MONASH University

MONASH BUSINESS SCHOOL

ETC3250 Business Analytics: Data Wrangling

Week 7, Class 1

Souhaib Ben Taieb, Di Cook

GROUP OF EIGHT AUSTRALIA

AACSB ACCREDITED EFMD EQUIS ACCREDITED ASSOCIATION OF MBAS ACCREDITED

Outline

- What is `tidy data`? Why do you want tidy data? Getting your data into tidy form using tidyr.
- Writing readable code using `pipes`
- Wrangling verbs: `filter`, `arrange`, `select`, `mutate`, `summarise`, with dplyr
- Date and time with lubridate
- String operations, working with `text`
- Reading different `data formats`
- Handling missing data

Your turn 1

What are the variables?

Inst	AvNumPubs	AvNumCits	PctCompleti
ARIZONA STATE UNIVERSITY	0.90	1.57	31
AUBURN UNIVERSITY	0.79	0.64	44
BOSTON COLLEGE	0.51	1.03	46
BOSTON UNIVERSITY	0.49	2.66	34
BRANDEIS UNIVERSITY	0.30	3.03	48
BROWN UNIVERSITY	0.84	2.31	54

Your turn 2

What's in the column names of this data? What are the experimental units? What are the measured variables?

id	WI-6.R1	WI-6.R2	WI-6.R4	WM-6.R1	WM-6.R2	WI-12
Gene 1	2.182424	2.2042219	4.195636	2.6273345	5.063641	4.540
Gene 2	1.464224	0.5854472	1.859238	0.5152242	2.882808	1.364
Gene 3	2.031792	0.8701078	3.281983	0.5330452	4.627315	2.182

Your turn 3

What are the variables? What are the records?

```
melbtemp <- read.fwf("../data/ASN00086282.dly",
   c(11, 4, 2, 4, rep(c(5, 1, 1, 1), 31)), fill=T)
kable(head(melbtemp[,c(1,2,3,4,seq(5,128,4))]))
```

V1	V2	V3	V4	V5	V9	V13	V17	V21	V25
ASN00086282	1970	7	TMAX	141	124	113	123	148	149
ASN00086282	1970	7	TMIN	80	63	36	57	69	47
ASN00086282	1970	7	PRCP	3	30	0	0	36	3
ASN00086282	1970	8	TMAX	145	128	150	122	109	112
ASN00086282	1970	8	TMIN	50	61	75	67	41	51
ASN00086282	1970	8	PRCP	0	66	0	53	13	3

Your turn 4

What are the variables? What are the experimental units?

```
tb <- read_csv("../data/tb.csv")
#tail(tb)
colnames(tb)
#>  [1] "iso2"   "year"   "m_04"   "m_514"  "m_014"  "m_1524"
#>  [8] "m_3544" "m_4554" "m_5564" "m_65"   "m_u"    "f_04"
#> [15] "f_014"  "f_1524" "f_2534" "f_3544" "f_4554" "f_5564"
#> [22] "f_u"
```

Your turn 5

What are the variables? What are the experimental units?

```
pew <- read.delim(
  file = "http://stat405.had.co.nz/data/pew.txt",
  header = TRUE,
  stringsAsFactors = FALSE,
  check.names = F
)
kable(pew[1:5, 1:5])
```

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

Your turn 6

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 is the experimental unit? What are the factors of the experiment? What was measured? What do you want to know?

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
- One type of experimental unit stored in multiple tables
- Dates in many different formats

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

Tidy Data

Figure 1:

Messy Data

Figure 2:

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

During a ten week sensory experiment, 12 individuals were asked to assess taste of french fries (HOT CHIPS!) on several scales (how potato-y, buttery, grassy, rancid, paint-y do the fries taste?)
French fries were fried in one of three different oils, and each week individuals had to assess six batches of french fries (all three oils, replicated twice)

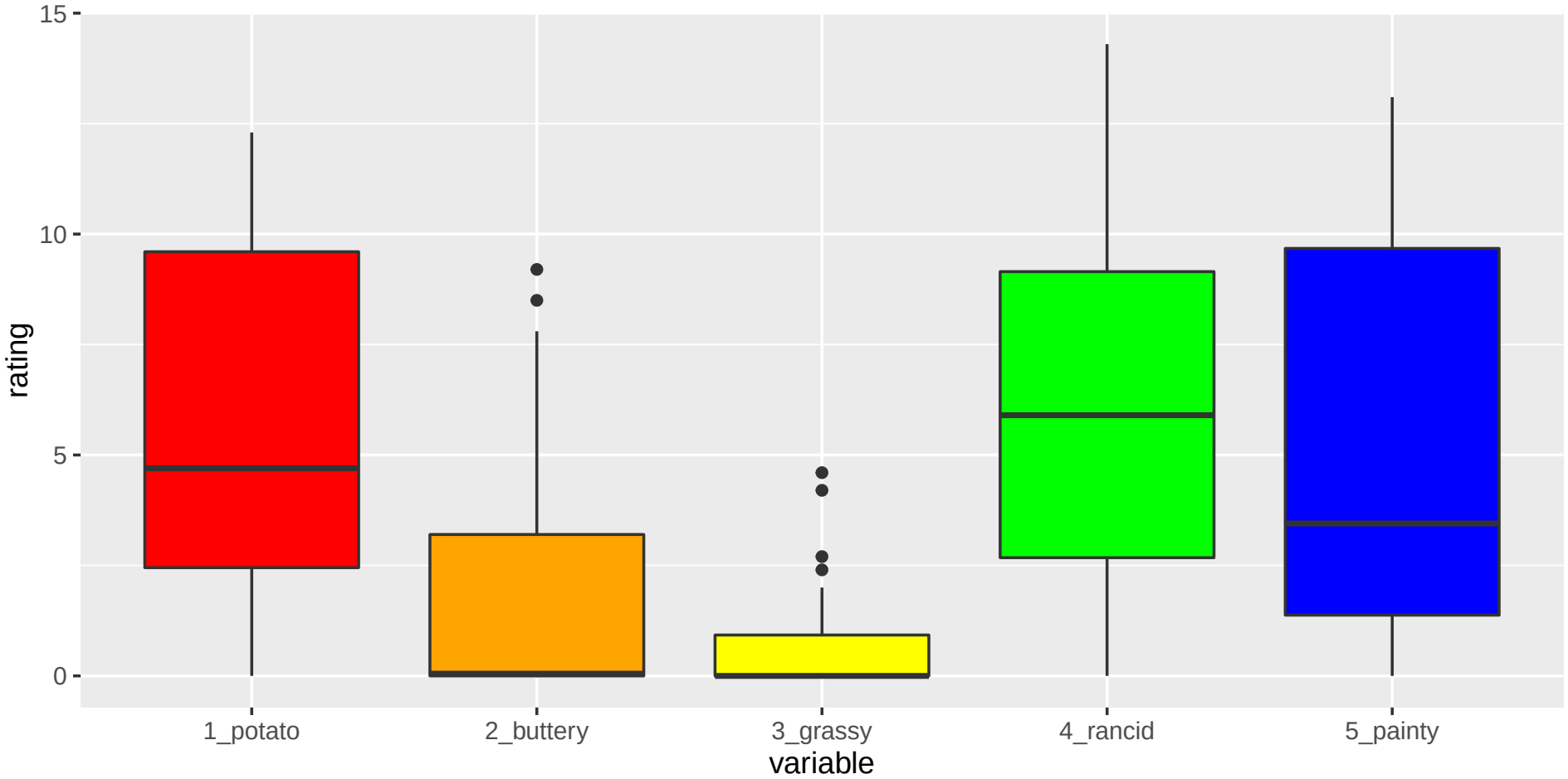
	time	treatment	subject	rep	potato	buttery	grassy	rancid	p
61	1	1	3	1	2.9	0.0	0.0	0.0	
25	1	1	3	2	14.0	0.0	0.0	1.1	
62	1	1	10	1	11.0	6.4	0.0	0.0	
26	1	1	10	2	9.9	5.9	2.9	2.2	
63	1	1	15	1	1.2	0.1	0.0	1.1	
27	1	1	15	2	8.8	3.0	3.6	1.5	

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("ora
 geom_boxplot(aes(x = "3_grassy", y = grassy), fill = I("yello
 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

What we have ..

We want to change this **wide format**:

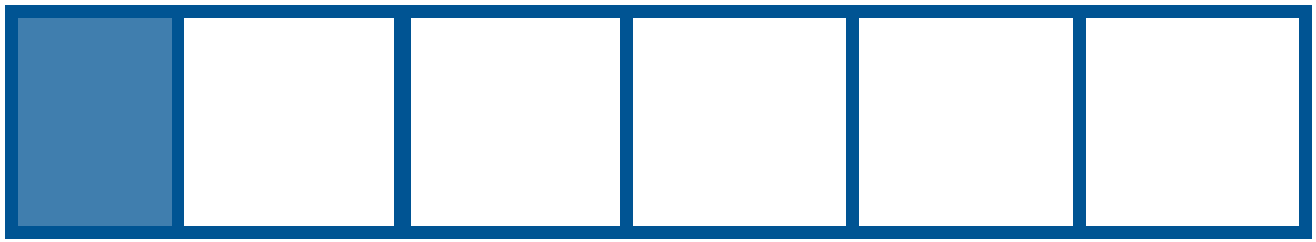
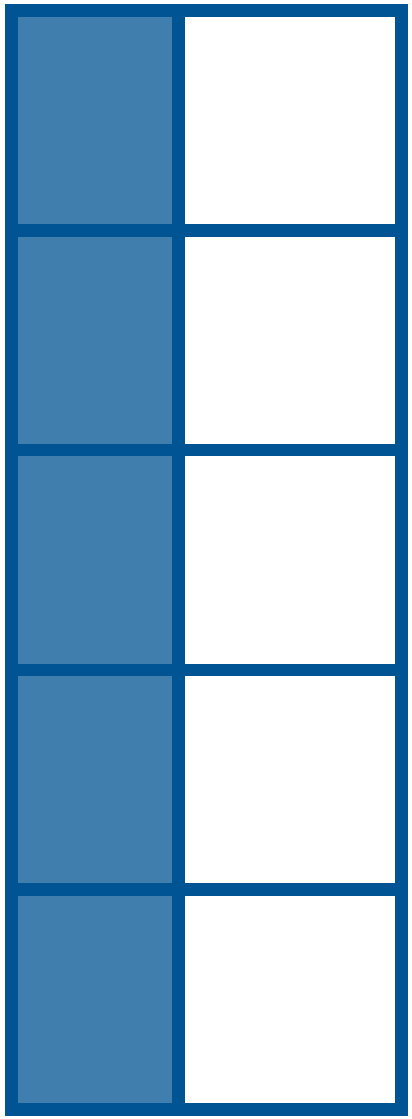


Figure 3: wide

and what we want

to this **long format**:



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

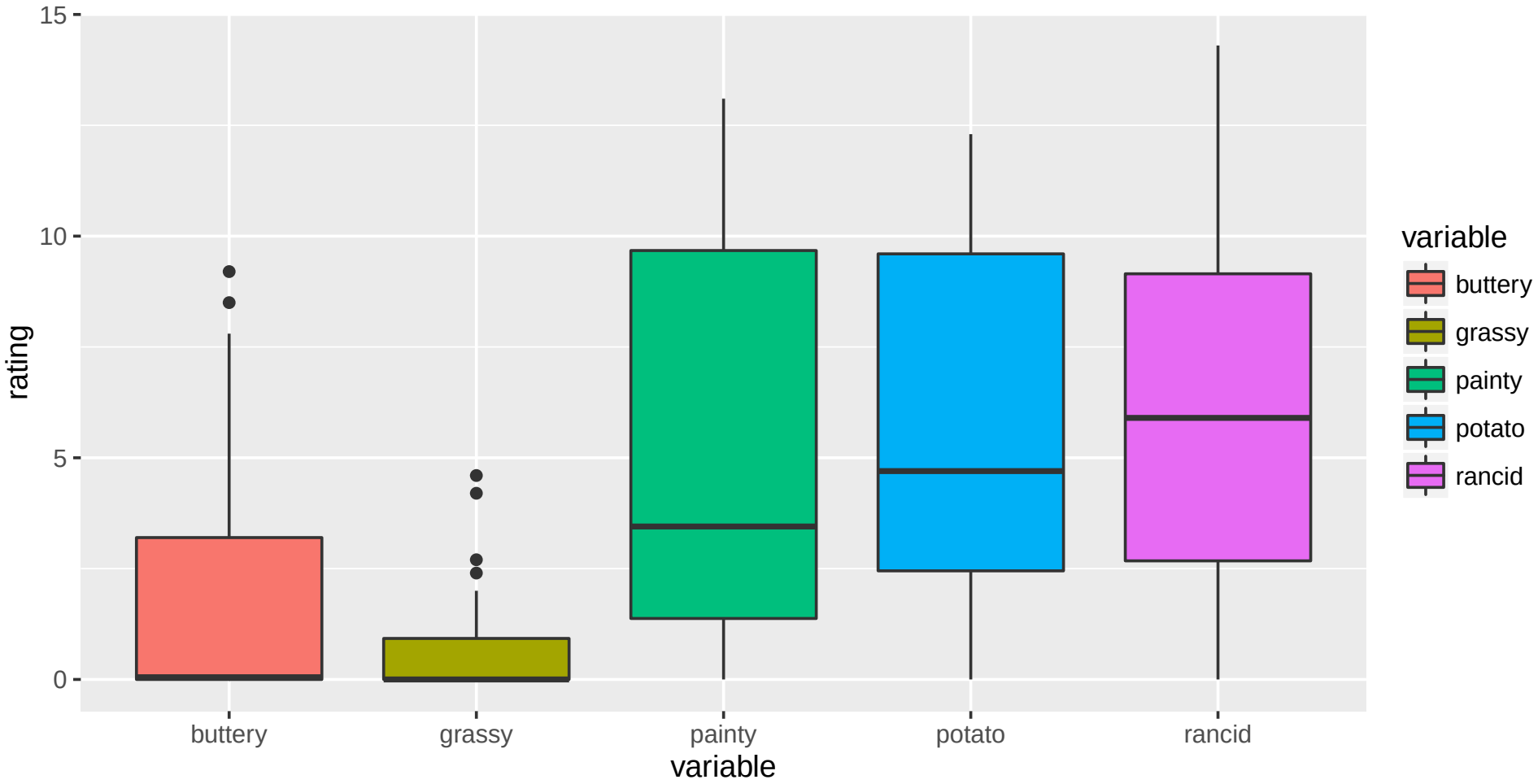
Gathering the French Fry Data

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

head(french_fries_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

```
french_fries_long_sub <- french_fries_long[
  french_fries_long$time == 10,]

ggplot(data = french_fries_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:

```
french_fries_wide <- spread(french_fries_long,
  key = variable, value = rating)
head(french_fries_wide)
```

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

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

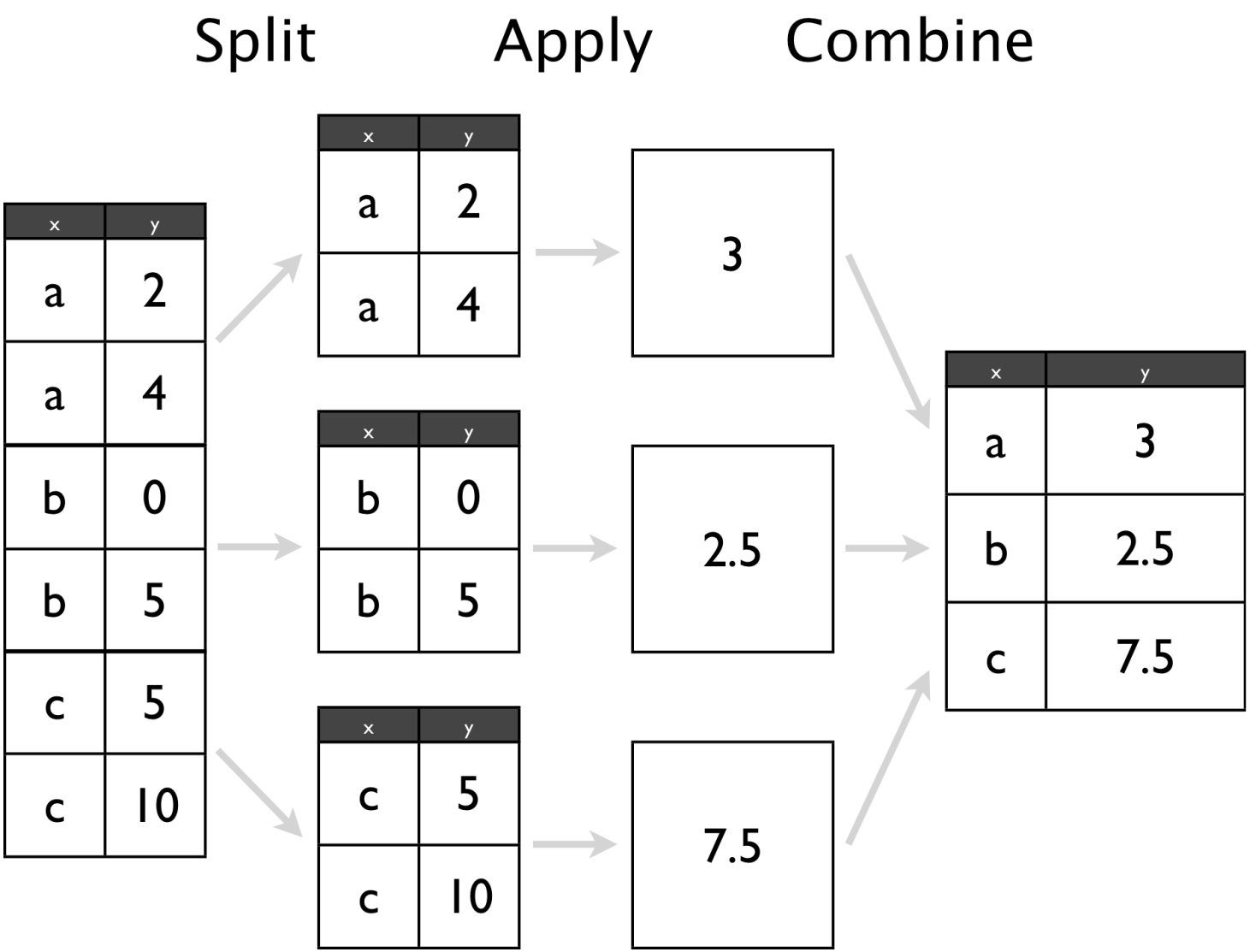


Figure 5: SAC

Split-Apply-Combine in dplyr

```
library(dplyr)
french_fries_split <- group_by(french_fries_long,
  variable) # SPLIT
french_fries_apply <- summarise(french_fries_split,
  m = mean(rating, na.rm = TRUE),
  s=sd(rating, na.rm=TRUE))
# APPLY + COMBINE
french_fries_apply
```

```
#> # 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
```

The pipe operator

- 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
```

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 p
#> 1    1         1       3   1    2.9     0.0    0.0    0.0
#> 2    1         1       3   2   14.0     0.0    0.0    1.1
#> 3    1         2       3   1   13.9     0.0    0.0    3.9
#> 4    1         2       3   2   13.4     0.1    0.0    1.5
#> 5    1         3       3   1   14.1     0.0    0.0    1.1
#> 6    1         3       3   2    9.5     0.0    0.6    2.8
```

```
#> # A tibble: 5 x 2
#>   variable    rating
#>      <chr>     <dbl>
#> 1  buttery 1.8236994
#> 2   grassy 0.6641727
#> 3   painty 2.5217579
#> 4   potato 6.9525180
```

Arrange

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

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

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
```

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
```

Dates and Times

- Dates are deceptively hard to work with in R.

Example: 02/05/2012. Is it February 5th, or May 2nd?
Other things are difficult too:

- Time zones
- POSIXct format in base R is challenging

The **lubridate** package helps tackle some of these issues.

Basic Lubridate Use

```
library(lubridate)

now()
today()
now() + hours(4)
today() - days(2)

#> [1] "2016-09-05 10:55:59 AEST"
#> [1] "2016-09-05"
#> [1] "2016-09-05 14:55:59 AEST"
#> [1] "2016-09-03"
```

Parsing Dates

```
ymd("2013-05-14")
mdy("05/14/2013")
dmy("14052013")
ymd_hms("2015:05:14 14:50:30", tz = "America/Chicago")
ymd_hms("2015:05:14 14:50:30", tz = "Australia/Melbourne")
today(tzone = "America/Chicago")
today(tzone = "Australia/Melbourne")
```

```
#> [1] "2013-05-14"
#> [1] "2013-05-14"
#> [1] "2013-05-14"
#> [1] "2015-05-14 14:50:30 CDT"
#> [1] "2015-05-14 14:50:30 AEST"
#> [1] "2016-09-04"
#> [1] "2016-09-05"
```

Dates example: Oscars date of birth

```
oscars <- read.csv("../data/oscars.csv", stringsAsFactors=FALS
summary(oscars$DOB)
head(oscars$DOB)
oscars$DOB <- as.Date(oscars$DOB, format="%m/%d/%Y")
summary(oscars$DOB)
```

```
#>    Length     Class      Mode
#>       423 character character
#> [1] "9/30/1895" "7/23/1884" "4/23/1894" "10/6/2006" "2/2/18
#>         Min.      1st Qu.       Median         Mean      3r
#> "1868-04-10" "1934-09-18" "1957-06-23" "1962-05-21" "2008-0
#>         Max.
#> "2029-12-13"
```

Calculating on dates

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

```
oscars$DOByr <- year(oscars$DOB)
summary(oscars$DOByr)
oscars %>% filter(DOByr == "2029") %>% select(Name, Sex, DOB)
oscars %>% filter(DOByr < 1950) %>% select(Name, Sex, DOB)
oscars %>% filter(DOByr > 2015) %>% select(Name, Sex, DOB)
```

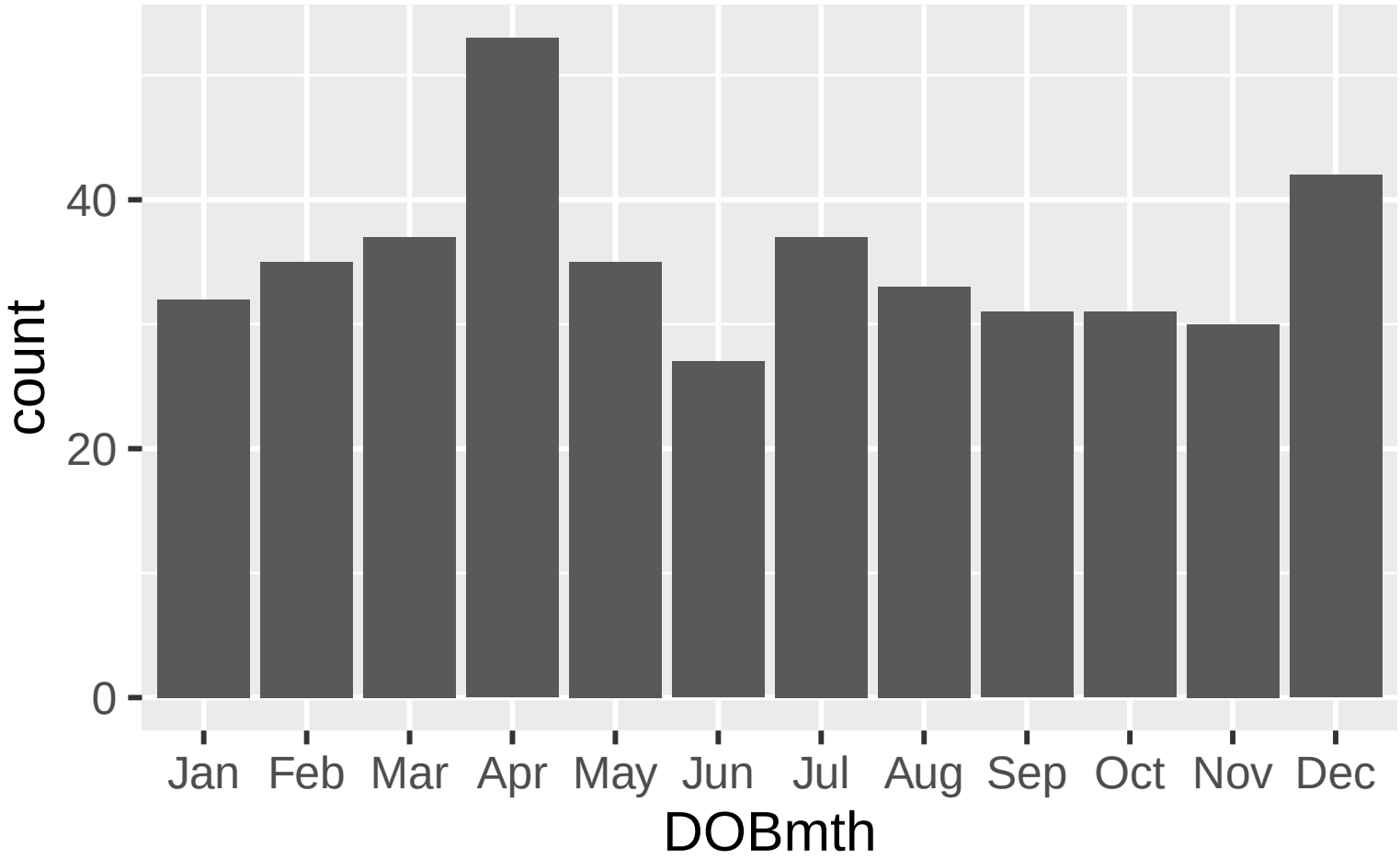
```
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>    1868    1934    1957    1962    2008    2029
#>                    Name    Sex         DOB
#> 1       Audrey Hepburn Female 2029-05-04
#> 2         Grace Kelly Female 2029-11-12
#> 3       Miyoshi Umeki Female 2029-04-03
#> 4 Christopher Plummer   Male 2029-12-13
```

Months

```
oscars$DOBmth <- month(oscars$DOB, )
table(oscars$DOBmth)
oscars$DOBmth <- factor(oscars$DOBmth, levels=1:12,
  labels=month.abb)

#>
#>  1  2  3  4  5  6  7  8  9 10 11 12
#> 32 35 37 53 35 27 37 33 31 31 30 42
```

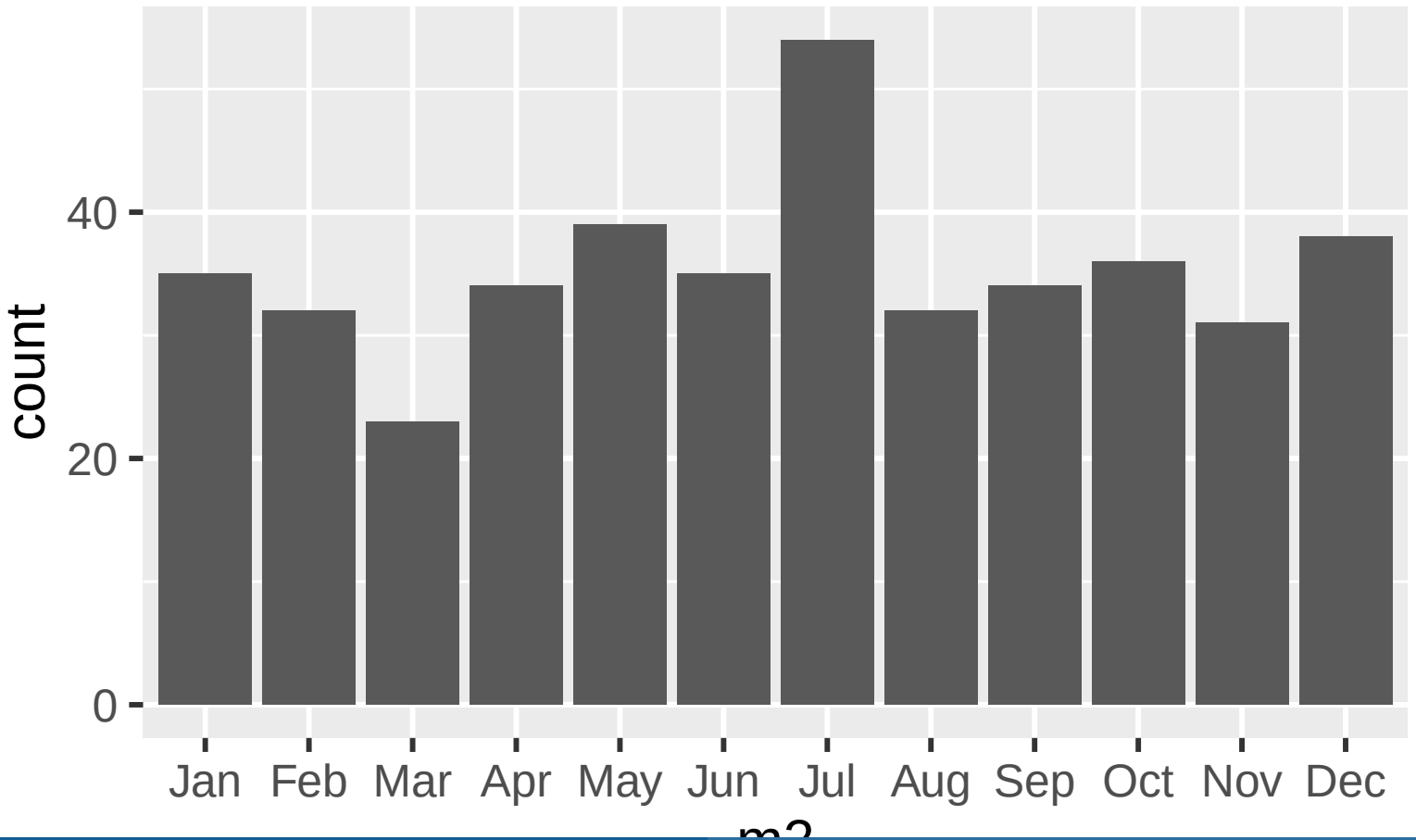
Now plot it

```
ggplot(data=oscars, aes(DOBmth)) + 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()
```



References

- tidy data concepts
- Split-apply-combine concepts
- Working with dates and times
- RStudio cheatsheets

Share and share alike