MONASH University

ETC3250 Business Analytics: Data Wrangling

Souhaib Ben Taieb, Di Cook, Rob Hyndman

September 7, 2015

Using the packages tidyr, dplyr

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)

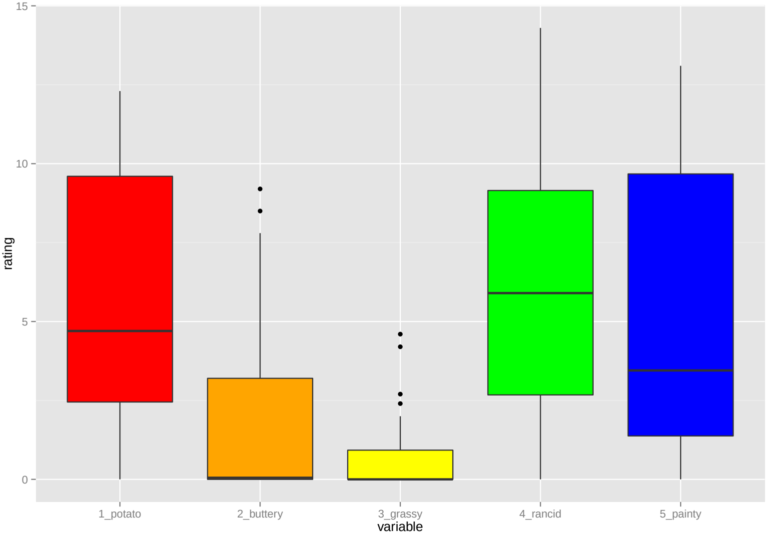
```
##    time treatment subject rep potato buttery grassy rancid
## 61    1         1       3   1    2.9     0.0    0.0    0.0
## 25    1         1       3   2   14.0     0.0    0.0    1.1
## 62    1         1      10   1   11.0     6.4    0.0    0.0
## 26    1         1      10   2    9.9     5.9    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,]
qplot("1_potato", potato, data = french_sub,
   fill = I("red"), geom = "boxplot") +
 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")
```

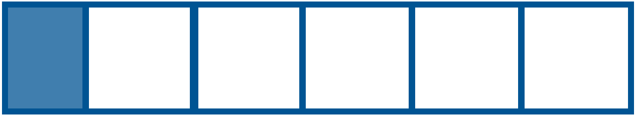
The Plot



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

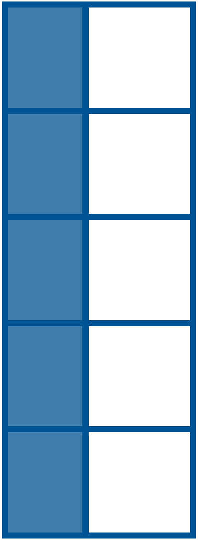
What we have ..

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



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, valu

head(french_fries_long)
```

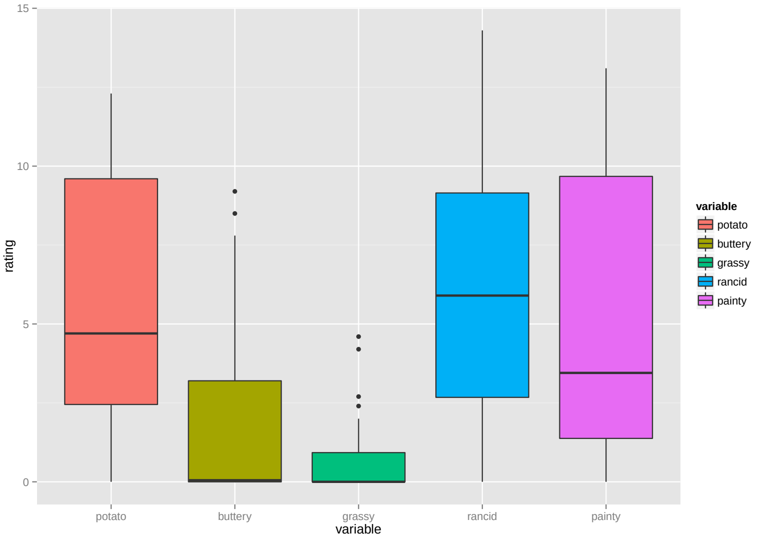
```
##   time treatment subject rep variable rating
## 1    1         1       3   1   potato    2.9
## 2    1         1       3   2   potato   14.0
## 3    1         1      10   1   potato   11.0
## 4    1         1      10   2   potato    9.9
## 5    1         1      15   1   potato    1.2
## 6    1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

qplot(variable, rating, data = french_fries_long_sub,
  fill = variable, geom = "boxplot")
```

And plot it


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

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:

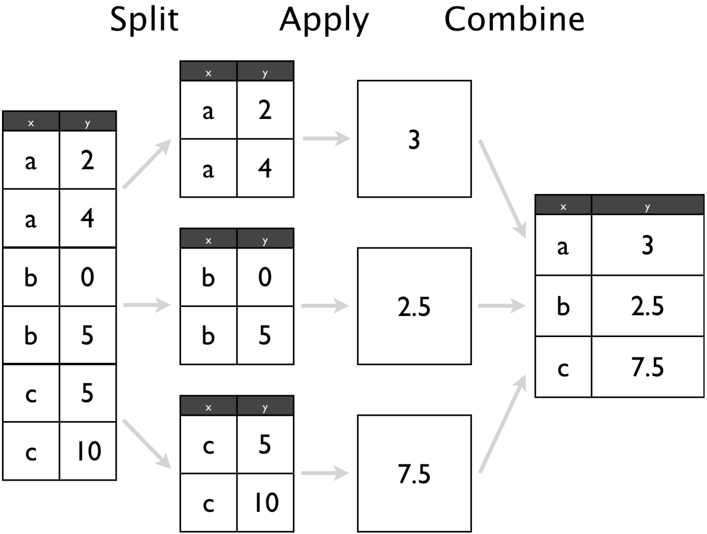
```
french_fries_wide <- spread(french_fries_long,
  key = variable, value = rating)
head(french_fries_wide)
```

```
##   time treatment subject rep 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         1      10   1   11.0     6.4    0.0    0.0
## 4    1         1      10   2    9.9     5.9    2.9    2.2
## 5    1         1      15   1    1.2     0.1    0.0    1.1
## 6    1         1      15   2    8.8     3.0    3.6    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)
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

## Source: local data frame [5 x 3]
##
##   variable         m        s
## 1   potato 6.9525180 3.584403
## 2  buttery 1.8236994 2.409758
## 3   grassy 0.6641727 1.320574
## 4   rancid 3.8522302 3.781815
## 5   painty 2.5217579 3.393717
```

The pipe operator

- dplyr allows us to chain together these data analysis tasks using the %>% (pipe) operator
- `x %>% f(y)` is shorthand for `f(x, y)`
- Example:

```
french_fries %>%
    gather(key = variable, value = rating, potato:painty) %>%
    group_by(variable) %>%
    summarise(rating = mean(rating, na.rm = TRUE))
```

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

## Source: local data frame [5 x 2]
##
##   variable    rating
## 1   potato 6.9525180
## 2  buttery 1.8236994
## 3   grassy 0.6641727
## 4   rancid 3.8522302
```

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(ranci

## Source: local data frame [30 x 4]
## Groups: time
##
##    time treatment mean_rancid sd_rancid
## 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
```

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] "2015-09-07 06:50:40 AEST"
## [1] "2015-09-07"
## [1] "2015-09-07 10:50:40 AEST"
## [1] "2015-09-05"
```

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 UTC"
## [1] "2013-05-14 UTC"
## [1] "2013-05-14 UTC"
## [1] "2015-05-14 14:50:30 CDT"
## [1] "2015-05-14 14:50:30 AEST"
## [1] "2015-09-06"
## [1] "2015-09-07"
```

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)
```

```
##    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
```

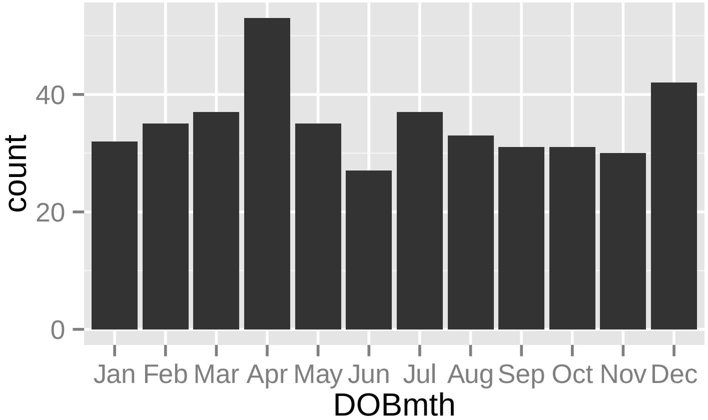
Months

```
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
```

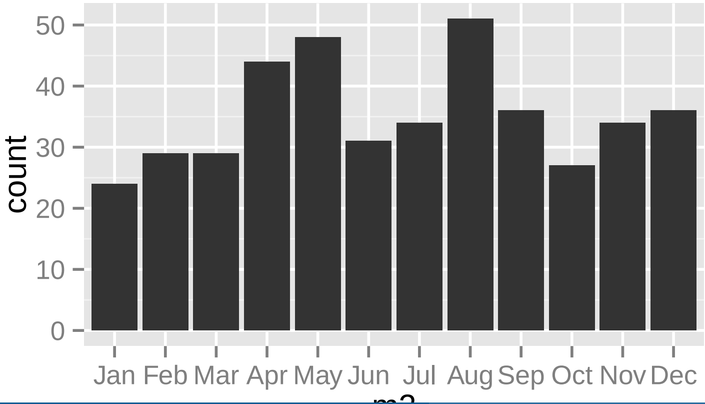
Now plot it

```
qplot(DOBmth, data=oscars)
```



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)
qplot(m2, data=df)
```



Working with strings

- Example: NBA salaries
- ESPN provides basketball players' salaries for the 2013-2014 season at `http://espn.go.com/nba/salaries`

```
##    RK                NAME                 TEAM      SALARY
## 1   1      Kobe Bryant, SG  Los Angeles Lakers $25,000,000
## 2   2      Joe Johnson, SF       Brooklyn Nets $24,894,863
## 3   3     LeBron James, SF Cleveland Cavaliers $22,970,500
## 4   4 Carmelo Anthony, SF     New York Knicks $22,875,000
## 5   5     Dwight Howard, C     Houston Rockets $22,359,364
## 6   6        Chris Bosh, C          Miami Heat $22,192,730
```

```
## 'data.frame':    423 obs. of  4 variables:
##  $ RK    : Factor w/ 394 levels "1","10","11",..: 1 12 23 3
##  $ NAME  : Factor w/ 394 levels "Al Jefferson, C",..: 31 24
##  $ TEAM  : Factor w/ 32 levels "Atlanta Hawks",..: 14 3 6 1
##  $ SALARY: Factor w/ 299 levels "$13,400,000",..: 35 34 33
```

Cleaning NBA salaries data

```
head(nba14$SALARY)

# get rid of $ and , in salaries and convert to numeric:
gsub("[$,]", "", head(as.character(nba14$SALARY)))
nba14$SALARY <- as.numeric(gsub("[$,]", "",
  as.character(nba14$SALARY)))
```

```
## [1] $25,000,000 $24,894,863 $22,970,500 $22,875,000 $22,35
## 299 Levels: $13,400,000 $13,437,500 $13,500,000 $13,600,00
```

```
## [1] "25000000" "24894863" "22970500" "22875000" "22359364"
```

```
## Warning: NAs introduced by coercion
```

- Where does the warning come from?

Cleaning NBA salaries data: hunting the warning

```
head(subset(nba14, is.na(SALARY)))
```

```
##    RK NAME TEAM SALARY
## 11 RK NAME TEAM     NA
## 22 RK NAME TEAM     NA
## 33 RK NAME TEAM     NA
## 54 RK NAME TEAM     NA
## 65 RK NAME TEAM     NA
## 76 RK NAME TEAM     NA
```

- We don't need these rows - delete all of them

```
dim(nba14)
nba14 <- nba14[-which(nba14$RK=="RK"),]
dim(nba14)
```

```
## [1] 423   4
```

Cleaning NBA data

- Separate names into first, last, and position

```
library(stringr)
splits <- str_split(as.character(nba14$NAME), pattern="(, )| '
splits[1:3]
library(plyr)
numnames <- ldply(splits, length)
summary(numnames) # some players have multiple names, ... sigh
```

```
## [[1]]
## [1] "Kobe"    "Bryant" "SG"
##
## [[2]]
## [1] "Joe"      "Johnson" "SF"
##
## [[3]]
## [1] "LeBron" "James"  "SF"
```

Cleaning data

- There's only limited possibilities in terms of what we can do automatically about people with multiple names - we will deal with them alongside the other ones and flag them ... maybe we should leave first and last name together.

```
head(splits[numnames>3], 5)
sum(numnames>3)
```

```
## [[1]]
## [1] "Otto"   "Porter" "Jr."    "SF"
##
## [[2]]
## [1] "Frank"    "Kaminsky" "III"      "C"
##
## [[3]]
## [1] "Kelly" "Oubre" "Jr."   "SF"
##
## [[4]]
```

Cleaning NBA data

```
splitnames <- ldply(splits, function(x)
  c(name=paste(x[-length(x)], collapse=" "),
  position=x[length(x)]))
head(splitnames) # looks OK
# now copy into the nba14 data frame
nba14$name <- as.factor(splitnames$name)
nba14$position <- as.factor(splitnames$position)
```

```
##                name position
## 1      Kobe Bryant       SG
## 2      Joe Johnson       SF
## 3     LeBron James       SF
## 4 Carmelo Anthony        SF
## 5    Dwight Howard        C
## 6       Chris Bosh        C
```

Cleaned data ...?

```
summary(nba14)
```

```
##          RK                            NAME
##  1         :  1   Al Jefferson, C       :  1   Portland Trail Bl
##  10        :  1   Andrew Bogut, C       :  1   Boston Celtics
##  11        :  1   Blake Griffin, PF     :  1   Brooklyn Nets
##  12        :  1   Carmelo Anthony, SF :  1   Charlotte Hornets
##  13        :  1   Chandler Parsons, SF:  1   Toronto Raptors
##  14        :  1   Chris Bosh, C         :  1   Atlanta Hawks
##  (Other):387   (Other)               :387   (Other)
##      SALARY                       name          position
##  Min.    :  525093   Aaron Brooks   :  1   C :57
##  1st Qu.: 1499187   Aaron Gordon   :  1   PF:90
##  Median : 3283181   Aaron Harrison:  1   PG:74
##  Mean   : 5267665   Adreian Payne :  1   SF:82
##  3rd Qu.: 7000000   Al Horford     :  1   SG:90
##  Max.   :25000000   Al Jefferson   :  1
```

Cleaned data . . . ?

- Numbers might still be wrong, but now we are in a position to check for that.

```
library(ggplot2)
qplot(SALARY, geom="histogram", data=nba14)
```

Show it

```
qplot(position, SALARY, geom="boxplot", data=nba14)
```