Day 4 Lecture

Cecilia Y. Sui

01/14/2022

Day 4 Outline:

1. Packages and Help Pages
2. Introduction to dplyr
3. Introduction to ggplot2
4. Tips on Using R Markdown

1. Packages and Help Pages

You are not the only person writing your own functions with R. "Many professors, programmers, and statisticians use R to design tools that can help people analyze data." They then make these tools free for anyone to use. "To use these tools, you just have to download them. They come as preassembled collections of functions and objects called **packages**."

1.1 Installing Packages

To use an R package, you must first **install** it on your computer and then **load** it in your current R session. The easiest way to install an R package is with the **install.packages()** R function. Open R and type the following into the command line:

```r
install.packages("<package name>")
```

This will search for the specified package in the collection of packages hosted on the CRAN site. When R finds the package, it will download it into a **library** folder on your computer. R can access the package here in future R sessions without reinstalling it.

Anyone can write an R package and disseminate it as they like; however, almost all R packages are published through the CRAN website. CRAN tests each R package before publishing it. This does not eliminate every bug inside a package, but it does mean that you can trust a package on CRAN to run in the current version of R on your OS.

You can **install multiple packages** at once by linking their names with R's concatenate function, **c()**.

For example, to install the ggplot2, data.table and dplyr packages, run:

```r
install.packages(c("ggplot2", "data.table", "dplyr"))
```

If this is your first time installing a package, R will prompt you to choose an online mirror to install from. Mirrors are listed by location. Your downloads should be the quickest if you select a mirror that is closest to you. If you want to download a new package, you can pick any of the mirrors listed under USA.

1.2 Loading Packages

Installing a package does not immediately place its functions at your fingertips. It just downloads them to your computer. To use an R package, you next have to **load** it in your R session with the command:

```
library(<package name>)

# or equivalently
library("<package name>")

# For example:
library(dplyr)
```

Quotation marks are **optional** for the **library()** command, but it is **required** for the **install.packages()** command.

library() will make all of the package's functions, data sets, and help files available to you until you close your current R session. The next time you begin an R session, you will have to reload the package with library if you want to use it, but you do not need to reinstall it. You only have to install each package once on your computer. After that, a copy of the package will live in your R library. To see which packages you currently have in your R library, run:

```
library()
```

library() also shows the path to your actual R library, which is the folder that contains your R packages. You may notice many packages that you do not remember installing. This is because R automatically downloads a set of useful packages when you first install R, which are sometimes refereed to as base R or the R base packages. For example, the min(), max(), summary(), and hist() functions that you have used all come with base R.

1.3 Help Pages

There are over 1,000 functions at the core of R, and new R functions are created all of the time. This can be a lot of material to memorize and learn! Luckily, each R function comes with its own help page, which you can access by typing the function's name after a question mark.

For example, each of these commands below will open a help page. Look for the pages to appear in the Help tab of RStudio's bottom-right pane!

```
help(lm)
help("lm") # quotation marks are optional
?lm
?"lm"
help("?")
?sample
```

To access help for a function in a package that is NOT currently loaded, you can specify the function name, in addition to the name of the package. For example, you can use the following code to obtain documentation for the **rlm()** function in the MASS package.

```
help(rlm, package = "MASS")
help(package="MASS")
?mean
```

Help pages contain useful information about what each function does. These help pages also serve as code documentation, so reading them can be bittersweet. They often seem to be written for people who already understand the function and do not need help.

Don't let this bother you! You can gain a lot from a help page by scanning it for information that makes sense and glossing over the rest. This technique will inevitably bring you to the most helpful part of each help page: **the bottom**. Here, almost every help page includes some example code that puts the function in action. Running this code is a great way to learn by example.

Each help page is divided into sections. Which sections appear can vary from help page to help page, but you can usually expect to find these useful topics:

- Description - A short summary of what the function does.
- Usage - An example of how you would type the function. Each argument of the function will appear in the order R expects you to supply it (if you don't use argument names).
- Arguments - A list of each argument the function takes, what type of information R expects you to supply for the argument, and what the function will do with the information.
- Details - A more in-depth description of the function and how it operates. The details section also gives the function author a chance to alert you to anything you might want to know when using the function.
- Value - A description of what the function returns when you run it.
- See Also - A short list of related R functions.
- References - Papers or Books that published this R package.
- Examples - Example code that uses the function and is guaranteed to work. The examples section of a help page usually demonstrates a couple different ways to use a function. This helps give you an idea of what the function is capable of.

Let's go through the parts of a help page together! First, open the help page. It will appear in the same pane in RStudio as your plots did (but in the Help tab, not the Plots tab):

```
?sample
```

Unfortunately, the **help()** function and the **?** operator are only useful if you already know the name of the function or package that you wish to use. If you would like to look up the help page for a function but have forgotten the function's name, you can search by keyword.

To do this, you can use the **help.search()** funciton or the **??** folowed by a keyword in your console. R will pull up a list of links to help pages related to the keyword. You can think of this as the help page for the help page.

For example:

```
??log
??dplyr
```

In-class exercises 5.1:

1. Install the packages **dplyr** and **ggplot2**.
2. Load the packages into your current R Session.
3. Use the help function to get a brief understanding of the two packages, and find out what functions are made available from these packages.

2. Introduction to dplyr

When working with data you must:

- Figure out what you want to do.
- Describe those tasks in the form of a computer program.
- Execute the program.

The dplyr package makes these steps fast and easy:

- By constraining your options, it helps you think about your data manipulation challenges.
- "It provides simple functions that correspond to the most common data manipulation tasks, to help you translate your thoughts into code."
- It uses efficient backends, so you spend less time waiting for the computer.

Here we will introduce you to dplyr's basic set of tools and show you how to apply them to dataframes.

```
# # If you have not installed the package dplyr,
# # please un-comment and run the following line in your console.
# install.packages("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
```

```
# We will use the same worldTFR dataset again for simplicity.
worldTFR <- read.csv("worldTFR.csv")
# View(worldTFR)
```

dplyr Verbs

dplyr aims to provide a function for each basic **verb** of data manipulation. These verbs can be organized into three categories based on the component of the dataset that they work with:

1. Rows:

- filter() chooses rows based on column values.
- slice() chooses rows based on location.
- arrange() changes the order of the rows.

2. Columns:

- select() changes whether or not a column is included.
- rename() changes the name of columns.
- mutate() changes the values of columns and creates new columns.
- relocate() changes the order of the columns.

3 Groups of rows:

- summarise() collapses a group into a single row.

The Pipe

All of the **dplyr** functions take a **dataframe** as the first argument. Rather than forcing the user to either save intermediate objects or nest functions, dplyr provides the **%>%** operator. For example, x %>% f(y) turns into f(x, y), so the result from one step is then "piped" into the next step. You can use the pipe to rewrite multiple operations that you can read left-to-right, top-to-bottom (reading the pipe operator as "then").

2.1 Filter rows with filter()

filter() allows you to select a subset of rows in a data frame. Like all single verbs, the first argument is the dataframe (or sometimes called the tibble). The second and subsequent arguments refer to **variables within that dataframe**. The function will select rows where the expression is evaluated to TRUE.

For example, we can select all rows with STATE as Missouri:

```
# check structure of your df first
filter(worldTFR, Year == "1950")
```

```
##                               Country Uncode Year    TFR InfMRateCME InfMRateUN
## 1                         Afghanistan      4 1950 7.4500          NA    304.594
## 2                             Albania      8 1950 5.9138          NA    164.781
...
```

```
# or equivalently using the pipe operator
# This is a cleaner approach
worldTFR %>% filter(Year == "1950")
```

```
##                               Country Uncode Year    TFR InfMRateCME InfMRateUN
## 1                         Afghanistan      4 1950 7.4500          NA    304.594
## 2                             Albania      8 1950 5.9138          NA    164.781
...
```

This is roughly equivalent to this base R code:

```
worldTFR[worldTFR$Year == "1950", ] # Don't forget the comma!
```

```
##                                 Country Uncode Year    TFR InfMRateCME
## 1                           Afghanistan      4 1950 7.4500          NA
## 67                              Albania      8 1950 5.9138          NA
...
```

You can always add more conditions to your expression using commas.

```
worldTFR %>% filter(Year == "1950", TFR >= 7.5, Uncode > 500)
```

```
##       Country Uncode Year    TFR InfMRateCME InfMRateUN U5MRateCME U5MRateUN
## 1 Philippines    608 1950 7.5710          NA    106.969         NA  152.8444
## 2      Rwanda    646 1950 7.8500          NA    169.113         NA  285.1094
## 3       Samoa    882 1950 7.6297          NA    117.558         NA  177.8467
## 4     Vanuatu    548 1950 7.8500          NA    188.977         NA  283.1024
##   LifeExpB MtoFbirth   MtoF04  Pop1564 Pop1564Female    GDPpc GDPpcGrowth
## 1   53.776      1.06 1.074300 52.82024      53.32406 1423.896          NA
## 2   38.529      1.01 1.004753 53.97987      54.76619       NA          NA
## 3   43.359      1.08 1.089769 50.00000      50.00000       NA          NA
## 4   38.947      1.07 1.102738 54.62708      51.53043       NA          NA
##   Yschooling YschoolF1549 GenrollPrim ChildBearing CountryCode
## 1       2.21    2.3009839          NA       30.703         PHL
## 2       0.32    0.2015992          NA       31.567         RWA
## 3         NA           NA          NA       30.270         WSM
## 4         NA           NA          NA       29.056         VUT
```

This function also offers an alternative way to filter NA values. For example:

```
worldTFR %>% filter(!is.na(GDPpc))
```

```
##                       Country Uncode Year     TFR InfMRateCME InfMRateUN
## 1                     Albania      8 1970 4.91000          NA    76.9580
## 2                     Albania      8 1971 4.77500          NA    73.1626
...
```

```
# You can also add more conditions here:
worldTFR %>% filter(!is.na(GDPpc), TFR > 5)
```

```
##                             Country Uncode Year     TFR InfMRateCME InfMRateUN
## 1                           Algeria     12 1960 7.52400       148.2   153.5210
## 2                           Algeria     12 1961 7.57300       148.1   151.3858
...
```

2.2 Arrange rows with arrange()

arrange() works similarly to filter() except that instead of filtering or selecting rows, it **reorders** them. It takes a data frame, and a set of column names (or more complicated expressions) to order by. If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns.

```
worldTFR.na <- worldTFR %>% filter(complete.cases(.))

# equivalently to: na.omit(worldTFR)
summary(worldTFR.na)
```

```
##    Country              Uncode           Year           TFR
##  Length:4455        Min.   :  8.0   Min.   :1970   Min.   :1.076
##  Class :character   1st Qu.:208.0   1st Qu.:1981   1st Qu.:1.931
...
```

```
worldTFR.na %>% arrange(Year, TFR)
```

```
##                       Country Uncode Year     TFR InfMRateCME InfMRateUN
## 1                  Luxembourg    442 1970 2.19200        19.3    20.9080
## 2                     Uruguay    858 1970 2.90200        48.6    47.1020
...
```

Use desc() to order a column in descending order:

```
# desc in Year, but still ascending in TFR
worldTFR.na %>% arrange(desc(Year), TFR)
```

```
##                       Country Uncode Year     TFR InfMRateCME InfMRateUN
## 1                 Korea, Rep.    410 2010 1.22600         3.5     3.5070
## 2                     Hungary    348 2010 1.25000         5.7     5.6570
...
```

2.3 Choose rows using their position with slice()

slice() lets you index rows by their (integer) locations. It allows you to select, remove, and duplicate rows.

We can get characters from row numbers 5 through 10.

```r
worldTFR.na %>% slice(5:10)
```

```
##   Country Uncode Year   TFR InfMRateCME InfMRateUN U5MRateCME U5MRateUN
## 1 Albania      8 1982 3.452        56.1    45.3844       67.8  55.61363
## 2 Albania      8 1983 3.383        52.4    44.6536       62.8  54.57924
...
```

It is accompanied by a number of helpers for common use cases.

2.3.1 slice_head() and slice_tail() select the first or last rows.

Get the first three rows:

```r
worldTFR.na %>% slice_head(n = 10)
```

```
##    Country Uncode Year   TFR InfMRateCME InfMRateUN U5MRateCME U5MRateUN
## 1  Albania      8 1978 3.841        73.0    51.3000       91.1  63.73139
## 2  Albania      8 1979 3.725        68.4    49.0730       84.7  60.70690
...
```

Get the last three rows:

```r
worldTFR.na %>% slice_tail(n = 3)
```

```
##    Country Uncode Year   TFR InfMRateCME InfMRateUN U5MRateCME U5MRateUN
## 1 Zimbabwe    716 2001 4.036        63.2     65.266      105.6  93.99998
## 2 Zimbabwe    716 2002 4.018        62.7     65.752      105.1  95.33407
...
```

2.3.2 slice_sample() randomly selects rows.

Each time you run the function, it should give you a different set of rows.

```r
worldTFR.na %>% slice_sample(n = 10)
```

```
##             Country Uncode Year   TFR InfMRateCME InfMRateUN U5MRateCME
## 1            Malawi    454 1977 7.589       170.6   165.0374      287.7
## 2       Netherlands    528 2008 1.770         4.0     4.4062        4.8
...
```

Use the option **prop** to choose a certain proportion of the cases.

```r
worldTFR.na %>% slice_sample(prop = 0.3) # 30% of rows
```

```
##                       Country Uncode Year     TFR InfMRateCME InfMRateUN
## 1                    Pakistan    586 2008 3.94200        76.1    79.7222
## 2                     Albania      8 1981 3.53000        59.9    46.1152
...
```

Use **replace = TRUE** to perform a bootstrap sample.

```r
worldTFR.na %>% slice_sample(prop = 0.1, replace = FALSE)
```

```
##                      Country Uncode Year     TFR InfMRateCME InfMRateUN
## 1                  Indonesia    360 1986 3.60600        71.6    77.6468
## 2                    Bolivia     68 1998 4.24100        64.1    77.4914
...
```

2.3.3 slice_min() and slice_max() select rows with highest or lowest values of a variable.

Note that we first must choose only the values which are not NA. Remember on Day 1 we covered that R can not make comparison that involves NA values.

```
worldTFR %>%
  filter(!is.na(GDPpc)) %>% # you can have multiple pipes
  slice_max(GDPpc, n = 3)
```

```
##                Country Uncode Year   TFR InfMRateCME InfMRateUN U5MRateCME
## 1 United Arab Emirates    784 1971 6.512        65.1    74.4908       89.1
## 2 United Arab Emirates    784 1970 6.605        70.9    78.8710       98.4
...
```

```
worldTFR %>%
  filter(!is.na(Yschooling)) %>%
  slice_max(Uncode, n = 3)
```

```
##    Country Uncode Year   TFR InfMRateCME InfMRateUN U5MRateCME U5MRateUN
## 1   Zambia    894 1950 6.450          NA   159.6410         NA 269.66845
## 2   Zambia    894 1951 6.500          NA   157.3702         NA 265.83513
...
```

2.4 Select columns with select()

It is often that when you work with large datasets with lots of columns, only a few are actually of interest to you. **select()** allows you to rapidly zoom in on a useful subset.

2.4.1 Select columns by name

Select the following three columns:

```
worldTFR %>% select(CountryCode, Year, TFR)
```

Be careful when running the select function after you loaded the “MASS” package. The select() function from dplyr will clash with the same select() function from MASS, which generates an error like this:

```
library(MASS)
worldTFR %>% select(CountryCode, Year, TFR)
detach(package:MASS, unload = TRUE) # or go to packages and unselect
# Error in select(., CountryCode, Year, TFR) : unused arguments (CountryCode, Year, TFR)
```

You can fix this error by using the following code instead. This explicitly tells R to use the select() function from the dplyr package.

```
# library(MASS)
worldTFR %>% dplyr::select(CountryCode, Year, TFR)
```

```
##       CountryCode Year     TFR
## 1             AFG 1950 7.45000
## 2             AFG 1951 7.45000
...
```

2.4.2 Select all columns between two columns

Select all columns between Year and LifeExpB (inclusive):

```
worldTFR.na %>% select(Year:LifeExpB)
```

```
##      Year     TFR InfMRateCME InfMRateUN U5MRateCME  U5MRateUN LifeExpB
## 1    1978 3.84100        73.0    51.3000       91.1  63.731387  69.7542
## 2    1979 3.72500        68.4    49.0730       84.7  60.706904  70.0097
...
```

2.4.3 Select all columns except some

Select all columns except those from NAME to STATE (inclusive):

```
worldTFR.na %>% select(!(Year:LifeExpB))
```

```
##                         Country Uncode MtoFbirth    MtoF04  Pop1564 Pop1564Female
## 1                       Albania      8    1.0700 1.0653960 57.69829      57.24580
## 2                       Albania      8    1.0700 1.0673750 58.18736      57.64706
...
```

```
worldTFR.na
```

```
##                         Country Uncode Year     TFR InfMRateCME InfMRateUN
## 1                       Albania      8 1978 3.84100        73.0    51.3000
## 2                       Albania      8 1979 3.72500        68.4    49.0730
...
```

```
worldTFR.na %>% select(!c(Country, Year, TFR))
```

```
##      Uncode InfMRateCME InfMRateUN U5MRateCME  U5MRateUN LifeExpB MtoFbirth
## 1         8        73.0    51.3000       91.1  63.731387  69.7542    1.0700
## 2         8        68.4    49.0730       84.7  60.706904  70.0097    1.0700
...
```

2.5 Add new columns with mutate()

Besides selecting sets of existing columns, it's often useful to add new columns that are functions of existing columns.

This is the job of **mutate()**.

For example:

```
worldTFR.na %>% mutate(ChildBearing_r = round(ChildBearing,0))
```

```
##                         Country Uncode Year     TFR InfMRateCME InfMRateUN
## 1                       Albania      8 1978 3.84100        73.0    51.3000
## 2                       Albania      8 1979 3.72500        68.4    49.0730
...
```

## 2.6 Storing the results to a new dataframe

You can always store your results to a new dataframe.

```
# Can you explain what this code block does?
df <- worldTFR %>%
  mutate(ChildBearing_r = round(ChildBearing,0)) %>%
  filter(!is.na(GDPpc), !is.na(Yschooling), TFR > 5) %>%
  select(CountryCode, Year, TFR, GDPpc, Yschooling, ChildBearing_r)
df
```

```
##      CountryCode Year     TFR       GDPpc Yschooling ChildBearing_r
## 1            DZA 1960 7.52400   5078.3080      0.880             30
## 2            DZA 1961 7.57300   4528.4084      0.924             30
...
```

```
# View(df)
```

Or you can also make these changes **in place** by storing it to your original dataframe.

```
worldTFR <- read.csv("worldTFR.csv") %>%
  mutate(ChildBearing_r = round(ChildBearing,0)) %>%
  filter(!is.na(GDPpc), !is.na(Yschooling), TFR > 5) %>%
  select(CountryCode, Year, TFR, GDPpc, Yschooling, ChildBearing_r)
```

If you would like to learn more about dplyr, you can read through the documentation for dplyr:

[https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html]

In-class exercises 5.2:

Let's apply the tools we learned with the **worldTFR** dataset!

1. Select the first 10 rows and last 3 rows of worldTFR.

```
dim(worldTFR)[1]
```

```
## [1] 12342
```

```
worldTFR  %>% slice(c(1:10 , 12340:12342))
```

```
##        Country Uncode Year    TFR InfMRateCME InfMRateUN U5MRateCME  U5MRateUN
## 1  Afghanistan      4 1950 7.4500          NA   304.5940         NA 438.010260
## 2  Afghanistan      4 1951 7.4500          NA   299.6836         NA 431.606470
## 3  Afghanistan      4 1952 7.4500          NA   294.7732         NA 425.202680
## 4  Afghanistan      4 1953 7.4500          NA   289.8628         NA 418.798890
## 5  Afghanistan      4 1954 7.4500          NA   284.9524         NA 412.395100
## 6  Afghanistan      4 1955 7.4500          NA   280.0420         NA 405.991310
## 7  Afghanistan      4 1956 7.4500          NA   275.1316         NA 399.587520
## 8  Afghanistan      4 1957 7.4500          NA   270.2212         NA 393.183730
## 9  Afghanistan      4 1958 7.4500          NA   265.3108         NA 386.779940
## 10 Afghanistan      4 1959 7.4500          NA   260.4004         NA 380.376150
## 11      Taiwan    158 2013 1.0838          NA     4.6062         NA   5.785116
## 12      Taiwan    158 2014 1.0948          NA     4.4206         NA   5.541063
## 13      Taiwan    158 2015 1.1058          NA     4.2350         NA   5.297009
##    LifeExpB MtoFbirth    MtoF04  Pop1564 Pop1564Female    GDPpc GDPpcGrowth
## 1   26.0690     1.060 0.9523352 56.08875      54.25678       NA          NA
## 2   26.5736     1.060 0.9524428 55.82908      54.06208       NA          NA
## 3   27.0782     1.060 0.9728808 55.74039      54.08136       NA          NA
## 4   27.5828     1.060 0.9952082 55.71038      54.13613       NA          NA
## 5   28.0874     1.060 1.0122050 55.71779      54.22245       NA          NA
## 6   28.5920     1.060 1.0222970 55.68319      54.21412       NA          NA
## 7   29.0966     1.060 1.0463900 55.48940      54.15216       NA          NA
## 8   29.6012     1.060 1.0493490 55.25483      53.91750       NA          NA
## 9   30.1058     1.060 1.0410410 55.11521      53.78352       NA          NA
## 10  30.6104     1.060 1.0313490 55.02604      53.76849       NA          NA
## 11  78.8234     1.094 1.0903848 74.01081      73.96104 42371.11  0.03437965
## 12  79.0402     1.092 1.0895734 74.06428      73.95233 44327.51  0.04513868
## 13  79.2570     1.090 1.0884003 73.97292      73.79027       NA          NA
##    Yschooling YschoolF1549 GenrollPrim ChildBearing CountryCode
## 1       0.270   0.08679564          NA      29.8350         AFG
## 2       0.278   0.08758149          NA      29.8350         AFG
## 3       0.286   0.08836733          NA      29.8350         AFG
## 4        0.294   0.08915317          NA      29.8350          AFG
## 5        0.302   0.08993901          NA      29.8350          AFG
## 6        0.310   0.09072485          NA      29.8350          AFG
## 7        0.322   0.09407387          NA      29.8350          AFG
## 8        0.334   0.09742289          NA      29.8350          AFG
## 9        0.346   0.10077190          NA      29.8350          AFG
## 10       0.358   0.10412092          NA      29.8350          AFG
## 11          NA           NA       98.62      30.3196          TWN
## 12          NA           NA       98.46      30.5738          TWN
## 13          NA           NA       98.36      30.8280          TWN
```

```
# View(df4)
```

2. Select the 100th to 110th rows.
3. Select all rows where Year is 1951 and TFR is greater than 6.
4. Order your result from Q2 by TFR in descending order.
5. Randomly sample 100 rows with replacement and order the rows by Year in ascending order.
6. Select columns: Country, Year, TFR, and store them into a new dataframe.
7. Select all columns except CountryCode.
8. Create a new column called **ChildBearing_sd** where you subtract the mean of the column from its original value and divide the result by the standard deviation of the column.
9. Create a new column called **GDPpc_sd** where you do a similar operation as in Q8.

3. Introduction to ggplot2

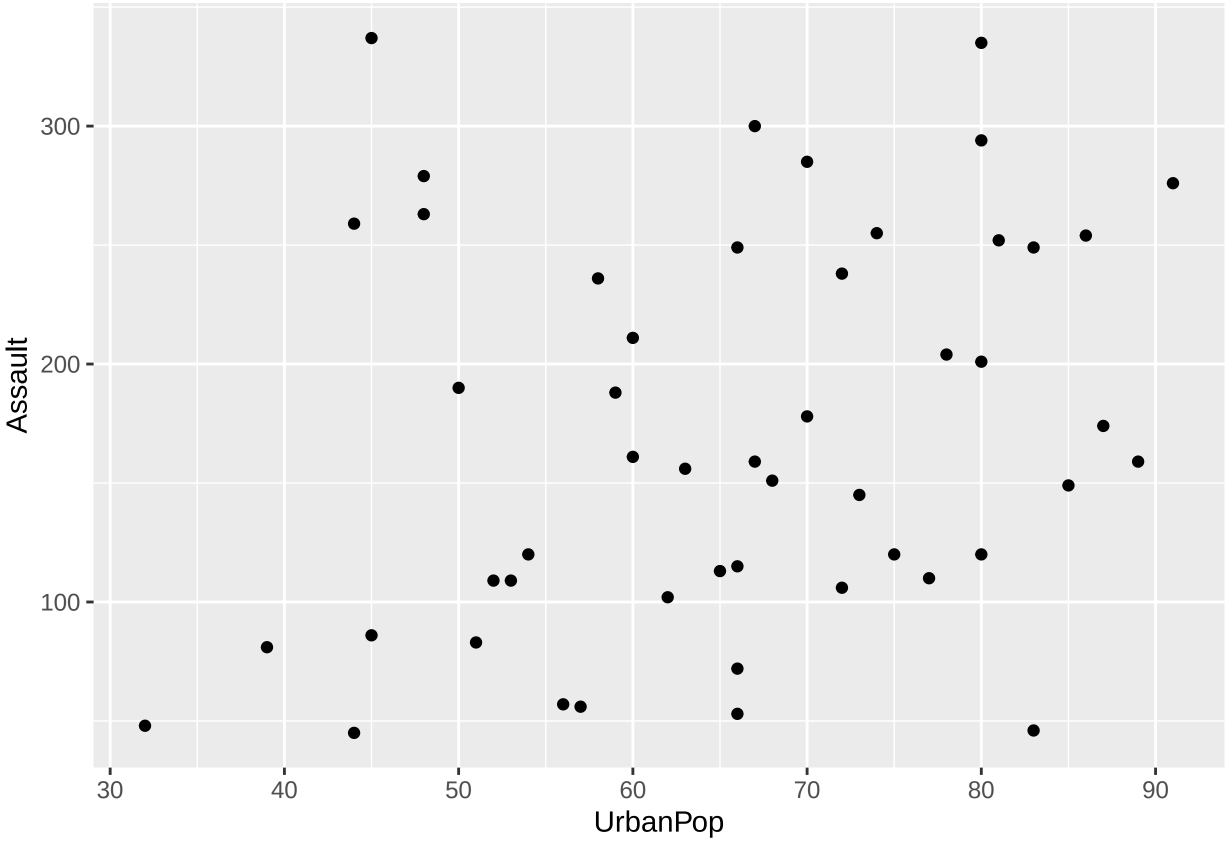
We need to first install and load the package.

```
# install.packages("ggplot2")
library(ggplot2)
```

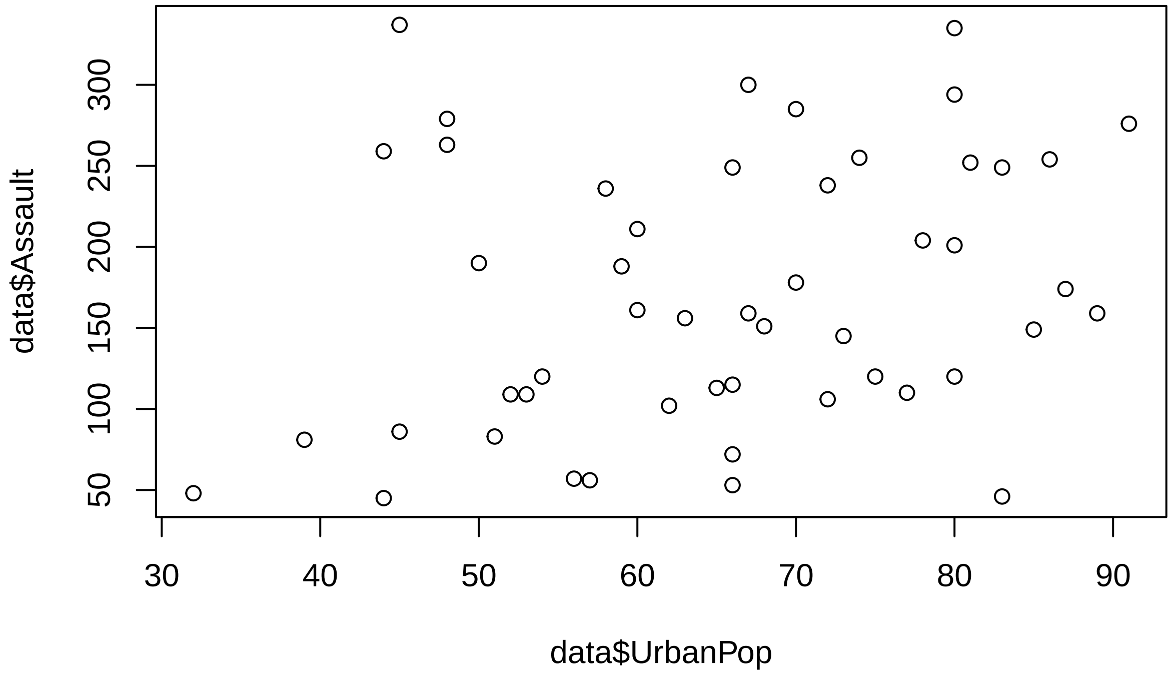
3.1 How to Make a Simple Scatterplot

```
data = USArrests # built-in dataset
?USArrests
# View(USArrests)

ggplot(data, aes(x = UrbanPop, y = Assault)) + geom_point()
```



```
# sort of equivalent to doing this with the baseplot
plot(data$UrbanPop, data$Assault)
```



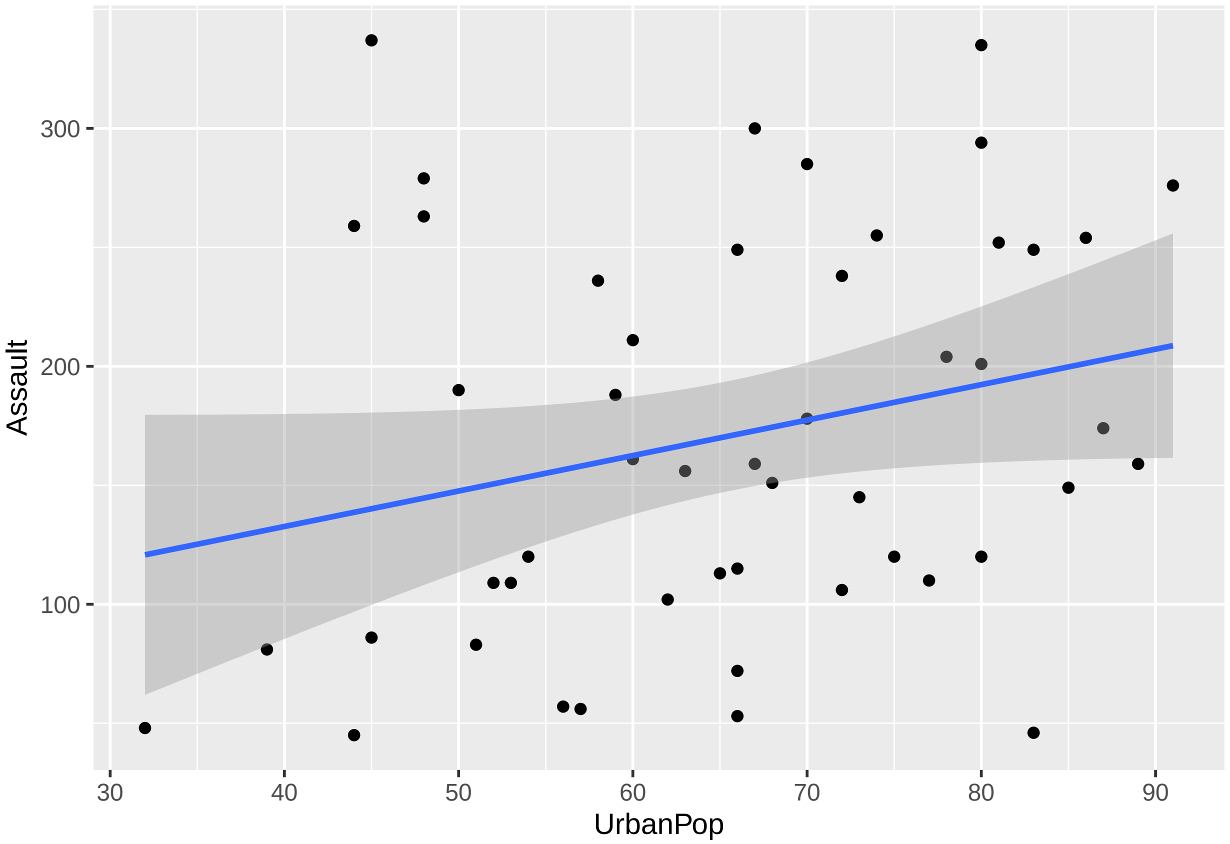
```
# ?ggplot
```

We got a basic scatterplot, where each point represents a US state. However, it lacks some basic components such as the plot title, meaningful axis labels, etc.

Like geom_point(), there are many such geom layers to use for visualization. For now, let's just add a smoothing layer using **geom_smooth(method = 'lm')**. Since the method is set as lm (short for linear model), it draws the line of best fit. The line of best fit is in blue by default. The shaded area is the confidence intervals.

```
ggplot(data, aes(x=UrbanPop, y=Assault)) +
  geom_point() +
  geom_smooth(method="lm", se = TRUE)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



3.2 Adjusting the X and Y Axis Limits

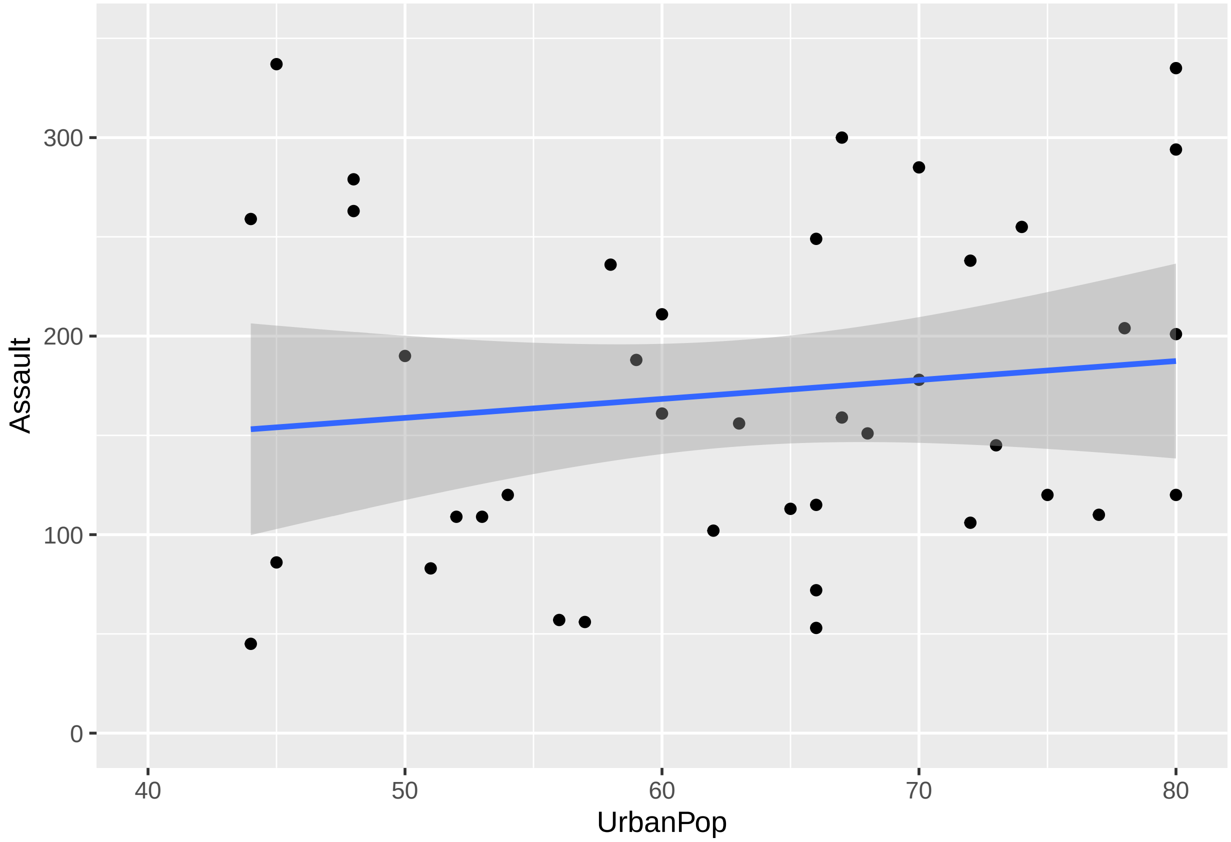
We can delete the points outside our range. This will change the lines of best fit or smoothing lines as compared to the original data. This can be done by **xlim()** and **ylim()**. You can pass a numeric vector of length 2 (with min and max values) or just the max and min values itself.

```
ggplot(data, aes(x=UrbanPop, y=Assault)) +
  geom_point() +
  geom_smooth(method="lm")  +
  xlim(c(40, 80)) + ylim(c(0, 350))   # deletes points
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning: Removed 10 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```

```
## Warning: Removed 10 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



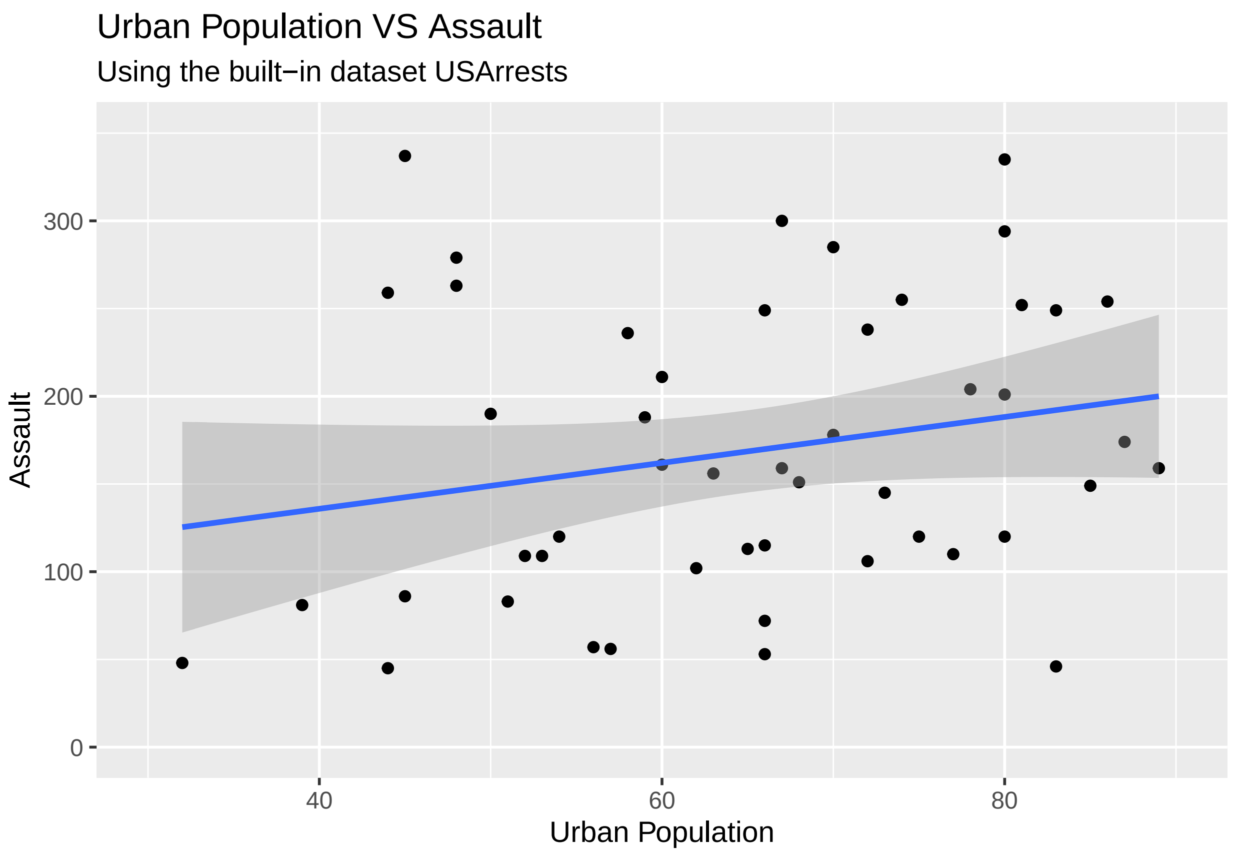
3.3 Change the Title and Axis Labels

```
ggplot(data, aes(x=UrbanPop, y=Assault)) +
  geom_point() +
  geom_smooth(method="lm") +
  xlim(c(30, 90)) + ylim(c(0, 350)) +
  labs(title="Urban Population VS Assault",
         subtitle="Using the built-in dataset USArrests",
       y="Assault", x="Urban Population") # adding labels here
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_smooth()`).
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_point()`).
```



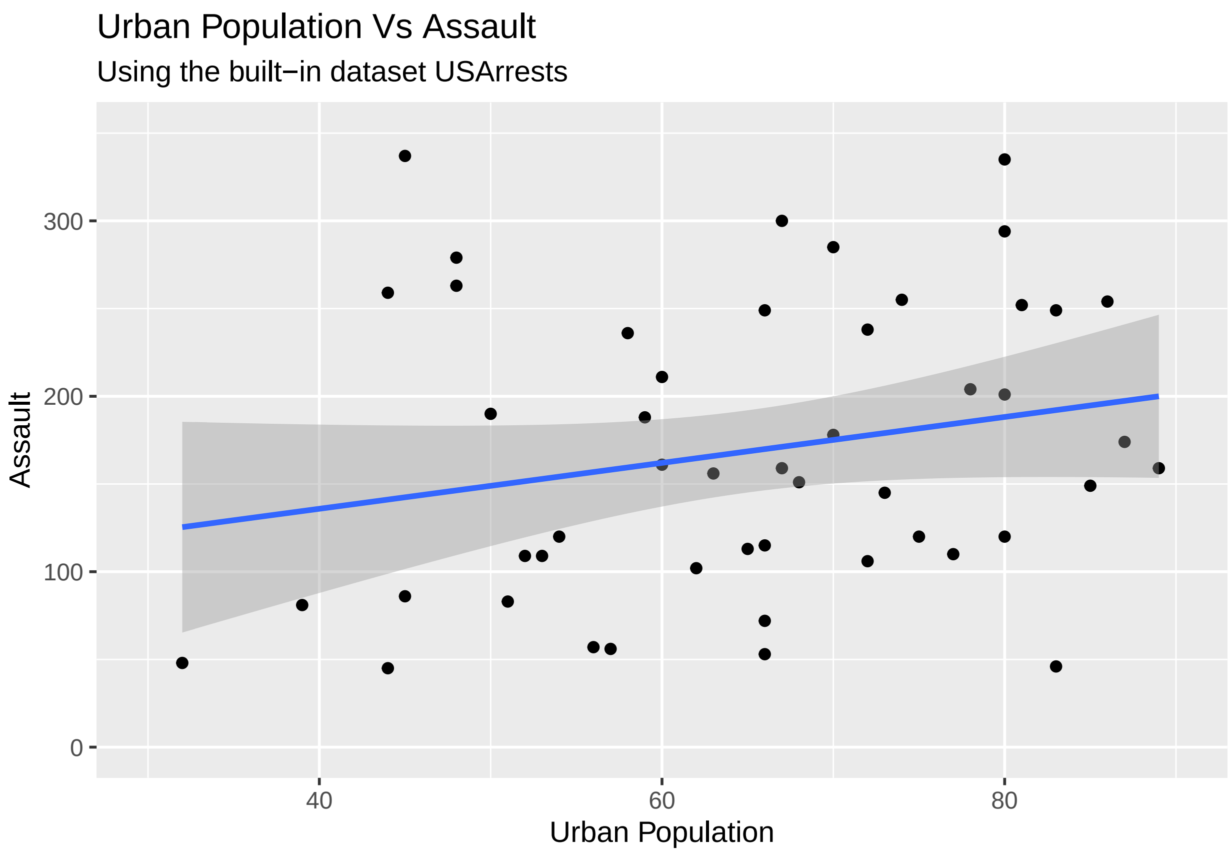
You can also get the same plot by adding the labels using an alternative way:

```r
ggplot(data, aes(x=UrbanPop, y=Assault)) +
  geom_point() +
  geom_smooth(method="lm") +
  xlim(c(30, 90)) + ylim(c(0, 350)) +
  ggtitle("Urban Population Vs Assault",
          subtitle="Using the built-in dataset USArrests") +
  xlab("Urban Population") +
  ylab("Assault")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_smooth()`).
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_point()`).
```



3.4 Change the Color and Size of Points

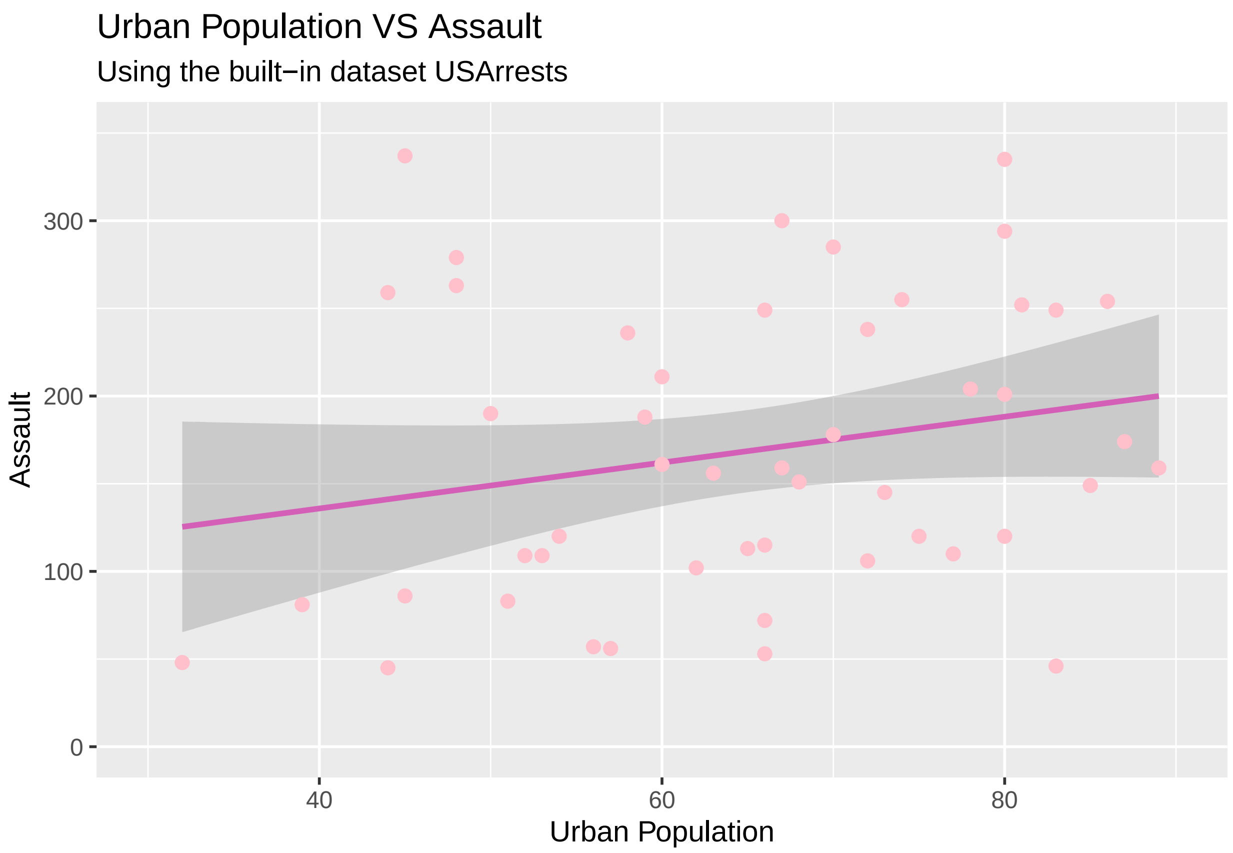
We can change the aesthetics of a geom layer by modifying the respective **geoms**. Let's change the color of the points and the line to a static value. R has a limited number of colors to use by their name, such as red, blue, yellow, etc. But you can always use html color codes to make your plots more colorful!

```
ggplot(data, aes(x=UrbanPop, y=Assault)) +
  geom_smooth(method="lm",  col="#D35FB7") + # change the color of line
  xlim(c(30, 90)) + ylim(c(0, 350)) +
  labs(title="Urban Population VS Assault",
         subtitle="Using the built-in dataset USArrests",
      y="Assault", x="Urban Population") +
  geom_point(col="pink", size=2) # Set static color and size for points
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_smooth()`).
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_point()`).
```



ggplot2 is a very powerful package for visualizing your data.

If you would like to learn more about what it is capable to do, check out this website:

http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html

3.5 Saving your plot

ggsave() is a convenient function for saving a plot.

It defaults to saving the last plot that you displayed, using the size of the current graphics device. It also guesses the type of graphics device from the extension.

It has many different arguments that you can customize. We will not cover how to use each argument, but you can learn more about them in the documentation for ggplot.

```r
# The following code chunk is not executable.
ggsave(
  filename,
  plot = last_plot(),
  device = NULL,
  path = NULL,
  scale = 1,
  width = NA,
  height = NA,
  units = c("in", "cm", "mm", "px"),
  dpi = 300,
  limitsize = TRUE,
  bg = NULL,
  ...
)
```

For example, you can save the plot above like this:

```
ggplot(data, aes(x=UrbanPop, y=Assault)) +
  geom_point() +
  geom_smooth(method="lm",  col="skyblue") +
  xlim(c(30, 90)) + ylim(c(0, 350)) +
  labs(title="Urban Population VS Assault",
         subtitle="Using the built-in dataset USArrests",
       y="Assault", x="Urban Population") +
  geom_point(col="pink", size=3)

ggsave("myplot.png")
```

When having multiple plots, you can also save your plots to objects and then export to images. For example:

```
g1 <- ggplot(data, aes(x=UrbanPop, y=Assault)) +
          geom_point() +
          geom_smooth(method="lm",  col="skyblue") +
          xlim(c(30, 90)) + ylim(c(0, 350)) +
          labs(title="Urban Population VS Assault",
                 subtitle="Using the built-in dataset USArrests",
               y="Assault", x="Urban Population") +
          geom_point(col="pink", size=3)

ggsave("myplot.pdf", plot = g1)
```

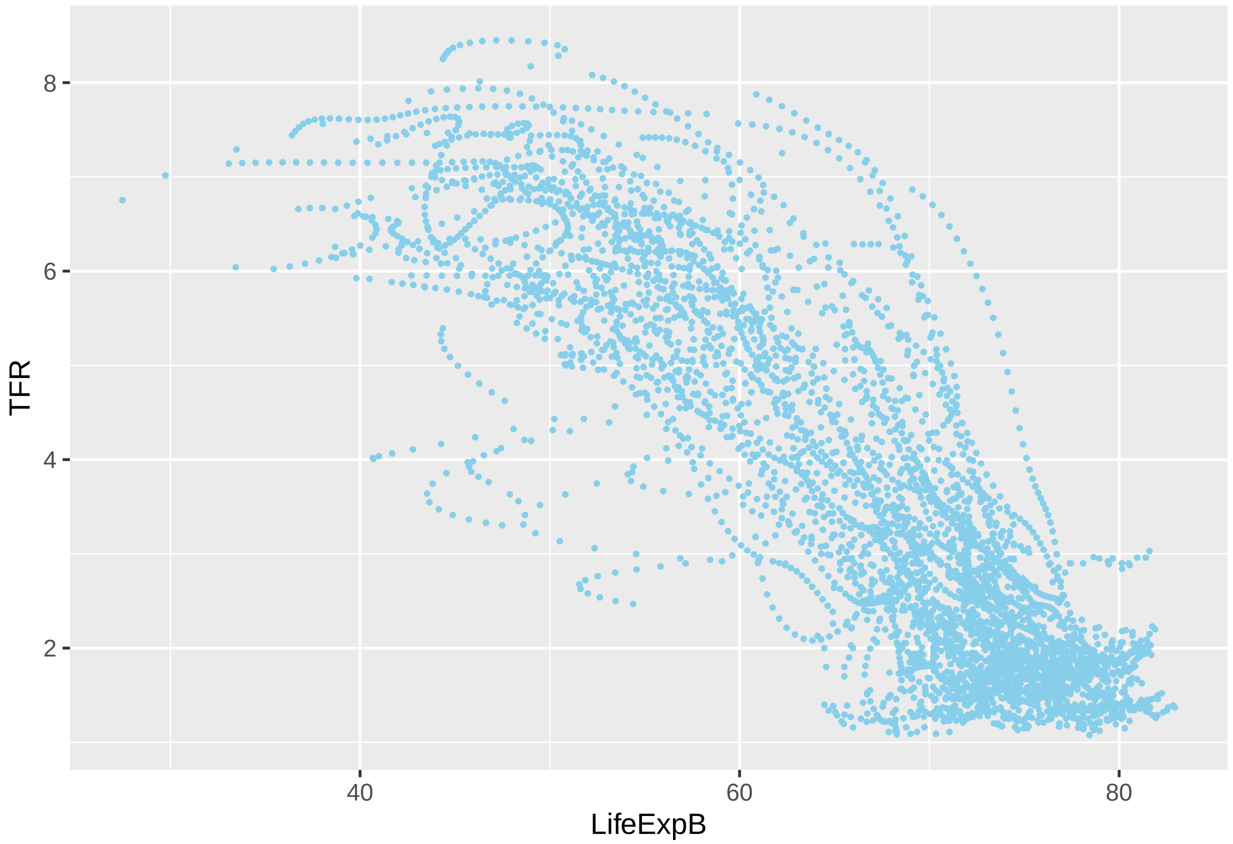
In-class exercises 5.3:

1. Omit all NA's from worldTFR.

```
df <- na.omit(worldTFR)
```

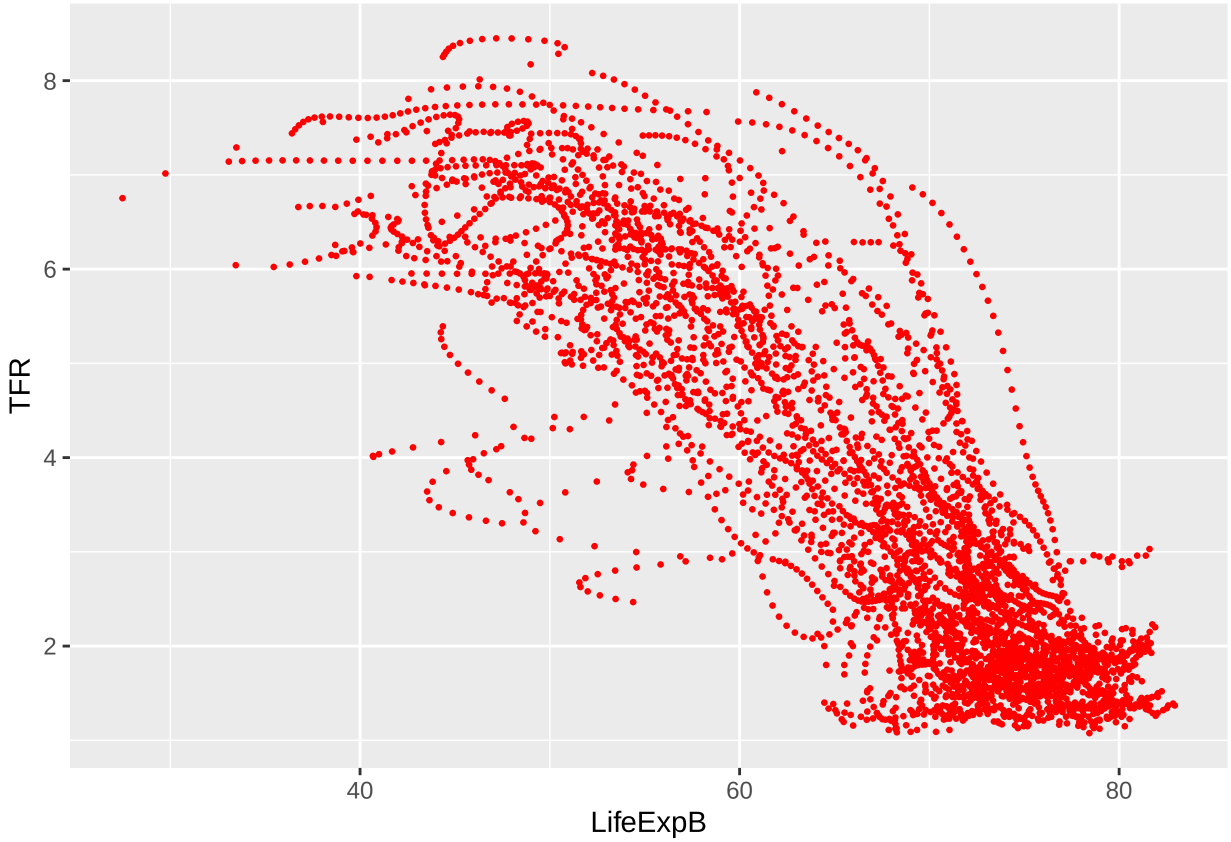
2. Make a plot where the horizontal axis is LifeExpB and vertical axis is TFR.

```
ggplot(df, aes(x = LifeExpB, y = TFR)) + geom_point(col = "skyblue", size = 0.5)
```



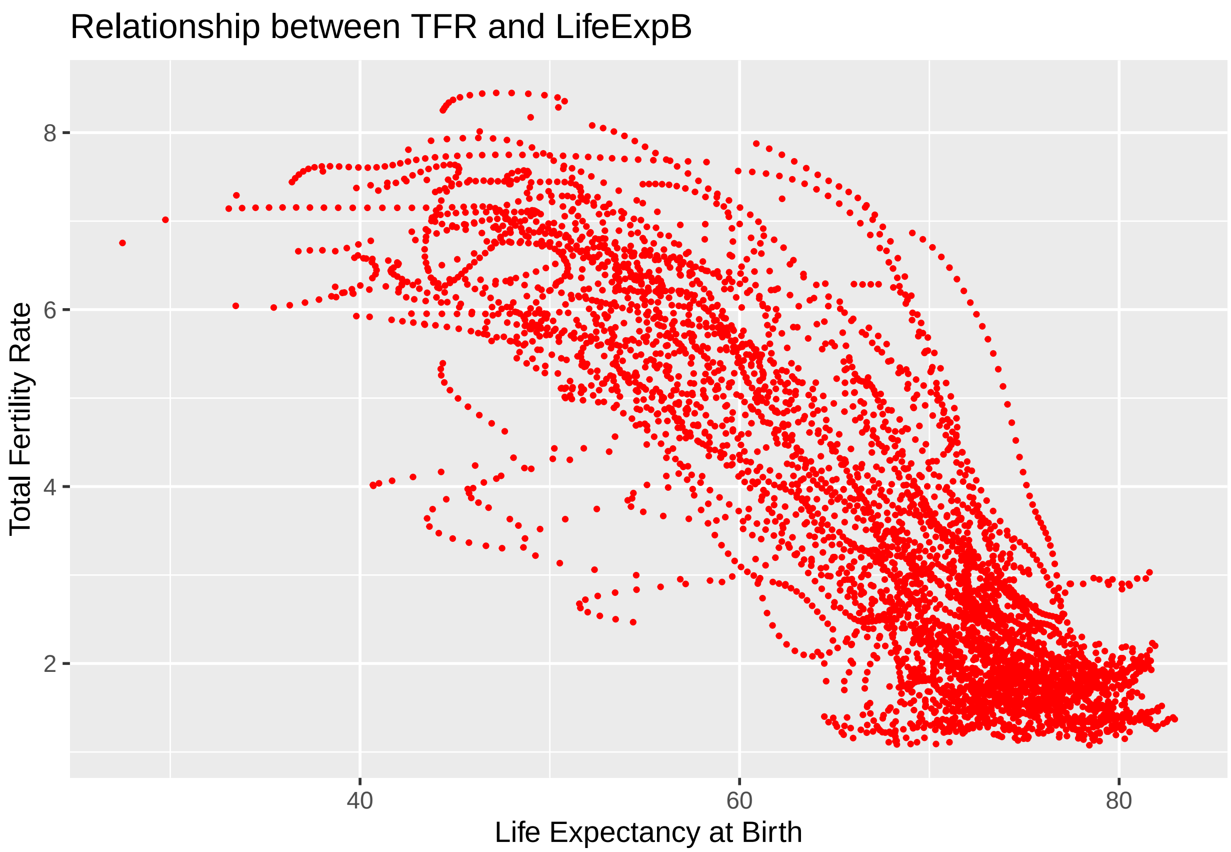
3. Set the color to red, and size for points as 0.5

```r
ggplot(df, aes(x = LifeExpB, y = TFR)) + geom_point(col = "red", size = 0.5)
```



4. Add appropriate label for the axis and the plot.

```
ggplot(df, aes(x = LifeExpB, y = TFR)) +
  geom_point(col = "red", size = 0.5) +
  labs(title="Relationship between TFR and LifeExpB",
       y="Total Fertility Rate", x="Life Expectancy at Birth")
```



5. Create a new directory named plots, and save your plot as TFR.png into plots.

```
dir.create("plots")
ggsave("TFR.png")
```

```
## Saving 6.5 x 4.5 in image
```

4. Tips on Using R Markdown

R Markdown is a file format for making dynamic documents with R. An R Markdown document is written in markdown (an easy-to-write plain text format) and contains chunks of embedded R code. For assignments in QPM I and II, you will often need to turn in your assignments in PDFs with LaTex content and R code chunks, not raw R scripts. Using R Markdown makes the process much easier!

For the content below, you need to knit the R markdown file into PDF to view how the content will be displayed.

$$\pi = 3.14$$

4.1 Formatted Text

The text in R markdown shares a similar syntax with LaTex that you have been using in Math Modeling.

You can use hashtags to create headers.

Header 1

Header 2

Header 3

Header 4 You can create italicized text with *asterisks*,

and bold text with **double asterisks.**

You can also create an ordered list like this:

1. item 1
2. item 2
3. item 3

Or an unordered list like this:

- item 1
- item 2
- item 3

4.2 Embedded R Code

The **knitr** package (which you all have installed on Day 1!) extends the basic markdown syntax to include chunks of executable R code.

You can type “‘ followed by {r} to start your R code block. Then close the code block using “‘.

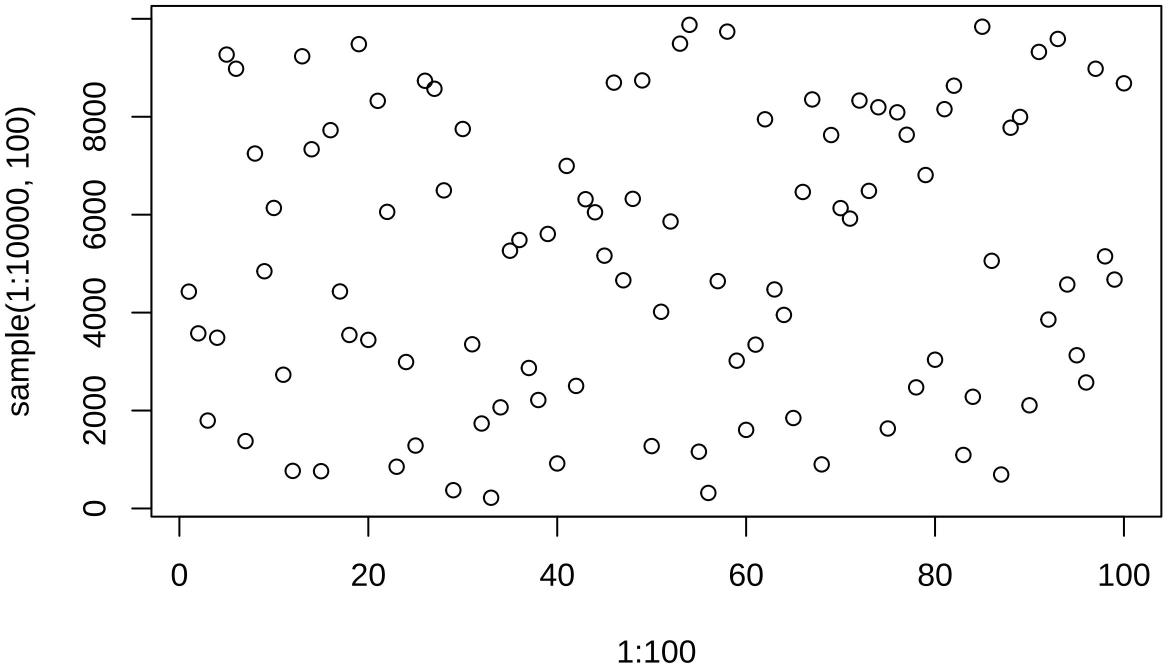
Alternatively, you can use the appropriate keyboard shortcut for your OS!

4.3 Adding Images

When you create an image in your code block in R Markdown, it will automatically be added to the PDF when you knit it.

For example,

```r
plot(1:100, sample(1:10000, 100))
```



However, when you need to add images from an image file, you can do it by typing:

```
![name of your image](path-to-image-here)
```

For example,

```
![Pongki Checkout her YouTube Channel: RuPong house](./pongki.jpeg){width=80%}
```

Figure 1: Pongki Checkout her YouTube Channel: RuPong house