Lab 0.2: dplyr

Introduction to Statistical Computing

Elena Conderana & Sergio Cuenca

02/02/2025

Contents

This lab is to be done outside of class time. You may collaborate with one classmate, but you must identify yourself and his/her name above, in the author's field, and you must submit **your own** lab as this completed .Rmd file.

Installing and loading packages

Below we install `tidyverse` which gives us the packages we need (`purrr` and `dplyr`) needed to complete this lab. We also install the `repurrrsive` package which has the Game of Thrones data set that we'll use for the first couple of questions. Since this may be the first time installing packages for some of you, we'll show you how. If you already have these packages installed, then you can of course skip this part. Note: *do not remove `eval=FALSE` from the above code chunk*, just run the lines below in your console. You can also select "Tools" –> "Install Packages" from the RStudio menu.

```
install.packages("tidyverse")
install.packages("repurrrsive")
```

Now we'll load the packages we need. Note: the code chunk below will cause errors if you try to knit this file without installing the packages first.

```
library(purrr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
library(repurrrsive)
```

Q1. Pipes to base R

For each of the following code blocks, which are written with pipes, write equivalent code in base R (to do the same thing).

1a.

```
letters %>%
  toupper %>%
  paste(collapse="+")
```

```
## [1] "A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z"
```

```r
paste(toupper(letters), collapse="+")
```

```
## [1] "A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z"
```

1b.

```r
"     Ceci n'est pas une pipe     " %>%
  gsub("une", "un", .) %>%
  trimws
```

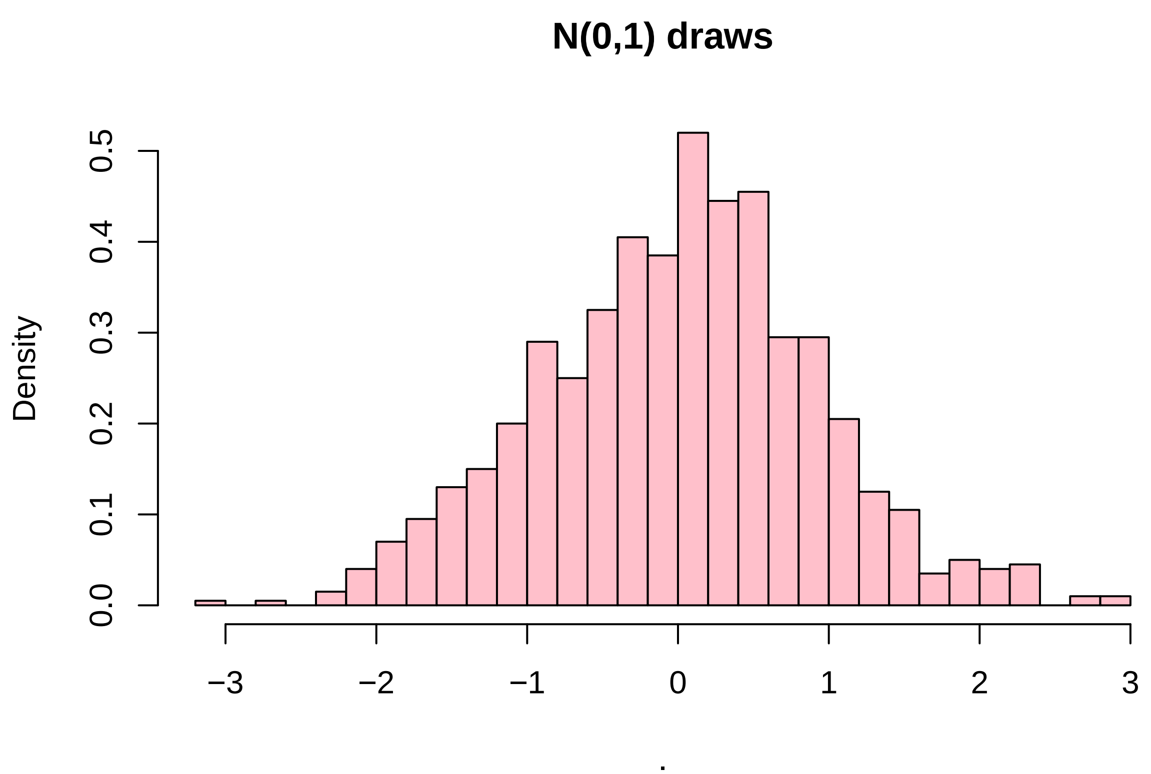
```
## [1] "Ceci n'est pas un pipe"
```

```r
trimws(gsub("une", "un", "     Ceci n'est pas une pipe     "))
```

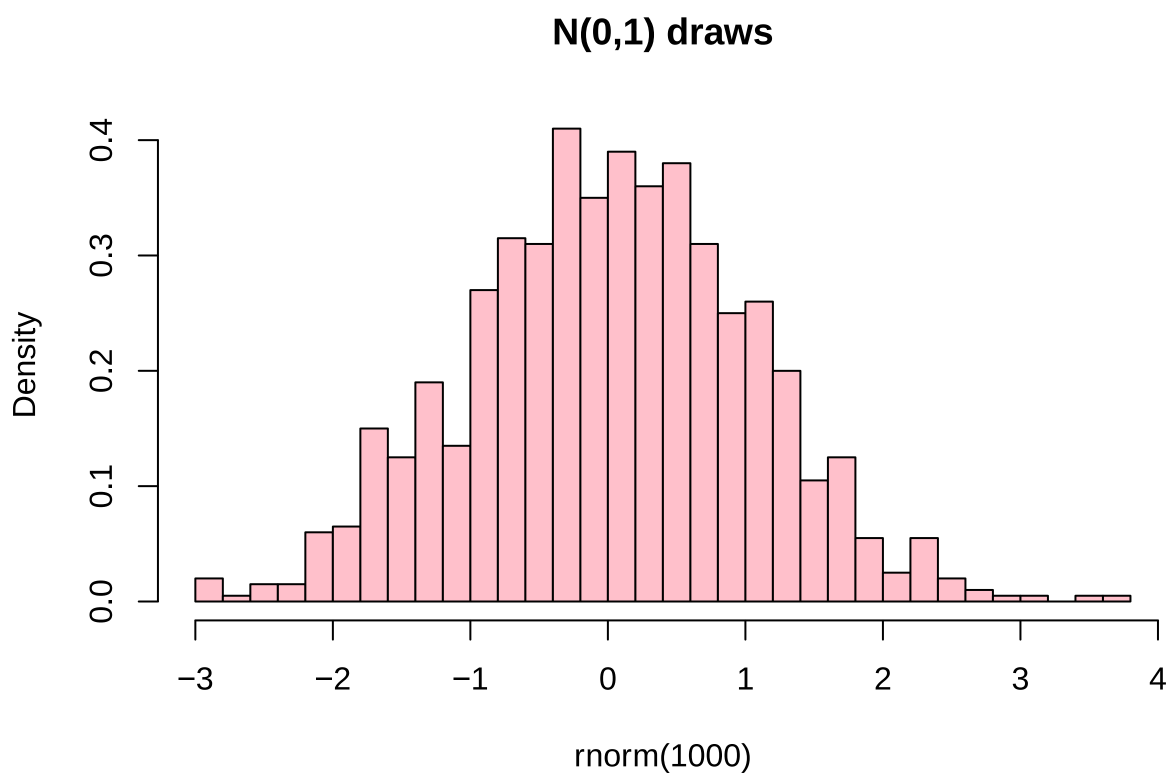
```
## [1] "Ceci n'est pas un pipe"
```

1c.

```r
rnorm(1000) %>%
  hist(breaks=30, main="N(0,1) draws", col="pink", prob=TRUE)
```



```
hist(rnorm(1000), breaks=30, main="N(0,1) draws", col="pink", prob=TRUE)
```



1d.

```
rnorm(1000) %>%
  hist(breaks=30, plot=FALSE) %>%
  `[[`("density") %>%
  max
```

```
## [1] 0.45
```

```
h <- hist(rnorm(1000), breaks=30, plot=FALSE)
max(h$density)
```

```
## [1] 0.425
```

Q2. Base R to pipes

For each of the following code blocks, which are written in base R, write equivalent code with pipes (to do the same thing).

2a. Hint: you'll have to use the dot `.`, as seen above in Q1b, or in the lecture notes.

```r
paste("Your grade is", sample(c("A","B","C","D","R"), size=1))
```

```
## [1] "Your grade is A"
```

```r
sample(c("A", "B", "C", "D", "R"), size=1) %>%
  paste("Your grade is", .)
```

```
## [1] "Your grade is B"
```

2b. Hint: you can use the dot `.` again, in order to index `state.name` directly in the last pipe command.

```r
state.name[which.max(state.x77[,"Illiteracy"])]
```

```
## [1] "Louisiana"
```

```r
state.x77[,"Illiteracy"] %>%
  which.max() %>% state.name[.]
```

```
## [1] "Louisiana"
```

2c. Note: `str.url` is defined for use in this and the next question; you can simply refer to it in your solution code (it is not part of the code you have to convert to pipes).

```r
str.url = "http://www.stat.cmu.edu/~ryantibs/statcomp/data/king.txt"

lines = readLines(str.url)
text = paste(lines, collapse=" ")
words = strsplit(text, split="[[:space:]]|[[:punct:]]")[[1]]
wordtab = table(words)
wordtab = sort(wordtab, decreasing=TRUE)
head(wordtab, 10)
```

```
## words
##        of  the   to  and    a   be will that   is
##  203   98   98   58   40   37   32   25   24   23
```

```r
readLines(str.url)%>%
  paste(collapse=" ") %>%
  strsplit(split="[[:space:]]|[[:punct:]]") %>%
  unlist() %>%
  table() %>%
  sort(decreasing=TRUE) %>%
  head(10)
```

```
## .
##        of  the   to  and    a   be will that   is
##  203   98   98   58   40   37   32   25   24   23
```

2d. Hint: the only difference between this and the last part is the line `words = words[words != ""]`. This is a bit tricky line to do with pipes: use the dot `.`, once more, and manipulate it as if were a variable name.

```
lines = readLines(str.url)
text = paste(lines, collapse=" ")
words = strsplit(text, split="[[:space:]]|[[:punct:]]")[[1]]
words = words[words != ""]
wordtab = table(words)
wordtab = sort(wordtab, decreasing=TRUE)
head(wordtab, 10)
```

```
## words
##   of  the   to  and    a   be will that   is   we
##   98   98   58   40   37   32   25   24   23   21
```

```
readLines(str.url) %>%
  paste(collapse=" ") %>%
  strsplit(split="[[:space:]]|[[:punct:]]") %>%
  unlist() %>%
  .[. != ""] %>%
  table() %>%
  sort(decreasing=TRUE) %>%
  head(10)
```

```
## .
##   of  the   to  and    a   be will that   is   we
##   98   98   58   40   37   32   25   24   23   21
```

Q3. Warming up with map

3a. Using the map functions from the `purrr` package, extract the names of the characters in `got_chars` so that you produce a character vector of length 30. Do this four different ways: (i) using `map()`, defining a custom function on-the-fly, and casting the resulting list into an appropriate data structure; (ii) using one of the `map_***()` functions, but still defining a custom function on-the-fly; (iii) using one of the `map_***()` functions, and using one of `` `[`() `` or `` `[[`() `` functions, as well as an additional argument; (iv) using one of the `map_***()` functions, and passing a string instead of a function (relying on its ability to define an appropriate extractor accordingly).

Store each of the results in a different vector and check that they are all identical.

```
name_vector1 <- map(got_chars, function(x) x$name) %>% unlist()
name_vector2 <- map_chr(got_chars, function(x) x$name)
name_vector3 <- map_chr(got_chars, `[[`, "name")
name_vector4 <- map_chr(got_chars, "name")

identical(name_vector1, name_vector2) &&
identical(name_vector2, name_vector3) &&
identical(name_vector3, name_vector4)
```

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

3b. Produce an integer vector that represents how many allegiances each character holds. Do this with whichever map function you'd like, and print the result to the console. Then use this (and your a saved object from the last question) to answer: which character holds the most allegiances? The least?

```
allegiances_count <- map_int(got_chars, ~length(.x$allegiances))
allegiances_count
```

```
##  [1] 1 1 1 0 1 0 0 1 1 2 1 1 2 2 0 1 3 2 1 1 1 2 1 1 1 0 1 1 1 2
```

```
max_allegiance_char <- got_chars[[which.max(allegiances_count)]]$name
print(paste("The character that holds the most allegiances is", max_allegiance_char))
```

```
## [1] "The character that holds the most allegiances is Brienne of Tarth"
```

```
min_allegiance_char <- got_chars[[which.min(allegiances_count)]]$name
print(paste("The character that holds the least allegiances is", min_allegiance_char))
```

```
## [1] "The character that holds the least allegiances is Will"
```

3c. Run the code below in your console. What does it do?

```
1:5 %in% 3:6
```

It checks if each element of the vector `1:5` (`c(1, 2, 3, 4, 5)`) is present in the vector `3:6` (`c(3, 4, 5, 6)`). The `%in%` operator returns a logical vector indicating whether each element of the left-hand side vector is found in the right-hand side one.

Using the logic you can infer about the `%in%` operator (you can also read its help file), craft a single line of code to compute a Boolean vector of length 6 that checks whether the first Game of Thrones character, stored in `got_chars[[1]]`, has appeared in each of the 6 TV seasons. Print the result to the console.

```
1:6 %in% as.numeric(gsub("Season ", "", got_chars[[1]]$tvSeries))
```

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

3d. Run the two lines of code below in their console. What do they do?

```
rbind(1:5, 6:10, 11:15)
do.call(rbind, list(1:5, 6:10, 11:15))
```

Both lines produce the same result: a 3-row matrix, where each row is one of the numeric sequences provided (`1:5`, `6:10`, `11:15`).

In the first line, `rbind()` stacks the given vectors row-wise. It takes the three vectors and arranges them into a 3-row, 5-column matrix. In the second, `do.call()` dynamically calls the function (`rbind`) on a list of elements. The list (`list(1:5, 6:10, 11:15)`) allows `rbind` to be applied iteratively over the list elements. This produces the same 3-row, 5-column matrix as the first command.

Using the logic you can infer about the `do.call()` function (you can also read its help file), as well as the logic from the last question, complete the following task. Using `map()`, a custom-defined function, as well as some post-processing of its results, produce a matrix that has dimension 30 x 6, with each column representing a TV season, and each row a character. The matrix should have a value of `TRUE` in position (i,j) if character i was in season j, and `FALSE` otherwise. Print the first 6 rows of the result to the console.

```r
season_matrix <- map(got_chars, ~ (1:6) %in% as.numeric(gsub("Season ", "", .x$tvSeries))) %>%
  simplify2array() %>%
  t()
rownames(season_matrix) <- map_chr(got_chars, "name")
colnames(season_matrix) <- paste0("Season", 1:6)

head(season_matrix)
```

```
##                   Season1 Season2 Season3 Season4 Season5 Season6
## Theon Greyjoy        TRUE    TRUE    TRUE    TRUE    TRUE    TRUE
## Tyrion Lannister     TRUE    TRUE    TRUE    TRUE    TRUE    TRUE
## Victarion Greyjoy   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## Will                FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## Areo Hotah          FALSE   FALSE   FALSE   FALSE    TRUE    TRUE
## Chett               FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
```

Challenge. Repeat the same task as in the last question, but using `map_df()` and no post-processing. The result will now be a data frame (not a matrix). Print the first 6 rows of the result to the console. Hint: `map_dfr()` will throw an error if it can't infer column names.

```r
season_df <- map_dfr(got_chars, ~ tibble(
  Name = .x$name,
  Season1 = 1 %in% as.numeric(gsub("Season ", "", .x$tvSeries)),
  Season2 = 2 %in% as.numeric(gsub("Season ", "", .x$tvSeries)),
  Season3 = 3 %in% as.numeric(gsub("Season ", "", .x$tvSeries)),
  Season4 = 4 %in% as.numeric(gsub("Season ", "", .x$tvSeries)),
  Season5 = 5 %in% as.numeric(gsub("Season ", "", .x$tvSeries)),
  Season6 = 6 %in% as.numeric(gsub("Season ", "", .x$tvSeries))
))

head(season_df)
```

```
## # A tibble: 6 x 7
##   Name              Season1 Season2 Season3 Season4 Season5 Season6
##   <chr>             <lgl>   <lgl>   <lgl>   <lgl>   <lgl>   <lgl>
## 1 Theon Greyjoy     TRUE    TRUE    TRUE    TRUE    TRUE    TRUE
## 2 Tyrion Lannister  TRUE    TRUE    TRUE    TRUE    TRUE    TRUE
## 3 Victarion Greyjoy FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## 4 Will              FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## 5 Areo Hotah        FALSE   FALSE   FALSE   FALSE   TRUE    TRUE
## 6 Chett             FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
```

Q4. Cultural studies

4a. Using `map_dfr()`, create a data frame of dimension 30 x 5, whose columns represent, for each Game of Thrones character, their name, birth date, death date, gender, and culture. Store it as `got_df` and print the last 3 rows to the console.

```r
got_df <- map_dfr(got_chars, ~ tibble(
  Name = .x$name,
```

```r
  Born = .x$born,
  Died = .x$died,
  Gender = .x$gender,
  Culture = .x$culture
))

tail(got_df, 3)
```

```
## # A tibble: 3 x 5
## Name Born Died Gender Culture
## <chr> <chr> <chr> <chr> <chr>
## 1 Quentyn Martell In 281 AC, at Sunspear, Dorne "In 300 AC, at M~ Male
Dornish
## 2 Samwell Tarly In 283 AC, at Horn Hill "" Male Andal
## 3 Sansa Stark In 286 AC, at Winterfell "" Female Northm~
```

4b. Using `got_df`, show that you can compute whether each character is alive or not, and compare this to what is stored in `got_chars`, demonstrating that the two ways of checking whether each character is alive lead to equal results.

```r
computed_alive <- got_df$Died == ""
stored_alive <- map_lgl(got_chars, "alive")

identical(computed_alive, stored_alive)
```

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

4c. Using `filter()`, print the subset of the rows of `got_df` that correspond to Ironborn characters. Then print the subset that correspond to female Northmen.

```r
# Ironborn characters
ironborn_chars <- filter(got_df, Culture == "Ironborn")
print(ironborn_chars)
```

```
## # A tibble: 4 x 5
## Name Born Died Gender Culture
## <chr> <chr> <chr> <chr> <chr>
## 1 Theon Greyjoy In 278 AC or 279 AC, at Pyke "" Male Ironbo~
## 2 Victarion Greyjoy In 268 AC or before, at Pyke "" Male Ironbo~
## 3 Asha Greyjoy In 275 AC or 276 AC, at Pyke "" Female Ironbo~
## 4 Aeron Greyjoy In or between 269 AC and 273 AC, at Py~ "" Male Ironbo~
```

```r
# Female Northmen characters
female_northmen <- filter(got_df, Culture == "Northmen", Gender == "Female")
print(female_northmen)
```

```
## # A tibble: 2 x 5
##   Name        Born                         Died  Gender Culture
##   <chr>       <chr>                        <chr> <chr>  <chr>
## 1 Arya Stark  In 289 AC, at Winterfell     ""    Female Northmen
## 2 Sansa Stark In 286 AC, at Winterfell     ""    Female Northmen
```

4d. Create a matrix of dimension (number of cultures) x 2 that counts how many women and men there are in each culture appearing in `got_df`. Print the results to the console. Hint: what happens if you pass `table()` two arguments?

```
gender_culture_matrix <- table(got_df$Culture, got_df$Gender)
print(gender_culture_matrix)
```

```
##
##                 Female Male
##                      1    5
##    Andal             0    1
##    Asshai            1    0
##    Dornish           1    1
##    Free Folk         0    1
##    Ironborn          1    3
##    Northmen          2    3
##    Norvoshi          0    1
##    Reach             0    1
##    Rivermen          1    1
##    Stormlands        0    1
##    Valyrian          1    0
##    Westerlands       0    1
##    Westerman         1    0
##    Westeros          0    2
```

4e. Using `group_by()` and `summarize()` on `got_df`, compute how many characters in each culture have died. Which culture—aside from the unknown category represented by " "—has the most deaths?

```
culture_deaths <- got_df %>%
  mutate(IsDead = Died != "") %>%
  group_by(Culture) %>%
  summarize(Deaths = sum(IsDead)) %>%
  arrange(desc(Deaths))

most_deaths_culture <- culture_deaths %>% filter(Culture != "") %>% slice(1)
print(most_deaths_culture)
```

```
## # A tibble: 1 x 2
##   Culture  Deaths
##   <chr>     <int>
## 1 Rivermen      2
```

Rio Olympics data set

This is a data set from the Rio Olympics data set that we saw in Lab 3. In the next question, we're going to repeat some calculations from Lab 3 but using `dplyr`.

```
rio = read.csv("http://www.stat.cmu.edu/~ryantibs/statcomp/data/rio.csv")
```

Q5. Practice with grouping and summarizing

5a. Using `group_by()` and `summarize()`, compute how many athletes competed for each country in the `rio` data frame? Print the results for the first 10 countries to the console. Building off your here answer, use an additional call to `filter()` to compute which country had the most number of athletes and how many that was. Hint: consider using `n()` from the `dplyr` package for the first part here.

```
athlete_counts <- rio %>%
  group_by(nationality) %>%
  summarize(Athletes = n()) %>%
  arrange(desc(Athletes))

print(head(athlete_counts, 10))
```

```
## # A tibble: 10 x 2
##    nationality Athletes
##    <chr>          <int>
##  1 USA              567
##  2 BRA              485
##  3 GER              441
##  4 AUS              431
##  5 FRA              410
##  6 CHN              404
##  7 GBR              374
##  8 JPN              346
##  9 CAN              321
## 10 ESP              313
```

```
most_athletes <- athlete_counts %>%
  filter(Athletes == max(Athletes))

print(most_athletes)
```

```
## # A tibble: 1 x 2
##   nationality Athletes
##   <chr>          <int>
## 1 USA              567
```

5b. Using `group_by()`, `summarize()`, and `filter()`, compute which country had the most number of total medals and how many that was.

```r
medal_counts <- rio %>%
  group_by(nationality) %>%
  summarize(TotalMedals = sum(gold + silver + bronze, na.rm = TRUE)) %>%
  arrange(desc(TotalMedals))

most_medals <- medal_counts %>%
  filter(TotalMedals == max(TotalMedals))

print(most_medals)
```

```
## # A tibble: 1 x 2
##   nationality TotalMedals
##   <chr>             <int>
## 1 USA                 264
```

5c. Using `group_by()`, `summarize()`, and `filter()`, compute which country—among those with zero total medals—had the most number of athletes. Hint: you will need to modify your `summarize()` command to compute the number of athletes; and you might need two calls to `filter()`.

```r
zero_medal_countries <- rio %>%
  group_by(nationality) %>%
  summarize(Athletes = n(), TotalMedals = sum(gold + silver + bronze, na.rm = TRUE)) %>%
  filter(TotalMedals == 0) %>%
  arrange(desc(Athletes))

most_athletes_no_medals <- zero_medal_countries %>%
  filter(Athletes == max(Athletes))

print(most_athletes_no_medals)
```

```
## # A tibble: 1 x 3
##   nationality Athletes TotalMedals
##   <chr>          <int>       <int>
## 1 CHI               42           0
```

5d. Using —yes, you guessed it— `group_by()`, `summarize()`, and `filter()`, compute the average weight of athletes in each sport, separately for men and women, and report the two sport with the highest average weights (one for each of men and women). Hint: `group_by()` can accept more than one grouping variable. Also, consider using `na.rm=TRUE` as an additional argument to certain arithmetic summary functions so that they will not be thrown off by `NA` or `NaN` values.

```r
average_weights <- rio %>%
  group_by(sport, sex) %>%
  summarize(AverageWeight = mean(weight, na.rm = TRUE), .groups = "drop") %>%
  arrange(desc(AverageWeight))

highest_weight_men <- average_weights %>%
  filter(sex == "male") %>%
  slice(1)

highest_weight_women <- average_weights %>%
  filter(sex == "female") %>%
```

```r
  slice(1)

print(highest_weight_men)
```

```
## # A tibble: 1 x 3
##   sport      sex   AverageWeight
##   <chr>      <chr>         <dbl>
## 1 basketball male           100.
```

```r
print(highest_weight_women)
```

```
## # A tibble: 1 x 3
##   sport      sex    AverageWeight
##   <chr>      <chr>          <dbl>
## 1 basketball female          75.4
```

Prostate cancer data set

Below we read in the prostate cancer data set, as visited in previous labs.

```
pros.df =
  read.table("http://www.stat.cmu.edu/~ryantibs/statcomp/data/pros.dat")
```

Q6. Practice with dplyr verbs

In the following, use pipes and `dplyr` verbs to answer questions on `pros.df`.

6a. Among the men whose `lcp` value is equal to the minimum value (across the entire data set), report the range (min and max) of `lpsa`.

```
pros.df %>%
  filter(lcp == min(lcp, na.rm = TRUE)) %>%
  summarize(MinLPSA = min(lpsa, na.rm = TRUE), MaxLPSA = max(lpsa, na.rm = TRUE))
```

```
##      MinLPSA  MaxLPSA
## 1 -0.4307829 4.129551
```

6b. Order the rows by decreasing `age`, then display the rows from men who are older than 70 and without SVI.

```
pros.df %>%
  arrange(desc(age)) %>%
  filter(age > 70, svi == 0)
```

```
## lcavol lweight age lbph svi lcp gleason pgg45 lpsa
## 78 2.5376572 4.354784 78 2.3263016 0 -1.3862944 7 10 3.4355988
## 72 1.1600209 3.341093 77 1.7491998 0 -1.3862944 7 25 3.0373539
## 3 -0.5108256 2.691243 74 -1.3862944 0 -1.3862944 7 20 -0.1625189
## 37 1.4231083 3.657131 73 -0.5798185 0 1.6582281 8 15 2.1575593
## 61 0.4574248 4.524502 73 2.3263016 0 -1.3862944 6 0 2.8419982
## 63 2.7757089 3.524889 72 -1.3862944 0 1.5581446 9 95 2.8535925
## 68 2.1983351 4.050915 72 2.3075726 0 -0.4307829 7 10 2.9626924
## 70 1.1939225 4.780383 72 2.3263016 0 -0.7985077 7 5 2.9729753
## 77 2.0108950 4.433789 72 2.1222615 0 0.5007753 7 60 3.3928291
## 33 1.2753628 3.037354 71 1.2669476 0 -1.3862944 6 0 2.0082140
```

6c. Order the rows by decreasing `age`, then decreasing `lpsa` score, and display the rows from men who are older than 70 and without SVI, but only the `age`, `lpsa`, `lcavol`, and `lweight` columns. Hint: `arrange()` can take two arguments, and the order you pass in them specifies the priority.

```
pros.df %>%
  arrange(desc(age), desc(lpsa)) %>%
  filter(age > 70, svi == 0) %>%
  select(age, lpsa, lcavol, lweight)
```

```
##    age       lpsa     lcavol  lweight
## 78  78  3.4355988  2.5376572 4.354784
## 72  77  3.0373539  1.1600209 3.341093
## 3   74 -0.1625189 -0.5108256 2.691243
## 61  73  2.8419982  0.4574248 4.524502
## 37  73  2.1575593  1.4231083 3.657131
## 77  72  3.3928291  2.0108950 4.433789
## 70  72  2.9729753  1.1939225 4.780383
## 68  72  2.9626924  2.1983351 4.050915
## 63  72  2.8535925  2.7757089 3.524889
## 33  71  2.0082140  1.2753628 3.037354
```

6d. We're going to resolve Q2c from Lab 3 using the tidyverse. Using `purrr` and `dplyr`, perform t-tests for each variable in the data set, between SVI and non-SVI groups. To be precise, you will perform a t-test for each column excluding the SVI variable itself, by running the function `t.test.by.ind()` below (which is just as in Q2c in Lab 3). Print the returned t-test objects out to the console.

```
t.test.by.ind = function(x, ind) {
  stopifnot(all(ind %in% c(0, 1)))
  return(t.test(x[ind == 0], x[ind == 1]))
}
```

```
t_test_results <- pros.df %>%
  select(-svi) %>%
  map(~ t.test.by.ind(.x, pros.df$svi))

print(t_test_results)
```

```
## $lcavol
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -8.0351, df = 51.172, p-value = 1.251e-10
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.917326 -1.150810
## sample estimates:
## mean of x mean of y
##  1.017892  2.551959
##
##
## $lweight
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -1.8266, df = 42.949, p-value = 0.07472
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.33833495  0.01674335
## sample estimates:
## mean of x mean of y
```

```
##  3.594131  3.754927
##
##
## $age
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -1.1069, df = 30.212, p-value = 0.2771
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -6.018547  1.786718
## sample estimates:
## mean of x mean of y
##  63.40789  65.52381
##
##
## $lbph
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = 0.88281, df = 34.337, p-value = 0.3835
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3914341  0.9930934
## sample estimates:
##  mean of x  mean of y
##  0.1654837 -0.1353460
##
##
## $lcp
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -8.8355, df = 31.754, p-value = 4.58e-10
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.797674 -1.749133
## sample estimates:
##  mean of x  mean of y
## -0.6715458  1.6018579
##
##
## $gleason
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -3.6194, df = 36.843, p-value = 0.0008816
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.8718223 -0.2459721
```

```
## sample estimates:
## mean of x mean of y
##  6.631579  7.190476
##
##
## $pgg45
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -4.9418, df = 31.288, p-value = 2.482e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -44.04052 -18.31537
## sample estimates:
## mean of x mean of y
##  17.63158  48.80952
##
##
## $lpsa
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -6.8578, df = 33.027, p-value = 7.879e-08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.047129 -1.110409
## sample estimates:
## mean of x mean of y
##  2.136592  3.715360
```

6e. Extend your code from the last part (append just one more line of code, glued together by a pipe) to extract the p-values from each of the returned t-test objects, and print them out to the console.

```r
t_test_pvalues <- pros.df %>%
  select(-svi) %>%
  map(~ t.test.by.ind(.x, pros.df$svi)) %>%
  map_dbl(~ .x$p.value)

print(t_test_pvalues)
```

```
## lcavol lweight age lbph lcp gleason
## 1.251040e-10 7.472088e-02 2.770533e-01 3.834772e-01 4.579752e-10
8.816293e-04
## pgg45 lpsa
## 2.482255e-05 7.879066e-08
```

Fastest 100m sprint times

Below, we read two data sets of the 1000 fastest times ever recorded for the 100m sprint, in men's and women's track. We scraped this data from http://www.alltime-athletics.com/m_100ok.htm and http://www.alltime-athletics.com/w_100ok.htm, in early September 2021. (Interestingly, the 2nd, 3rd, 4th, 7th, and 8th fastest women's times were all set at the most recent Tokyo Olympics, or after! Meanwhile, the top 10 men's times are all from about a decade ago.)

```
sprint.m.df = read.table(
  file="http://www.stat.cmu.edu/~ryantibs/statcomp/data/sprint.m.txt",
  sep="\t", quote="", header=TRUE)
sprint.w.df = read.table(
  file="http://www.stat.cmu.edu/~ryantibs/statcomp/data/sprint.w.txt",
  sep="\t", quote="", header=TRUE)
```

Q7. More practice with data frame computations

7a. Confirm that both `sprint.m.df` and `sprint.w.df` are data frames. Delete the `Rank` column from each data frame, then display the first and last 3 rows of each.

```
is.data.frame(sprint.m.df)
```

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

```
is.data.frame(sprint.w.df)
```

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

```
sprint.m.df <- sprint.m.df %>%
  select(-Rank)
sprint.w.df <- sprint.w.df %>%
  select(-Rank)

head(sprint.m.df, 3)
```

```
##   Time Wind       Name Country Birthdate    City       Date
## 1 9.58  0.9 Usain Bolt     JAM  21.08.86  Berlin 16.08.2009
## 2 9.63  1.5 Usain Bolt     JAM  21.08.86  London 05.08.2012
## 3 9.69  0.0 Usain Bolt     JAM  21.08.86 Beijing 16.08.2008
```

```
tail(sprint.m.df, 3)
```

```
##      Time Wind           Name Country Birthdate      City       Date
## 998  9.99  2.0 Travis Padgett     USA  13.12.86 Clermont 21.05.2011
## 999  9.99  1.3   Mike Rodgers     USA  24.04.85    Eugene 24.06.2011
## 1000 9.99  1.0   Nesta Carter     JAM  11.10.85 Lausanne 30.06.2011
```

```r
head(sprint.w.df, 3)
```

```
## Time Wind Name Country Birthdate City Date
## 1 10.49 0.0 Florence Griffith-Joyner USA 21.12.59 Indianapolis 16.07.1988
## 2 10.54 0.9 Elaine Thompson-Herah JAM 28.06.92 Eugene 21.08.2021
## 3 10.60 1.7 Shelly-Ann Fraser-Pryce JAM 27.12.86 Lausanne 26.08.2021
```

```r
tail(sprint.w.df, 3)
```

```
## Time Wind Name Country Birthdate City Date
## 998 10.99 1.5 Shania Collins USA 14.11.96 Knoxville 13.05.2018
## 999 10.99 1.8 Dezerea Bryant USA 27.04.93 Des Moines 21.06.2018
## 1000 10.99 1.2 Wei Yongli CHN 11.10.91 La Chaux-de-Fonds 01.07.2018
```

7b. Recompute the ranks for the men's data set from the `Time` column and add them back as a `Rank` column to `sprint.m.df`. Do the same for the women's data set. After adding back the rank columns, print out the first 10 rows of each data frame, but only the `Time`, `Name`, `Date`, and `Rank` columns. Hint: consider using `rank()`.

```r
sprint.m.df <- sprint.m.df %>%
  mutate(Rank = rank(Time, ties.method = "min"))

sprint.w.df <- sprint.w.df %>%
  mutate(Rank = rank(Time, ties.method = "min"))

sprint.m.df %>% select(Time, Name, Date, Rank) %>% head(10)
```

```
##    Time          Name       Date Rank
## 1  9.58    Usain Bolt 16.08.2009    1
## 2  9.63    Usain Bolt 05.08.2012    2
## 3  9.69    Usain Bolt 16.08.2008    3
## 4  9.69     Tyson Gay 20.09.2009    3
## 5  9.69   Yohan Blake 23.08.2012    3
## 6  9.71     Tyson Gay 16.08.2009    6
## 7  9.72    Usain Bolt 31.05.2008    7
## 8  9.72  Asafa Powell 02.09.2008    7
## 9  9.74  Asafa Powell 09.09.2007    9
## 10 9.74 Justin Gatlin 15.05.2015    9
```

```r
sprint.w.df %>% select(Time, Name, Date, Rank) %>% head(10)
```

```
##      Time                     Name       Date Rank
## 1   10.49 Florence Griffith-Joyner 16.07.1988    1
## 2   10.54    Elaine Thompson-Herah 21.08.2021    2
## 3   10.60  Shelly-Ann Fraser-Pryce 26.08.2021    3
## 4   10.61 Florence Griffith-Joyner 17.07.1988    4
## 5   10.61    Elaine Thompson-Herah 31.07.2021    4
## 6   10.62 Florence Griffith-Joyner 24.09.1988    6
## 7   10.63  Shelly-Ann Fraser-Pryce 05.06.2021    7
## 8   10.64          Carmelita Jeter 20.09.2009    8
## 9   10.64    Elaine Thompson-Herah 26.08.2021    8
## 10 10.65A             Marion Jones 12.09.1998   10
```

7c. Using base R functions, compute, for each country, the number of sprint times from this country that appear in the men's data set. Call the result `sprint.m.counts`. Do the same for the women's data set, and call the result `sprint.w.counts`. What are the 5 most represented countries, for the men, and for the women? (Interesting side note: go look up the population of Jamaica, compared to that of the US. Pretty impressive, eh?)

```r
sprint.m.counts <- table(sprint.m.df$Country)
sprint.w.counts <- table(sprint.w.df$Country)

sort(sprint.m.counts, decreasing = TRUE)[1:5]
```

```
##
## USA JAM TTO CAN GBR
## 432 258  50  38  34
```

```r
sort(sprint.w.counts, decreasing = TRUE)[1:5]
```

```
##
## USA JAM CIV GDR TTO
## 357 291  46  39  33
```

7d. Repeat the same calculations as in last part but using `dplyr` functions, and print out again the 5 most represented countries for men and women. (No need to save new variables.) Hint: consider using `arrange()` from the `dplyr` library.

```r
sprint.m.df %>%
  count(Country, sort = TRUE) %>%
  head(5)
```

```
##   Country   n
## 1     USA 432
## 2     JAM 258
## 3     TTO  50
## 4     CAN  38
## 5     GBR  34
```

```r
sprint.w.df %>%
  count(Country, sort = TRUE) %>%
  head(5)
```

```
##   Country   n
## 1     USA 357
## 2     JAM 291
## 3     CIV  46
## 4     GDR  39
## 5     TTO  33
```

7e. Are there any countries that are represented by women but not by men, and if so, what are they? Viceversa, represented by men and not women? Hint: consider using the `%in%` operator.

```r
men_countries <- unique(sprint.m.df$Country)
women_countries <- unique(sprint.w.df$Country)

# Countries in women but not men
setdiff(women_countries, men_countries)
```

```
##  [1] "RUS" "BUL" "GDR" "UKR" "GRE" "GER" "SUI" "BRA" "BLR" "POL" "CMR" "ECU"
```

```r
# Countries in men but not women
setdiff(men_countries, women_countries)
```

```
##  [1] "ITA" "NAM" "POR" "KEN" "BAR" "ZIM" "ANT" "QAT" "TUR" "AUS" "AHO" "SKN"
## [13] "CAY" "JPN" "OMA" "GHA" "CUB"
```

Q8. More practice with `dplyr` functions

8a. Using `dplyr` functions, compute, for each country, the fastest time among athletes who come from that country. Do this for each of the men's and women's data sets, and display the first 10 rows of the result.

```r
sprint.m.df %>%
  group_by(Country) %>%
  summarize(FastestTime = min(Time, na.rm = TRUE)) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    Country FastestTime
##    <chr>   <chr>
##  1 AHO     9.93
##  2 ANT     9.91
##  3 AUS     9.93
##  4 BAH     9.91
##  5 BAR     9.87A
##  6 CAN     9.84
##  7 CAY     9.95
##  8 CHN     9.83
##  9 CIV     9.93
## 10 CUB     9.98
```

```r
sprint.w.df %>%
  group_by(Country) %>%
  summarize(FastestTime = min(Time, na.rm = TRUE)) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    Country FastestTime
##    <chr>   <chr>
##  1 BAH     10.84
##  2 BLR     10.92
##  3 BRA     10.91
##  4 BUL     10.77
```

```
##  5 CAN     10.98
##  6 CHN     10.79
##  7 CIV     10.78
##  8 CMR     10.98
##  9 ECU     10.99
## 10 FRA     10.73
```

8b. With the most minor modification to your code possible, do the same computations as in the last part, but now display the first 10 results ordered by increasing time. Hint: recall `arrange()`.

```
sprint.m.df %>%
  group_by(Country) %>%
  summarize(FastestTime = min(Time, na.rm = TRUE)) %>%
  arrange(FastestTime) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    Country FastestTime
##    <chr>   <chr>
##  1 JAM     9.58
##  2 USA     9.69
##  3 ITA     9.80
##  4 TTO     9.82
##  5 CHN     9.83
##  6 CAN     9.84
##  7 RSA     9.84
##  8 NGR     9.85
##  9 FRA     9.86
## 10 KEN     9.86
```

```
sprint.w.df %>%
  group_by(Country) %>%
  summarize(FastestTime = min(Time, na.rm = TRUE)) %>%
  arrange(FastestTime) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    Country FastestTime
##    <chr>   <chr>
##  1 USA     10.49
##  2 JAM     10.54
##  3 FRA     10.73
##  4 BUL     10.77
##  5 RUS     10.77
##  6 CIV     10.78
##  7 CHN     10.79
##  8 NGR     10.79
##  9 GDR     10.81
## 10 NED     10.81
```

8c. Rewrite your solution in the last part using base R. Hint: `tapply()` gives probably the easiest route here. Note: your code here shouldn't be too much more complicated than your code in the last part.

```r
fastest_men <- tapply(sprint.m.df$Time, sprint.m.df$Country, min, na.rm = TRUE)
fastest_women <- tapply(sprint.w.df$Time, sprint.w.df$Country, min, na.rm = TRUE)

head(sort(fastest_men), 10)
```

```
##    JAM    USA    ITA    TTO    CHN    CAN    RSA    NGR    FRA    KEN
## "9.58" "9.69" "9.80" "9.82" "9.83" "9.84" "9.84" "9.85" "9.86" "9.86"
```

```r
head(sort(fastest_women), 10)
```

```
## USA JAM FRA BUL RUS CIV CHN NGR GDR NED
## "10.49" "10.54" "10.73" "10.77" "10.77" "10.78" "10.79" "10.79" "10.81"
"10.81"
```

8d. Using `dplyr` functions, compute, for each country, the quadruple: name, city, country, and time, corresponding to the athlete with the fastest time among athletes from that country. Do this for each of the men's and women's data sets, and display the first 10 rows of the result, ordered by increasing time. If there are ties, then show all the results that correspond to the fastest time. Hint: consider using `select()` from the `dplyr` library.

```r
sprint.m.df %>%
  group_by(Country) %>%
  filter(Time == min(Time, na.rm = TRUE)) %>%
  select(Name, City, Country, Time) %>%
  arrange(Time) %>%
  head(10)
```

```
## # A tibble: 10 x 4
## # Groups: Country [9]
## Name City Country Time
## <chr> <chr> <chr> <chr>
## 1 Usain Bolt Berlin JAM 9.58
## 2 Tyson Gay Shanghai USA 9.69
## 3 Lamont Marcell Jacobs Tokyo ITA 9.80
## 4 Richard Thompson Port of Spain TTO 9.82
## 5 Su Bingtian Tokyo CHN 9.83
## 6 Donovan Bailey Atlanta CAN 9.84
## 7 Bruny Surin Sevilla CAN 9.84
## 8 Akani Simbine Sz&eacute;kesfeh&eacute;rv&aacute;r RSA 9.84
## 9 Olusoji Adekotunbo Fasuba Ad-Dawhah NGR 9.85
## 10 Frank Fredericks Lausanne NAM 9.86
```

```r
sprint.w.df %>%
  group_by(Country) %>%
  filter(Time == min(Time, na.rm = TRUE)) %>%
  select(Name, City, Country, Time) %>%
  arrange(Time) %>%
  head(10)
```

```
## # A tibble: 10 x 4
## # Groups:   Country [9]
```

```
##    Name                       City         Country Time
##    <chr>                      <chr>        <chr>   <chr>
##  1 Florence Griffith-Joyner   Indianapolis USA     10.49
##  2 Elaine Thompson-Herah      Eugene       JAM     10.54
##  3 Christine Arron            Budapest     FRA     10.73
##  4 Irina Privalova            Lausanne     RUS     10.77
##  5 Ivet Lalova-Collio         Plovdiv      BUL     10.77
##  6 Murielle Ahour&eacute;     Montverde    CIV     10.78
##  7 Marie-Jos&eacute;e Ta Lou Tokyo        CIV     10.78
##  8 Li Xuemei                  Shanghai     CHN     10.79
##  9 Blessing Okagbare          London       NGR     10.79
## 10 Marlies G&ouml;hr          Berlin       GDR     10.81
```

8e. Rewrite your solution in the last part using base R. Hint: there are various routes to go; one strategy is to use `split()`, followed by `lapply()` with a custom function call, and then `rbind()` to get things in a data frame form. Note: your code here will probably be more complicated, or at least less intuitive, than your code in the last part.

```r
fastest_per_country <- function(df) {
  split_df <- split(df, df$Country)

  result <- do.call(rbind, lapply(split_df, function(sub_df) {
    min_time <- min(sub_df$Time, na.rm = TRUE)
    sub_df[sub_df$Time == min_time, c("Name", "City", "Country", "Time")]
  }))

  result[order(result$Time), ]
}

fastest_men <- fastest_per_country(sprint.m.df)
fastest_women <- fastest_per_country(sprint.w.df)

head(fastest_men, 10)
```

```
## Name City Country
## JAM Usain Bolt Berlin JAM
## USA Tyson Gay Shanghai USA
## ITA Lamont Marcell Jacobs Tokyo ITA
## TTO Richard Thompson Port of Spain TTO
## CHN Su Bingtian Tokyo CHN
## CAN.78 Donovan Bailey Atlanta CAN
## CAN.79 Bruny Surin Sevilla CAN
## RSA Akani Simbine Sz&eacute;kesfeh&eacute;rv&aacute;r RSA
## NGR Olusoji Adekotunbo Fasuba Ad-Dawhah NGR
## FRA.141 Jimmy Vicaut Saint-Denis FRA
## Time
## JAM 9.58
## USA 9.69
## ITA 9.80
## TTO 9.82
## CHN 9.83
## CAN.78 9.84
## CAN.79 9.84
```

```
## RSA 9.84
## NGR 9.85
## FRA.141 9.86
```

```r
head(fastest_women, 10)
```

```
##                              Name         City Country  Time
## USA     Florence Griffith-Joyner Indianapolis     USA 10.49
## JAM        Elaine Thompson-Herah       Eugene     JAM 10.54
## FRA              Christine Arron     Budapest     FRA 10.73
## BUL           Ivet Lalova-Collio      Plovdiv     BUL 10.77
## RUS              Irina Privalova     Lausanne     RUS 10.77
## CIV.72    Murielle Ahour&eacute;    Montverde     CIV 10.78
## CIV.78 Marie-Jos&eacute;e Ta Lou        Tokyo     CIV 10.78
## CHN                    Li Xuemei     Shanghai     CHN 10.79
## NGR              Blessing Okagbare       London     NGR 10.79
## GDR              Marlies G&ouml;hr       Berlin     GDR 10.81
```

8f. Order the rows by increasing `Wind` value, and then display only the women who ran at most 10.7 seconds.

```r
sprint.w.df %>%
  arrange(Wind) %>%
  filter(Time <= 10.7)
```

```
## Time Wind Name Country Birthdate City
## 1 10.61 -0.6 Elaine Thompson-Herah JAM 28.06.92 Tokyo
## 2 10.67 -0.1 Carmelita Jeter USA 24.11.79 Thessalon&iacute;ki
## 3 10.49 0.0 Florence Griffith-Joyner USA 21.12.59 Indianapolis
## 4 10.54 0.9 Elaine Thompson-Herah JAM 28.06.92 Eugene
## 5 10.62 1.0 Florence Griffith-Joyner USA 21.12.59 Seoul
## 6 10.65A 1.1 Marion Jones USA 12.10.75 Johannesburg
## 7 10.61 1.2 Florence Griffith-Joyner USA 21.12.59 Indianapolis
## 8 10.64 1.2 Carmelita Jeter USA 24.11.79 Shanghai
## 9 10.63 1.3 Shelly-Ann Fraser-Pryce JAM 27.12.86 Kingston
## 10 10.60 1.7 Shelly-Ann Fraser-Pryce JAM 27.12.86 Lausanne
## 11 10.64 1.7 Elaine Thompson-Herah JAM 28.06.92 Lausanne
## Date Rank
## 1 31.07.2021 4
## 2 13.09.2009 11
## 3 16.07.1988 1
## 4 21.08.2021 2
## 5 24.09.1988 6
## 6 12.09.1998 10
## 7 17.07.1988 4
## 8 20.09.2009 8
## 9 05.06.2021 7
## 10 26.08.2021 3
## 11 26.08.2021 8
```

8g. Order the rows by terms of increasing `Time`, then increasing `Wind`, and again display only the women who ran at most 10.7 seconds, but only the `Time`, `Wind`, `Name`, and `Date` columns.

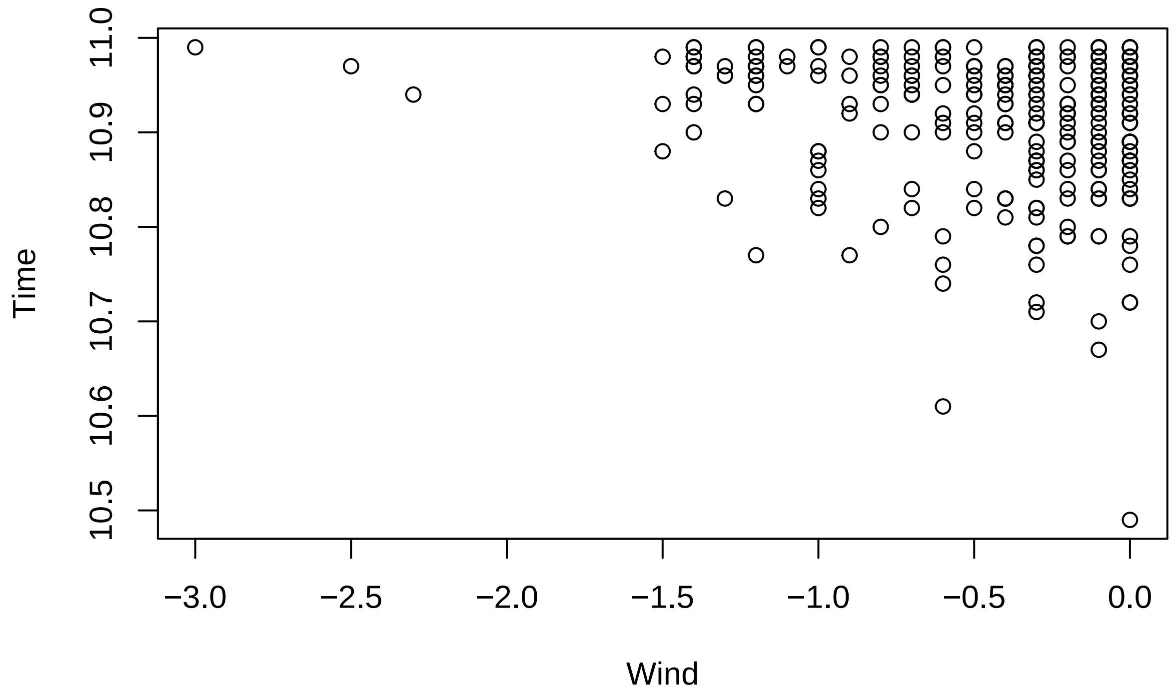
```r
sprint.w.df %>%
  arrange(Time, Wind) %>%
  filter(Time <= 10.7) %>%
  select(Time, Wind, Name, Date)
```

```
##      Time Wind                     Name       Date
## 1   10.49  0.0 Florence Griffith-Joyner 16.07.1988
## 2   10.54  0.9    Elaine Thompson-Herah 21.08.2021
## 3   10.60  1.7  Shelly-Ann Fraser-Pryce 26.08.2021
## 4   10.61 -0.6    Elaine Thompson-Herah 31.07.2021
## 5   10.61  1.2 Florence Griffith-Joyner 17.07.1988
## 6   10.62  1.0 Florence Griffith-Joyner 24.09.1988
## 7   10.63  1.3  Shelly-Ann Fraser-Pryce 05.06.2021
## 8   10.64  1.2          Carmelita Jeter 20.09.2009
## 9   10.64  1.7    Elaine Thompson-Herah 26.08.2021
## 10 10.65A  1.1             Marion Jones 12.09.1998
## 11  10.67 -0.1          Carmelita Jeter 13.09.2009
```

8h. Plot the `Time` versus `Wind` columns, but only using data where `Wind` values that are nonpositive. Hint: note that for a data frame, `df` with columns `colX` and `colY`, you can use `plot(colY ~ colX, data=df)`, to plot `df$colY` (y-axis) versus `df$colX` (x-axis).

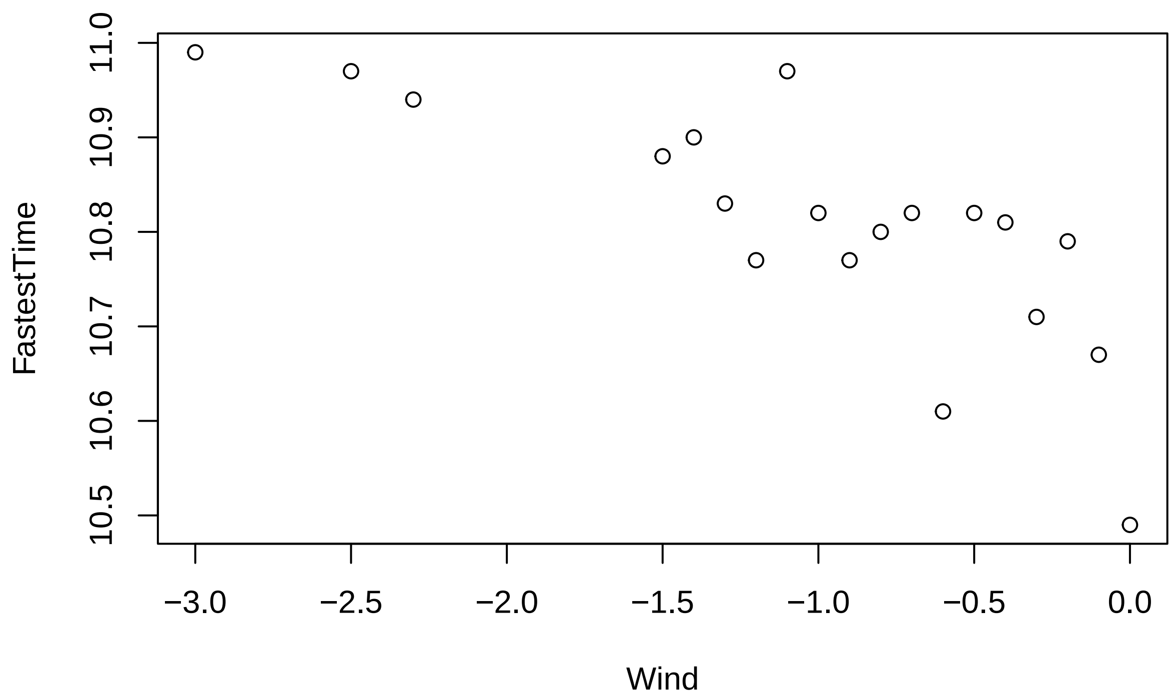
```r
sprint.w.df %>%
  filter(Wind <= 0) %>%
  plot(Time ~ Wind, data = .)
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): NAs introduced by coercion
```



8i. Extend your code from the last part (append just two more lines of code, glued together by a pipe) to plot the single fastest `Time` per `Wind` value. (That is, your plot should be as in the last part, but among points that share the same x value, only the point with the lowest y value should be drawn.)

```
sprint.w.df %>%
  filter(Wind <= 0) %>%
  group_by(Wind) %>%
  summarize(FastestTime = min(Time)) %>%
  plot(FastestTime ~ Wind, data = .)
```



Q9. Practice pivoting wider and longer

In the following, use pipes and `dplyr` and `tidyr` verbs to answer questions on `sprint.m.df`. In some parts, it might make more sense to use direct indexing, and that's perfectly fine.

9a. Confirm that the `Time` column is stored as character data type. Why do you think this is? Convert the `Time` column to numeric. Hint: after converting to numeric, there will be `NA` values; look at the position of one such `NA` value and revisit the original `Time` column to see why it was stored as character type in the first place.

```
class(sprint.m.df$Time)
```

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

```
sprint.m.df.new <- sprint.m.df %>%
  mutate(Time = as.numeric(Time))
```

```
## Warning: There was 1 warning in ‘mutate()‘.
## i In argument: ‘Time = as.numeric(Time)‘.
## Caused by warning:
## ! NAs introduced by coercion
```

```r
class(sprint.m.df.new$Time)
```

```
## [1] "numeric"
```

```r
which(is.na(sprint.m.df.new$Time))
```

```
## [1] 155 312 327 352 355 366 418 453 495 509 514 516 574 575 592 601 663 703
813
## [20] 872 886 891 961 967
```

The Time column is stored as character because some values contain non-numeric characters, such as:

- "10.54w" (wind-assisted time, "w")
- "10.67A" (altitude-assisted time, "A")

9b. Define a reduced data frame `dat.reduced` as follows. For each athlete, and each city, keep the fastest of all times they recorded in this city. Then drop all rows with an `NA` value in the `Time` column Your new data frame `dat.reduced` should have 600 rows and 3 columns (`Name`, `City`, `Time`). Confirm that it has these dimensions, and display its first 10 rows. Hint: `drop_na()` in the `tidyr` package allows you to drop rows based on `NA` values.

```r
dat.reduced <- sprint.m.df.new %>%
  drop_na(Time) %>%
  group_by(Name, City) %>%
  summarize(Time = min(Time, na.rm = TRUE), .groups = "drop") %>%
  ungroup() %>%
  select(Name, City, Time)

dim(dat.reduced) # 600 x 3
```

```
## [1] 600   3
```

```r
head(dat.reduced, 10)
```

```
## # A tibble: 10 x 3
##    Name                  City              Time
##    <chr>                 <chr>            <dbl>
##  1 Aaron Brown           Montr&eacute;al  9.96
##  2 Aaron Brown           Montverde         9.96
##  3 Abdul Aziz Zakari     Ath&iacute;nai    9.99
##  4 Abdul Aziz Zakari     Rieti             9.99
##  5 Abdul Hakim Sani Brown Austin           9.97
##  6 Adam Gemili           Birmingham        9.97
##  7 Akani Simbine         Ad-Dawhah         9.93
##  8 Akani Simbine         Gwangju           9.97
##  9 Akani Simbine         London            9.93
## 10 Akani Simbine         Madrid            9.98
```

9c. The data frame `dat.reduced` is said to be in "long" format: it has observations on the rows, and variables (`Name`, `City`, `Time`) on the columns. Arrange the rows alphabetically by city; convert this data frame into "wide" format; and then order the rows so that they are alphabetical by sprinter name. Call the result `dat.wide`. To be clear, here the first column should be the athlete names, and the remaining columns should correspond to the cities. Confirm that your data frame has dimension 141 x 152. Do these dimensions make sense to you?

```
dat.wide <- dat.reduced %>%
  pivot_wider(names_from = City, values_from = Time) %>%
  arrange(Name)

dim(dat.wide)  # 141 x 152
```

```
## [1] 141 152
```

The dimensions make sense as each row represents a unique sprinter, the first column is the sprinter's name, and the other 151 columns correspond to different cities where the times were recorded.

9d. Not counting the names in the first column, how many non-`NA` values does `dat.wide` have? How could you have guessed this number ahead of time, directly from `dat.reduced` (before any pivoting at all)?

```
non_na_count <- sum(!is.na(dat.wide[,-1]))
print(non_na_count)
```

```
## [1] 600
```

```
nrow(dat.reduced)
```

```
## [1] 600
```

The number of non-NA values directly could have been predicted directly from `dat.reduced` by simply counting its rows. Each row in `dat.reduced` represents a valid (Name, City, Time) combination. Since `dat.wide` is just `dat.reduced` pivoted, the total number of non-NA values remains the same. As a result, the number of non-NA values in `dat.wide` should be equal to the number of rows in `dat.reduced`, which is 600.

9e. From `dat.wide`, look at the row for "Usain Bolt", and determine the city names that do not have `NA` values. These should be the cities in which he raced. Determine these cities directly from `dat.reduced`, and confirm that they match.

```
usain_bolt_row <- dat.wide %>%
  filter(Name == "Usain Bolt")

usain_bolt_cities_wide <- names(usain_bolt_row)[which(!is.na(usain_bolt_row))][-1]

usain_bolt_cities_long <- dat.reduced %>%
  filter(Name == "Usain Bolt") %>%
  pull(City)

identical(sort(usain_bolt_cities_wide), sort(usain_bolt_cities_long))
```

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

```r
print(usain_bolt_cities_wide)
```

```
##  [1] "London"         "Monaco"         "Rio de Janeiro" "Roma"
##  [5] "Beijing"        "Berlin"         "Bruxelles"      "Kingston"
##  [9] "Lausanne"       "Oslo"           "Ostrava"        "Port of Spain"
## [13] "Saint-Denis"    "Stockholm"      "Z&uuml;rich"    "Zagreb"
## [17] "New York City"  "Moskva"         "Daegu"          "Warszawa"
```

9f. Convert `dat.wide` back into "long" format, and call the result `dat.long`. Remove rows that have `NA` values (hint: you can do this by setting `values_drop_na = TRUE` in the call to the pivoting function), and order the rows alphabetically by athlete and city name. Once you've done this, `dat.long` should have matching entries to `dat.reduced`; confirm that this is the case.

```r
dat.long <- dat.wide %>%
  pivot_longer(cols = -Name,
               names_to = "City",
               values_to = "Time",
               values_drop_na = TRUE) %>%
  arrange(Name, City)

dim(dat.long)
```

```
## [1] 600   3
```