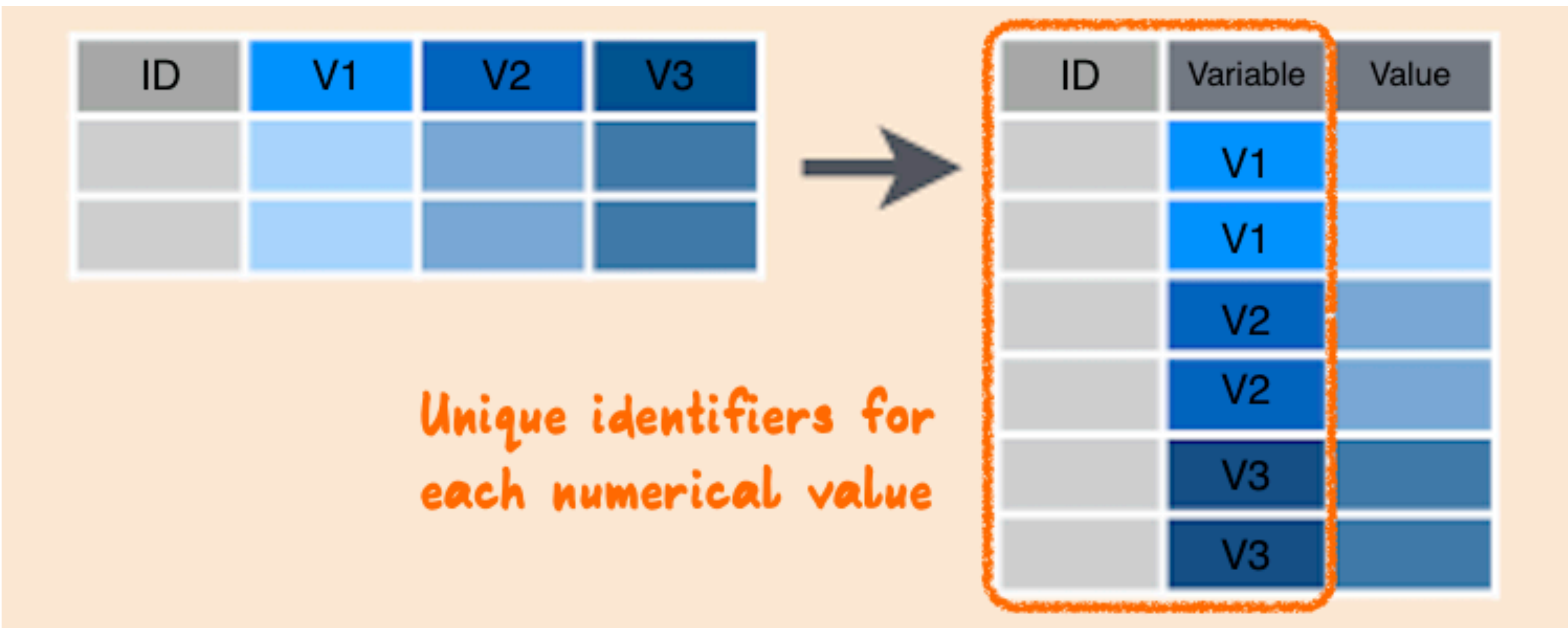
What code would be needed to plot each of the ratings over time as a different color?

```
library(ggplot2)
french_sub <- french_fries[french_fries$time == 10,]
ggplot(data = french_sub) +
  geom_boxplot(aes(x="1_potato", y=potato), fill = I("red")) +
 geom_boxplot(aes(x = "2_buttery", y = buttery), fill = I("orange")) +
 geom_boxplot(aes(x = "3_grassy", y = grassy), fill = I("yellow")) +
 geom_boxplot(aes(x = "4_rancid", y = rancid), fill = I("green")) +
 geom_boxplot(aes(x = "5_painty", y = painty), fill = I("blue")) +
    xlab("variable") + ylab("rating")
```

The plot


15
10
5
0
rating
1_potato
2_buttery
3_grassy
4_rancid
5_painty
variable

Wide to long


ID
V1
V2
V3
ID
Variable
Value
V1
V1
V2
V2
V3
V3
Unique identifiers for each numerical value

Gathering

- When gathering, you need to specify the keys (identifiers) and the values (measures).
- Keys/Identifiers:
  - Identify a record (must be unique)
  - Example: Indices on an random variable
  - Fixed by design of experiment (known in advance)
  - May be single or composite (may have one or more variables)
- Values/Measures:
  - Collected during the experiment (not known in advance)
  - Usually numeric quantities

Gathering the French Fries Data

```
ff_long <- gather(french_fries, key = variable,
  value = rating, potato:painty)

head(ff_long)
```

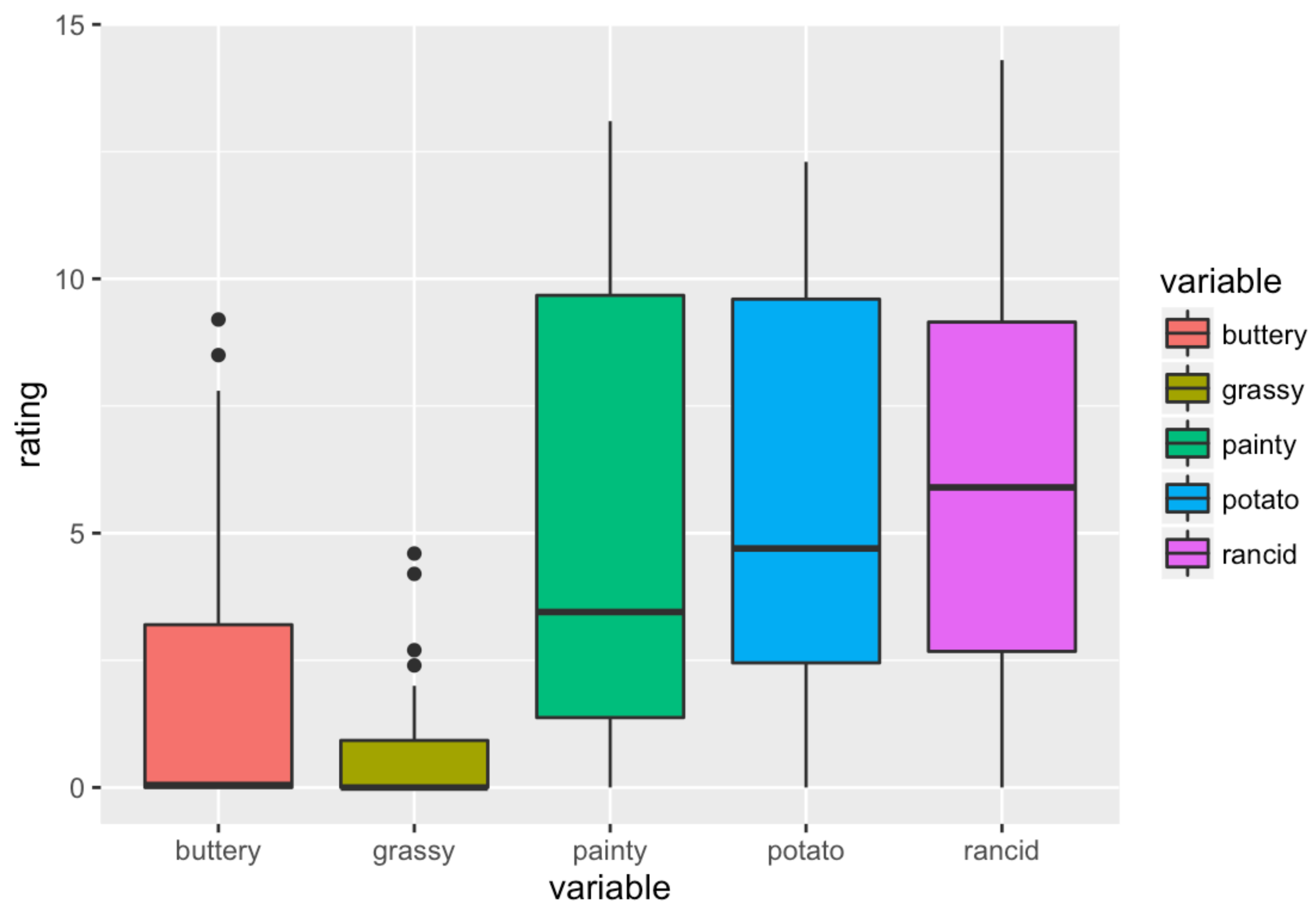
```
#>   time treatment subject rep variable rating
#> 1    1         1       3   1   potato    2.9
#> 2    1         1       3   2   potato   14.0
#> 3    1         1      10   1   potato   11.0
#> 4    1         1      10   2   potato    9.9
#> 5    1         1      15   1   potato    1.2
#> 6    1         1      15   2   potato    8.8
```

Let's re-write the code for our Plot

```
ff_long_sub <- ff_long[
  french_fries_long$time == 10,]

ggplot(data = ff_long_sub,
  aes(x=variable, y=rating,  fill = variable)) +
   geom_boxplot()
```

And plot it



15
10
5
0
rating
buttery
grassy
painty
potato
rancid
variable
variable
buttery
grassy
painty
potato
rancid

Long to Wide

In certain applications, we may wish to take a long dataset and convert it to a wide dataset (Perhaps displaying in a table).

```
#>   time treatment subject rep variable rating
#> 1    1         1       3   1   potato    2.9
#> 2    1         1       3   2   potato   14.0
#> 3    1         1      10   1   potato   11.0
#> 4    1         1      10   2   potato    9.9
#> 5    1         1      15   1   potato    1.2
#> 6    1         1      15   2   potato    8.8
```

Spread

We use the spread function from tidyr to do this:

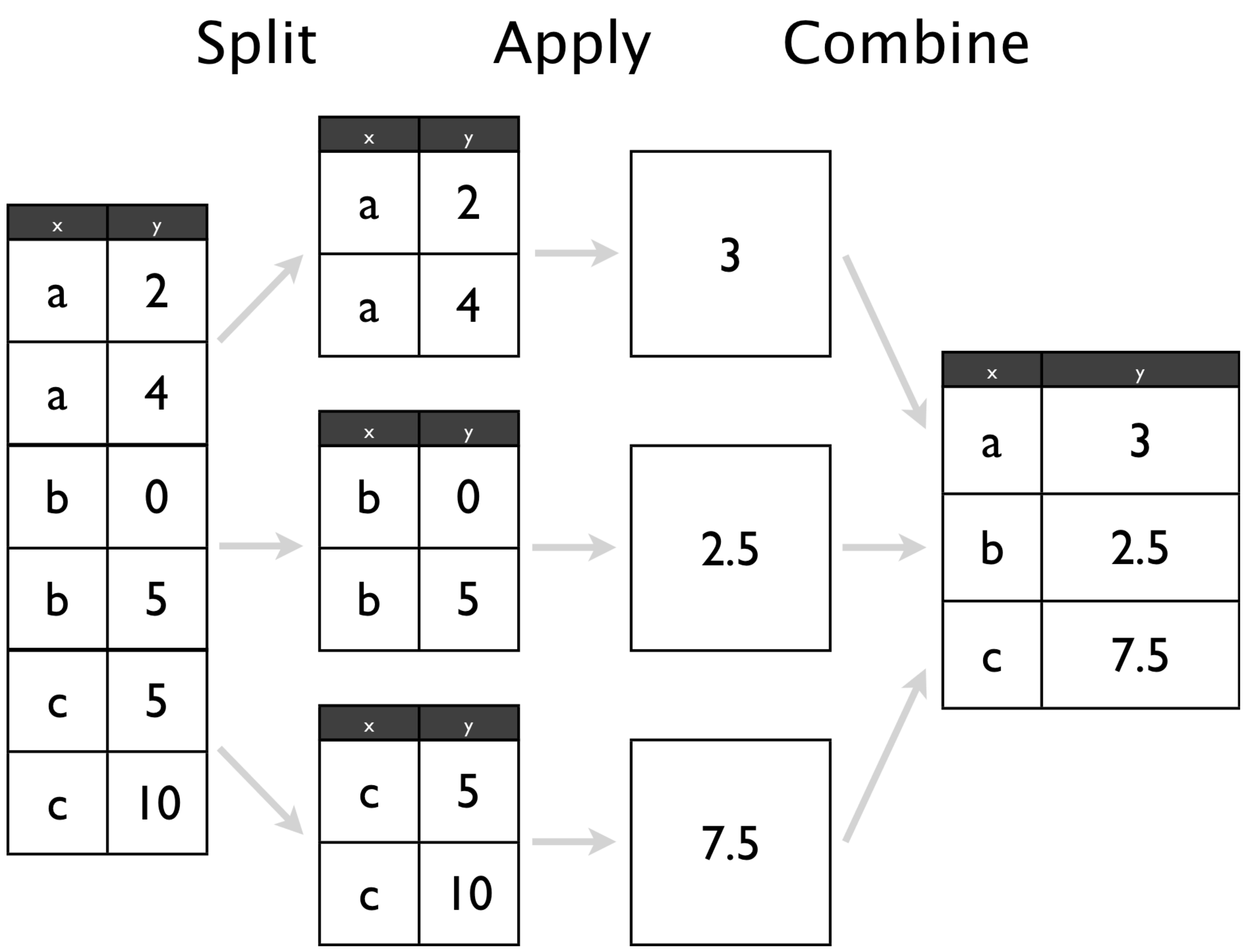
```
ff_wide <- spread(ff_long,
  key = variable, value = rating)
head(ff_wide)
```

```
#>   time treatment subject rep buttery grassy painty potato rancid
#> 1    1         1       3   1     0.0    0.0    5.5    2.9    0.0
#> 2    1         1       3   2     0.0    0.0    0.0   14.0    1.1
#> 3    1         1      10   1     6.4    0.0    0.0   11.0    0.0
#> 4    1         1      10   2     5.9    2.9    0.0    9.9    2.2
#> 5    1         1      15   1     0.1    0.0    5.1    1.2    1.1
#> 6    1         1      15   2     3.0    3.6    2.3    8.8    1.5
```

The Split-Apply-Combine Approach

- Split a dataset into many smaller sub-datasets
- Apply some function to each sub-dataset to compute a result
- Combine the results of the function calls into a one dataset

The Split-Apply-Combine Approach



Split-Apply-Combine in dplyr

```
library(dplyr)
ff_summary <- group_by(ff_long, variable) %>% # SPLIT
  summarise(
      m = mean(rating, na.rm = TRUE),
      s=sd(rating, na.rm=TRUE)) # APPLY + COMBINE
ff_summary
```

```
#> # A tibble: 5 x 3
#>   variable         m        s
#>      <chr>     <dbl>    <dbl>
#> 1  buttery 1.8236994 2.409758
#> 2   grassy 0.6641727 1.320574
#> 3   painty 2.5217579 3.393717
#> 4   potato 6.9525180 3.584403
#> 5   rancid 3.8522302 3.781815
```

Pipes

- Pipes historically enable data analysis pipelines
- Pipes allow the code to be read like a sequence of operations
- dplyr allows us to chain together these data analysis tasks using the `%>%` (pipe) operator
- `x %>% f(y)` is shorthand for `f(x, y)`
- Example:

```
student2012.sub <- readRDS("../data/student_sub.rds")
student2012.sub %>%
  group_by(CNT) %>%
  tally()
#> # A tibble: 43 x 2
#>      CNT     n
#>    <chr> <int>
#> 1    ARE 11500
#> 2    AUS 14481
#> 3    AUT  4755
#> 4    BEL  8597
#> 5    BGR  5282
#> 6    BRA  5506
#> 7    CAN 21544
#> 8    CHL  6856
#> 9    COL  9073
#> 10   CZE  5327
#> # ... with 33 more rows
```

dplyr verbs

There are five primary dplyr `verbs`, representing distinct data analysis tasks:

- `Filter`: Remove the rows of a data frame, producing subsets
- `Arrange`: Reorder the rows of a data frame
- `Select`: Select particular columns of a data frame
- `Mutate`: Add new columns that are functions of existing columns
- `Summarise`: Create collapsed summaries of a data frame

Filter

```
french_fries %>%
    filter(subject == 3, time == 1)
```

```
#>   time treatment subject rep potato buttery grassy rancid painty
#> 1    1         1       3   1    2.9     0.0    0.0    0.0    5.5
#> 2    1         1       3   2   14.0     0.0    0.0    1.1    0.0
#> 3    1         2       3   1   13.9     0.0    0.0    3.9    0.0
#> 4    1         2       3   2   13.4     0.1    0.0    1.5    0.0
#> 5    1         3       3   1   14.1     0.0    0.0    1.1    0.0
#> 6    1         3       3   2    9.5     0.0    0.6    2.8    0.0
```

Arrange

```
french_fries %>%
    arrange(desc(rancid)) %>%
    head
```

```
#>   time treatment subject rep potato buttery grassy rancid painty
#> 1    9         2      51   1    7.3     2.3      0   14.9    0.1
#> 2   10         1      86   2    0.7     0.0      0   14.3   13.1
#> 3    5         2      63   1    4.4     0.0      0   13.8    0.6
#> 4    9         2      63   1    1.8     0.0      0   13.7   12.3
#> 5    5         2      19   2    5.5     4.7      0   13.4    4.6
#> 6    4         3      63   1    5.6     0.0      0   13.3    4.4
```

Select

```
french_fries %>%
    select(time, treatment, subject, rep, potato) %>%
    head
```

```
#>    time treatment subject rep potato
#> 61    1         1       3   1    2.9
#> 25    1         1       3   2   14.0
#> 62    1         1      10   1   11.0
#> 26    1         1      10   2    9.9
#> 63    1         1      15   1    1.2
#> 27    1         1      15   2    8.8
```

Mutate

```
french_fries %>%
    mutate(yucky = grassy+rancid+painty) %>%
  head
```

```
#>   time treatment subject rep potato buttery grassy rancid painty yucky
#> 1    1         1       3   1    2.9     0.0    0.0    0.0    5.5   5.5
#> 2    1         1       3   2   14.0     0.0    0.0    1.1    0.0   1.1
#> 3    1         1      10   1   11.0     6.4    0.0    0.0    0.0   0.0
#> 4    1         1      10   2    9.9     5.9    2.9    2.2    0.0   5.1
#> 5    1         1      15   1    1.2     0.1    0.0    1.1    5.1   6.2
#> 6    1         1      15   2    8.8     3.0    3.6    1.5    2.3   7.4
```

Summarise

```
french_fries %>%
    group_by(time, treatment) %>%
    summarise(mean_rancid = mean(rancid),
    sd_rancid = sd(rancid))
```

```
#> Source: local data frame [30 x 4]
#> Groups: time [?]
#>
#>      time treatment mean_rancid sd_rancid
#>    <fctr>    <fctr>       <dbl>     <dbl>
#> 1       1         1    2.758333  3.212870
#> 2       1         2    1.716667  2.714801
#> 3       1         3    2.600000  3.202037
#> 4       2         1    3.900000  4.374730
#> 5       2         2    2.141667  3.117540
#> 6       2         3    2.495833  3.378767
#> 7       3         1    4.650000  3.933358
#> 8       3         2    2.895833  3.773532
#> 9       3         3    3.600000  3.592867
#> 10      4         1    2.079167  2.394737
#> # ... with 20 more rows
```

Dates and Times

- Dates are deceptively hard to work with
- 02/05/2012. Is it February 5th, or May 2nd?
- Time zones
- Different starting times of stock markets, airplane departure and arrival

Basic Lubridate Use

```
library(lubridate)

now()
#> [1] "2016-09-14 19:52:52 AEST"
now(tz = "America/Chicago")
#> [1] "2016-09-14 04:52:52 CDT"
today()
#> [1] "2016-09-14"
now() + hours(4)
#> [1] "2016-09-14 23:52:52 AEST"
today() - days(2)
#> [1] "2016-09-12"
ymd("2013-05-14")
#> [1] "2013-05-14"
mdy("05/14/2013")
#> [1] "2013-05-14"
dmy("14052013")
#> [1] "2013-05-14"
```

Dates example: Oscars date of birth

```
oscars <- read_csv("../data/oscars.csv")
oscars <- oscars %>% mutate(DOB = mdy(DOB))
head(oscars$DOB)
#> [1] "1895-09-30" "1884-07-23" "1894-04-23" "2006-10-06" "1886-02-02"
#> [6] "1892-04-08"
summary(oscars$DOB)
#>         Min.      1st Qu.       Median         Mean      3rd Qu.
#> "1868-04-10" "1934-09-18" "1957-06-23" "1962-05-21" "2008-04-05"
#>         Max.
#> "2029-12-13"
```

Calculating on dates

- You should never ask a woman her age, but … really!

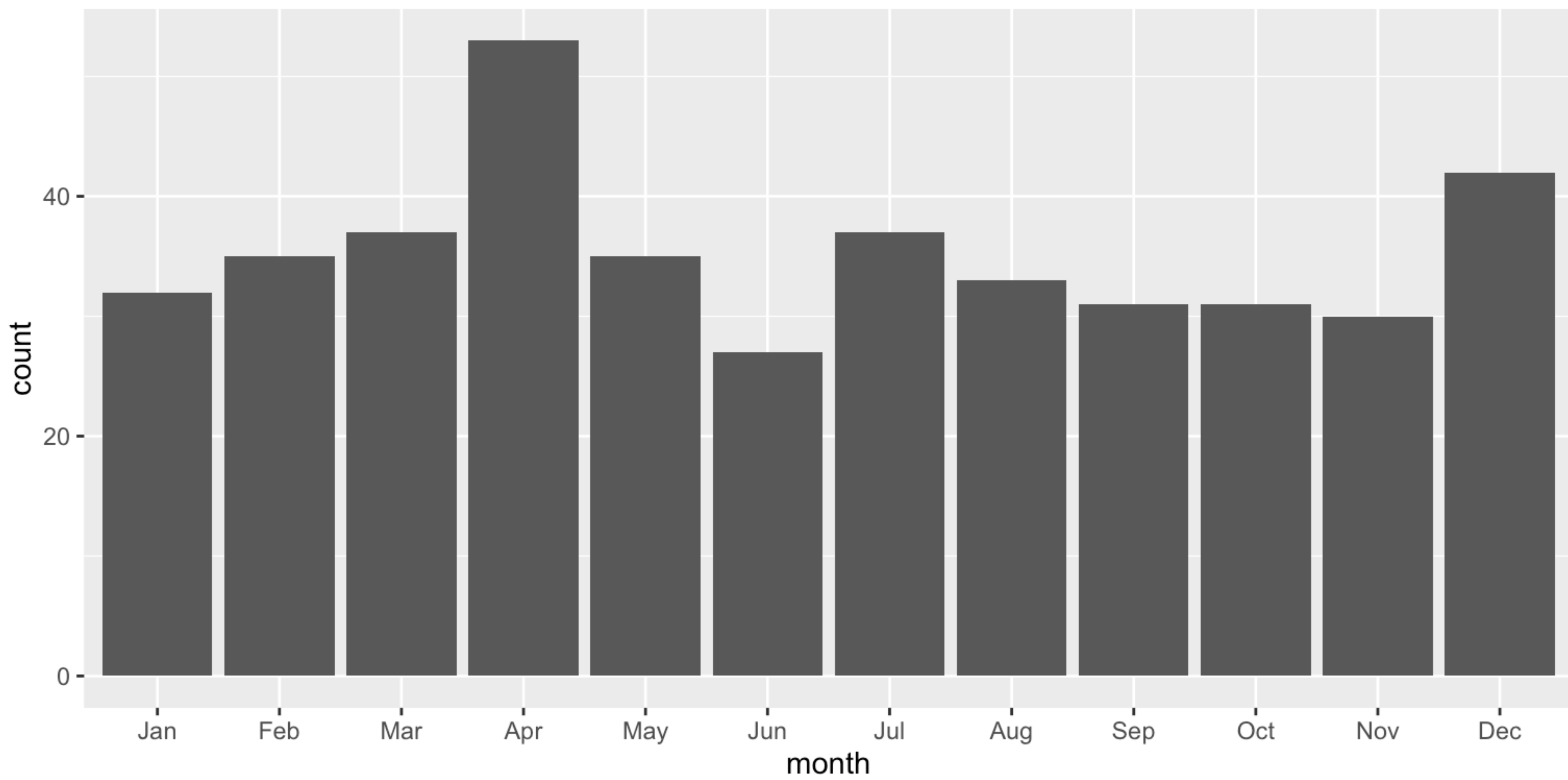
```
oscars <- oscars %>% mutate(year=year(DOB))
summary(oscars$year)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>    1868    1934    1957    1962    2008    2029
oscars %>% filter(year == "2029") %>%
  select(Name, Sex, DOB)
#> # A tibble: 4 x 3
#>                  Name    Sex        DOB
#>                 <chr>  <chr>     <date>
#> 1      Audrey Hepburn Female 2029-05-04
#> 2         Grace Kelly Female 2029-11-12
#> 3       Miyoshi Umeki Female 2029-04-03
#> 4 Christopher Plummer   Male 2029-12-13
```

Months

```
oscars <- oscars %>% mutate(month=month(DOB, label = TRUE, abbr = TRUE))
table(oscars$month)
#> 
#> Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec 
#>  32  35  37  53  35  27  37  33  31  31  30  42
```

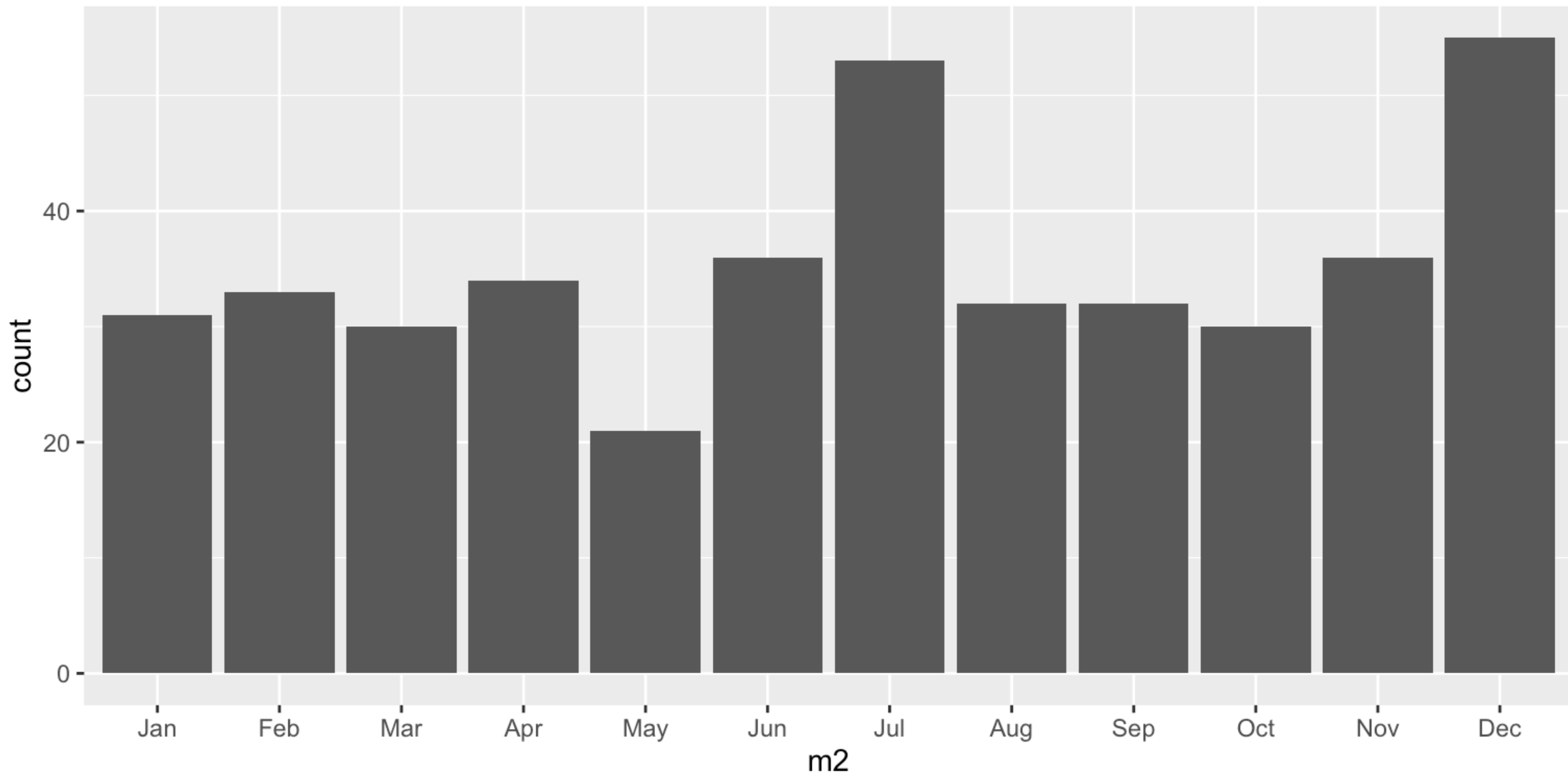
Now plot it

```
ggplot(data=oscars, aes(month)) + geom_bar()
```



Should you be born in April?

```
df <- data.frame(m=sample(1:12, 423, replace=TRUE))
df$m2 <- factor(df$m, levels=1:12,
  labels=month.abb)
ggplot(data=df, aes(x=m2)) + geom_bar()
```



Resources

- [Tidy data]
- [Split-apply-combine]
- [RStudio cheat sheets]
- [Working with dates and times]
- [R for Data Science]

Share and share alike