Data Wrangling II

Meghan Muse

July 16, 2024

Lesson Objectives

At the end of this lecture you should be able to:

1. Use pipes in dplyr
2. Subset data using dplyr
3. Move between wide and long data frames in tidyr
4. Generate simple summary tables

Resources

Cheat Sheet for Functions in dplyr: https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Pipes in tidyverse: https://style.tidyverse.org/pipes.html

```
# Install all tidyverse associated packages
#install.packages('tidyverse')

# If you only want to install the packages we will use in this lecture:
#install.packages('dplyr')
#install.packages('tidyr')
```

Data Set

We're going to start this lecture by generating the same random data set that we used in the last lecture. As always, don't forget to set a random seed so that our data is comparable across lectures.

```
# Set a random seed
set.seed(103)

# Define a data frame with our randomly generated data
randomData <- data.frame('SubjectID' = seq(1:1000),
                         'Systolic.BP' = rnorm(n = 1000,mean = 128,sd = 20),
                         'Diastolic.BP' = rnorm(n = 1000,mean = 71,sd = 10),
                         'Age' = trunc(runif(n = 1000,min = 18,max = 70)),
                         'Male' = rbinom(n = 1000,size = 1,prob = 0.5))
# Define binary variable for biological sex
randomData$BiologicalSex <- factor(ifelse(randomData$Male == 1,'Male','Female'))
# Define variable specifying age above 65 (medicare eligible)
randomData$MedicareAge <- ifelse(randomData$Age < 65,F,T)
```

Pipes

If you've looked at a lot of sample code online before, you've probably run into this syntax: %>%. This is a pipe! Pipes are used in tidyverse to keep code clean and prevent the defining of a lot of unnecessary intermediate variables. One of the main goals of this syntax is to keep a lot of white space in your code to help make it as readable as possible for anyone reading through your code.

Pipes will get more complex as we go through the lecture but first lets start of with something simple to start to see what they do. First, lets define a subset of our data that reflects only individuals eligible for medicare. Last lecture, we used the following syntax:

```
# Subset to only those at medicare age using our binary variable
medicareData <- randomData[which(randomData$MedicareAge == T),]
head(medicareData)
```

```
##    SubjectID Systolic.BP Diastolic.BP Age Male BiologicalSex MedicareAge
## 7          7    144.5196     66.34599  69    0        Female        TRUE
## 8          8    151.8032     51.76280  67    0        Female        TRUE
## 23        23    123.0903     61.61668  67    0        Female        TRUE
## 28        28    151.7242     73.91189  66    0        Female        TRUE
## 33        33    139.8668     84.28383  66    0        Female        TRUE
## 40        40    133.1999     61.29819  67    1          Male        TRUE
```

```
# Subset to only those at medicare age using a continuous variable
medicareData <- randomData[randomData$Age >= 65,]
head(medicareData)
```

```
##    SubjectID Systolic.BP Diastolic.BP Age Male BiologicalSex MedicareAge
## 7          7    144.5196     66.34599  69    0        Female        TRUE
## 8          8    151.8032     51.76280  67    0        Female        TRUE
## 23        23    123.0903     61.61668  67    0        Female        TRUE
## 28        28    151.7242     73.91189  66    0        Female        TRUE
## 33        33    139.8668     84.28383  66    0        Female        TRUE
## 40        40    133.1999     61.29819  67    1          Male        TRUE
```

Now, we can generate the same data set using a pipe and the filter function in dplyr.

```
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
```

```r
# Subset without pipe
medicareData <- filter(randomData,Age >= 65)

head(medicareData)
```

```
##   SubjectID Systolic.BP Diastolic.BP Age Male BiologicalSex MedicareAge
## 1         7    144.5196     66.34599  69    0        Female        TRUE
## 2         8    151.8032     51.76280  67    0        Female        TRUE
## 3        23    123.0903     61.61668  67    0        Female        TRUE
## 4        28    151.7242     73.91189  66    0        Female        TRUE
## 5        33    139.8668     84.28383  66    0        Female        TRUE
## 6        40    133.1999     61.29819  67    1          Male        TRUE
```

```r
# Subset with a pipe
medicareData <- randomData %>%
  filter(Age >= 65)

head(medicareData)
```

```
##   SubjectID Systolic.BP Diastolic.BP Age Male BiologicalSex MedicareAge
## 1         7    144.5196     66.34599  69    0        Female        TRUE
## 2         8    151.8032     51.76280  67    0        Female        TRUE
## 3        23    123.0903     61.61668  67    0        Female        TRUE
## 4        28    151.7242     73.91189  66    0        Female        TRUE
## 5        33    139.8668     84.28383  66    0        Female        TRUE
## 6        40    133.1999     61.29819  67    1          Male        TRUE
```

Based on the the use of the filter function above, can you describe the syntax of how a pipe works?

Pipes might not seem too useful when we are only providing it a single function, but what if we want it to work through multiple steps?

```r
medicareData <- randomData %>%
  dplyr::filter(Age >= 65) %>%
  dplyr::select(SubjectID,Systolic.BP,Diastolic.BP,BiologicalSex,Age)

head(medicareData)
```

```
##   SubjectID Systolic.BP Diastolic.BP BiologicalSex Age
## 1         7    144.5196     66.34599        Female  69
## 2         8    151.8032     51.76280        Female  67
## 3        23    123.0903     61.61668        Female  67
## 4        28    151.7242     73.91189        Female  66
## 5        33    139.8668     84.28383        Female  66
## 6        40    133.1999     61.29819          Male  67
```

What is the select function doing?

```r
medicareData <- randomData %>%
  filter(Age >= 65) %>%
  select(SubjectID,Systolic.BP,Diastolic.BP,BiologicalSex,Age) %>%
  mutate(MedicareID = row_number()) %>%
```

```
  mutate(BP.Diff = Systolic.BP - Diastolic.BP)

head(medicareData)
```

```
##   SubjectID Systolic.BP Diastolic.BP BiologicalSex Age MedicareID   BP.Diff
## 1         7    144.5196     66.34599        Female  69          1  78.17358
## 2         8    151.8032     51.76280        Female  67          2 100.04039
## 3        23    123.0903     61.61668        Female  67          3  61.47363
## 4        28    151.7242     73.91189        Female  66          4  77.81236
## 5        33    139.8668     84.28383        Female  66          5  55.58301
## 6        40    133.1999     61.29819          Male  67          6  71.90172
```

Based on these examples, what is the mutate function doing?

Wide -> Long Data in *tidyverse*

Last class, we moved from a wide to a long data frame using the *melt* function in *reshape2*.

```
# Melt the data frame into a long form
longData1 <- reshape2::melt(randomData[,c('SubjectID','Systolic.BP','Diastolic.BP','Age','BiologicalSex
                            id.vars = c('SubjectID','Age','BiologicalSex'),value.name = 'BP',
                            variable.name = 'BP.Type')

head(longData1)
```

```
##   SubjectID Age BiologicalSex     BP.Type        BP
## 1         1  52        Female Systolic.BP 112.28054
## 2         2  56        Female Systolic.BP 129.09478
## 3         3  25          Male Systolic.BP 104.54879
## 4         4  41        Female Systolic.BP 124.65374
## 5         5  41        Female Systolic.BP  90.69937
## 6         6  31        Female Systolic.BP 125.59120
```

In *dplyr*, we will use the the *gather* function or the *pivot_longer*.

```
longData2 <- randomData %>%
  tidyr::gather(Systolic.BP,Diastolic.BP,key = BP.Type, value = BP)
head(longData2)
```

```
##   SubjectID Age Male BiologicalSex MedicareAge     BP.Type        BP
## 1         1  52    0        Female       FALSE Systolic.BP 112.28054
## 2         2  56    0        Female       FALSE Systolic.BP 129.09478
## 3         3  25    1          Male       FALSE Systolic.BP 104.54879
## 4         4  41    0        Female       FALSE Systolic.BP 124.65374
## 5         5  41    0        Female       FALSE Systolic.BP  90.69937
## 6         6  31    0        Female       FALSE Systolic.BP 125.59120
```

```
longData <- randomData %>%
  tidyr::pivot_longer(cols = c(Systolic.BP,Diastolic.BP),names_to = 'BP.Type', values_to = 'BP')
head(longData)
```

```
## # A tibble: 6 x 7
##   SubjectID   Age  Male BiologicalSex MedicareAge BP.Type         BP
##       <int> <dbl> <int> <fct>         <lgl>       <chr>        <dbl>
## 1         1    52     0 Female        FALSE       Systolic.BP  112.
## 2         1    52     0 Female        FALSE       Diastolic.BP  75.5
## 3         2    56     0 Female        FALSE       Systolic.BP  129.
## 4         2    56     0 Female        FALSE       Diastolic.BP  60.0
## 5         3    25     1 Male          FALSE       Systolic.BP  105.
## 6         3    25     1 Male          FALSE       Diastolic.BP  74.5
```

What is different about the syntax we used here vs. what we used in the last class?

We can also use some more pipes to clean this up even more:

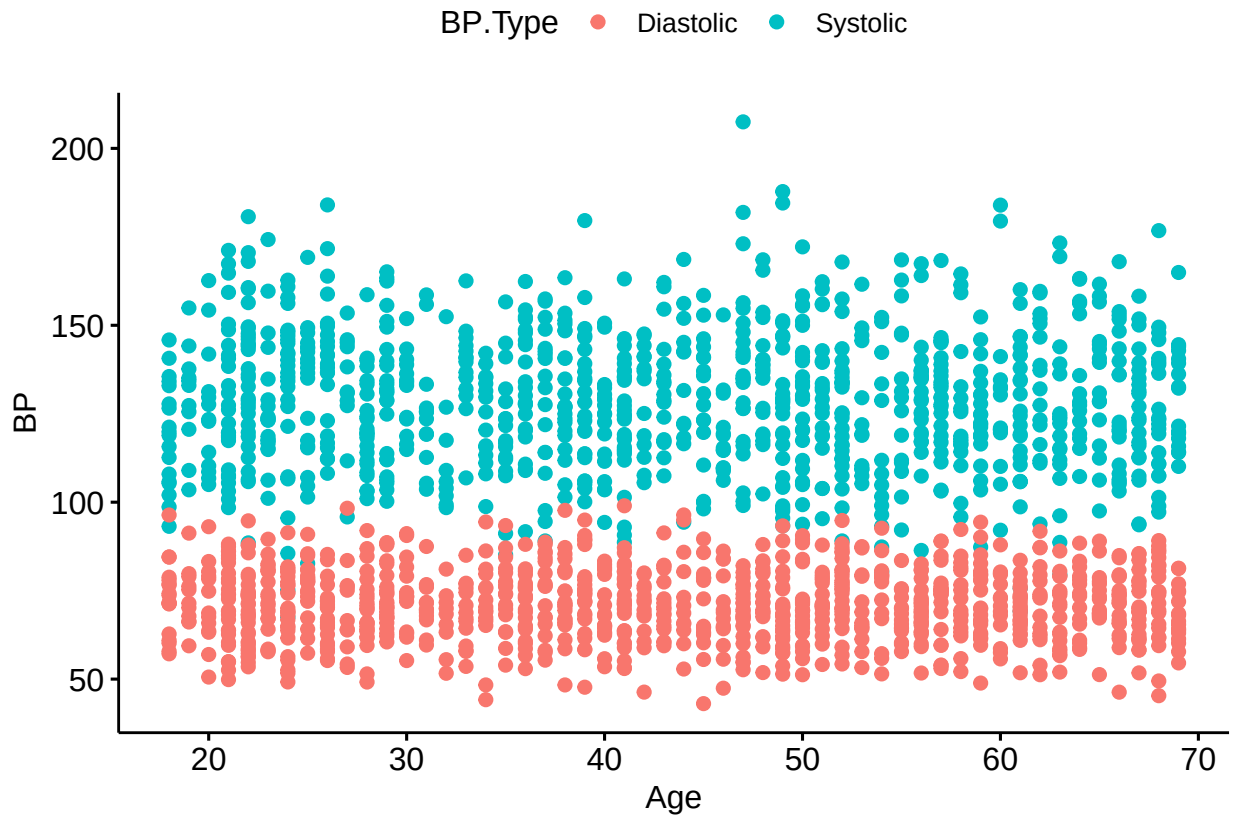
```
longData <- randomData %>%
  # Convert to long format
  tidyr::gather(key = BP.Type, value = BP,c('Systolic.BP','Diastolic.BP')) %>%
  # Split into two seperate variables
  tidyr::separate(col = BP.Type, into = c('BP.Type','Bad.ID')) %>%
  # Remove the bad ID variable
  select(-Bad.ID)

head(longData)
```

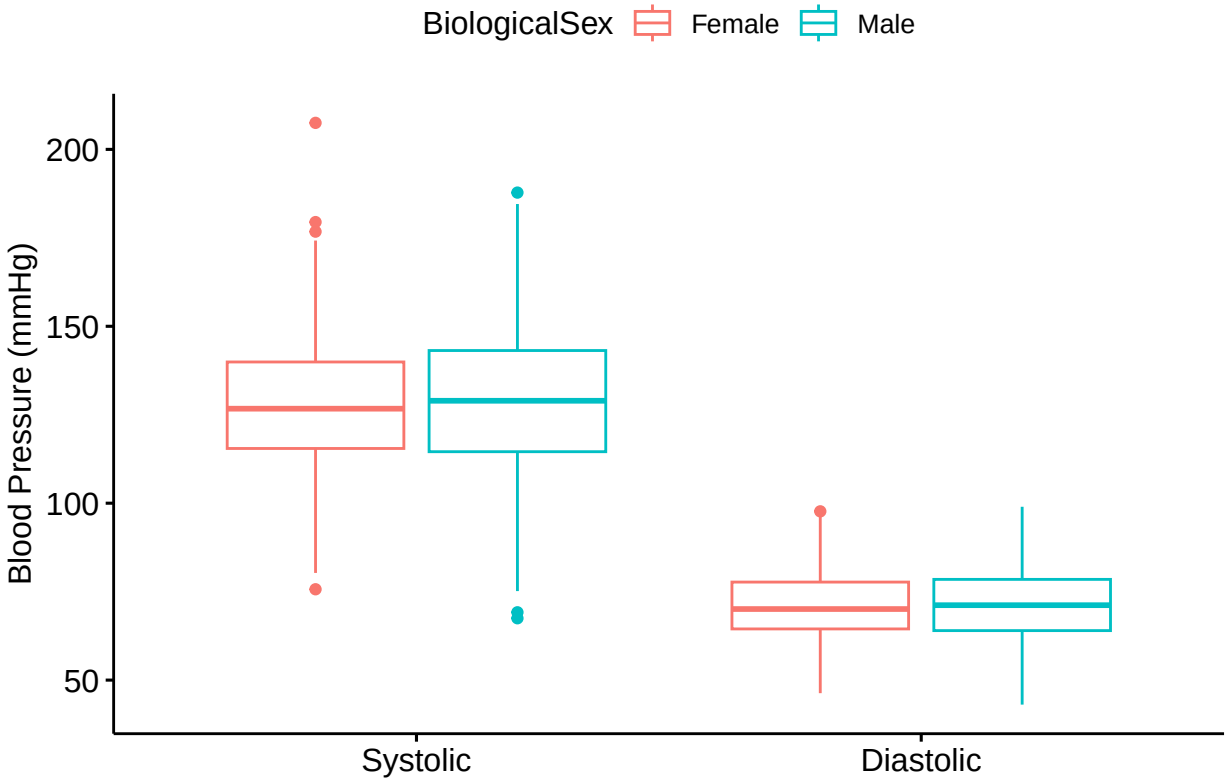
```
##   SubjectID Age Male BiologicalSex MedicareAge  BP.Type        BP
## 1         1  52    0        Female       FALSE Systolic 112.28054
## 2         2  56    0        Female       FALSE Systolic 129.09478
## 3         3  25    1          Male       FALSE Systolic 104.54879
## 4         4  41    0        Female       FALSE Systolic 124.65374
## 5         5  41    0        Female       FALSE Systolic  90.69937
## 6         6  31    0        Female       FALSE Systolic 125.59120
```

Now that our data is in a long format, we can generate plots with both measures of blood pressure in one plot.

```
# Generate a scatter plot of age by systolic blood pressure
ggpubr::ggscatter(longData,x = 'Age',y = 'BP',color = 'BP.Type')
```




```
# Generate a boxplot for diastolic blood pressure distrubition by biological sex in our original datase
ggpubr::ggboxplot(longData,x = 'BP.Type',y = 'BP',color = 'BiologicalSex',
                  ylab = 'Blood Pressure (mmHg)',xlab = '')
```



Take a minute and comment what each step of this pipe is doing.

Long -> Wide Data in *tidyverse*

We can go back to a wide format in *tidyr* using the *spread* or *pivot_wider* function.

```
# Convert using spread
wideData1 <- longData %>%
  tidyr::spread(key = BP.Type,value = BP)
head(wideData1)
```

```
##   SubjectID Age Male BiologicalSex MedicareAge Diastolic  Systolic
## 1         1  52    0        Female       FALSE  75.52894 112.28054
## 2         2  56    0        Female       FALSE  59.95778 129.09478
## 3         3  25    1          Male       FALSE  74.51568 104.54879
## 4         4  41    0        Female       FALSE  52.99577 124.65374
## 5         5  41    0        Female       FALSE  71.17388  90.69937
## 6         6  31    0        Female       FALSE  69.50961 125.59120
```

```
# Convert using pivot_wider
wideData <- longData %>%
  tidyr::pivot_wider(names_from = BP.Type,values_from = BP)
head(wideData)
```

```
## # A tibble: 6 x 7
```

```
##   SubjectID   Age  Male BiologicalSex MedicareAge Systolic Diastolic
##       <int> <dbl> <int> <fct>         <lgl>          <dbl>     <dbl>
## 1         1    52     0 Female        FALSE          112.       75.5
## 2         2    56     0 Female        FALSE          129.       60.0
## 3         3    25     1 Male          FALSE          105.       74.5
## 4         4    41     0 Female        FALSE          125.       53.0
## 5         5    41     0 Female        FALSE           90.7      71.2
## 6         6    31     0 Female        FALSE          126.       69.5
```

And, just like in *reshape2* we can also create summary tables using the *group_by* and *summarise* functions.

```
longData %>%
  tidyr::spread(key = BP.Type,value = BP) %>%
  dplyr::mutate(MedicareAge = ifelse(Age >= 65,T,F)) %>%
  dplyr::group_by(BiologicalSex,MedicareAge) %>%
  dplyr::summarise(Mean.Age = mean(Age),Mean.Sys = mean(Systolic),Mean.Dias = mean(Diastolic))
```

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

```
## # A tibble: 4 x 5
## # Groups:   BiologicalSex [2]
##   BiologicalSex MedicareAge Mean.Age Mean.Sys Mean.Dias
##   <fct>         <lgl>          <dbl>    <dbl>     <dbl>
## 1 Female        FALSE           41.0     128.      70.6
## 2 Female        TRUE            67.1     129.      71.1
## 3 Male          FALSE           41.0     129.      71.4
## 4 Male          TRUE            66.9     127.      70.3
```

In Class Activity

This is the same activity you did last class but now I want you to come up with all your solutions using *tidyverse*. Do not copy and paste any functions or syntax directly from your notes. Make sure you type everything out again to help re-enforce these new functions.

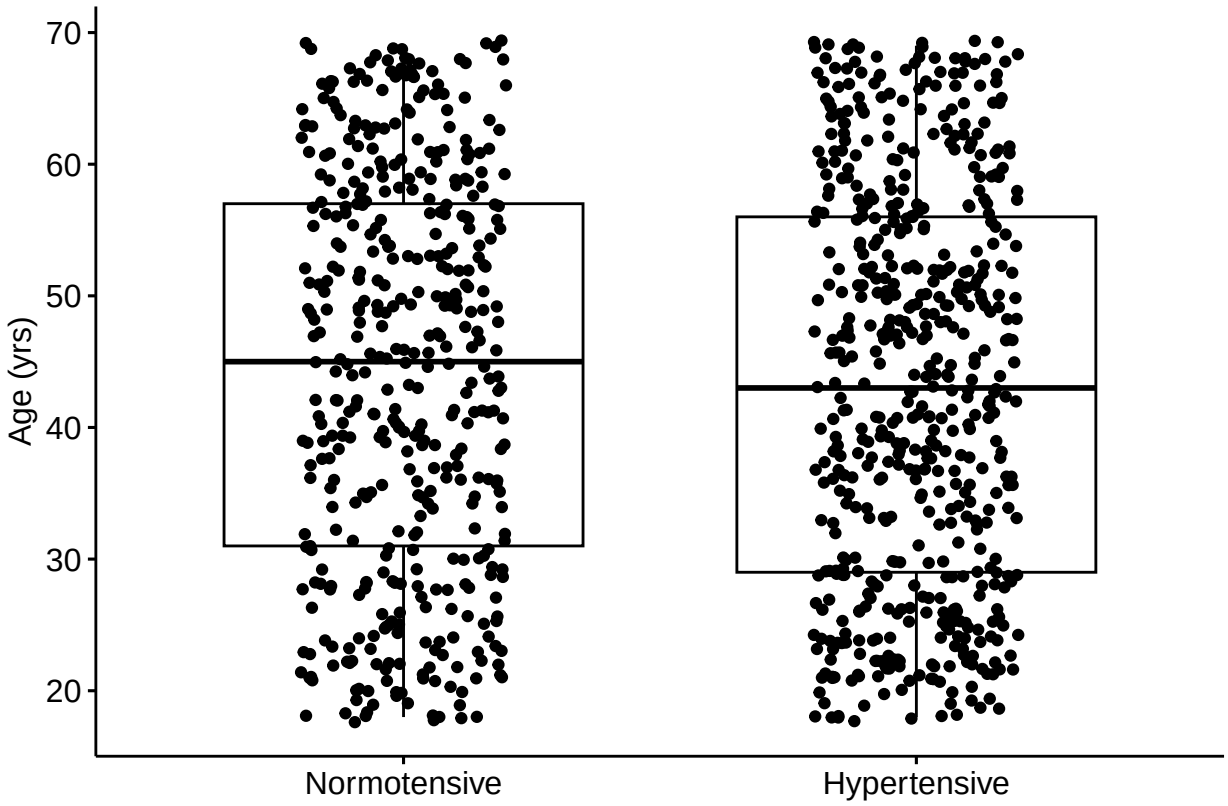
1. Define a new variable for hypertension in our original dataset (randomData). Here we will define hypertension as systolic blood pressure over 130 or a diastolic blood pressure over 80. Plot the distrubution in age for individuals with and without hypertension using boxplots. Note: this step does not require *tidyverse* but try to come up with a different method than you used in last class or than was provided in the solutions for last class.

```
# This was the solution I provided last class
randomData$Hypertension <- ifelse(randomData$Systolic.BP > 130 | randomData$Diastolic.BP > 80,
                                  'Hypertensive','Normotensive')

# This is another way to do it
randomData$Hypertension <- 'Normotensive'
randomData[randomData$Systolic.BP > 130,'Hypertension'] <- 'Hypertensive'
randomData[randomData$Diastolic.BP > 80,'Hypertension'] <- 'Hypertensive'

# Or using the mutate function:
randomData %>%
```

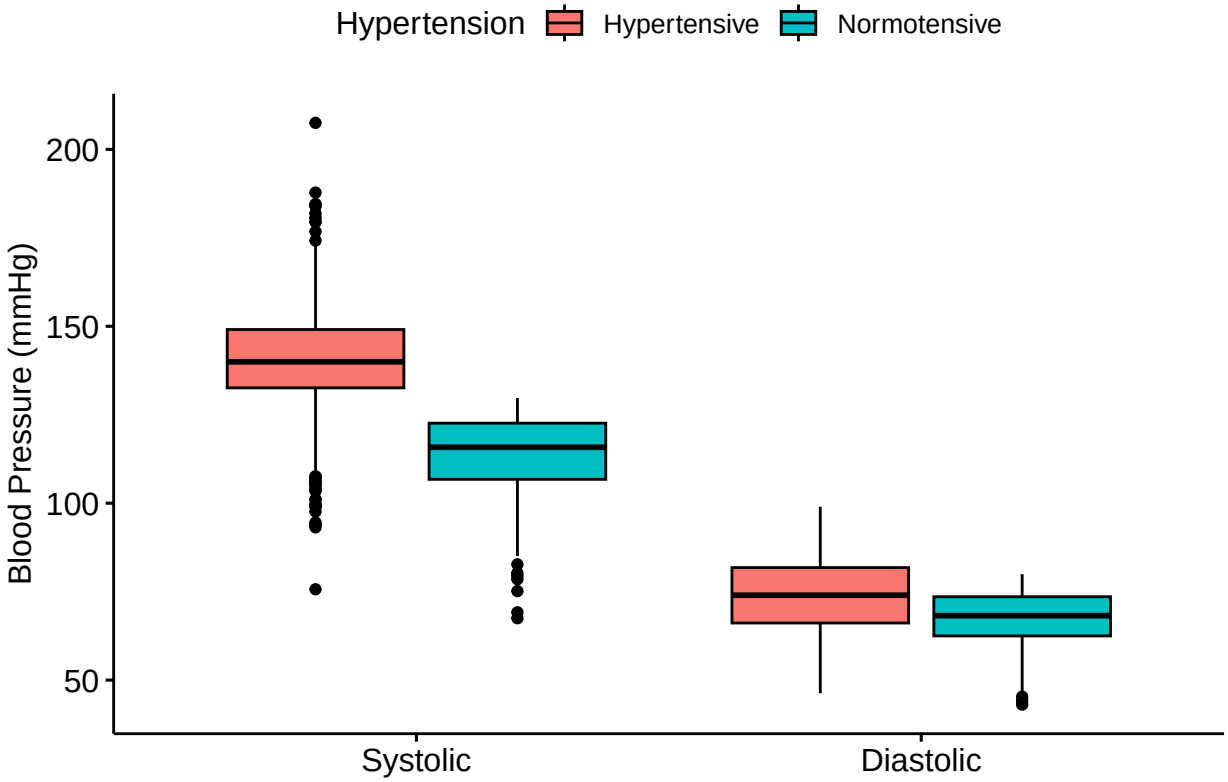
```
  mutate('Hypertension' = ifelse(Systolic.BP > 130 | Diastolic.BP > 80,
                                 'Hypertensive','Normotensive')) %>%
  ggpubr::ggboxplot(x = 'Hypertension',y = 'Age',xlab = '',ylab = 'Age (yrs)',add = 'jitter')
```



2. Use the *gather* or *pivot_longer* function to generate boxplots of the distribution of systolic and diastolic blood pressure in hypertensive vs. normotensive individuals (color should be based on hypertension status).

```
longData <- randomData %>%
  # Convert to long format
  #tidyr::gather(key = BP.Type, value = BP,c('Systolic.BP','Diastolic.BP')) %>%
  tidyr::pivot_longer(cols = c(Systolic.BP,Diastolic.BP),
                      values_to = 'BP',names_to = 'BP.Type') %>%
  # Seperate BP names into two columns, "Bad.ID" contains only "BP" for all observations
  tidyr::separate(col = BP.Type, into = c('BP.Type','Bad.ID')) %>%
  # Select only the required columns to keep
  select(SubjectID,Age,BiologicalSex,Hypertension,BP.Type,BP)

ggpubr::ggboxplot(longData,x = 'BP.Type',y = 'BP',fill = 'Hypertension',
                  xlab = '',ylab = 'Blood Pressure (mmHg)')
```



3. Use the *spread* or *pivot_wider* function to generate a table summarizing the mean age, systolic, and diastolic BP for males and females, seperately, with and without hypertension. Your table should have 4 rows. Order your table output such that it lists values for normotensive individuals first and hypertensive individuals second. We actually didn't talk about how to do this yet but look up the syntax for the *arrange* function online and try to figure out how to do it. You'll notice that the order in which you put this function in your pipe will be very important.

```
longData %>%
  #tidyr::spread(key = BP.Type,value = BP) %>%
  tidyr::pivot_wider(names_from = 'BP.Type',values_from = 'BP') %>%
  dplyr::group_by(Hypertension,BiologicalSex) %>%
  dplyr::summarise(Mean.Age = mean(Age),Mean.Sys = mean(Systolic),Mean.Dias = mean(Diastolic))  %>%
  dplyr::arrange(desc(Hypertension))
```

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

```
## # A tibble: 4 x 5
## # Groups:   Hypertension [2]
##   Hypertension BiologicalSex Mean.Age Mean.Sys Mean.Dias
##   <chr>        <fct>            <dbl>    <dbl>     <dbl>
## 1 Normotensive Female            43.6     115.      67.2
## 2 Normotensive Male              44.6     113.      67.6
## 3 Hypertensive Female            44.0     139.      73.6
## 4 Hypertensive Male              42.3     141.      74.0
```