lab4

Irlanda Ayon-Moreno

4/7/2018

Lab 4: Data Frame Basics

Gaston Sanchez

Learning Objectives:

- Importing Data Tables in R
- Default reading-table functions
- R package `"readr"`
- Get started with data frames
- Understand basic operations
- Understand bracket and dollar notations

Abalone Data Set

The first data set that you will working with is the **Abalone Data Set** that is part of the UCI Machine Learning Repository

The location of the data file is:

http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data

The location of the data dictionary (description of the data) is:

http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.names

Look at both the dataset file, and the file with its description, and answer the following questions:

- What's the character delimiter?
  A comma

- Is there a row for column names?
  No

- Are there any missing values? If so, how are they codified?
  No missing values

- What is the data type of each column?
  Integer

One basic way to read this file in R is by passing the url location of the file directly to any of the `read.table()` functions:

```
url <- "http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
abalone <- read.table(url, sep = ",")
```

Getting a Local Copy of the Data

My suggestion when reading datasets from the Web, is to always try to get a local copy of the data file in your machine (as long as you have enough free space to save it in your computer). To do this, you can use the function `download.file()` and specify the url address, and the name of the file that will be created in your computer. For instance, to save the abalone data file in **your working directory**, type the following commands directly on the R console:

```
# download copy to your working directory
origin <- 'http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data'
destination <- 'abalone.data'
download.file(origin, destination)
```

Some Bash Commands

Before describing some of the reading-table functions in R, let's practice some basic bash commands to inspect the downloaded data file.

- For those of you using Gitbash, go to your browser and open a tab to get the *Linux Man Pages Online* available in the website: http://man.he.net/
- Open the terminal (e.g. Mac terminal or Gitbash) and change directories to the folder that contains the downloaded file `abalone.data`.
- Use the `file` command to know what type of file is `abalone.data`.
- Use the *word count* command `wc` to obtain information about: 1) line count, 2) word count, and 3) byte count, of the `abalone.data` file.
- See the `man` documentation of `wc` and learn what option you should use to otabin only the number of lines in `abalone.data`.
- Use `head` to take a peek at the first lines (10 lines by default) of `abalone.data`
- See the `man` documentation of `head` and learn what option you should use to display only the first 5 files in `abalone.data`.
- Use `tail` to take a peek at the last lines (10 lines by default) of `abalone.data`
- See the `man` documentation of `tail` and learn what option you should use to display only the last 3 files in `abalone.data`.
- Use the `less` command to look at the contents of `abalone.data` (this command opens a *paginator* so you can move up and down the contents of the file).

Basic Importing

Now that you have a local copy of the dataset, you can read it in R with `read.table()` like so:

```
# reading data from your working directory
abalone <- read.table("abalone.data", sep = ",")
```

Once you read a data table, you may want to start looking at its contents, usually taking a peek at a few rows. This can be done with `head()` and/or with `tail()`:

```
# take a peek of first rows
head(abalone, 3)
```

```
##   V1    V2    V3    V4     V5     V6     V7   V8 V9
## 1  M 0.455 0.365 0.095 0.5140 0.2245 0.1010 0.15 15
## 2  M 0.350 0.265 0.090 0.2255 0.0995 0.0485 0.07  7
## 3  F 0.530 0.420 0.135 0.6770 0.2565 0.1415 0.21  9
```

```
# take a peek of last rows
tail(abalone, 3)
```

```
##      V1    V2    V3    V4     V5     V6     V7    V8 V9
## 4175  M 0.600 0.475 0.205 1.1760 0.5255 0.2875 0.308  9
## 4176  F 0.625 0.485 0.150 1.0945 0.5310 0.2610 0.296 10
## 4177  M 0.710 0.555 0.195 1.9485 0.9455 0.3765 0.495 12
```

Likewsie, you may also want to examine how R has decided to take care of the storage details (what data type is used for each column?). Use the function `str()` to check the structure of the data frame:

```
# check data frame's structure
str(abalone, vec.len = 1)
```

```
## 'data.frame':    4177 obs. of  9 variables:
##  $ V1: Factor w/ 3 levels "F","I","M": 3 3 ...
##  $ V2: num  0.455 0.35 ...
##  $ V3: num  0.365 0.265 ...
##  $ V4: num  0.095 0.09 ...
##  $ V5: num  0.514 ...
##  $ V6: num  0.225 ...
##  $ V7: num  0.101 0.0485 ...
##  $ V8: num  0.15 0.07 ...
##  $ V9: int  15 7 ...
```

Detailed information about the columns

So far we have been able to read the data file in R. But we are missing a few things. First, we don't have names for the columns. Second, it would be nice if we could specify the data types of each column instead of letting R guess how to handle each data type.

According to the description of the Abalone data set, we can assign the following data types to each of the columns as:

Name	Data Type
Sex	character
Length	continuous
Diameter	continuous
Height	continuous
Whole weight	continuous
Shucked weight	continuous
Viscera weight	continuous
Shell weight	continuous
Rings	integer

Let's create a vector of columns names, and another vector of data types:

```
# vector of column names
column_names <- c(
    'sex',
    'length',
    'diameter',
    'height',
    'whole_weight',
    'shucked_weight',
    'viscera_weight',
    'shell_weight',
    'rings'
)

# vector of data types (for each column)
column_types <- c(
    'character',
    'real',
    'real',
    'real',
    'real',
    'real',
    'real',
    'real',
    'integer'
)
```

Optionally, we could also specify a type "factor" for the variable `sex` since this is supposed to be in nominal scale (i.e. it is a categorical variable). Also note that the variable `rings` is supposed to be integers, therefore we can choose an `integer` vector for this column.

Now we can re-read the table in a more complete (and usually more efficient) way:

```r
abalone <- read.table(
    'abalone.data',
    col.names = column_names,
    colClasses = column_types,
    sep = ","
)

# check its structure again
str(abalone, vec.len = 1)
```

```
## 'data.frame':    4177 obs. of  9 variables:
##  $ sex           : chr  "M" ...
##  $ length        : num  0.455 0.35 ...
##  $ diameter      : num  0.365 0.265 ...
##  $ height        : num  0.095 0.09 ...
##  $ whole_weight  : num  0.514 ...
##  $ shucked_weight: num  0.225 ...
##  $ viscera_weight: num  0.101 0.0485 ...
##  $ shell_weight  : num  0.15 0.07 ...
##  $ rings         : int  15 7 ...
```

Your turn

- Read the Abalone data with the `read.csv()` function.
- Use the inputs `col.names` and `colClasses` to specify column names and their data types.
- Look at the data description in the following link:

http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.names and confirm the following statistics:

	Length	Diam	Height	Whole	Shucked	Viscera	Shell	Rings
Min	0.075	0.055	0.000	0.002	0.001	0.001	0.002	1
Max	0.815	0.650	1.130	2.826	1.488	0.760	1.005	29
Mean	0.524	0.408	0.140	0.829	0.359	0.181	0.239	9.934
SD	0.120	0.099	0.042	0.490	0.222	0.110	0.139	3.224

Import Abalone data with `read_csv()`

In addition to the built-in functions for importing tables in R, there is also the set of functions from the R package `"readr"`:

- `read_delim()`
- `read_csv()`
- `read_tsv()`
- `read_csv2()`
- `read_fwf()`
- `read_table()`

Take a look at this post for some examples:

https://blog.rstudio.com/2015/04/09/readr-0-1-0/

- Use `read_csv()` to import the abalone data set.
- Use the argument `col_names` to specify the column names.
- Learn how to use the argument `col_types` to specify the data type for each column.

```
library(readr)
help("read_csv")
abalone <- read_csv('abalone.data',
                    col_names = c('sex',
                                  'length',
                                  'diameter',
                                  'height',
                                  'whole_weight',
                                  'shucked_weight',
                                  'viscera_weight',
                                  'shell_weight',
                                  'rings'
                    ),
                    col_types = list(col_character(),
                                     col_double(),
                                     col_double(),
                                     col_double(),
                                     col_double(),
                                     col_double(),
                                     col_double(),
                                     col_double(),
                                     col_integer()
                                     )
                    )


# check its structure again
str(abalone)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    4177 obs. of  9 variables:
##  $ sex           : chr  "M" "M" "F" "M" ...
##  $ length        : num  0.455 0.35 0.53 0.44 0.33 0.425 0.53 0.545 0.475 0.55 ...
##  $ diameter      : num  0.365 0.265 0.42 0.365 0.255 0.3 0.415 0.425 0.37 0.44 ...
##  $ height        : num  0.095 0.09 0.135 0.125 0.08 0.095 0.15 0.125 0.125 0.15 ...
##  $ whole_weight  : num  0.514 0.226 0.677 0.516 0.205 ...
##  $ shucked_weight: num  0.2245 0.0995 0.2565 0.2155 0.0895 ...
##  $ viscera_weight: num  0.101 0.0485 0.1415 0.114 0.0395 ...
##  $ shell_weight  : num  0.15 0.07 0.21 0.155 0.055 0.12 0.33 0.26 0.165 0.32 ...
##  $ rings         : int  15 7 9 10 7 8 20 16 9 19 ...
##  - attr(*, "spec")=List of 2
##   ..$ cols   :List of 9
```

```
##   .. ..$ sex           : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ length        : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ diameter      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ height        : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ whole_weight  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ shucked_weight: list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ viscera_weight: list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ shell_weight  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ rings         : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   ..$ default: list()
##   .. ..- attr(*, "class")= chr  "collector_guess" "collector"
##   ..- attr(*, "class")= chr "col_spec"
```

Pittsburgh Bridges Data Set

This data set is part of the UCI Machine Learning Repository:

http://archive.ics.uci.edu/ml/datasets/Pittsburgh+Bridges

The data Description is here:

http://archive.ics.uci.edu/ml/machine-learning-databases/bridges/bridges.names

The Data file is here:

http://archive.ics.uci.edu/ml/machine-learning-databases/bridges/bridges.data.version1

Read the description, and take a look at the data set:

- Are there column names?
- What is the field separator?
- Are there any missing values?
- What is the character for missing values (if any)?
- What is the data type of each variable (i.e. column)?
- Download a copy of the data to your computer (use `download.file()`) and save it in a file named `bridges.data`

Reading the Data

- Create a vector of column names

```
bnames <- c('id', 'river', 'location', 'erected', 'purpose', 'length', 'lanes', 'clearg', 'td'
```

- Create a vector of column types
- Use the function `read.table()` to import the data. Name it `bridges1`.

```
bridges1 <- read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/bridges/bridg
                       stringsAsFactors = FALSE,
                       sep = ",",
                       col.names = bnames
                       )
head(bridges1,3)
```

```
##   id river location erected  purpose length lanes clearg      td material
## 1 E1     M        3    1818  HIGHWAY      ?     2      N THROUGH     WOOD
## 2 E2     A       25    1819  HIGHWAY   1037     2      N THROUGH     WOOD
## 3 E3     A       39    1829 AQUEDUCT      ?     1      N THROUGH     WOOD
##    span rel type
## 1 SHORT   S WOOD
## 2 SHORT   S WOOD
## 3     ?   S WOOD
```

- Use the function `read.csv()` to import the data. Name it `bridges2`.

```
bridges2 <- read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/bridges/bridges
                     stringsAsFactors = FALSE,
                     header = FALSE)
```

- How would you specify the argument `colClasses` to import just the first five columns? (check the documentation of `?read.table`). Name this data frame `bridges3`.

Basic Inspection

Use functions to start examining the `bridges1` data frame:

- `str()`
- `summary()`
- `head()` and `tail()`
- `dim()`
- `names()`
- `colnames()`
- `nrow()`
- `ncol()`

Optional: Want to do more?

Write R code to find:

- Year of the oldest erected bridge

```
min(bridges1[ , 'erected'])
```

```
## [1] 1818
```

- Year of the most recent erected bridge

```
max(bridges1[ , 'erected'])
```

```
## [1] 1986
```

- Frequency of bridges by purpose

```
table(bridges1[ , 'purpose'])
```

```
##
## AQUEDUCT  HIGHWAY       RR     WALK
##        4       71       32        1
```

- Frequency of materials

```
table(bridges1[ , 'material'])
```

```
##
##     ?  IRON STEEL  WOOD
##     2    11    79    16
```

- Average length of the bridges

```
mean(bridges1[ , 'length'])
```

```
## Warning in mean.default(bridges1[, "length"]): argument is not numeric or
## logical: returning NA
```

```
## [1] NA
```

Creating Data Frames

Here's a table with the starting lineup of the Golden State Warriors:

Player	Position	Salary	Points	PPG	Rookie
Thompson	SG	16,663,575	1742	22.3	FALSE
Curry	PG	12,112,359	1999	25.3	FALSE
Green	PF	15,330,435	776	10.2	FALSE
Durant	SF	26,540,100	1555	25.1	FALSE
Pachulia	C	2,898,000	426	6.1	FALSE

- Start by creating vectors for each of the columns.

```
player <- c('Thompson', 'Curry', 'Green', 'Durant', 'Pachulia')
position <- c('SG', 'PG', 'PF', 'SF', 'C')
salary <- c(16663575, 12112359, 15330435, 26540100, 2898000)
points <- c(1742, 1999, 776, 1555, 426)
ppg <- c(22.3, 25.3, 10.2, 25.1, 6.1)
rookie <- rep(FALSE, 5)
```

- Use the vectors to create a first data frame with `data.frame()`.

```
df1 <- data.frame(
  player,
  position,
  salary,
  points,
  ppg,
  rookie
)
df1
```

```
##     player position   salary points  ppg rookie
## 1 Thompson       SG 16663575   1742 22.3  FALSE
## 2    Curry       PG 12112359   1999 25.3  FALSE
## 3    Green       PF 15330435    776 10.2  FALSE
## 4   Durant       SF 26540100   1555 25.1  FALSE
## 5 Pachulia        C  2898000    426  6.1  FALSE
```

- Check that this data frame is of class `"data.frame"`, and that it is also a list.

```
class(df1)
```

```
## [1] "data.frame"
```

```
is.list(df1)
```

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

- Create another data frame by first starting with a `list()`, and then passing the list to `data.frame()`.

```
variables_list <- list(
  player = player,
  position = position,
  salary = salary,
  points = points,
  ppg = ppg,
  rookie = rookie
)

df2 <- data.frame(variables_list)
```

- What would you do to obtain a data frame such that when you check its structure `str()` the variables are:

  - *Player* as character
  - *Position* as factor
  - *Salary* as numeric or real (ignore the commas)
  - *Points* as integer
  - *PPG* as numeric or real
  - *Rookie* as logical

```
df3 <- data.frame(
  player = player,
  position = as.factor(position),
  salary = salary,
  points = as.integer(points),
  ppg = ppg,
  rookie = rookie,
  stringsAsFactors = FALSE
)
str(df3)
```

```
## 'data.frame':    5 obs. of  6 variables:
##  $ player  : chr  "Thompson" "Curry" "Green" "Durant" ...
##  $ position: Factor w/ 5 levels "C","PF","PG",..: 5 3 2 4 1
##  $ salary  : num  16663575 12112359 15330435 26540100 2898000
##  $ points  : int  1742 1999 776 1555 426
##  $ ppg     : num  22.3 25.3 10.2 25.1 6.1
##  $ rookie  : logi  FALSE FALSE FALSE FALSE FALSE
```

- Find out how to use the *column binding* function `cbind()` to create a tabular object with the vectors created in step 1 (inspect what class of object is obtained with `cbind()`).

```
df4 <- cbind(
  player,
  position,
  salary,
  points,
  ppg,
  rookie
)
class(df4)
```

```
## [1] "matrix"
```

- How could you convert the object in the previous step into a data frame?

```
df4 <- data.frame(df4)
```

NBA Players Data

Now that you've seen some of the most basic operations to manipulate data frames, let's apply them on a data set about NBA players. The corresponding data file is `nba2017-players.csv`, located in

the `data/` folder of the course github repository. This file contains 15 variables measured on 441 players.

First download a copy of the csv file to your computer.

```
# download csv file into your working directory
csv <- "https://github.com/ucb-stat133/stat133-fall-2017/raw/master/data/nba2017-players.csv"
download.file(url = csv, destfile = 'nba2017-players.csv')
```

To import the data in R you can use the function `read.csv()`:

```
dat <- read.csv('nba2017-players.csv', stringsAsFactors = FALSE)
```

Notice that I'm specifying the argument `stringsAsFactors = FALSE` to avoid the conversion of characters into R factors. Why do you think this is a good practice?

All the default reading table functions generate a data frame. Typically, everytime I read a new data set which I'm not familiar with, or a data set that I haven't worked on in a long time, I always like to call a couple of functions to inspect its contents:

- `dim()`
- `head()`
- `tail()`
- `str()`
- `summary()`

A first check-up is to examine the dimensions of the data frame with `dim()`:

```
# dimensions (# of rows, # of columns)
dim(dat)
```

```
## [1] 441  15
```

If you know in advanced how many rows and columns are in the data table, this is a good way to make sure that R was able to read all the records.

Then, depending on the size of the data, you may want to take a peek at its contents with `head()` or `tail()`, just to get an idea of what the data looks like:

```
# display first few rows
head(dat)
```

```
##              player team position height weight age experience
## 1        Al Horford  BOS        C     82    245  30          9
## 2      Amir Johnson  BOS       PF     81    240  29         11
## 3     Avery Bradley  BOS       SG     74    180  26          6
## 4 Demetrius Jackson  BOS       PG     73    201  22          0
## 5      Gerald Green  BOS       SF     79    205  31          9
## 6     Isaiah Thomas  BOS       PG     69    185  27          5
##                          college   salary games minutes points points3
## 1         University of Florida 26540100    68    2193    952      86
## 2                               12000000    80    1608    520      27
## 3 University of Texas at Austin  8269663    55    1835    894     108
## 4      University of Notre Dame  1450000     5      17     10       1
```

```
## 5                                1410598    47     538    262      39
## 6      University of Washington  6587132    76    2569   2199     245
##   points2 points1
## 1     293     108
## 2     186      67
## 3     251      68
## 4       2       3
## 5      56      33
## 6     437     590
```

For a more detailed description of how R is treating the data type in each column, you should use the structure function `str()`.

```
# check the structure
str(dat, vec.len = 1)
```

```
## 'data.frame':    441 obs. of  15 variables:
##  $ player    : chr  "Al Horford" ...
##  $ team      : chr  "BOS" ...
##  $ position  : chr  "C" ...
##  $ height    : int  82 81 ...
##  $ weight    : int  245 240 ...
##  $ age       : int  30 29 ...
##  $ experience: int  9 11 ...
##  $ college   : chr  "University of Florida" ...
##  $ salary    : num  26540100 ...
##  $ games     : int  68 80 ...
##  $ minutes   : int  2193 1608 ...
##  $ points    : int  952 520 ...
##  $ points3   : int  86 27 ...
##  $ points2   : int  293 186 ...
##  $ points1   : int  108 67 ...
```

This function `str()` displays the dimensions of the data frame, and then a list with the name of all the variables, and their data types (e.g. `chr` character, `num` real, etc). The argument `vec.len = 1` indicates that just the first element in each column should be displayed.

When working with data frames, remember to always take some time inspecting the contents, and checking how R is handling the data types. It is in these early stages of data exploration that you can catch potential issues in order to avoid disastrous consequences or bugs in subsequent stages.

Your turn:

Use bracket notation, the dollar operator, as well as concepts of logical subsetting and indexing to:

- Display the last 5 rows of the data.
- Display those rows associated to players having height less than 70 inches tall.

```
dat[dat[ ,'height'] < 70, ]
```

```
##           player team position height weight age experience
## 6  Isaiah Thomas  BOS       PG     69    185  27          5
## 24    Kay Felder  CLE       PG     69    176  21          0
##                     college  salary games minutes points points3 points2
## 6  University of Washington 6587132    76    2569   2199     245     437
## 24       Oakland University  543471    42     386    166       7      55
##    points1
## 6      590
## 24      35
```

- Of those players that are centers (position `C`), display their names and salaries.

```
head(dat[dat$position == 'C', c('player', 'salary')], 3)
```

```
##          player   salary
## 1    Al Horford 26540100
## 12 Kelly Olynyk  3094014
## 15 Tyler Zeller  8000000
```

- Create a data frame `durant` with Kevin Durant's information (i.e. row).

```
durant <- dat[dat$player == 'Kevin Durant', ]
durant
```

```
##           player team position height weight age experience
## 228 Kevin Durant  GSW       SF     81    240  28          9
##                            college   salary games minutes points points3
## 228 University of Texas at Austin 26540100    62    2070   1555     117
##     points2 points1
## 228     434     336
```

- Create a data frame `ucla` with the data of players from college UCLA ("University of California, Los Angeles").

```
ucla <- dat[dat["college"] == "University of California, Los Angeles",]
head(ucla, 3)
```

- Create a data frame `rookies` with those players with 0 years of experience.

```
rookies <- dat[dat["experience"] == 0, ]
head(rookies, 3)
```

- Create a data frame `rookie_centers` with the data of Center rookie players.

```
rookie_centers <- rookies[rookies$position == 'C', ]
head(rookie_centers, 3)
```

- Create a data frame `top_players` for players with more than 50 games and more than 100 minutes played.

```
top_players <- dat[(dat$games > 50 & dat$minutes >100), ]
head(top_players, 3)
```

- What's the largest height value?

```
max(dat$height)
```

```
## [1] 87
```

- What's the minimum height value?

```
min(dat$height)
```

```
## [1] 69
```

- What's the overall average height?

```
mean(dat$height)
```

```
## [1] 79.1542
```

- Who is the tallest player?

```
dat[dat$height == max(dat$height), "player"]
```

```
## [1] "Edy Tavares"        "Boban Marjanovic"   "Kristaps Porzingis"
```

- Who is the shortest player?

```
dat[dat$height == min(dat$height), "player"]
```

```
## [1] "Isaiah Thomas" "Kay Felder"
```

- Which are the unique teams?

```
unique_teams <- unique(dat$team)
unique_teams
```

```
##  [1] "BOS" "CLE" "TOR" "WAS" "ATL" "MIL" "IND" "CHI" "MIA" "DET" "CHO"
## [12] "NYK" "ORL" "PHI" "BRK" "GSW" "SAS" "HOU" "LAC" "UTA" "OKC" "MEM"
## [23] "POR" "DEN" "NOP" "DAL" "SAC" "MIN" "LAL" "PHO"
```

- How many different teams?

```
length(unique_teams)
```

```
## [1] 30
```

- Who is the oldest player?

```
dat[dat$age == max(dat$age), "player"]
```

```
## [1] "Vince Carter"
```

- What is the median salary of all players?

```
median(dat$salary)
```

```
## [1] 3500000
```

- What is the median salary of the players with 10 years of experience or more?

```
median(dat[dat$experience >= 10, 'salary'])
```

```
## [1] 5644034
```

- What is the median salary of Shooting Guards (SG) and Point Guards (PG)?

```
median(dat[(dat$position == 'SG' | dat$position == 'PG'), 'salary'])
```

```
## [1] 3230690
```

- What is the median salary of Power Forwards (PF), 29 years or older, and 74 inches tall or less?

```
median(dat$salary[dat$position == 'PG' & dat$age >= 29 & dat$height <= 74])
```

```
## [1] 4770262
```

- How many players scored 4 points or less?

```
sum(dat$points <= 4)
```

```
## [1] 7
```

- Who are those players who scored 4 points or less?

```
dat$player[dat$points <= 4]
```

```
## [1] "Chris McCullough" "Michael Gbinije"  "Patricio Garino"
## [4] "Isaiah Taylor"    "Brice Johnson"    "Roy Hibbert"
## [7] "Elijah Millsap"
```

- Who is the player with 0 points?

```
dat$player[dat$points == 0]
```

```
## [1] "Patricio Garino"
```

- How many players are from “University of California, Berkeley”?

```
sum(dat$college == "University of California, Berkeley")
```

```
## [1] 0
```

- Are there any players from “University of Notre Dame”? If so how many and who are they?

```
sum(dat$college == "University of Notre Dame")
```

```
## [1] 3
```

```
dat$player[dat$college == "University of Notre Dame"]
```

```
## [1] "Demetrius Jackson" "Jerian Grant"      "Pat Connaughton"
```

- Are there any players with weight greater than 260 pounds? If so how many and who are they?

```r
sum(dat$weight > 260)
```

```
## [1] 21
```

```r
dat$player[dat$weight > 260]
```

```
##  [1] "Jonas Valanciunas" "Dwight Howard"     "Greg Monroe"
##  [4] "Al Jefferson"      "Kevin Seraphin"    "Cristiano Felicio"
##  [7] "Hassan Whiteside"  "Andre Drummond"    "Boban Marjanovic"
## [10] "Jahlil Okafor"     "Brook Lopez"       "JaVale McGee"
## [13] "Zaza Pachulia"     "DeAndre Jordan"    "Derrick Favors"
## [16] "Jusuf Nurkic"      "Roy Hibbert"       "DeMarcus Cousins"
## [19] "Kosta Koufos"      "Ivica Zubac"       "Timofey Mozgov"
```

- How many players did not attend a college in the US?

```r
sum(dat$college == "")
```

```
## [1] 85
```

- Who is the player with the maximum rate of points per minute?

```r
dat$player[which.max(dat$points / dat$minutes)]
```

```
## [1] "Russell Westbrook"
```

- Who is the player with the maximum rate of three-points per minute?

```r
dat$player[which.max(dat$points3 / dat$minutes)]
```

```
## [1] "Stephen Curry"
```

- Who is the player with the maximum rate of two-points per minute?

```r
dat$player[which.max(dat$points2 / dat$minutes)]
```

```
## [1] "JaVale McGee"
```

- Who is the player with the maximum rate of one-points (free-throws) per minute?

```r
dat$player[which.max(dat$points1 / dat$minutes)]
```

```
## [1] "Russell Westbrook"
```

- Create a data frame `gsw` with the name, height, weight of Golden State Warriors (GSW)

```r
gsw <- dat[dat$team == "GSW", c("player", "height", "weight")]
head(gsw, 3)
```

```
##             player height weight
## 221 Andre Iguodala     78    215
## 222   Damian Jones     84    245
## 223     David West     81    250
```

- Display the data in `gsw` sorted by height in increasing order (hint: see `?sort` and `?order`)

```
order(gsw$height)
```

```
##  [1]  5 14  1  4 10 11 12 13  3  6  8  9 15  2  7
```

```
gsw[order(gsw$height), ]
```

```
##                    player height weight
## 225             Ian Clark     75    175
## 234         Stephen Curry     75    190
## 221        Andre Iguodala     78    215
## 224        Draymond Green     79    230
## 230         Klay Thompson     79    215
## 231           Matt Barnes     79    226
## 232         Patrick McCaw     79    185
## 233     Shaun Livingston     79    192
## 223           David West     81    250
## 226 James Michael McAdoo     81    230
## 228          Kevin Durant     81    240
## 229          Kevon Looney     81    220
## 235         Zaza Pachulia     83    270
## 222          Damian Jones     84    245
## 227          JaVale McGee     84    270
```

- Display the data in gsw by weight in decreasing order (hint: see `?sort` and `?order`)

```
gsw[order(gsw$weight, decreasing = TRUE), ]
```

- Display the player name, team, and salary, of the top 5 highest-paid players (hint: see `?sort` and `?order`)

```
head(dat[order(dat$salary, decreasing = TRUE), c('player', 'team', 'salary')], 5)
```

- Display the player name, team, and points3, of the top 10 three-point players (hint: see `?sort` and `?order`)

```
head(dat[order(dat$points3, decreasing = TRUE), c('player', 'team', 'points3')], 10)
```

Group By

Group-by operations are very common in data analytics. Without dedicated functions, these operations tend to be very hard (labor intensive).

Quick try: Using just bracket notation, try to create a data frame with two columns: the team name, and the team payroll (addition of all player salaries).

So what functions can you use in R to perform group by operations? In base R, the main function for group-by operations is `aggregate()`.

Here's an example using `aggregate()` to get the median salary, grouped by team:

```
aggregate(dat$salary, by = list(dat$team), FUN = median)
```

```
##    Group.1       x
## 1      ATL 3279291
## 2      BOS 4743000
## 3      BRK 1790902
## 4      CHI 2112480
## 5      CHO 6000000
## 6      CLE 5239437
## 7      DAL 2898000
## 8      DEN 3500000
## 9      DET 4625000
## 10     GSW 1551659
## 11     HOU 1508400
## 12     IND 4000000
## 13     LAC 3500000
## 14     LAL 5281680
## 15     MEM 3332940
## 16     MIA 3449000
## 17     MIL 4184870
## 18     MIN 3650000
## 19     NOP 3789125
## 20     NYK 2898000
## 21     OKC 3140517
## 22     ORL 5000000
## 23     PHI 2318280
## 24     PHO 2941440
## 25     POR 4943123
## 26     SAC 5200000
## 27     SAS 2898000
## 28     TOR 5300000
## 29     UTA 2433334
## 30     WAS 4365326
```

The same example above can also be obtained with `aggreagte()` using formula notation like this:

```
aggregate(salary ~ team, data = dat, FUN = median)
```

```
##    team  salary
## 1   ATL 3279291
## 2   BOS 4743000
## 3   BRK 1790902
## 4   CHI 2112480
## 5   CHO 6000000
## 6   CLE 5239437
## 7   DAL 2898000
## 8   DEN 3500000
## 9   DET 4625000
## 10  GSW 1551659
## 11  HOU 1508400
## 12  IND 4000000
```

```
## 13  LAC 3500000
## 14  LAL 5281680
## 15  MEM 3332940
## 16  MIA 3449000
## 17  MIL 4184870
## 18  MIN 3650000
## 19  NOP 3789125
## 20  NYK 2898000
## 21  OKC 3140517
## 22  ORL 5000000
## 23  PHI 2318280
## 24  PHO 2941440
## 25  POR 4943123
## 26  SAC 5200000
## 27  SAS 2898000
## 28  TOR 5300000
## 29  UTA 2433334
## 30  WAS 4365326
```

Here's another example using `aggregate()` to get the average height and average weight, grouped by position:

```
aggregate(dat[ ,c('height', 'weight')], by = list(dat$position), FUN = mean)
```

```
##   Group.1   height   weight
## 1       C 83.25843 250.7978
## 2      PF 81.50562 235.8539
## 3      PG 74.30588 188.5765
## 4      SF 79.63855 220.4699
## 5      SG 77.02105 204.7684
```

The same example above can also be obtained with `aggreagte()` using formula notation like this:

```
aggregate(. ~ position, data = dat[ ,c('position', 'height', 'weight')],
          FUN = mean)
```

```
##   position   height   weight
## 1        C 83.25843 250.7978
## 2       PF 81.50562 235.8539
## 3       PG 74.30588 188.5765
## 4       SF 79.63855 220.4699
## 5       SG 77.02105 204.7684
```

Your turn

- Create a data frame with the average height, average weight, and average age, grouped by position

```
aggregate(. ~ position, data = dat[ ,c('position', 'height', 'weight', 'age')], FUN = mean)
```

- Create a data frame with the average height, average weight, and average age, grouped by team

```
aggregate(dat[ ,c('height', 'weight', 'age')], by = list(dat$team), FUN = mean)
```

- Create a data frame with the average height, average weight, and average age, grouped by team and position.

```
hwa_by_tp <- aggregate(
  dat[ ,c('height', 'weight', 'age')],
  by = list(dat$team, dat$position),
  FUN = "mean")
```

- Difficult: Create a data frame with the minimum salary, median salary, mean salary, and maximum salary, grouped by team and position.

```
salary_by_tp <- aggregate(
  dat$salary, by = list(team = dat$team),
  FUN = function(x) c(min = min(x), med = median(x), avg = mean(x), max = max(x)))
```
