Lab 11: Tidyverse II: Tidyr and Advanced Dplyr

Name: Rufus Petrie

This week's agenda: practicing grouping, pivoting wider and longer, and joins.

```
# Load the tidyverse!
library(tidyverse)
assertthat::assert_that(utils::packageVersion("tidyr") > "0.8.99")
```

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

Practice with grouping

Below we read in a data frame `sprint.m.df` containing the top men's times in the 100m sprint, as seen in previous labs. In the following, unless stated otherwise, use pipes and `dplyr` verbs to solve each part as cleanly/succintly as you can.

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

- **1a.** Compute, for each country, the fastest time among athletes who come from that country, and display the first 10 results, ordered alphabetically by country. Also compute, for each city, the fastest time among athletes who ran in that city, and display the first 10 results, ordered alphabetically by city. Hint: `group_by()`, `summarise()`.

```
sprint.m.df %>%
  group_by(Country) %>%
  summarize(Time = min(Time)) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    Country  Time
##    <fct>   <dbl>
##  1 AHO      9.93
##  2 ANT      9.91
##  3 AUS      9.93
##  4 AZE     10.1
##  5 BAH      9.91
##  6 BAR      9.87
##  7 BRA     10.0
##  8 CAN      9.84
##  9 CAY      9.95
## 10 CHN      9.99
```

```
sprint.m.df %>%
  group_by(City) %>%
  summarize(Time = min(Time)) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    City                  Time
##    <fct>                <dbl>
##  1 &Eacute;vry-Bondoufle 10.1
##  2 Abbotsford            10.0
##  3 Abilene               10.1
##  4 Abuja                  9.95
##  5 Ad-Dawhah              9.74
##  6 Air Force Academy      9.93
##  7 Aix-les-Bains          9.95
##  8 Albi                   9.92
##  9 Albuquerque           10.1
## 10 Almaty                10.1
```

- **1b.** 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: `arrange()`.

```
sprint.m.df %>%
  group_by(Country) %>%
  summarize(Time = min(Time)) %>%
  arrange(Time) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    Country  Time
##    <fct>   <dbl>
##  1 JAM      9.58
##  2 USA      9.69
##  3 TTO      9.82
##  4 CAN      9.84
##  5 NGR      9.85
##  6 FRA      9.86
##  7 NAM      9.86
##  8 POR      9.86
##  9 BAR      9.87
## 10 GBR      9.87
```

```
sprint.m.df %>%
  group_by(Country) %>%
  summarize(Time = min(Time)) %>%
  arrange(Time) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    Country  Time
##    <fct>   <dbl>
##  1 JAM      9.58
##  2 USA      9.69
##  3 TTO      9.82
##  4 CAN      9.84
##  5 NGR      9.85
##  6 FRA      9.86
##  7 NAM      9.86
##  8 POR      9.86
##  9 BAR      9.87
## 10 GBR      9.87
```

- **1c.** 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.

```
x1 <- as.data.frame(sort(tapply(sprint.m.df$Time, sprint.m.df$Country, FUN = min),
                         decreasing = FALSE)[1:10])
colnames(x1) <- "Time"
x2 <- as.data.frame(sort(tapply(sprint.m.df$Time, sprint.m.df$City, FUN = min),
                         decreasing = FALSE)[1:10])
colnames(x2) <- "Time"
x1
```

```
##     Time
## JAM 9.58
## USA 9.69
## TTO 9.82
## CAN 9.84
## NGR 9.85
## FRA 9.86
## NAM 9.86
## POR 9.86
## BAR 9.87
## GBR 9.87
```

```
x2
```

```
##               Time
## Berlin        9.58
## London        9.63
## Beijing       9.69
## Lausanne      9.69
## Shanghai      9.69
## New York City 9.72
## Ad-Dawhah     9.74
## Rieti         9.74
## Kingston      9.75
## Roma          9.75
```

- **1d.** Compute, for each country, the quadruple: (Name, City, Country, Time) corresponding to the athlete with the fastest time among athletes from that country. Display the first 10 results, ordered by increasing time. If there are ties, then show all the results that correspond to the fastest time. Repeat the same computation, but for the fastest time per city. Hint: `group_by()`, `filter()`, `select()`.

```
sprint.m.df %>%
  group_by(Country) %>%
  filter(Time == min(Time)) %>%
  select(Name, City, Country, Time) %>%
  arrange(Time)
```

```
## # A tibble: 51 x 4
## # Groups:   Country [46]
##    Name                         City          Country  Time
##    <fct>                        <fct>         <fct>   <dbl>
##  1 Usain Bolt                   Berlin        JAM      9.58
##  2 Tyson Gay                    Shanghai      USA      9.69
##  3 Richard Thompson             Port of Spain TTO      9.82
##  4 Donovan Bailey               Atlanta       CAN      9.84
##  5 Bruny Surin               Sevilla        CAN      9.84
##  6 Adekotunbo Olusoji Fasuba Ad-Dawhah      NGR      9.85
##  7 Frank Fredericks          Lausanne       NAM      9.86
##  8 Francis Obikwelu          Ath&iacute;nai POR      9.86
##  9 Jimmy Vicaut              Saint-Denis    FRA      9.86
## 10 Jimmy Vicaut              Montreuil      FRA      9.86
## # ... with 41 more rows
```

```
sprint.m.df %>%
  group_by(City) %>%
  filter(Time == min(Time)) %>%
  select(Name, City, Country, Time) %>%
  arrange(Time)
```

```
## # A tibble: 331 x 4
## # Groups:   City [320]
##    Name          City          Country  Time
##    <fct>         <fct>         <fct>   <dbl>
##  1 Usain Bolt    Berlin        JAM      9.58
##  2 Usain Bolt    London        JAM      9.63
##  3 Usain Bolt    Beijing       JAM      9.69
##  4 Tyson Gay     Shanghai      USA      9.69
##  5 Yohan Blake   Lausanne      JAM      9.69
##  6 Usain Bolt    New York City JAM      9.72
##  7 Asafa Powell  Rieti         JAM      9.74
##  8 Justin Gatlin Ad-Dawhah     USA      9.74
##  9 Yohan Blake   Kingston      JAM      9.75
## 10 Justin Gatlin Roma          USA      9.75
## # ... with 321 more rows
```

- **1e.** Rewrite the rest of your solution in the last part using base R. You should end up with two data frames (per country, and per city) with the exact same structure as in the last part, and display the top 10 rows of each, ordered by increasing time. 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.

```
c1 <- lapply(split(sprint.m.df, sprint.m.df$Country), function(y) y[y$Time==min(y$Time),])
c1 <- do.call("rbind", c1)
c1 <- c1[,c("Name", "City", "Time")]
c1 <- c1[order(c1$Time),]

c2 <- lapply(split(sprint.m.df, sprint.m.df$City), function(y) y[y$Time==min(y$Time),])
c2 <- do.call("rbind", c2)
c2 <- c2[,c("Name", "Country", "Time")]
c2 <- c2[order(c2$Time),]

head(c1, 10)
```

```
##                              Name           City Time
## JAM                    Usain Bolt         Berlin 9.58
## USA                     Tyson Gay       Shanghai 9.69
## TTO              Richard Thompson  Port of Spain 9.82
## CAN.70             Donovan Bailey        Atlanta 9.84
## CAN.71                Bruny Surin        Sevilla 9.84
## NGR     Adekotunbo Olusoji Fasuba      Ad-Dawhah 9.85
## FRA.127             Jimmy Vicaut    Saint-Denis 9.86
## FRA.129             Jimmy Vicaut      Montreuil 9.86
## NAM             Frank Fredericks       Lausanne 9.86
## POR             Francis Obikwelu Ath&iacute;nai 9.86
```

```
head(c2, 10)
```

```
##                         Name Country Time
## Berlin           Usain Bolt     JAM 9.58
## London           Usain Bolt     JAM 9.63
## Beijing          Usain Bolt     JAM 9.69
## Lausanne        Yohan Blake     JAM 9.69
## Shanghai          Tyson Gay     USA 9.69
## New York City    Usain Bolt     JAM 9.72
## Ad-Dawhah     Justin Gatlin     USA 9.74
## Rieti          Asafa Powell     JAM 9.74
## Kingston        Yohan Blake     JAM 9.75
## Roma          Justin Gatlin     USA 9.75
```

- **1f.** With the most minor modification to your code possible, do the same computations as in Q1d, but now when there are ties, pick only one of the relevant results arbitrarily (e.g., uniformly at random is fine).

```
sprint.m.df %>%
  group_by(Country) %>%
  filter(Time == min(Time)) %>%
  sample_n(1) %>%
  select(Name, City, Country, Time) %>%
  arrange(Time)
```

```
## # A tibble: 46 x 4
## # Groups:   Country [46]
##    Name                      City           Country  Time
##    <fct>                     <fct>          <fct>   <dbl>
##  1 Usain Bolt                Berlin         JAM      9.58
##  2 Tyson Gay                 Shanghai       USA      9.69
##  3 Richard Thompson          Port of Spain  TTO      9.82
##  4 Bruny Surin               Sevilla        CAN      9.84
##  5 Adekotunbo Olusoji Fasuba Ad-Dawhah      NGR      9.85
##  6 Jimmy Vicaut              Saint-Denis    FRA      9.86
##  7 Frank Fredericks          Lausanne       NAM      9.86
##  8 Francis Obikwelu          Ath&iacute;nai POR      9.86
##  9 Obadele Thompson          Johannesburg   BAR      9.87
## 10 Linford Christie          Stuttgart      GBR      9.87
## # ... with 36 more rows
```

```
sprint.m.df %>%
  group_by(City) %>%
  filter(Time == min(Time)) %>%
  sample_n(1) %>%
  select(Name, City, Country, Time) %>%
  arrange(Time)
```

```
## # A tibble: 320 x 4
## # Groups:   City [320]
##    Name          City          Country  Time
##    <fct>         <fct>         <fct>   <dbl>
##  1 Usain Bolt    Berlin        JAM      9.58
##  2 Usain Bolt    London        JAM      9.63
##  3 Usain Bolt    Beijing       JAM      9.69
##  4 Yohan Blake   Lausanne      JAM      9.69
##  5 Tyson Gay     Shanghai      USA      9.69
##  6 Usain Bolt    New York City JAM      9.72
##  7 Justin Gatlin Ad-Dawhah     USA      9.74
##  8 Asafa Powell  Rieti         JAM      9.74
##  9 Yohan Blake   Kingston      JAM      9.75
## 10 Justin Gatlin Roma          USA      9.75
## # ... with 310 more rows
```

Practice with pivoting wider and longer

In the following, use pipes and `dplyr` or `tidyr` verbs to solve each part as cleanly/succintly as you can. In some parts, it might make more sense to use direct indexing, and that's perfectly fine.

- **2a.** From `sprint.m.df`, define a reduced data frame `dat.reduced` as follows. For each athlete, and each city, compute the median of all times they recorded in this city. Your new data frame `dat.reduced` should have 1787 rows and 3 columns (Name, City, Time). Confirm that it has these dimensions, and display its first 10 entries.

```
dat.reduced <- sprint.m.df %>%
  group_by(Name, City) %>%
  summarise(Time = median(Time)) %>%
  select(Name, City, Time)
```

```
## `summarise()` has grouped output by 'Name'. You can override using the `.groups` argument.
```

```
dat.reduced
```

```
## # A tibble: 1,787 x 3
## # Groups:   Name [307]
##    Name              City             Time
##    <fct>             <fct>           <dbl>
##  1 Aaron Armstrong   Port of Spain  10.0
##  2 Aaron Brown       Edmonton       10.1
##  3 Aaron Brown       Eugene         10.1
##  4 Aaron Brown       Los Angeles    10.1
##  5 Aaron Brown       Montverde       9.98
##  6 Abdul Aziz Zakari Ath&iacute;nai 10.0
##  7 Abdul Aziz Zakari Berlin         10.0
##  8 Abdul Aziz Zakari Bruxelles      10.0
##  9 Abdul Aziz Zakari Hani&aacute;   10.0
## 10 Abdul Aziz Zakari Helsinki       10
## # ... with 1,777 more rows
```

- **2b.** 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. Use `pivot_wider()` to convert this into "wide" format, and call the result `dat.wide`. Here the first column should be the athlete names, and the remaining columns should correspond to the cities. *Please you the `arrange` function (2x) to get the columns (beside the `Name` column) and rows in alphabetical order. Apart from the first column, each entry gives the median time recorded by the athlete in this city. What are the dimensions of `dat.wide`, and do these make sense to you?

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

dat.wide
```

```
## # A tibble: 307 x 321
## # Groups:   Name [307]
##    Name   `&Eacute;vry-Bo~ Abbotsford Abilene Abuja `Ad-Dawhah` `Air Force Acad~
##    <fct>             <dbl>      <dbl>   <dbl> <dbl>       <dbl>            <dbl>
##  1 Aaron~               NA         NA      NA  NA          NA                  NA
##  2 Aaron~               NA         NA      NA  NA          NA                  NA
##  3 Abdul~               NA         NA      NA  NA          NA                  NA
##  4 Abdul~               NA         NA      NA  NA          NA                  NA
##  5 Abdul~               NA         NA      NA  NA          NA                  NA
##  6 Adam ~               NA         NA      NA  NA          NA                  NA
##  7 Adeko~               NA         NA      NA  10.1         9.89               NA
##  8 Akani~               NA         NA      NA  NA           9.99               NA
##  9 Alex ~               NA         NA      NA  NA          NA                  NA
## 10 Alons~               NA         NA      NA  NA          NA                  NA
## # ... with 297 more rows, and 314 more variables: Aix-les-Bains <dbl>,
## #   Albi <dbl>, Albuquerque <dbl>, Almaty <dbl>, Ames <dbl>, Amman <dbl>,
## #   Amsterdam <dbl>, Angers <dbl>, Ankara <dbl>, Antananarivo <dbl>,
## #   Arlington <dbl>, Ath&iacute;nai <dbl>, Athens <dbl>, Atlanta <dbl>,
## #   Auburn <dbl>, Auckland <dbl>, Austin <dbl>, Azusa <dbl>,
## #   Baie-Mahault <dbl>, Baku <dbl>, Bangkok <dbl>, Barcelona <dbl>,
## #   Basseterre <dbl>, Baton Rouge <dbl>, Bauchi <dbl>, Bedford <dbl>, ...
```

The dimensions are now 307x321, which makes sense because there were 307 names in the previous dataframe and 321 total races sounds about right.

- **2c.** Not counting the names in the first column, how many non-NA values does `dat.wide` have? Does this make sense to you? It should. Reason how could you have guessed this number ahead of time, without even calling `pivot_wider()`, based only on `dat.reduced`?

```
sum(sapply(dat.wide, function(x) sum(is.na(x))))
```

```
## [1] 96453
```

There are 96453 total NAs of a possible 98547. This makes sense because races usually only have <10 people, so each race will exclude the majority of the athletes in this data. You could have reasoned this out from dat.reduced by taking the number of athletes times the total amount of cities and subtracting the number of individual times.

- **2d.** 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.

```
not_na <- function(x) {!(is.na(x))}
dat.wide %>%
  filter(Name=="Usain Bolt") %>%
  select_if(not_na)
```

```
## # A tibble: 1 x 25
## # Groups:   Name [1]
##   Name       Beijing Berlin Bruxelles Daegu Georgetown Kingston Lausanne London
##   <fct>        <dbl>  <dbl>     <dbl> <dbl>      <dbl>    <dbl>    <dbl>  <dbl>
## 1 Usain Bolt    9.88   9.89      9.78  9.96       10.1      9.94     9.82   9.89
## # ... with 16 more variables: Monaco <dbl>, Moskva <dbl>, New York City <dbl>,
## #   Oslo <dbl>, Ostrava <dbl>, Port of Spain <dbl>, R&eacute;thymno <dbl>,
## #   Rio de Janeiro <dbl>, Roma <dbl>, Saint-Denis <dbl>, Spanish Town <dbl>,
## #   Stockholm <dbl>, Toronto <dbl>, Warszawa <dbl>, Z&uuml;rich <dbl>,
## #   Zagreb <dbl>
```

```
dat.reduced[dat.reduced$Name=="Usain Bolt",]
```

```
## # A tibble: 24 x 3
## # Groups:   Name [1]
##    Name       City        Time
##    <fct>      <fct>      <dbl>
##  1 Usain Bolt Beijing     9.88
##  2 Usain Bolt Berlin      9.89
##  3 Usain Bolt Bruxelles   9.78
##  4 Usain Bolt Daegu       9.96
##  5 Usain Bolt Georgetown 10.1
##  6 Usain Bolt Kingston    9.94
##  7 Usain Bolt Lausanne    9.82
##  8 Usain Bolt London      9.89
##  9 Usain Bolt Monaco      9.91
## 10 Usain Bolt Moskva      9.92
## # ... with 14 more rows
```

- **2e.** Use `pivot_longer()` to 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 `pivot_longer()`), and order the rows alphabetically by athlete and city name. Once you've done this, `dat.wide` should have matching entries to `dat.reduced`; confirm that this is the case.

```
dat.long <- pivot_longer(dat.wide, names_to = "City", values_to = "Time",
                         cols = 2:321, values_drop_na = TRUE)
dat.long <- dat.long %>%
  arrange(Name, City)
head(dat.reduced)
```

```
## # A tibble: 6 x 3
## # Groups:   Name [3]
##   Name              City            Time
##   <fct>             <fct>          <dbl>
## 1 Aaron Armstrong   Port of Spain  10.0
## 2 Aaron Brown       Edmonton       10.1
## 3 Aaron Brown       Eugene         10.1
## 4 Aaron Brown       Los Angeles    10.1
## 5 Aaron Brown       Montverde       9.98
## 6 Abdul Aziz Zakari Ath&iacute;nai 10.0
```

```
head(dat.long)
```

```
## # A tibble: 6 x 3
## # Groups:   Name [3]
##   Name              City            Time
##   <fct>             <chr>          <dbl>
## 1 Aaron Armstrong   Port of Spain  10.0
## 2 Aaron Brown       Edmonton       10.1
## 3 Aaron Brown       Eugene         10.1
## 4 Aaron Brown       Los Angeles    10.1
## 5 Aaron Brown       Montverde       9.98
## 6 Abdul Aziz Zakari Ath&iacute;nai  10.0
```

Practice with joins

Below we read in a data frame `sprint.w.df` containing the top women's times in the 100m sprint, as seen in previous labs. In the following, use pipes and `dplyr` verbs to solve each part as cleanly/succintly as you can. Note: you'll receive warnings when you make joins about the conversion of factors to characters, and that's fine, don't worry about it.

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

- **3a.** As in Q1f, compute for each country, the triplet (Name, Country, Time) corresponding to the male athlete with the fastest time among athletes from that country, and breaking ties arbitrarily. Instead of displaying the results, save the resulting data frame as `dat.m`. Importantly, at the end of your flow of pipe commands used to define `dat.m`, make sure to call `ungroup()`. This will assure that `dat.m` has no groupings associated with it. Do the same for the women, and call the result `dat.w`. Report the dimensions of `dat.m` and `dat.w`, and check that they make sense to you.

```
dat.m <- sprint.m.df %>%
  group_by(Country) %>%
  filter(Time == min(Time)) %>%
  sample_n(1) %>%
  select(Name, Country, Time) %>%
  arrange(Time) %>%
  ungroup

dat.w <- sprint.w.df %>%
  group_by(Country) %>%
  filter(Time == min(Time)) %>%
  sample_n(1) %>%
  select(Name, Country, Time) %>%
  arrange(Time) %>%
  ungroup

head(dat.m)
```

```
## # A tibble: 6 x 3
##   Name                      Country  Time
##   <fct>                     <fct>   <dbl>
## 1 Usain Bolt                JAM      9.58
## 2 Tyson Gay                 USA      9.69
## 3 Richard Thompson          TTO      9.82
## 4 Donovan Bailey            CAN      9.84
## 5 Adekotunbo Olusoji Fasuba NGR      9.85
## 6 Jimmy Vicaut              FRA      9.86
```

```
head(dat.w)
```

```
## # A tibble: 6 x 3
##   Name                     Country  Time
##   <fct>                    <fct>   <dbl>
## 1 Florence Griffith-Joyner USA      10.5
## 2 Zhanna Block             UKR      10.6
## 3 Juliet Cuthbert          JAM      10.7
## 4 Blessing Okagbare        NGR      10.7
## 5 Svetlana Goncharenko     RUS      10.7
## 6 Christine Arron          FRA      10.7
```

- **3b.** Perform an inner join, using `inner_join()`, of `dat.m` and `dat.w`, with the join done by the Country column. Call the resulting data frame `dat.ij`, and display its first 10 rows. How many rows does it have in total? Show how could you have arrived at this number ahead of time, from `dat.m$Country` and `dat.w$Country` (hint: `intersect()`). Count the number of `NA` values in `dat.ij`: this should be zero.

```
dat.ij <- inner_join(x=dat.m, y=dat.w, by="Country")
head(dat.ij, 10)
```

```
## # A tibble: 10 x 5
##    Name.x                    Country Time.x Name.y                   Time.y
##    <fct>                     <fct>    <dbl> <fct>                     <dbl>
##  1 Usain Bolt                JAM       9.58 Juliet Cuthbert            10.7
##  2 Tyson Gay                 USA       9.69 Florence Griffith-Joyner   10.5
##  3 Richard Thompson          TTO       9.82 Kelly-Ann Baptiste         10.8
##  4 Donovan Bailey            CAN       9.84 Angela Bailey              11.0
##  5 Adekotunbo Olusoji Fasuba NGR       9.85 Blessing Okagbare          10.7
##  6 Jimmy Vicaut              FRA       9.86 Christine Arron            10.7
##  7 Linford Christie          GBR       9.87 Dina Asher-Smith           11.0
##  8 Akani Simbine             RSA       9.89 Evette de Klerk            11.1
##  9 Derrick Atkins            BAH       9.91 Chandra Sturrup            10.8
## 10 Churandy Martina          NED       9.91 Dafne Schippers            10.8
```

```
length(dat.ij$Name.x)
```

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

```
length(intersect(dat.w$Country, dat.m$Country))
```

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

```
sum(colSums(is.na(dat.ij)))
```

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

There are 21 total rows. You could have found this value beforehand by intersecting the country columns and counting the total.

- **3c.** Perform a left join, using `left_join()`, of `dat.m` and `dat.w`, with the join again done by the Country column. Call the resulting data frame `dat.lj`, and display its first 10 rows. How many rows does it have in total? Explain why this makes sense. Count the number of `NA` values in `dat.lj`: this should be 50. Show how you could have arrived at this number from `dat.m$Country` and `dat.w$Country` (hint: `setdiff()`).

```
dat.lj <- left_join(x=dat.m, y=dat.w, by="Country")
head(dat.lj, 10)
```

```
## # A tibble: 10 x 5
##    Name.x                    Country Time.x Name.y                   Time.y
##    <fct>                     <fct>    <dbl> <fct>                     <dbl>
##  1 Usain Bolt                JAM       9.58 Juliet Cuthbert            10.7
##  2 Tyson Gay                 USA       9.69 Florence Griffith-Joyner   10.5
##  3 Richard Thompson          TTO       9.82 Kelly-Ann Baptiste         10.8
##   4 Donovan Bailey            CAN       9.84 Angela Bailey            11.0
##   5 Adekotunbo Olusoji Fasuba NGR       9.85 Blessing Okagbare        10.7
##   6 Jimmy Vicaut              FRA       9.86 Christine Arron          10.7
##   7 Frank Fredericks          NAM       9.86 <NA>                     NA
##   8 Francis Obikwelu          POR       9.86 <NA>                     NA
##   9 Obadele Thompson          BAR       9.87 <NA>                     NA
##  10 Linford Christie          GBR       9.87 Dina Asher-Smith         11.0
```

```
length(dat.lj$Name.x)
```

```
## [1] 46
```

```
sum(colSums(is.na(dat.lj)))
```

```
## [1] 50
```

```
2*length(setdiff(dat.m$Country, dat.w$Country))
```

```
## [1] 50
```

It has 46 total rows. This makes sense because it will now include male observations from countries with no female observation. You could have arrived at this number beforehand by counting male observations with no female counterpart and calculating two times this number.

- **3d.** Finally, perform an full join, using `full_join()`, of `dat.m` and `dat.w`, with the join again done by the Country column. Call the resulting data frame `dat.fj`. How many rows does it have in total? Show how you could have arrived at this number from `dat.m$Country` and `dat.w$Country` (hint: `union()`). Count the number of `NA` values in `dat.fj`: this should be 80. **Challenge**: show how you could have arrived at this number from `dat.m$Country` and `dat.w$Country`.

```
dat.fj <- full_join(x=dat.m, y=dat.w, by="Country")
length(dat.fj$Name.x)
```

```
## [1] 61
```

```
length(union(dat.m$Country, dat.w$Country))
```

```
## [1] 61
```

```
sum(colSums(is.na(dat.fj)))
```

```
## [1] 80
```

```
2*length(setdiff(dat.m$Country, dat.w$Country)) + 2*length(setdiff(dat.w$Country, dat.m$Country))
```

```
## [1] 80
```

This dataframe has 61 rows. You could have computed this beforehand by taking the size of the union of the countries from each dataframe. You could have counted the total amount of NA values beforehand by taking the total number of countries not in the union and multiplying it by two, as I did above.

More grouping and joining

Below is some solution code from Lab 8, where we convert the Birthdate and Date columns in the `sprint.m.df` and `sprint.w.df` data frames to numeric form. In what follows, you will resolve some of the questions from Lab 8, but using pipes and `dplyr`, `tidyr`.

```
date.to.numeric = function(val) {
  val = as.character(val)
  vec = strsplit(val, split  = "\\.")[[1]]
  if (nchar(vec[3]) == 2) vec[3] = paste0("19", vec[3])
  vec = as.numeric(vec)
  vec[3]*10^4 + vec[2]*10^2 + vec[1]
}

sprint.m.df$Birthdate = sapply(sprint.m.df$Birthdate, date.to.numeric)
sprint.m.df$Date = sapply(sprint.m.df$Date, date.to.numeric)
sprint.w.df$Birthdate = sapply(sprint.w.df$Birthdate, date.to.numeric)
sprint.w.df$Date = sapply(sprint.w.df$Date, date.to.numeric)

head(sprint.m.df, 5)
```

```
##   Rank Time Wind        Name Country Birthdate     City     Date
## 1    1 9.58  0.9  Usain Bolt     JAM  19860821   Berlin 20090816
## 2    2 9.63  1.5  Usain Bolt     JAM  19860821   London 20120805
## 3    3 9.69  0.0  Usain Bolt     JAM  19860821  Beijing 20080816
## 4    3 9.69  2.0   Tyson Gay     USA  19820809 Shanghai 20090920
## 5    3 9.69 -0.1 Yohan Blake     JAM  19891226 Lausanne 20120823
```

```
head(sprint.w.df, 5)
```

```
##   Rank  Time Wind                     Name Country Birthdate         City
## 1    1 10.49  0,0 Florence Griffith-Joyner     USA  19591221 Indianapolis
## 2    2 10.61 +1,2 Florence Griffith-Joyner     USA  19591221 Indianapolis
## 3    3 10.62 +1,0 Florence Griffith-Joyner     USA  19591221        Seoul
## 4    4 10.64 +1,2          Carmelita Jeter     USA  19791124     Shanghai
## 5    5 10.65 +1,1             Marion Jones     USA  19751012 Johannesburg
##       Date
## 1 19880716
## 2 19880717
## 3 19880924
## 4 20090920
## 5 19980912
```

- **4a.** Here you'll effectively resolve Q2c and Q2d from Lab 8, using one single flow of pipe commands, for each of the `sprint.m.df` and `sprint.w.df` data frames. In particular, define a new column CityDate given by concatenating the City and Date columns separated by a "." (hint: `unite()`), then keep only the row with the fastest time for each value of CityDate (breaking ties arbitrarily), then sort the rows by increasing Time Call the resulting data frames `dat.m.cd` and `dat.w.cd`. Make sure in the last line of pipe commands use to define them, you call `ungroup()`. Check that these data frames have dimensions 1253 x 7 and 921 x 7, respectively, and display the first 5 rows of each.

```
dat.m.cd <- sprint.m.df %>%
  unite(CityDate, City, Date, sep=".") %>%
  group_by(CityDate) %>%
  filter(Time==min(Time)) %>%
  sample_n(1) %>%
  arrange(Time) %>%
  ungroup

dat.w.cd <- sprint.w.df %>%
  unite(CityDate, City, Date, sep=".") %>%
  group_by(CityDate) %>%
  filter(Time==min(Time)) %>%
  sample_n(1) %>%
  arrange(Time) %>%
  ungroup

head(dat.m.cd, 5)
```

```
## # A tibble: 5 x 7
##    Rank  Time  Wind Name        Country Birthdate CityDate
##   <int> <dbl> <dbl> <fct>       <fct>       <dbl> <chr>
## 1     1  9.58   0.9 Usain Bolt  JAM      19860821 Berlin.20090816
## 2     2  9.63   1.5 Usain Bolt  JAM      19860821 London.20120805
## 3     3  9.69   0   Usain Bolt  JAM      19860821 Beijing.20080816
## 4     3  9.69  -0.1 Yohan Blake JAM      19891226 Lausanne.20120823
## 5     3  9.69   2   Tyson Gay   USA      19820809 Shanghai.20090920
```

```
head(dat.w.cd, 5)
```

```
## # A tibble: 5 x 7
##    Rank  Time Wind  Name                      Country Birthdate CityDate
##   <int> <dbl> <fct> <fct>                     <fct>       <dbl> <chr>
## 1     1  10.5 0,0   Florence Griffith-Joyner USA      19591221 Indianapolis.198~
## 2     1  10.6 0,0   Zhanna Block              UKR      19720706 Kiev.19970612
## 3     2  10.6 +1,2  Florence Griffith-Joyner USA      19591221 Indianapolis.198~
## 4     3  10.6 +1,0  Florence Griffith-Joyner USA      19591221 Seoul.19880924
## 5     4  10.6 +1,2  Carmelita Jeter           USA      19791124 Shanghai.20090920
```

- **4b.** Now you'll effectively resolve Q3 on Lab 8, using one single flow of pipe commands, for each of the `sprint.m.df` and `sprint.w.df` data frames. In particular, do an inner join between `dat.m.cd` and `dat.w.cd` by CityDate, then drop the Rank.x, Rank.y, Birthdate.x, Birthdate.y columns. Call the resulting data frame `dat.cd` and check that its dimensions are 377 x 9. Display its first 10 rows, and check that it has no `NA` values.

```
dat.cd <- inner_join(x=dat.m.cd, y=dat.w.cd, by="CityDate") %>%
  select(-c("Rank.x", "Rank.y", "Birthdate.x", "Birthdate.y"))
dim(dat.cd)
```

```
## [1] 377   9
```

```
head(dat.cd, 10)
```

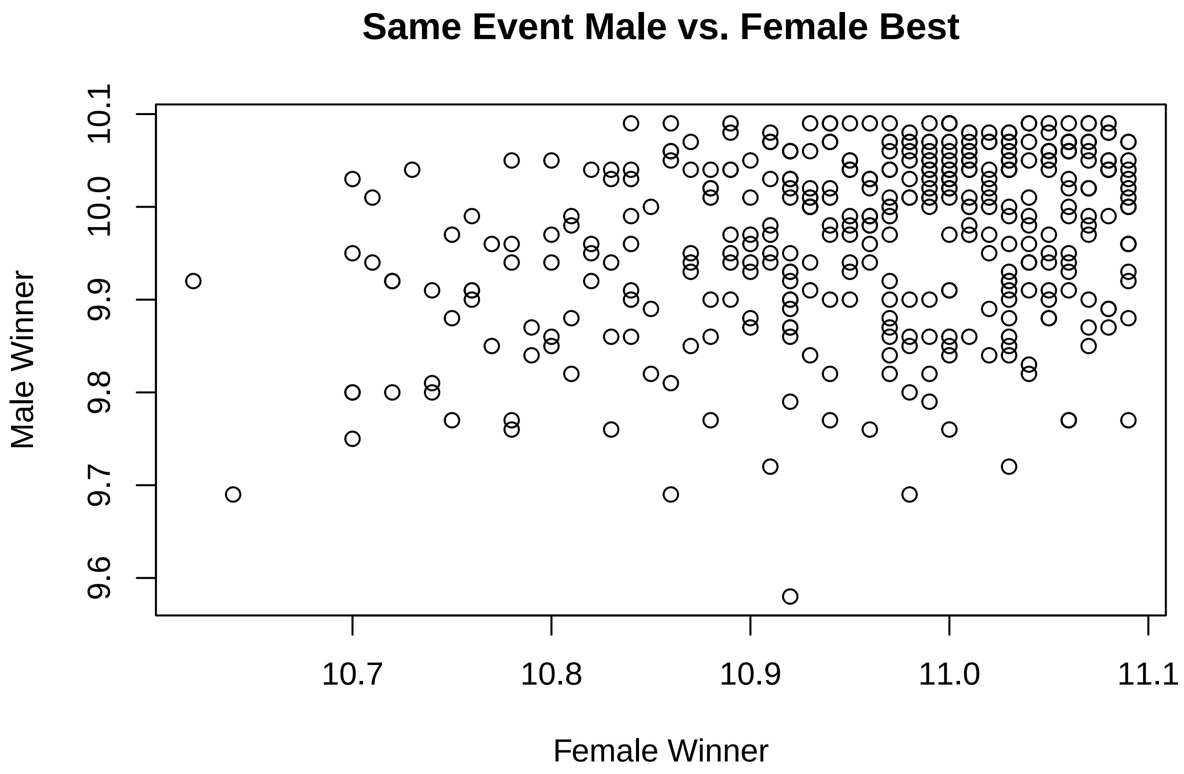
```
## # A tibble: 10 x 9
##    Time.x Wind.x Name.x       Country.x CityDate  Time.y Wind.y Name.y Country.y
##     <dbl>  <dbl> <fct>        <fct>     <chr>      <dbl> <fct>  <fct>  <fct>
##  1   9.58    0.9 Usain Bolt   JAM       Berlin.2~   10.9 -0,3   Kerro~ JAM
##  2   9.69    0   Usain Bolt   JAM       Beijing.~   11.0 +0,4   Kerro~ JAM
##  3   9.69   -0.1 Yohan Blake  JAM       Lausanne~   10.9 -0,1   Carme~ USA
##  4   9.69    2   Tyson Gay    USA       Shanghai~   10.6 +1,2   Carme~ USA
##  5   9.72    0.2 Asafa Powell JAM       Lausanne~   11.0 +0,2   Shell~ JAM
##  6   9.72    1.7 Usain Bolt   JAM       New York~   10.9 +0,9   Veron~ JAM
##  7   9.75    1.1 Yohan Blake  JAM       Kingston~   10.7 +0,6   Shell~ JAM
##  8   9.76    1.3 Usain Bolt   JAM       Bruxelle~   10.8 +0,4   Carme~ USA
##  9   9.76    1.8 Usain Bolt   JAM       Kingston~   11.0 +0,9   Kerro~ JAM
## 10   9.76   -0.1 Usain Bolt   JAM       Roma.201~   11   0,0    Murie~ CIV
```

```
sum(colSums(is.na(dat.cd)))
```

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

- **4c.** Reproduce the plot you made in Q3d on Lab 8, of Time.y (women's time) versus Time.x (men's time), from the `dat.cd` data frame. As a reminder, a positive correlation here would indicate some kind of "track meet effect". Call `cor.test()` on Time.x and Time.y and report the p-value. This should all look exactly the same as in Q3d from Lab 8, it's just a check of reproducibility.

```
plot(dat.cd$Time.y, dat.cd$Time.x,
     main="Same Event Male vs. Female Best", xlab="Female Winner", ylab="Male Winner")
```

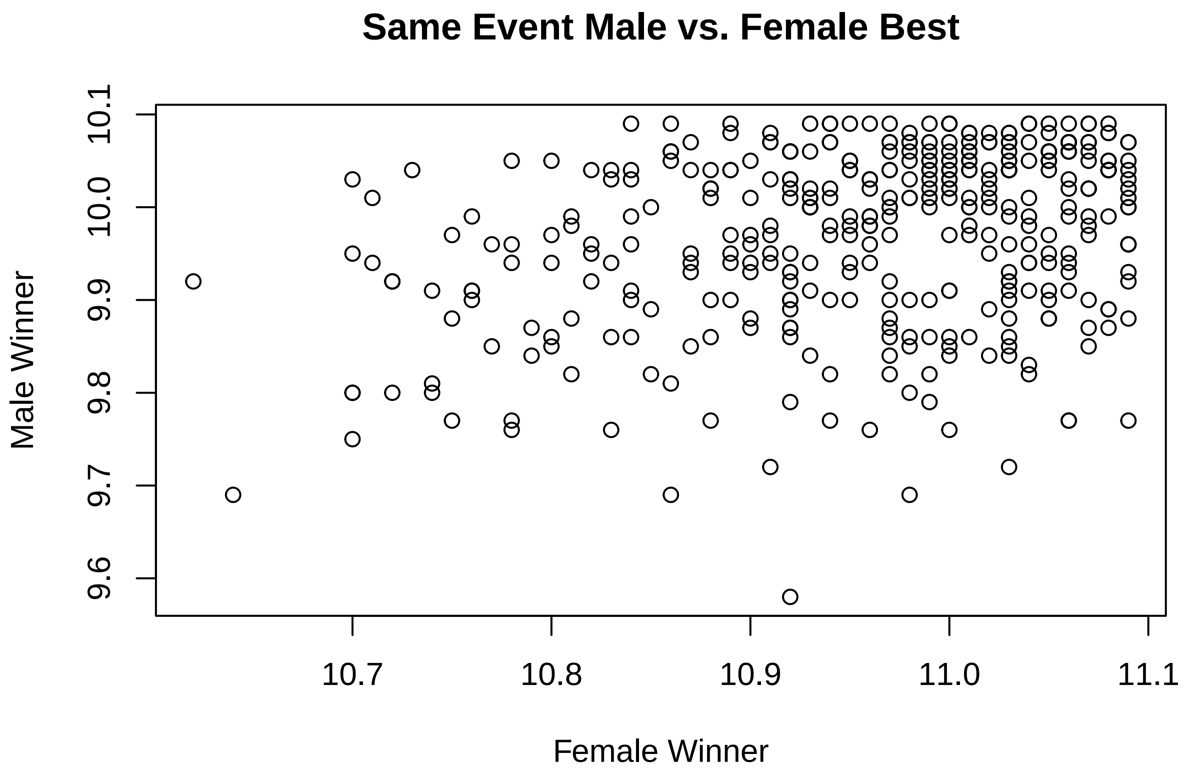


```
cor.test(dat.cd$Time.y, dat.cd$Time.x)
```

```
##
##  Pearson's product-moment correlation
##
## data:  dat.cd$Time.y and dat.cd$Time.x
## t = 5.9721, df = 375, p-value = 5.441e-09
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1996416 0.3842650
## sample estimates:
##      cor
## 0.294701
```

- **Challenge.** In one single flow of pipe commands, for each of `sprint.m.df` and `sprint.w.df` (i.e., without saving an intermediate object `dat.cd`), reproduce the results in Q4b and Q4c. You don't have to worry about reporting the dimensions of the joined data frame or displaying its first 10 rows; just complete the inner join, produce the plot, and report the p-value from `cor.test()`. Hint: to produce the plot *before* you report the p-value from `cor.test()`, you're going to have to use the "tee" operator `%T>%` so that the pipe flow doesn't terminate prematurely.

```r
# dplyr inserts df into cor.test regardless of using dot operator...
dat.m.cd %>%
  inner_join(y=dat.w.cd, by="CityDate") %T>%
  plot(Time.x ~ Time.y, data=.,
       main="Same Event Male vs. Female Best", xlab="Female Winner", ylab="Male Winner") %>%
  summarize(coef = cor.test(.$Time.x, .$Time.y)$estimate,
            pval = cor.test(.$Time.x, .$Time.y)$p.value)
```



```
## # A tibble: 1 x 2
##    coef          pval
##   <dbl>         <dbl>
## 1 0.295 0.00000000544
```

Split-apply-combine with nesting (optional)

Sometimes you'd like to preform analysis conditional on a set of groups (think back to the times you've used `tapply`). There's a paradigm called "split-apply-combine" that defines the steps you'd need to take to preform this type of analysis. In the "tidyverse" this approach can be done using the `nesting` commands from `tidyr`.

More specifically, this problem with introduce you to nesting (`nest` and `unnest`) as well as the functions `purrr::map` and some functions from the package `broom`. Lecture slide #21 provides a link to a lecture (Rmd, html) that covers most of the material in this problem.

For this problem we'll be looking at a slightly different dataset that can be loaded in using the following:

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

This dataset contains information about the best sprinters (conditional on gender) for each year. It contains 3 new columns compared to the above data frames:

1. `Gender` (factor): indicates which gender the runner was
2. `Year` (integer): which year the time was recorded
3. `Year.centered` (integer): relative year

Suppose we were interested in examine the relationship between the best time relative to the year and wind speed conditional on gender. In a linear model, we could model

`Time ~ Wind*Gender + Year.centered*Gender + Gender`

but today we will instead look at making 2 models (filtering the data by gender) and then looking at the below relationship:

`Time ~ Wind + Year.centered`

- **6a.** Run the following line of code (note you'll need to remove the "eval = FALSE"). What is the size of nested.df? What are the column names? Examine the element `nested.df$data[[1]]` and describe it (please also identify what subgroup it belongs to).

```
nested.df = sprint.best.full.df %>%
  group_by(Gender) %>%
  nest()
```

nested.df is 2x2. The column names are Gender and data. nested.df$data[[1]] is equal to sprint.m.df, the dataframe containing men's sprint times.

- **6b.** You probably noticed in the last part that the `nest` function "nested" the proportion of the `sprint.best.df.full` associated to the specific gender into the column `data` in the `nested.df`. The `nest` function along with the `map` function from purrr allows us to preform similar operation in a "tidyverse" way as you learned when you used things like `tapply` and `lapply`.

Suppose, at the end of the day we wanted to compare linear model β coefficients between the two models (1 built with male data, one with female data). The first thing we'd need to do would be to run the linear also as described above. For a single dataset we could do something like what is demonstrated below.

```
purrr::map(nested.df$data[1],function(df) lm(Time ~ Wind + Year.centered, data = df))
#or
lapply(nested.df$data[1],function(df) lm(Time ~ Wind + Year.centered, data = df))
```

In "tidyverse" land, let's use `purrr::map`. We can create (and store) these linear models into our data frames using mutate, specifically we can do the following (make sure to change the "eval = T":

```
nested.df = nested.df %>%
  mutate(model = map(data, function(df) lm(Time ~ Wind + Year.centered, data = df)))
# if for some reason the above doesn't work, try:
# my.lm.func = function(df) lm(Time ~ Wind + Year.centered, df)
# nested.df = nested.df %>%
#   mutate(model = map(data, my.lm.func))
```

Check what columns nested.df contains. What is the new column's name? What class of object is stored in each element?

```
colnames(nested.df)
```

```
## [1] "Gender" "data"   "model"
```

```
typeof(nested.df[1, "model"])
```

```
## [1] "list"
```

The new column's name is model and it contains a list.

- **6c.** Now, we want to grab out the coefficents (and for now, suppose also the full summary). Update the nested.df such that we have a summary of each model in a new column called sum. Remember you should use map and that you're applying summary to the models, not the data.

```
nested.df <- nested.df %>%
  mutate(sum = map(model, summary))
```

- **6d.** (No work, just reading) What you should be noticing is that this approach allows you to interatively write your code (which has it's benefits). Sadly we need a final step (which we provide for you). We will discuss why in the last part of this question (summary). (Make sure to correct eval = F.)

```
nested.df <- nested.df %>%
  mutate(sum2 = map(sum, broom::tidy))
```

- **6e.** Now we'd like to pull out the the summary information out of this "nested" format. To do so we use the function unnest. We provide the code for you below. Why do you think we use select(Gender, coef2)? Express in words how the unnested data frame changes if we don't include that line of code.

```
unnested.df <- nested.df %>%
  select(Gender, sum2) %>%
  unnest(sum2)
```

If we don't select gender and sum2, then we would see the rest of the data in the nested dataframe.

- **6f.** Finally, create a table using that has 2 rows (for each gender) and contains the beta coefficents of each of the terms in the model (define this "table" as beta.model.df and print it out). Hint: you'll probably use a pivot_* and a select call. Looking at this table and back at unnested.df does it appear that the effect of year (conditional on Wind speed) is stronger for male runners or female runners? (Note these models isn't super amazing—so you shouldn't really see this as a take away.)

```
df.wide <- pivot_wider(unnested.df, names_from="term", values_from="estimate") %>%
  group_by(Gender) %>%
  summarize(Intercept = max(`(Intercept)`, na.rm=TRUE),
            Wind = max(Wind, na.rm=TRUE),
            Year = max(Year.centered, na.rm=TRUE))
```

From this model, it appears that the men's times decrease more than women's times as the years progress.

- **Summary.** You've now gotten a test of the "split-apply-combine" paradigm using nest/unnest, purrr and a little bit of functions from the broom library. This approach should appear similar to apply style coding but a bit more iterative. You may have noticed that we had a previous extra step in Q6d that you might not have expected; as tidyverse emphasis is on data.frames, we end up needing to work with data frame to make sure that the unnesting works as expected.

Finally, we call this approach "split-apply-combine" based on the sequence of steps one takes, in this example we could seperate these sequences into:

1. split: group_by call
2. apply: all the mutate(purrr::map) style steps

3. combine: the use of `pivot_*` to alter the final output