Data frame

Yufeng Huang

Associate Professor of Marketing, Simon Business School

August 2/4, 2022

Data frame

- More about data frames
  - Attributes of a data frame
  - How select elements and groups of elements
  - Add rows and columns to a data frame
- Combining data frames: merge()
- Collapsing / summarizing data frames: aggregate()
- Reshaping data frames: melt() and cast()

Attributes of a data frame

str(): general structure of a data frame

```
# let's construct a data frame
cdata <- data.frame(
        name = c("Anita", "Linda", "Harikesh", "Yufeng"),
        age = 26:29,
        female = c(T, T, F, F),
        entry = c(2014, 2015, 2015, 2016)
)

# str() summarizes the structure of cdata
a <- str(cdata) # usually don't assign something to str(), see next slide

## 'data.frame': 4 obs. of  4 variables:
##  $ name  : chr  "Anita" "Linda" "Harikesh" "Yufeng"
##  $ age   : int  26 27 28 29
##  $ female: logi  TRUE TRUE FALSE FALSE
##  $ entry : num  2014 2015 2015 2016
```

Function str()

- Function str() provides
  - number of rows
  - number of variables
  - name of each variable/column
  - mode (i.e. type) of each column (e.g. num, int, chr, factor)
  - levels of factor variables
- str() is good for visual inspection but does not directly give you access to any display information

```
# previously we assigned a to str(cdata)
a

## NULL
```

Basic information of a data frame

```
# dim() gives dimensions
dim(cdata)

## [1] 4 4

# nrow() gives #rows (ncol for #columns)
nrow(cdata)

## [1] 4

# names() gives names of columns
names(cdata)

## [1] "name"   "age"    "female" "entry"

# dimnames() gives row and column names
dimnames(cdata)

## [[1]]
## [1] "1" "2" "3" "4"
##
## [[2]]
## [1] "name"   "age"    "female" "entry"
```

Function object.size()

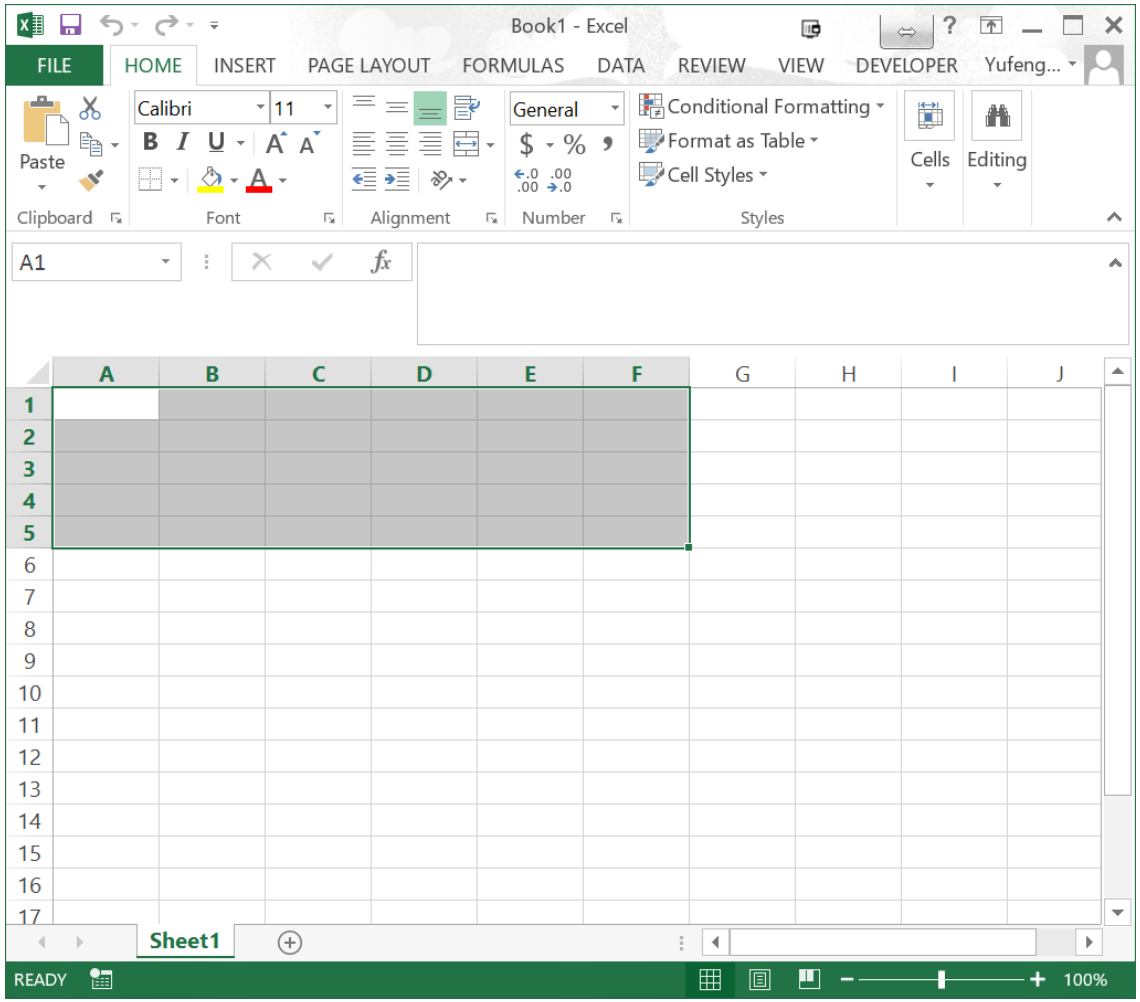
```
# object.size() lets you know how much memory are allocated to an object
#    useful to check whether something is too large
#    in case you need to delete it to free up space

object.size(cdata)

## 1416 bytes

object.size(cdata$name) # works on other objects

## 312 bytes
```

Function summary()

```
# summary() computes descriptive statistics
#    very useful to have a first impression on a dataset

summary(cdata)

##      name                age          female            entry
##  Length:4           Min.   :26.00   Mode :logical   Min.   :2014
##  Class :character   1st Qu.:26.75   FALSE:2         1st Qu.:2015
##  Mode  :character   Median :27.50   TRUE :2         Median :2015
##                     Mean   :27.50                   Mean   :2015
##                     3rd Qu.:28.25                   3rd Qu.:2015
##                     Max.   :29.00                   Max.   :2016

# note: can also summarize other things e.g. statistical analysis outputs
```

Review: accessing information in a data frame

Excel: how would you select A1:F5?



Single cell

```
# recall that notation [] is for subsets

# first cell
cdata[1, 1]

## [1] "Anita"

# cell 4, 4
cdata[4, 4]

## [1] 2016
```

A set of cells

```
# adjacent cells
cdata[1:3, 1:3]
```

```
##       name age female
## 1    Anita  26   TRUE
## 2    Linda  27   TRUE
## 3 Harikesh  28  FALSE
```

```
# permuted cells
cdata[2:1, 3:1]
```

```
##   female age  name
## 2   TRUE  27 Linda
## 1   TRUE  26 Anita
```

```
# non-adjacent cells
cdata[c(2, 4), c(1, 3)]
```

```
##     name female
## 2  Linda   TRUE
## 4 Yufeng  FALSE
```

```
# much like what we did with arrays/vectors
```

A set of cells (con'd)

```
# excluding cells
cdata[-c(1:3), -c(1:2)]
```

```
##   female entry
## 4  FALSE  2016
```

```
# mixed
cdata[2:1, -c(1:2)]
```

```
##   female entry
## 2   TRUE  2015
## 1   TRUE  2014
```

```
# recall that with vectors, things like a[c(-1, 2)] is not allowed
```

Entire rows

```
# entire rows
cdata[1:2, ]

##    name age female entry
## 1 Anita  26   TRUE  2014
## 2 Linda  27   TRUE  2015

# excluding entire rows
cdata[-(1:2), ]

##       name age female entry
## 3 Harikesh  28  FALSE  2015
## 4   Yufeng  29  FALSE  2016
```

Entire column

```
# entire column, but as a vector
cdata[, 3]
```

```
## [1]  TRUE  TRUE FALSE FALSE
```

```
# entire column maintaining the shape
cdata[, 3, drop = F]
```

```
##   female
## 1   TRUE
## 2   TRUE
## 3  FALSE
## 4  FALSE
```

```
# multiple columns
cdata[, c(1, 3)]
```

```
##       name female
## 1    Anita   TRUE
## 2    Linda   TRUE
## 3 Harikesh  FALSE
## 4   Yufeng  FALSE
```

Select elements like in a list

```
cdata$name

## [1] "Anita"    "Linda"    "Harikesh" "Yufeng"

cdata[["name"]]

## [1] "Anita"    "Linda"    "Harikesh" "Yufeng"

cdata[[1]]

## [1] "Anita"    "Linda"    "Harikesh" "Yufeng"
```

Previous arguments are identical

```
identical(cdata$name, cdata[["name"]])
```

```
## [1] TRUE
```

```
identical(cdata[["name"]], cdata[[1]])
```

```
## [1] TRUE
```

If you want to maintain the data frame structure...

```
# recall that single brackets [] maintains the list structure

cdata["entry"]

##   entry
## 1  2014
## 2  2015
## 3  2015
## 4  2016

identical(cdata["entry"], cdata[, 4, drop = F])

## [1] TRUE
```

Subsetting a data frame by conditions

Subset a data frame by logical conditions

- A combination of
  - comparison statement to locate cells that satisfy the given condition
  - and to select those cells
- Saw similar things in lists

Subset

```
# find female
cdata[cdata$female == T, ]

##    name age female entry
## 1 Anita  26   TRUE  2014
## 2 Linda  27   TRUE  2015

# find female with certain height
cdata[cdata$female == T & cdata$age > 27, ]     # no entry

## [1] name   age    female entry
## <0 rows> (or 0-length row.names)
```

How does it work?

```
# cdata$female is a vector of factors
cdata$female
```

```
## [1]  TRUE  TRUE FALSE FALSE
```

```
# cdata$female == T is a vector of logicals
#  indicates in which row the female variable is TRUE
cdata$female == T
```

```
## [1]  TRUE  TRUE FALSE FALSE
```

```
# can also call which()
which(cdata$female == T)
```

```
## [1] 1 2
```

How does it work (con'd)

```
# we then call the corresponding rows
cdata[cdata$female == T, ]

# just as if we are calling a series of logicals
cdata[c(T, T, F, F), ]

# also equivalent if we just call the row number
cdata[c(2, 2), ]
```

Can also use subset()

```
# subset() reduces the verbose statements
subset(cdata, female == T)
```

```
##    name age female entry
## 1 Anita  26   TRUE  2014
## 2 Linda  27   TRUE  2015
```

```
# multiple conditions
subset(cdata, female == T & age > 27)
```

```
## [1] name   age    female entry
## <0 rows> (or 0-length row.names)
```

```
# can also select columns
subset(cdata,
       female == T,
       select = c(name, age))
```

```
##    name age
## 1 Anita  26
## 2 Linda  27
```

Adding and removing rows or columns

Can add columns to a data frame

```
eyecol <- c("blue", "brown", "black", "black")

# preferred way to add is cbind()
cdata.wider <- cbind(cdata, eyecol)
cdata.wider
```

```
##       name age female entry eyecol
## 1    Anita  26   TRUE  2014   blue
## 2    Linda  27   TRUE  2015  brown
## 3 Harikesh  28  FALSE  2015  black
## 4   Yufeng  29  FALSE  2016  black
```

```
# a different way, in case you want to
#    but, note that this modifies the original data frame
cdata$eyecol <- eyecol
```

[Outdated] Data frame converts character to factors [update: not any more (Yay!)]

```
# eyecol is character vector
typeof(eyecol)

## [1] "character"

# eyecol in data frame used to be a factor, but not any more from R4.0
typeof(cdata.wider$eyecol)

## [1] "character"
```

[Outdated] If you do not want factor conversion you have to specify...

```
# back in the days, have to say stringsAsFactors, but not any more
cdata.string <- data.frame(
        name = c("Anita", "Linda", "Harikesh", "Yufeng"),
        age = 26:29,
        female = c(T, T, F, F),
        entry = c(2014, 2015, 2015, 2016),
        stringsAsFactors = FALSE        # easily misspelled
)

typeof(cdata.string[, 1])
```

```
## [1] "character"
```

Add a row and a column

```
# adding a row: need to add a data frame
drac_data <- data.frame(name = "Dracula",
                        age = 589,
                        female = F,
                        entry = 1531,
                        eyecol = "green")
cdata.new <- rbind(cdata.wider, drac_data)

# examine the new data
cdata.new
```

```
##       name age female entry eyecol
## 1    Anita  26   TRUE  2014   blue
## 2    Linda  27   TRUE  2015  brown
## 3 Harikesh  28  FALSE  2015  black
## 4   Yufeng  29  FALSE  2016  black
## 5  Dracula 589  FALSE  1531  green
```

How to remove rows and columns?

```
# remove row by re-assignment
cdata.new <- cdata.new[1:4, ]   # NULL does not work

# remove column
cdata.new$eyecol <- NULL        # re-assignment also works

# back to original?
cdata.new       # note that factor level not the same
```

```
##       name age female entry
## 1    Anita  26   TRUE  2014
## 2    Linda  27   TRUE  2015
## 3 Harikesh  28  FALSE  2015
## 4   Yufeng  29  FALSE  2016
```

Taking stock

- Data frames are stored as lists but displayed as tables

```
# Column operation: which one of these three is different?
#   and no this is not something I ask in exams
mtcars["mpg"]

mtcars[["mpg"]]

mtcars[, "mpg"]
```

```
# Example row operations
mtcars[-1, ]    # anything but not first row

mtcars[mtcars$mpg > 20, ]       # get rows where $mpg > 20

subset(mtcars, mpg > 20)        # identical

mtcars[order(mtcars$mpg, decreasing = T), ]
#    order mtcars in descending of $mpg
```

- Next step: data frame operations – merge, aggregate, reshape

Recall the contact data example

```r
# let's construct a data frame
cdata <- data.frame(
        name = c("Anita", "Linda", "Harikesh", "Yufeng"),
        age = 26:29,
        female = c(T, T, F, F),
        entry = c(2014, 2015, 2015, 2016)
)

# visually inspect the data
cdata
```

```
##       name age female entry
## 1    Anita  26   TRUE  2014
## 2    Linda  27   TRUE  2015
## 3 Harikesh  28  FALSE  2015
## 4   Yufeng  29  FALSE  2016
```

Merge

What if we want to combine two data sets? cbind()?

```
# define a namesheet to only keep name and entry year
id <- cdata[c(1, 4)]

# suppose the assistant gives you a list of grades
grades.gba464 <- data.frame(gba464 = c("A", "A-", "A-", "B"))

# you can merge it by cbind?
grade.data <- cbind(id, grades.gba464)
grade.data
```

```
##       name entry gba464
## 1    Anita  2014      A
## 2    Linda  2015     A-
## 3 Harikesh  2015     A-
## 4   Yufeng  2016      B
```

Your assistant then says:

> *Oh by the way, I sorted the grades in descending order and they don't necessarily correspond to names...*

```
# It turns out that Linda got the A

# so you manually entered the data, carefully this time
grades.gba464 <- data.frame(gba464 = c("A-", "A", "A-", "B"))

# and you cbind it again
grade.data <- cbind(id, grades.gba464)
grade.data
```

```
##       name entry gba464
## 1    Anita  2014     A-
## 2    Linda  2015      A
## 3 Harikesh  2015     A-
## 4   Yufeng  2016      B
```

```r
# here's a name-grade mapping:
grades.gba464 <- data.frame(
        name = c("Harikesh", "Yufeng", "Linda", "Anita"),
        gba464 = c("A-", "B", "A", "A-")
)

# compare these two data frames
id
```

```
##       name entry
## 1    Anita  2014
## 2    Linda  2015
## 3 Harikesh  2015
## 4   Yufeng  2016
```

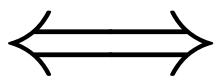
```r
grades.gba464
```

```
##       name gba464
## 1 Harikesh     A-
## 2   Yufeng      B
## 3    Linda      A
## 4    Anita     A-
```

Merge by variables that uniquely defines each observation

- Notice that *name* **uniquely defines** each observation
- So can merge by this variable despite different sorting rules

name	entry
Anita	2014
Linda	2015
Harikesh	2015
Yufeng	2016



name	year	mkt440
Linda	2015	A
Harikesh	2016	A-
Anita	2015	A-
Yufeng	2016	B

Use merge() in R

```
# and you can merge the two by names
grade.data <- merge(id, grades.gba464, by = "name")
grade.data
```

```
##       name entry gba464
## 1    Anita  2014     A-
## 2 Harikesh  2015     A-
## 3    Linda  2015      A
## 4   Yufeng  2016      B
```

```
# this is a 1-1 merge since 'name' in both data frames define an observation
```

Side: merge by factors

```
# for illustration purposes: what if factors are not the same?
id2 <- id
id2$name <- factor(id2$name,
        ordered = TRUE, # what if name is ordered in one data frame?
        levels = c("Harikesh", "Linda", "Anita", "Yufeng"))

# what would happen to the merge?
grade.data2 <- merge(id2, grades.gba464, by = "name")
```

Side: merge by factors

```r
# for illustration purposes: what if factors are not the same?
id2 <- id
id2$name <- factor(id2$name,
        ordered = TRUE, # what if name is ordered in one data frame?
        levels = c("Harikesh", "Linda", "Anita", "Yufeng"))

# what would happen to the merge?
grade.data2 <- merge(id2, grades.gba464, by = "name")
```

```r
# exactly the same because R matches factor levels not labels
grade.data2
```

```
##       name entry gba464
## 1    Anita  2014     A-
## 2 Harikesh  2015     A-
## 3    Linda  2015      A
## 4   Yufeng  2016      B
```

Some students took MKT440 twice and some never took it

- Some took MKT440 twice
  - so names cannot clearly define who has which grade
- Can we merge MKT440 grades into the name list?
  - can you tell me what the result is?

name	entry
Anita	2014
Linda	2015
Harikesh	2015
Yufeng	2016

$\Longleftrightarrow$

name	year	mkt440
Harikesh	2015	A-
Harikesh	2016	A
Anita	2014	A-
Anita	2015	A
Yufeng	2016	B

How would you do the merge now?

```
# here's a name-grade mapping and:
grades.mkt440 <- data.frame(
        name = c("Harikesh", "Harikesh", "Anita", "Anita", "Yufeng"),
        year = c(2015, 2016, 2014, 2015, 2016),
        mkt440 = c("A-", "A", "A-", "A", "B")
)

# note that this is a 1-n merge

# and you can merge the two according to names (also note missing names)
grade.data <- merge(id, grades.mkt440, by = "name", all = TRUE)
grade.data
```

```
##       name entry year mkt440
## 1    Anita  2014 2014     A-
## 2    Anita  2014 2015      A
## 3 Harikesh  2015 2015     A-
## 4 Harikesh  2015 2016      A
## 5    Linda  2015   NA   <NA>
## 6   Yufeng  2016 2016      B
```

'all = FALSE' will drop all the NAs

```
# 'all = FALSE', or not specifying the argument 'all',
#    will drop rows with NAs
#    also consult help for arguments all.x and all.y

grade.data2 <- merge(id, grades.mkt440, by = "name")
grade.data2
```

```
##       name entry year mkt440
## 1    Anita  2014 2014     A-
## 2    Anita  2014 2015      A
## 3 Harikesh  2015 2015     A-
## 4 Harikesh  2015 2016      A
## 5   Yufeng  2016 2016      B
```

What to do with duplicated names?

- We don't have student list and we only have two grade lists
- Each grade sheet has names that potentially appear in multiple places

name	year	gba464
Linda	2015	A-
Harikesh	2016	A
Anita	2014	A
Yufeng	2016	B
Yufeng	2017	B

$\Longleftrightarrow$

name	year	mkt440
Harikesh	2015	A-
Harikesh	2016	A
Anita	2014	A-
Anita	2015	A
Yufeng	2016	B

WRONG! Merging two grade lists with duplicated names

```
# here's a grade sheet for GBA 464
grades.gba464 <- data.frame(
        name = c("Linda", "Harikesh", "Anita", "Yufeng", "Yufeng"),
        year = c(2015, 2016, 2014, 2016, 2017),
        gba464 = c("A-", "A", "A", "B", "B")
)

# try if merge them by names
grade.data <- merge(grades.mkt440, grades.gba464,
        by = "name", all = TRUE)
grade.data
```

```
##       name year.x mkt440 year.y gba464
## 1    Anita   2014     A-   2014      A
## 2    Anita   2015      A   2014      A
## 3 Harikesh   2015     A-   2016      A
## 4 Harikesh   2016      A   2016      A
## 5    Linda     NA   <NA>   2015     A-
## 6   Yufeng   2016      B   2016      B
## 7   Yufeng   2016      B   2017      B
```

```
# what's wrong?
#    some years are duplicated to match two tables
#    in other words these n-n merge don't really work
```

Keys of a dataset

- Primary keys are variables that **uniquely identify** a single observation in a table
  - LHS table: name
  - RHS table: name and year
- There is no formal definition of keys in data.frame
  - the terminology comes from SQL
- However, such knowledge is very useful when merging/organizing data sets

name	entry
Anita	2014
Linda	2015
Harikesh	2015
Yufeng	2016

name	year	mkt440
Harikesh	2015	A-
Harikesh	2016	A
Anita	2014	A-
Anita	2015	A
Yufeng	2016	B

CORRECT! Merging with name+year

```
# correct way is to merge it by name and year
grade.data <- merge(grades.mkt440, grades.gba464,
        by = c("name", "year"), all = TRUE)
grade.data
```

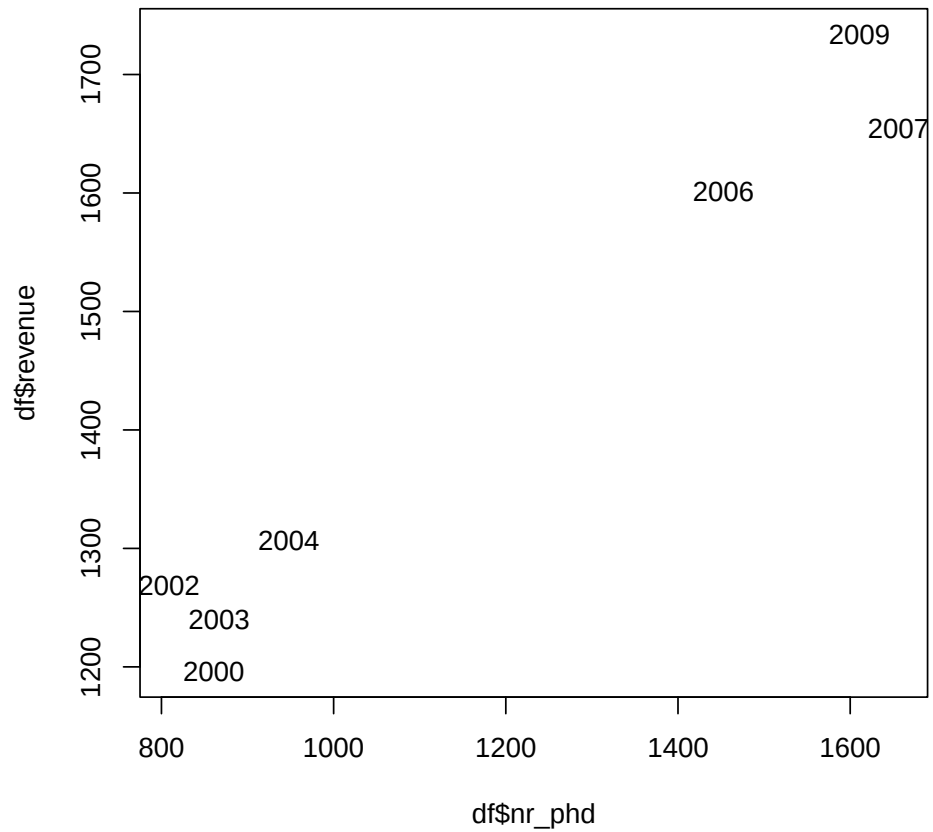
```
##       name year mkt440 gba464
## 1    Anita 2014     A-      A
## 2    Anita 2015      A   <NA>
## 3 Harikesh 2015     A-   <NA>
## 4 Harikesh 2016      A      A
## 5    Linda 2015   <NA>     A-
## 6   Yufeng 2016      B      B
## 7   Yufeng 2017   <NA>      B
```

A simple example[1]

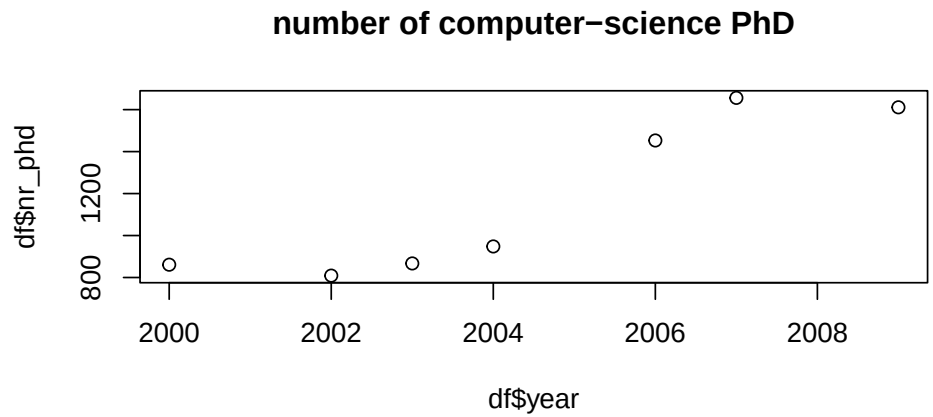
```
# read.table to read text data (source: tylervigen.com)
phd <- read.table('phd.txt', header = T) # nr. computer science phds (NSF)
rev <- read.table('rev.txt', header = T) # nr. arcade game revenue (Census)

# show structure (data not sorted and contain missing, on purpose)
head(phd)

##   year nr_phd
## 1 2000    861
## 2 2002    809
## 3 2003    867
## 4 2005   1129
## 5 2007   1656
## 6 2009   1611

head(rev)

##   year revenue
## 1 2000    1196
## 2 2001    1176
## 3 2002    1269
## 4 2006    1601
## 5 2007    1654
## 6 2008    1803
```

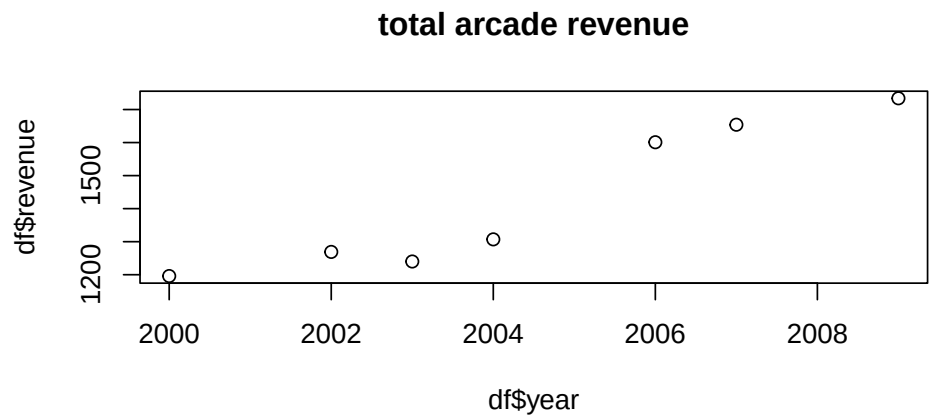
[1] Some graphical arguments here might be new to you; we'll discuss some of those later this week, but you can always help()

```r
# merge them and check correlations
df <- merge(phd, rev, by = "year")

# plot, but omit marker
plot(df$nr_phd, df$revenue, cex = 0)    # cex: marker size

# add text that symbolizes which year this is
text(df$nr_phd, df$revenue, labels = df$year)
```



```r
# a better figure is to *realize*
# that the two series are not really related
par(mfrow = c(2, 1))    # create an area of two plots
plot(df$year, df$nr_phd, main = "number of computer-science PhD")
plot(df$year, df$revenue, main = "total arcade revenue")
```





Merge summary

- Never use cbind to merge data sets
- Data frames *should* have keys
- Can merge two data sets if defined by keys
  - given keys, can do 1-1, 1-n and n-1 merge
  - cannot do n-n merge
- Do not need to worry about whether data are sorted

Now, suppose I give you data of grades by name-year-course

```
# display the dataset (I'll tell you where it comes from later)
grade.data.long
```

```
##        name year course grade
## 1     Anita 2014 mkt440    A-
## 2     Anita 2015 mkt440     A
## 3  Harikesh 2015 mkt440    A-
## 4  Harikesh 2016 mkt440     A
## 6    Yufeng 2016 mkt440     B
## 8     Anita 2014 gba464     A
## 11 Harikesh 2016 gba464     A
## 12    Linda 2015 gba464    A-
## 13   Yufeng 2016 gba464     B
## 14   Yufeng 2017 gba464     B
```

```
# I will use this data frame to illustrate aggregate()
```

Appendix: code for the previous slide

```
# first get the 'reshape' library
library('reshape')

# 'melt' the wide data into long
grade.data.long <- melt(grade.data, id = c("name", "year"))
grade.data.long <- subset(grade.data.long, !is.na(value))
colnames(grade.data.long)[3:4] <- c("course", "grade")  # rename column, more l
```

Aggregate

Objective: collapse data into a sub-data and generate summary statistics

Start with a long data in the sense that each observation is defined by multiple keys $\Rightarrow$ collapse it into a "shorter" data with fewer keys

ID	year	income
Anita	2010	80
Anita	2011	90
Linda	2011	70
Linda	2012	80
Linda	2013	90
Yufeng	2009	60
Yufeng	2010	70

$\Rightarrow$

name	avg_inc
Anita	85
Linda	80
Yufeng	65

Aggregate

- One of the most useful functions if you want to compute summary statistics from a data frame
  - and you can define your own way to summarize what you want – useful when combined with self-defined functions
- How it works
  - splits data frame into subsets
  - applies a function (built-in or user-defined) on the data frame, by subsets
  - returns the result in a "convenient form"
- Two different notations

```
# notation 1: x-by-FUN
aggregate(x = data.frame$var2, by = list(data.frame$var1), FUN = mean)

# notation 2: formula-data-FUN
aggregate(formula = var2 ~ var1, data = data.frame, FUN = mean)
```

- Example 1: what is the *first year* that *each* student ever took any class (i.e. enrolled)?

```
# notation 1: x-by-FUN
first.year <- aggregate(
        x = grade.data.long$year,       # x is the variable to summarize
        by = list(grade.data.long$name),        # argument by must take a list
        FUN = min       # function to apply, here 'min' is function name
)

# check first two rows of result, note that column names are lost
head(first.year, 2)
```

```
##    Group.1    x
## 1    Anita 2014
## 2 Harikesh 2015
```

```
# assign correct column names
colnames(first.year) <- c("name", "first.year")

# now correct result
first.year
```

```
##       name first.year
## 1    Anita       2014
## 2 Harikesh       2015
## 3    Linda       2015
## 4   Yufeng       2016
```

- Example 2: how many times *each* student ever took *each* class?

```
# still do x-by-FUN, but *use lists to add variable names*
number.attempts <- aggregate(
        x       = list(attempts = grade.data.long$grade), # generate this
        by      = list(name = grade.data.long$name,     # by two var's
                       course = grade.data.long$course),
        FUN     = length        # function is length of the subset
)

# check results
number.attempts

##       name course attempts
## 1    Anita mkt440        2
## 2 Harikesh mkt440        2
## 3   Yufeng mkt440        1
## 4    Anita gba464        1
## 5 Harikesh gba464        1
## 6    Linda gba464        1
## 7   Yufeng gba464        2
```

- Example 2 again: how many times *each* student ever took *each* class?

```
# now turn to formula-data notation
number.attempts.2 <- aggregate(
        formula = grade ~ name + course, # summarize grade by name and coure
        data    = grade.data.long, # specify the data frame to work on
        FUN     = length         # function (same as before)
)

## Error in aggregate.formula(formula = grade ~ name + course, data =
grade.data.long, :  argument 'x' is  missing - it has been renamed from
'formula'

# check results
number.attempts.2

## Error in eval(expr, envir, enclos):  object 'number.attempts.2' not
found

# note that now variable names are preserved
#    but we do not necessarily want the name "grade"
```

- Example 3: how many unique classes are offered each year?

```
# first define a user-generated function (more on this wk 5/6)
length.unique <- function(x) length(unique(x))

# use formula-data-FUN notation first
number.classes <- aggregate(formula = course ~ year, # formula
        data = grade.data.long,          # specifies data frame
        FUN = length.unique)             # uses user-defined function

## Error in aggregate.formula(formula = course ~ year, data =
grade.data.long, :  argument 'x' is  missing - it has been renamed from
'formula'

# check results (don't need to rename columns)
number.classes

## Error in eval(expr, envir, enclos):  object 'number.classes' not found
```

```
# almost equivalent will be
number.classes <- aggregate(x = grade.data.long$course,
                        by = list(grade.data.long$year),
                        FUN = length.unique)
```

Another example: state.x77 data

- state.x77 dataset describes the population, income and some other variables for all 50 US states
- This is a built-in example in the help file for aggregate()
- It shows some other uses of aggregate() that we have not seen before

```
# first convert into data frame (was a matrix)
state.x77.df <- data.frame(state.x77)

# "see" the data
head(state.x77.df)

##            Population Income Illiteracy Life.Exp Murder HS.Grad Frost   Area
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20  50708
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65  51945
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
## Colorado         2541   4884        0.7    72.06    6.8    63.9   166 103766

# variables: population, average income, rate of illiteracy, life expectancy,
#    murder per 100,000, % high-school grad, number of days below freezing, area in sq. ft.
```

- Example 1: summary statistics of all the columns, by region and for states that are "cold" (i.e. frost days $>$ 130)

```
# state.region is a factor that match this data
head(state.region)

## [1] South West  West  South West  West
## Levels: Northeast South North Central West

# NOTE: can aggregate on a variable that is temporarily defined
aggregate(x = state.x77.df,
                by = list(Region = state.region,
                          Cold = state.x77.df$Frost > 130),       # cold = more than 130 days of frost
                FUN = mean)

##          Region  Cold Population   Income Illiteracy Life.Exp    Murder
## 1     Northeast FALSE  8802.8000 4780.400  1.1800000 71.12800  5.580000
## 2         South FALSE  4208.1250 4011.938  1.7375000 69.70625 10.581250
## 3 North Central FALSE  7233.8333 4633.333  0.7833333 70.95667  8.283333
## 4          West FALSE  4582.5714 4550.143  1.2571429 71.70000  6.828571
## 5     Northeast  TRUE  1360.5000 4307.500  0.7750000 71.43500  3.650000
## 6 North Central  TRUE  2372.1667 4588.833  0.6166667 72.57667  2.266667
## 7          West  TRUE   970.1667 4880.500  0.7500000 70.69167  7.666667
##    HS.Grad    Frost      Area
## 1 52.06000 110.6000  21838.60
## 2 44.34375  64.6250  54605.12
## 3 53.36667 120.0000  56736.50
## 4 60.11429  51.0000  91863.71
## 5 56.35000 160.5000  13519.00
## 6 55.66667 157.6667  68567.50
## 7 64.20000 161.8333 184162.17
```

- Example 2: what are the min, median and max of population in a region?

```
# NOTE: can use aggregate on functions defined "on the fly"
#    formally, known as "anonymous function"
aggregate(
        x = list(Population = state.x77.df$Population),
        by = list(Region = state.region),
        FUN = function(x) c(min = min(x), median = median(x), max = max(x))) # inside: anonymous function

##          Region Population.min Population.median Population.max
## 1     Northeast          472.0            3100.0        18076.0
## 2         South          579.0            3710.5        12237.0
## 3 North Central          637.0            4255.0        11197.0
## 4          West          365.0            1144.0        21198.0
```

Taking stock

- When we want to connect data frames and stack them together, can use rbind
- But whenever we want to combine data frames "row-wise", use merge, never use cbind
- Aggregate is a useful function to "collapse" a data frame to compute summary statistics
  - two classes of notations that give us almost equivalent results

```
# notation 1: x-by-FUN
aggregate(x = data.frame$var2, by = list(data.frame$var1), FUN = mean)

# notation 2: formula-data-FUN
aggregate(formula = var2 ~ var1, data = data.frame, FUN = mean)
```

  - there are a few other classes of functions that are more flexible: reshape (today) and data.table

The reshape package: melt() and cast()

Reshape

- How do we re-structure data?
- Specifically, how do we make data frames that are
  - "wide", i.e. with many columns?
  - or "long", i.e. with few columns but more key variables?
- Let's go back to the original grades example

Let's say that there are many many other classes...

```
# Recall the original grades data
grade.data
```

```
##       name year mkt440 gba464
## 1    Anita 2014     A-      A
## 2    Anita 2015      A   <NA>
## 3 Harikesh 2015     A-   <NA>
## 4 Harikesh 2016      A      A
## 5    Linda 2015   <NA>     A-
## 6   Yufeng 2016      B      B
## 7   Yufeng 2017   <NA>      B
```

```
# Let's say they had two more classes, GBA 462 and MKT 436

grades.gba462 <- data.frame(
        name = c("Harikesh", "Yufeng", "Linda", "Linda", "Anita"),
        year = c(2015, 2016, 2015, 2016, 2015),
        gba462 = c("A-", "B", "A", "A-", "A-")
)

grades.mkt436 <- data.frame(
        name = c("Harikesh", "Yufeng", "Yufeng", "Linda", "Anita"),
        year = c(2015, 2016, 2017, 2016, 2014),
        mkt436 = c("A-", "B", "B", "A", "A-")
)
```

...and soon the grade sheet becomes "wide"

```
# merge GBA 462 into the two grade lists
grade.data <- merge(grade.data, grades.gba462,
        by = c("name", "year"), all = TRUE)

# merge MKT 436 into the three grade lists
grade.data <- merge(grade.data, grades.mkt436,
        by = c("name", "year"), all = TRUE)

# and soon the grade sheet becomes wide
grade.data
```

```
##       name year mkt440 gba464 gba462 mkt436
## 1    Anita 2014     A-      A   <NA>     A-
## 2    Anita 2015      A   <NA>     A-   <NA>
## 3 Harikesh 2015     A-   <NA>     A-     A-
## 4 Harikesh 2016      A      A   <NA>   <NA>
## 5    Linda 2015   <NA>     A-      A   <NA>
## 6    Linda 2016   <NA>   <NA>     A-      A
## 7   Yufeng 2016      B      B      B      B
## 8   Yufeng 2017   <NA>      B   <NA>      B
```

More formal example

```
# wide format
df1

##   index0 var1 var2 var3 var4
## 1      1    2    5    4    2
## 2      2    3    6   -1    9

# long format
df2

##   index0 index1 var
## 1      1   var1   2
## 2      2   var1   3
## 3      1   var2   5
## 4      2   var2   6
## 5      1   var3   4
## 6      2   var3  -1
## 7      1   var4   2
## 8      2   var4   9
```

melt()

- Objective is to get the following "long" data frame from the "wide" data frame after merge:

name	year	course	grade
Anita	2014	gba464	A
Anita	2014	mkt436	A-
Anita	2014	mkt440	A-
Anita	2015	gba462	A-
Anita	2015	mkt440	A
Harikesh	2015	gba462	A-
	...		
Linda	2016	gba462	A-
Linda	2016	mkt436	A

- Can achieve this using melt() in the reshape package
  - combined with cast(), can achieve powerful results

```r
# use melt() in reshape library

# first get the 'reshape' library
library('reshape')

# 'melt' the wide data into long
#    argument id is the keys of the wide data format
grade.data.long <- melt(grade.data, id = c("name", "year"))

# check results
grade.data.long
```

```
##        name year variable value
## 1     Anita 2014   mkt440    A-
## 2     Anita 2015   mkt440     A
## 3  Harikesh 2015   mkt440    A-
## 4  Harikesh 2016   mkt440     A
## 5     Linda 2015   mkt440  <NA>
## 6     Linda 2016   mkt440  <NA>
## 7    Yufeng 2016   mkt440     B
## 8    Yufeng 2017   mkt440  <NA>
## 9     Anita 2014   gba464     A
## 10    Anita 2015   gba464  <NA>
## 11 Harikesh 2015   gba464  <NA>
## 12 Harikesh 2016   gba464     A
## 13    Linda 2015   gba464    A-
## 14    Linda 2016   gba464  <NA>
## 15   Yufeng 2016   gba464     B
## 16   Yufeng 2017   gba464     B
```

```r
# clean up

# rename variables (melt always name variables into "variable" and "value")
colnames(grade.data.long)[c(3, 4)] <- c("course", "grade")

# clear NAs
grade.data.long <- grade.data.long[!is.na(grade.data.long$grade), ]

# check results
grade.data.long
```

```
##         name year course grade
## 1      Anita 2014 mkt440    A-
## 2      Anita 2015 mkt440     A
## 3   Harikesh 2015 mkt440    A-
## 4   Harikesh 2016 mkt440     A
## 7     Yufeng 2016 mkt440     B
## 9      Anita 2014 gba464     A
## 12  Harikesh 2016 gba464     A
## 13     Linda 2015 gba464    A-
## 15    Yufeng 2016 gba464     B
## 16    Yufeng 2017 gba464     B
## 18     Anita 2015 gba462    A-
## 19  Harikesh 2015 gba462    A-
## 21     Linda 2015 gba462     A
## 22     Linda 2016 gba462    A-
## 23    Yufeng 2016 gba462     B
## 25     Anita 2014 mkt436    A-
## 27  Harikesh 2015 mkt436    A-
```

cast()

- We saw that melt() is a specific function aimed at "melting" data into long format
- But cast() is a very flexible function
- For example, it can be used to reverse melt() and get us back to wide data
- It can also be used on a wide variety of tasks that involve restructuring data

```
# common usage looks like:
new_data <- cast(data = melted_data, formula = x ~ y, value = "var")

# understanding what the formula means is key to understand cast()
#    and yes it's different from the formula in aggregate()
```

What does the formula mean in cast()?

- Formula in cast (and specifically in cast!) specifies which variables go to which *dimension*
- For example, the following example specifies that rows are defined by unique values of v1 and v2 (i.e., v1 and v2 combined are keys), whereas each unique value of v3 will occupy a column

```
cast(data = melted_data, formula = v1 + v2 ~ v3, value = "var")
```

- Example 1: reverse melt() and get back to wide data
  - long data frame is on keys *name* + *year* + *course*
  - resulting data frame should be on keys *name* + *year*, with *course* being in separate columns

```
# could simply "cast()" it back
grade.data.wide <- cast(data = grade.data.long, # which data to work on
                        formula = name + year ~ course, # formula = dimension/structure
                        value = "grade")        # variable to work on
# explanation: think about x being name + year (keys of the new data frame)
#    and course being separate columns, hence takes "y"

# check results
grade.data.wide
```

```
##       name year mkt440 gba464 gba462 mkt436
## 1    Anita 2014     A-      A   <NA>     A-
## 2    Anita 2015      A   <NA>     A-   <NA>
## 3 Harikesh 2015     A-   <NA>     A-     A-
## 4 Harikesh 2016      A      A   <NA>   <NA>
## 5    Linda 2015   <NA>     A-      A   <NA>
## 6    Linda 2016   <NA>   <NA>     A-      A
## 7   Yufeng 2016      B      B      B      B
## 8   Yufeng 2017   <NA>      B   <NA>      B
```

- Example 2: number of classes taken in the entire year
  - resulting data frame should be on keys *name* + *year*
  - one value column taking the length of each group of values

```
# cast can be used similar to aggregate
#    in which case add the fun.aggregate argument
number.courses <- cast(data = grade.data.long,
                  formula = name + year ~ .,    # '.' means no variable, aggregate into single scalar
                  fun.aggregate = length,       # function to apply when aggregate
                  value = "grade")     # can skip because it does not matter for length

# then rename and clean up
colnames(number.courses)[3] <- "number.courses"
# explanation: x is still name + year but y is collapsed, and therefore use '.'

# check results
number.courses

##       name year number.courses
## 1    Anita 2014              3
## 2    Anita 2015              2
## 3 Harikesh 2015              3
## 4 Harikesh 2016              2
## 5    Linda 2015              2
## 6    Linda 2016              2
## 7   Yufeng 2016              4
## 8   Yufeng 2017              2
```

- Example 3a: number of unique classes ever taken by each student

```
# can further collpase on the x dimension
number.courses.ever <- cast(data = grade.data.long,
                            formula = name ~ .,      # . aggregate into single scalar
                            fun.aggregate = length.unique,  # we have defined length.unique before
                            value = "course")        # note: default would have been grade, cannot ski

colnames(number.courses.ever)[2] <- "number.courses.ever"

# check results
number.courses.ever
```

```
##       name number.courses.ever
## 1    Anita                   4
## 2 Harikesh                   4
## 3    Linda                   3
## 4   Yufeng                   4
```

- Example 3b: number of classes taken in each year, but in wide format

```
# can further collpase on the x dimension
number.courses.wide <- cast(grade.data.long,
                            formula = name ~ year,  # . aggregate into single scalar
                            fun.aggregate = length) # function to apply when aggregate / skipped value

## Using grade as value column.   Use the value argument to cast to override this choice

# check results
number.courses.wide

##       name 2014 2015 2016 2017
## 1    Anita    3    2    0    0
## 2 Harikesh    0    3    2    0
## 3    Linda    0    2    2    0
## 4   Yufeng    0    0    4    2

# some column naming issues that need fixing
```

- Example 4: let's now look at airquality data
  - to illustrate some of the more advanced usage of cast()
  - first let's load data and melt it

```
# use built-in airquality data; change variable names to lower case
names(airquality) <- tolower(names(airquality))

# show the data
head(airquality)

##   ozone solar.r wind temp month day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6

# QUESTION FOR YOU:
#  what are the keys for this data set?

# melt
aqm <- melt(airquality, id = c("month", "day"), na.rm=TRUE)     # na.rm removes NAs
head(aqm)

##   month day variable value
## 1     5   1    ozone    41
## 2     5   2    ozone    36
## 3     5   3    ozone    12
## 4     5   4    ozone    18
## 5     5   6    ozone    28
## 6     5   7    ozone    23
```

- Example 4a: we can cast data frame into 3 dimensional arrays
  - for each variable, give a matrix of data
  - structure it so that days are in rows and months are in columns

```
# can cast the data frame into a 3-dimensional array
res1 <- cast(aqm, formula = day ~ month ~ variable)     # what are the 3 dimensions?

# show results, only for first 7 days
res1[1:7, , ]

## , , variable = ozone
##
##    month
## day  5  6   7   8  9
##   1 41 NA 135  39 96
##   2 36 NA  49   9 78
##   3 12 NA  32  16 73
##   4 18 NA  NA  78 91
##   5 NA NA  64  35 47
##   6 28 NA  40  66 32
##   7 23 29  77 122 20
##
## , , variable = solar.r
##
##    month
## day   5   6   7   8   9
##   1 190 286 269  83 167
##   2 118 287 248  24 197
##   3 149 242 236  77 183
##   4 313 186 101  NA 189
##   5  NA 220 175  NA  95
##   6  NA 264 314  NA  92
##   7 299 127 276 255 252
##
## , , variable = wind
##
```

- Example 4b: calculate averages of each variable, by month
  - we now essentially have something like an "aggregate by"

```
res2 <- cast(aqm, month ~ . | variable, fun.aggregate = mean)

# note:   'month ~ .' represents collapse everything into month level
#         '| variable' inidicates do everything for each variable

# show part of the results
res2[1:2]

## $ozone
##   month    (all)
## 1     5 23.61538
## 2     6 29.44444
## 3     7 59.11538
## 4     8 59.96154
## 5     9 31.44828
##
## $solar.r
##   month    (all)
## 1     5 181.2963
## 2     6 190.1667
## 3     7 216.4839
## 4     8 171.8571
## 5     9 167.4333
```

- Example 4c: store multiple results and expand them into different columns
  - use result_variable in the formula

```
# aggregate by month and for each variable
#    now operate on multiple functions at the same time
res3 <- cast(aqm, month ~ variable + result_variable, fun.aggregate = c(min, median, mean, max))
head(res3)

##   month ozone_min ozone_median ozone_mean ozone_max solar.r_min solar.r_median
## 1     5         1           18   23.61538       115           8          194.0
## 2     6        12           23   29.44444        71          31          188.5
## 3     7         7           60   59.11538       135           7          253.0
## 4     8         9           52   59.96154       168          24          197.5
## 5     9         7           23   31.44828        96          14          192.0
##   solar.r_mean solar.r_max wind_min wind_median wind_mean wind_max temp_min
## 1     181.2963         334      5.7        11.5 11.622581     20.1       56
## 2     190.1667         332      1.7         9.7 10.266667     20.7       65
## 3     216.4839         314      4.1         8.6  8.941935     14.9       73
## 4     171.8571         273      2.3         8.6  8.793548     15.5       72
## 5     167.4333         259      2.8        10.3 10.180000     16.6       63
##   temp_median temp_mean temp_max
## 1          66  65.54839       81
## 2          78  79.10000       93
## 3          84  83.90323       92
## 4          82  83.96774       97
## 5          76  76.90000       93
```

Taking stock

- Recall that each dataset should be define by a set of keys
  - what are keys?
- The reshape package focuses on changing the "shape" of the dataset by changing its keys
- Most important feature of cast(): turn a dataset with keys X + Y ~ . (e.g. grades, by name and year) into X ~ Y (e.g. grades in each year as separate variables, by name)
- In addition, reshape can also substitute aggregate() to some extent
  - and later we'll see related functionalities from data.table

Summary

- Data frame is a type of data structure in R that is very easy to work with
  - tabular (similar to a matrix), but permits multiple data types
  - (similar to a list) permits multiple data types but it is tabular
- Although not required by definition, we should organize data with clearly-defined keys
- Manipulations of data structure (these few commands can carry you a long way)
  - merge() combines two data frames by the keys of at least one of them
  - aggregate() collapses data to a subset of keys
  - melt() converts a data frame to "long format' with more keys
  - cast() reshapes data into a form with the desired set of keys
  - what's left on the table about data structure: data.table