Lab 5: Dplyr, Pipes, and More

Statistical Computing, 36-350

Name: Rufus Petrie

This week's agenda: mastering the pipe operator `%>%`, practicing `dplyr` verbs, and pivoting using `tidyr`.

Loading the tidyverse

Now we'll load the tidyverse suite of packages. (You should already have `tidyverse` installed from the last lab; but if for some reason you still need to install again, then you can just look back at the last lab's instructions.) This gives us access to the pipe operator `%>%` as well as the `dplyr` and `tidyr` packages needed to complete this lab.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

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"
```

```
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.**

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

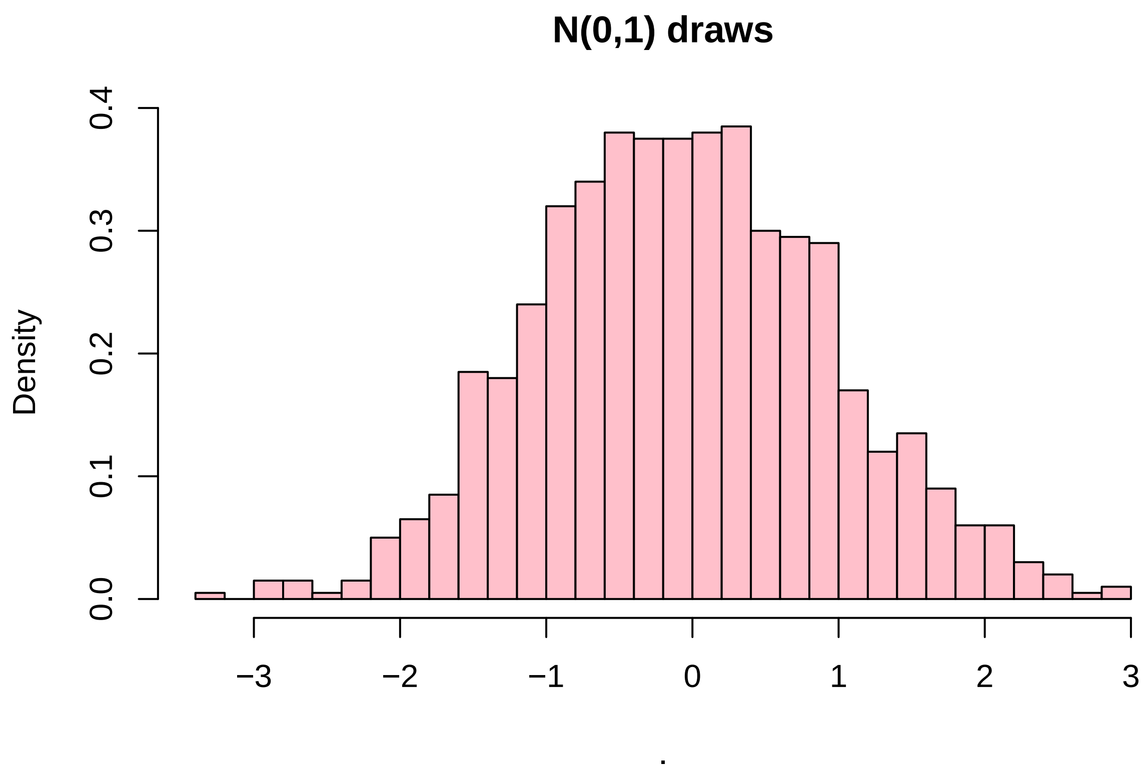
```
## [1] "Ceci n'est pas un pipe"
```

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

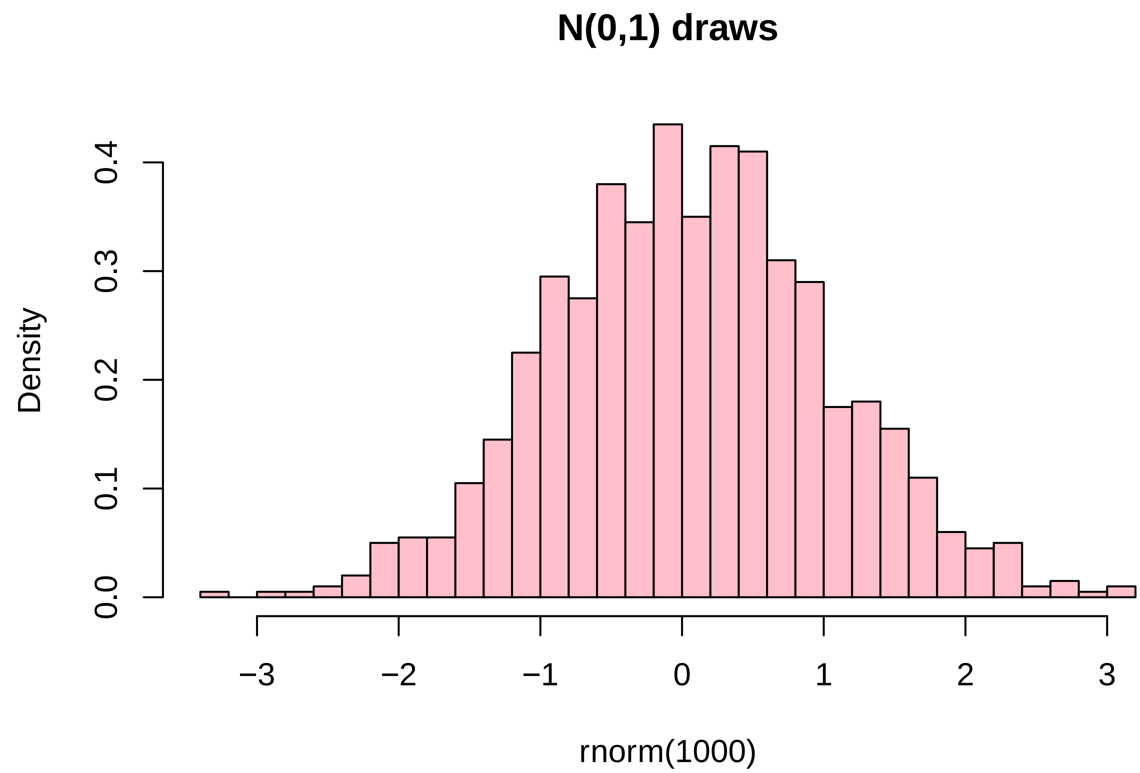
```
## [1] "Ceci n'est pas un pipe"
```

- **1c.**

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




```r
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] 0.43
```

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

```
## [1] 0.435
```

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.

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

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

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

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

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

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

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

```
as.data.frame(state.x77) %>%
  select(Illiteracy) %>%
  arrange(desc(Illiteracy)) %>%
  head(1)
```

```
##           Illiteracy
## Louisiana        2.8
```

- **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).

```
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
```

```
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(text, 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
```

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")
```

Q3. Practice with dplyr verbs

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

- **3a.** 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)) %>%
  filter(lpsa == max(lpsa) | lpsa == min(lpsa)) %>%
  select(lpsa)
```

```
##          lpsa
## 1  -0.4307829
## 92  4.1295508
```

- **3b.** 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
```

- **3c.** 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
```

- **3d.** 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]))
}
```

```
pros.df %>%
  select(-c("svi")) %>%
  lapply(., t.test.by.ind, ind = pros.df[,5])
```

```
## $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
```

- **3e.** 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.

```
pros.df %>%
  select(-c("svi")) %>%
  lapply(., t.test.by.ind, ind = pros.df[,5]) %>%
  sapply(., '[[', 'p.value')
```

```
## 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 in two data sets of the 1000 fastest times ever recorded for the 100m sprint, in men's and women's track., as seen in the last lab.

```
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)
```

Q4. More practice with `dplyr` verbs

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

- **4a.** Order the rows by increasing `Wind` value, and then display only the women who ran at most 10.7 seconds.

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

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

- **4b.** 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.

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

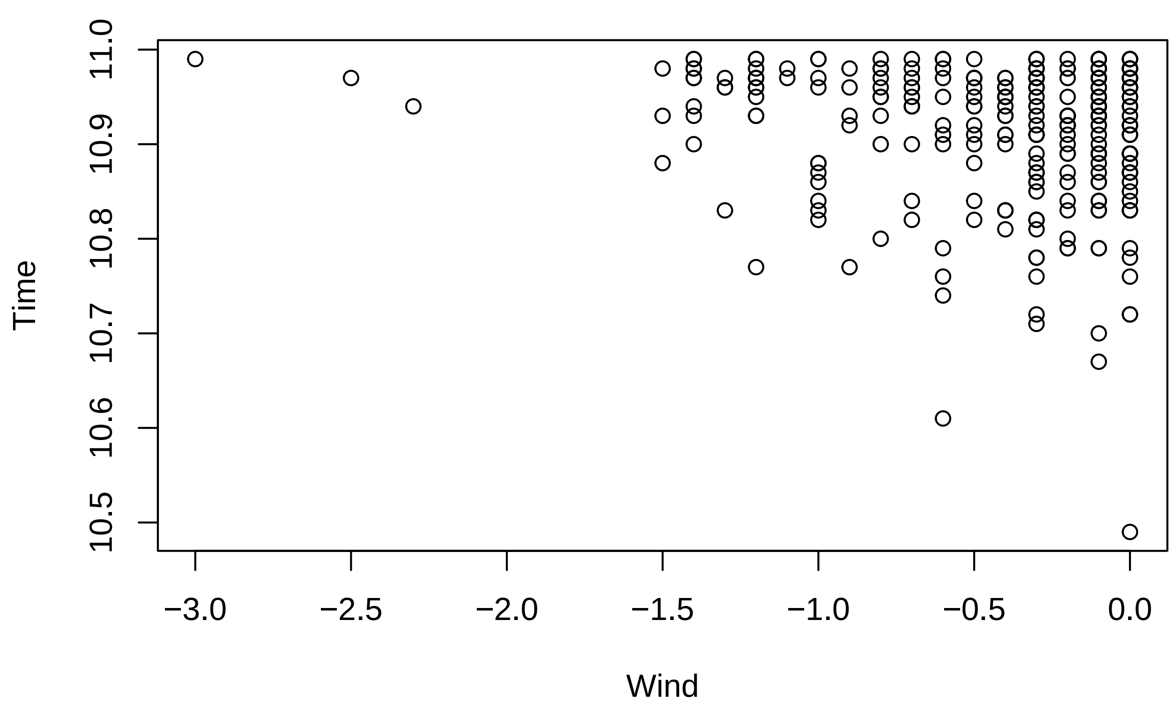
```
##      Time Wind                     Name       Date
## 1   10.49  0.0 Florence Griffith-Joyner 16.07.1988
## 2   10.54  0.9    Elaine Thompson-Herah 21.08.2021
## 3   10.60  1.7   Shelly-Ann Fraser-Pryce 26.08.2021
```

```
## 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
```

- **4c.** 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).

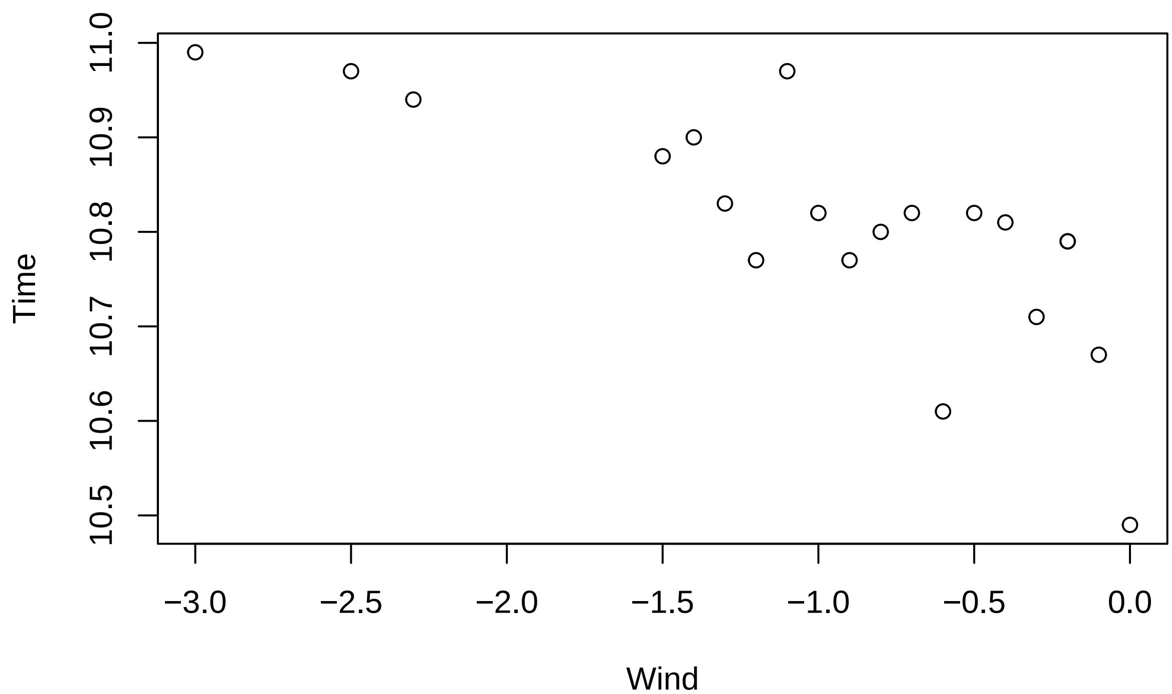
```r
sprint.w.df %>%
  filter(Wind <= 0) %>%
  plot(Time ~ Wind, data = .)
```

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



- **4d.** 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.)

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



Q5. 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.

- **5a.** 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.

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

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

```
sprint.m.df <- sprint.m.df %>%
  mutate_at("Time", as.numeric)
```

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

One such NA value had the time "9.98A", which means that the time variable could be a character because of an input error.

- **5b.** 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.

```
dat.reduced <- sprint.m.df %>%
  group_by(Name, City) %>%
  filter(rank(Time, ties.method ="first") == 1) %>%
  drop_na(., Time) %>%
  select(Name, City, Time)
head(dat.reduced, 10)
```

```
## # A tibble: 10 x 3
## # Groups:   Name, City [10]
##    Name          City           Time
##    <chr>         <chr>         <dbl>
##  1 Usain Bolt    Berlin         9.58
##  2 Usain Bolt    London         9.63
##  3 Usain Bolt    Beijing        9.69
##  4 Tyson Gay     Shanghai       9.69
##  5 Yohan Blake   Lausanne       9.69
##  6 Tyson Gay     Berlin         9.71
##  7 Usain Bolt    New York City  9.72
##  8 Asafa Powell  Lausanne       9.72
##  9 Asafa Powell  Rieti          9.74
## 10 Justin Gatlin Ad-Dawhah      9.74
```

- **5c.** 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 alphebtically 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 %>%
  arrange(City) %>%
  pivot_wider(names_from = City,
              values_from = Time)
```

- **5d.** 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)?

```
sum(is.na(dat.wide))
```

```
## [1] 20691
```

You could have surmised this number from 'dat.reduced' by taking the total number of (athletes x cities) and subtracting the amount of times.

- **5e.** 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.

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

```
## # A tibble: 1 x 152
## # Groups: Name [1]
## Name Abuja `Ad-Dawhah` `Aix-les-Bains` Albi Amman Andorf Angers
## <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Usain Bolt NA NA NA NA NA NA NA
```

```
## # i 144 more variables: `Ath&iacute;nai` <dbl>, `Athens GA` <dbl>,
## # Atlanta <dbl>, Auburn <dbl>, Austin <dbl>, Barcelona <dbl>,
## # Basseterre <dbl>, `Baton Rouge` <dbl>, Bedford <dbl>, Beijing <dbl>,
## # `Bel&eacute;m` <dbl>, Bellinzona <dbl>, Beograd <dbl>, Bergen <dbl>,
## # Berkeley <dbl>, Berlin <dbl>, Birmingham <dbl>, Boise <dbl>, Boston
<dbl>,
## # Bottrop <dbl>, Braga <dbl>, Bruxelles <dbl>, Budapest <dbl>, Burnaby
<dbl>,
## # Bursa <dbl>, `Camag&uuml;ey` <dbl>, Carson <dbl>, Cayenne <dbl>, ...
```

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

```
## # A tibble: 20 x 3
## # Groups:   Name, City [20]
##    Name       City            Time
##    <chr>      <chr>          <dbl>
##  1 Usain Bolt Berlin          9.58
##  2 Usain Bolt London          9.63
##  3 Usain Bolt Beijing         9.69
##  4 Usain Bolt New York City   9.72
##  5 Usain Bolt Kingston        9.76
##  6 Usain Bolt Bruxelles       9.76
##  7 Usain Bolt Roma            9.76
##  8 Usain Bolt Moskva          9.77
##  9 Usain Bolt Saint-Denis     9.79
## 10 Usain Bolt Oslo            9.79
## 11 Usain Bolt Z&uuml;rich     9.81
## 12 Usain Bolt Rio de Janeiro  9.81
## 13 Usain Bolt Lausanne        9.82
## 14 Usain Bolt Zagreb          9.85
## 15 Usain Bolt Daegu           9.86
## 16 Usain Bolt Monaco          9.88
## 17 Usain Bolt Stockholm       9.89
## 18 Usain Bolt Ostrava         9.91
## 19 Usain Bolt Port of Spain   9.92
## 20 Usain Bolt Warszawa        9.98
```

- **5f.** 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.

```
dat.long <- dat.wide %>%
  pivot_longer(names_to = "City",
               values_to = "Time",
               cols = 2:152,
               values_drop_na = TRUE) %>%
  arrange(Name, City)
dat.reduced <- dat.reduced %>%
  arrange(Name, City)
head(dat.long)
```

```
## # A tibble: 6 x 3
## # Groups:   Name [4]
##   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
```

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

```
## # A tibble: 6 x 3
## # Groups:   Name, City [6]
##   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
```

The dataframes have the same entries.