An example of creating modular code in R - Efficient scientific programming

Learning Objectives

After completing this tutorial, you will be able to:

-

What you need

You will need a computer with internet access to complete this lesson and the data that we already downloaded for week 6 of the course.

{% include/data_subsets/course_earth_analytics/_data-week6-7.md %}

```
library("dplyr")
library("ggplot2")
```

Direct data access

Now that we have a general understanding of 2 ways to programmatically access data, let's give each way a go. We will start by directly downloading data from a website of interest.

In week one, we used `download.file()` to download a file from the web directly to our computer. When we did this, we were literally downloading that file, which happened to be in `.csv` (comma separated value or basic text format) format to our computer.

We specified the location where that file would download to, using the `destfile=` argument. Notice below, I specified week 10 as the download location given that is our current class week.

```
# download text file to a specified location on our computer
download.file(url = "https://ndownloader.figshare.com/files/7010681",
              destfile = "data/week10/boulder-precip-aug-oct-2013.csv")
```

If `R` was able to communicate with the server (in this case Figshare) and download the file, we could then open up the file and plot it.

```
# read data into R
boulder_precip <- read.csv("data/week10/boulder-precip-aug-oct-2013.csv")

# fix date
boulder_precip$DATE <- as.Date(boulder_precip$DATE)
# plot data with ggplot
ggplot(boulder_precip, aes(x = DATE, y=PRECIP)) +
  geom_point() +
      labs(x="Date (2013)",
           y="Precipitation (inches)",
          title="Precipitation - Boulder, CO ",
          subtitle = "August - October 2013")
```

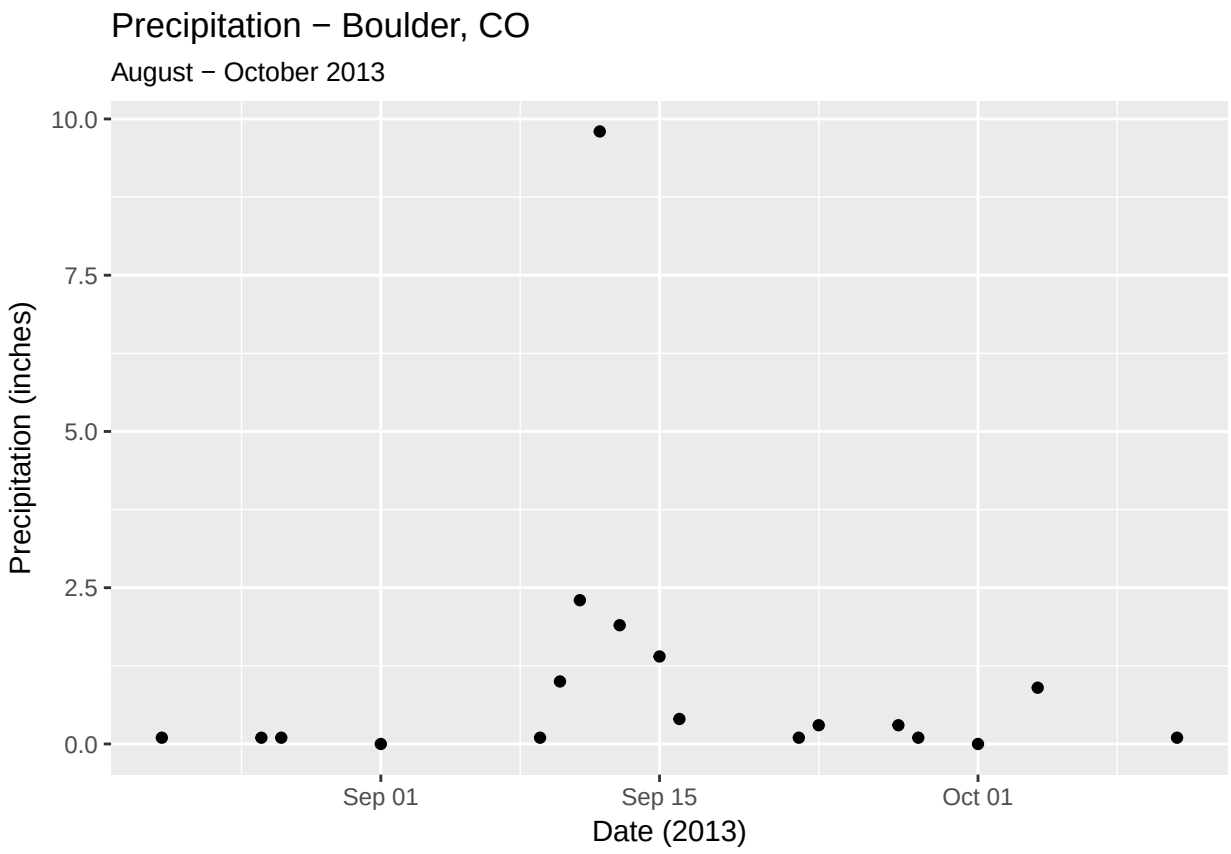
Precipitation − Boulder, CO
August − October 2013
Precipitation (inches)
10.0
7.5
5.0
2.5
0.0
Sep 01
Sep 15
Oct 01
Date (2013)

Figure 1:

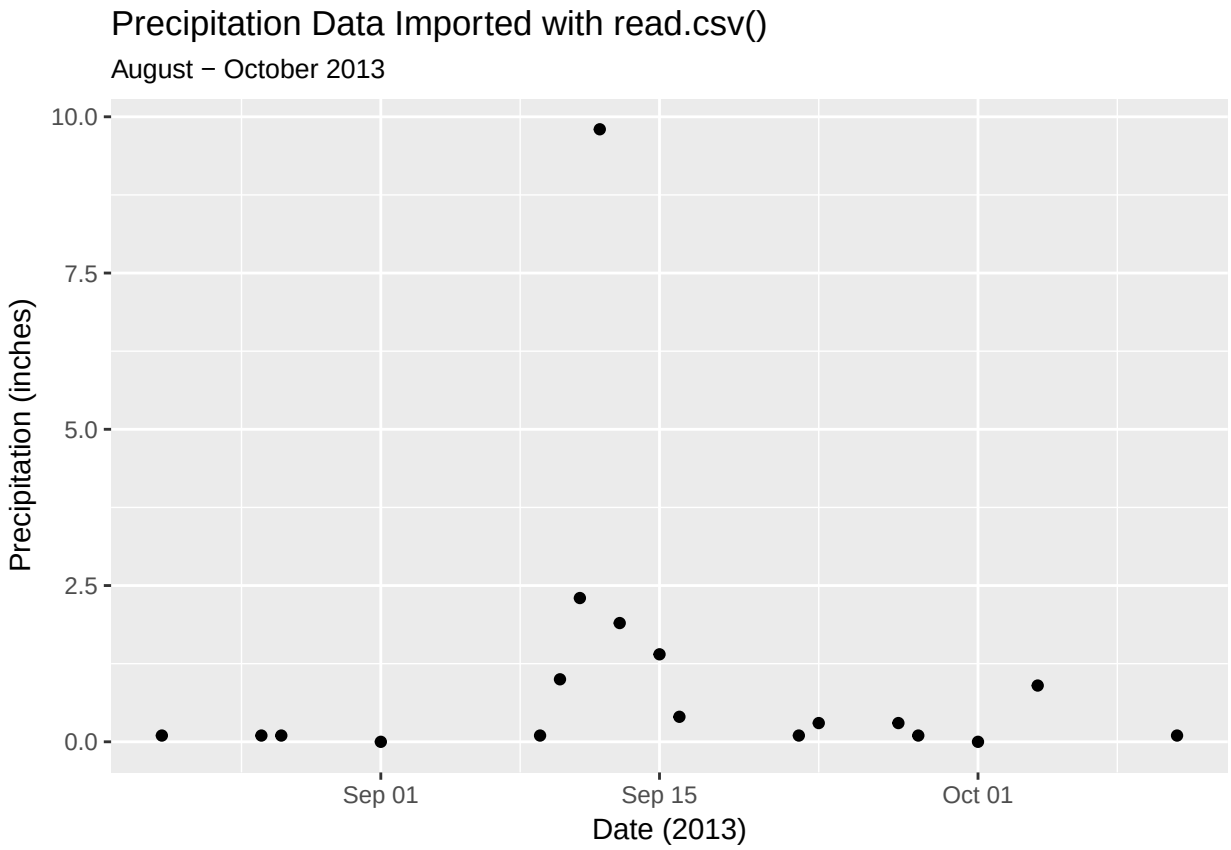

Figure 2:

Get data via human readable url

There are several ways that we download data from the internet using `R`. The simplest option is to download a text (e.g., `.csv`, `.txt`) file containing data via a `URL` like we did above.

However we can also import data directly into R rather than downloading it, using the `read.csv()` and/or `read.table` functions. Let's do that next.

Data Tip: If we have a secure url (secure transfer protocols - i.e., `https`) we may not be able to use read.csv. Instead, we need to use `RCurl` (which we'll see later). {: .notice–warning}

Use read.csv() to read in data from a URL

```
boulder_precip2 <- read.csv("https://ndownloader.figshare.com/files/7010681")
# fix date
boulder_precip2$DATE <- as.Date(boulder_precip2$DATE)
# plot data with ggplot
ggplot(boulder_precip2, aes(x = DATE, y=PRECIP)) +
  geom_point() +
      labs(x="Date (2013)",
           y="Precipitation (inches)",
          title="Precipitation Data Imported with read.csv()",
          subtitle = "August - October 2013")
```

Explore other data

Let's try to access another dataset available on a different site to practice what we just learned. Birth rate data on birth rates for several countries are available via a Princeton University website.

The dataset contains 3 variables: * Birth rate * Index of social setting * Index of family planning effort

We can read these data in `R` using the `read.table()` function.

Note that we are using read.table() rather than read.csv because in this instance, the data are not stored in a comma separated format. Rather, they are stored in a .dat format.

```
base = "http://data.princeton.edu/wws509"  # Base url
file = "/datasets/effort.dat"  # File name
birth_rates = read.table(paste0(base, file))
```

About paste0()

Also note that we are building the URL programmatically using the `paste0()` function. This function simply *pastes* together 2 or more strings of text (or variables into a new variable). It is useful to build a url this way when we plan to use the same API base url over and over, but may be calling various subsets of data available from that API.

In this case there may be other datasets in addition to the one located at `/datasets/effort.dat`.

```
# paste the base url together with the file name
paste0(base, file)
## [1] "http://data.princeton.edu/wws509/datasets/effort.dat"
```

Working with Web Data

The (`birth_rates`) data that we just accessed, imports into the `data.frame` format. We can analyze and visualize the data using `ggplot()` just like we did with the precipitation data above. For example:

Here's the top (or 'head') of the `data.frame`:

```
str(birth_rates)
## 'data.frame':    20 obs. of  3 variables:
##  $ setting: int  46 74 89 77 84 89 68 70 60 55 ...
##  $ effort : int  0 0 16 16 21 15 14 6 13 9 ...
##  $ change : int  1 10 29 25 29 40 21 0 13 4 ...
head(birth_rates)
##           setting effort change
## Bolivia        46      0      1
## Brazil         74      0     10
## Chile          89     16     29
## Colombia       77     16     25
## CostaRica      84     21     29
## Cuba           89     15     40
```

About the birth rate data

The birth rate data show how much effort went into considering family planning efforts that were in place to attempt to reduce birth rates in various countries. The outcome variable is the associated percent decline in birth rate by country over 10 years. An excerpt from the website where we are getting the data is below.

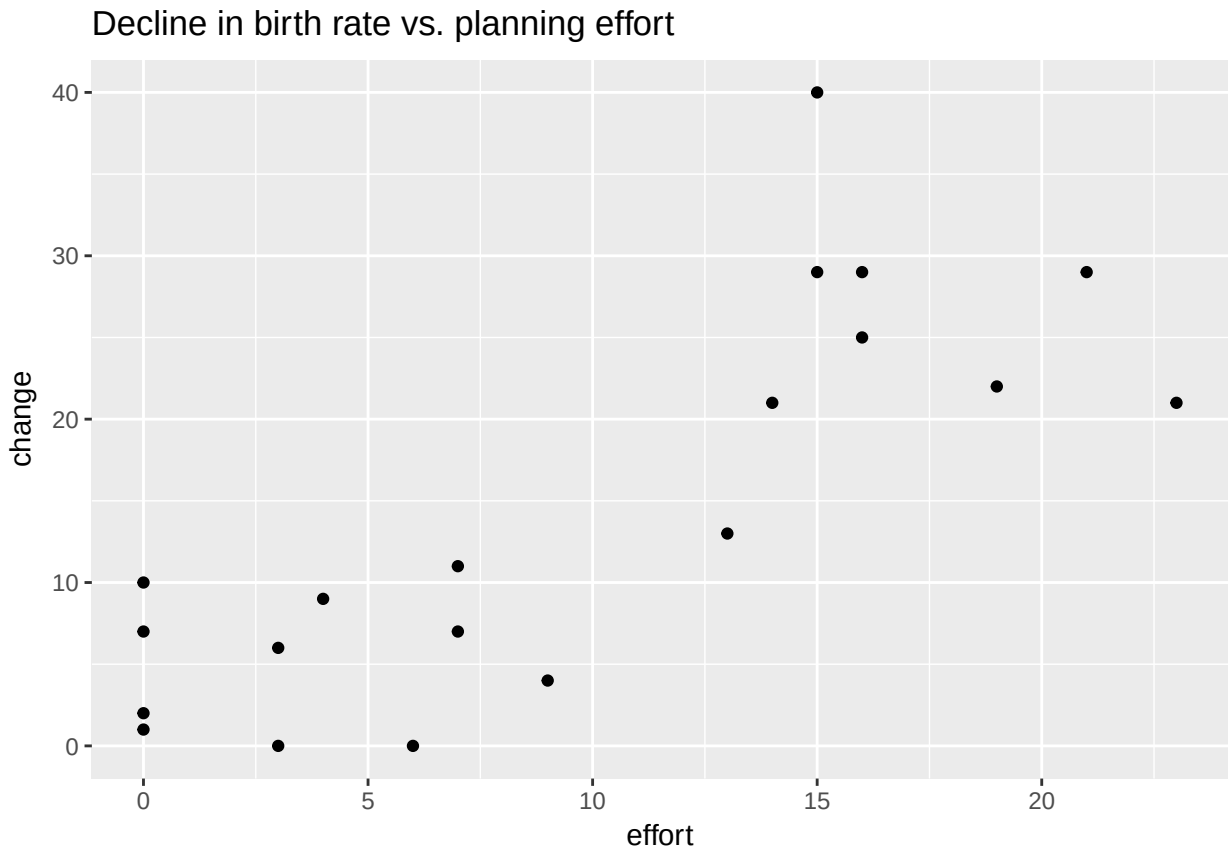

Figure 3:

> Here are the famous program effort data from Mauldin and Berelson. This extract consist of observations on an index of social setting, an index of family planning effort, and the percent decline in the crude birth rate (CBR) between 1965 and 1975, for 20 countries in Latin America.

We can plot these data to see the relationships between effort and % change in birth rates.

```
ggplot(birth_rates, aes(x=effort, y=change)) +
  geom_point() + ggtitle("Decline in birth rate vs. planning effort")
```

Remember that here, we've imported a tabular data set directly from the website. The data file itself is NOT on our computer. We are now moving towards a more programmatic approach.

Data Tip: Consider when you directly access a dataset via an API that - that data may not always be available. It is a good idea to save backup copies of certain datasets on your computer in many cases - in the event that the data API goes down, is taken away, etc. {: .notice–warning }

Use RCurl to download data

Sometimes the direct download base `R` protocols that we used above do not work. Specifically there are problems associated with downloading from secure, https URLs. RCurl is a powerful package that:

- Provides a set of tools to allow `R` to act like a *web client*
- Provides a number of helper functions to grab data files from the web:

The `getURL()` function works for most secure web download protocols (e.g., `http(s)`, `ftp(S)`). It also helps with web scraping, direct access to web resources, and even APIs

Download data with RCURL

Gapminder Data

Let's grab the gapminder data from a secure URL located on a GitHub website. @jennybryan provides an `R` package to access the Gapminder data for teaching. However, while we could access these data using the R gapminder package, we will instead use `RCURL` to get it from Jenny Bryan's Github Page to practice using RCURL.

```
library(RCurl)  # Load RCurl (note cases)
# Store base url (note the secure url)
file = "https://raw.githubusercontent.com/jennybc/gapminder/master/inst/gapminder.tsv"
temp = getURL(file)  # grab the data!
```

Ask carson why RCurl is required here. is it more a windows thing?? read.csv(file)

```
# this works --
head(read.csv(file, sep="\t"))
##       country continent year lifeExp      pop gdpPercap
## 1 Afghanistan      Asia 1952  28.801  8425333  779.4453
## 2 Afghanistan      Asia 1957  30.332  9240934  820.8530
## 3 Afghanistan      Asia 1962  31.997 10267083  853.1007
## 4 Afghanistan      Asia 1967  34.020 11537966  836.1971
## 5 Afghanistan      Asia 1972  36.088 13079460  739.9811
## 6 Afghanistan      Asia 1977  38.438 14880372  786.1134
```

Get Data with getURL()

Now that we have a connection to the Github `url`, we can treat it like a text file, and read in the file using `read.csv()` via a `textConnection()` function:

```
# Use textConnection to read content of temp as tsv
gap_data = read.csv(textConnection(temp))
head(gap_data)
##             country.continent.year.lifeExp.pop.gdpPercap
## 1  Afghanistan\tAsia\t1952\t28.801\t8425333\t779.4453145
## 2  Afghanistan\tAsia\t1957\t30.332\t9240934\t820.8530296
## 3   Afghanistan\tAsia\t1962\t31.997\t10267083\t853.10071
## 4  Afghanistan\tAsia\t1967\t34.02\t11537966\t836.1971382
## 5 Afghanistan\tAsia\t1972\t36.088\t13079460\t739.9811058
## 6   Afghanistan\tAsia\t1977\t38.438\t14880372\t786.11336
```

Looking at our data, we have a separator. In this case it's `\t`. We can account fo this using `read.csv()` by using the `sep=` argument.

```
# Use textConnection to read content of temp as tsv
gap_data <- read.csv(textConnection(temp),
                     sep="\t")
head(gap_data)
##       country continent year lifeExp      pop gdpPercap
## 1 Afghanistan      Asia 1952  28.801  8425333  779.4453
```

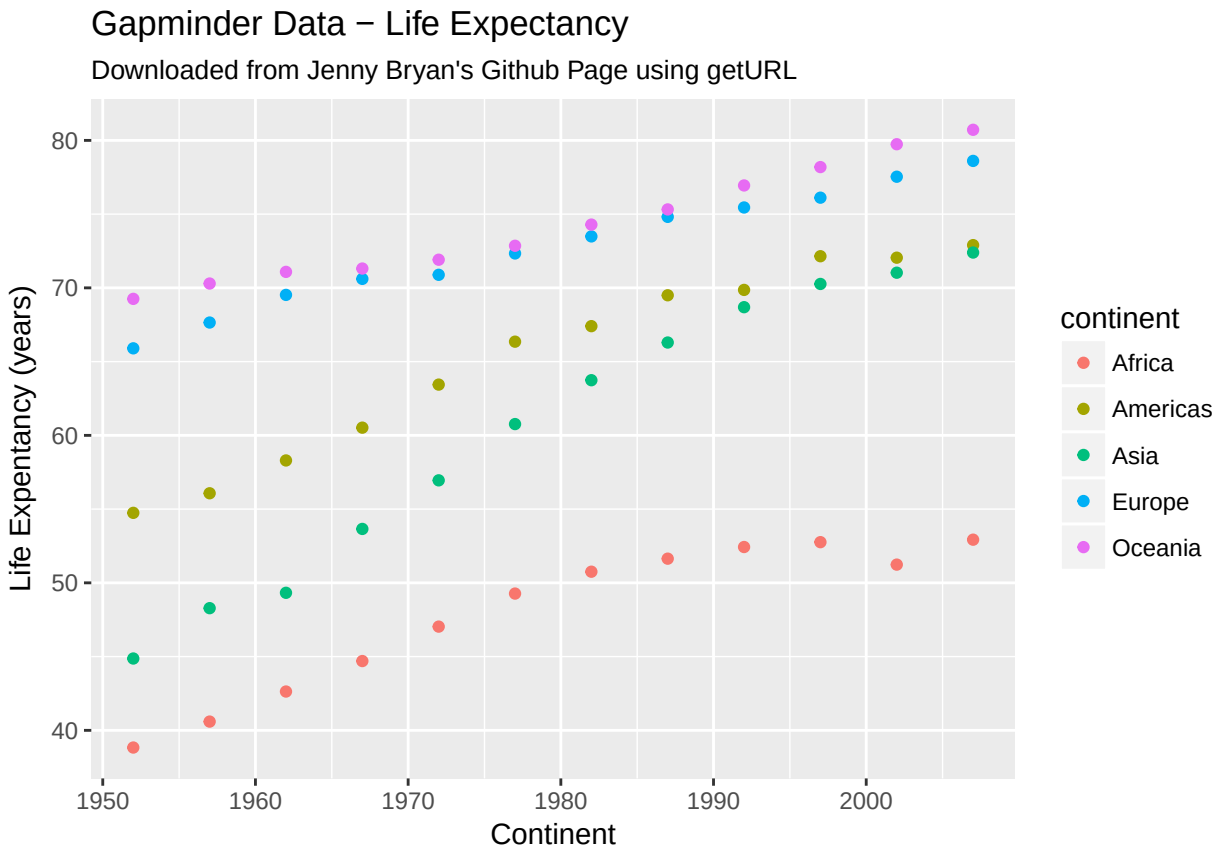

Figure 4:

```
## 2 Afghanistan      Asia 1957  30.332  9240934  820.8530
## 3 Afghanistan      Asia 1962  31.997 10267083  853.1007
## 4 Afghanistan      Asia 1967  34.020 11537966  836.1971
## 5 Afghanistan      Asia 1972  36.088 13079460  739.9811
## 6 Afghanistan      Asia 1977  38.438 14880372  786.1134
```

Next, we can summarize and plot the data!

```
# summarize the data - median value by content and year
summary_life_exp <-  gap_data %>%
   group_by(continent, year) %>%
   summarise(median_life = median(lifeExp))

ggplot(summary_life_exp, aes(x=year, y=median_life, colour = continent)) +
  geom_point() +
      labs(x="Continent",
           y="Life Expentancy (years)",
          title="Gapminder Data - Life Expectancy",
          subtitle = "Downloaded from Jenny Bryan's Github Page using getURL")
```

Notice that when we import the data from github, using read.csv, it imports into a `data.frame` format. Thus, we can plot the data using ggplot() like we are used to. Below, we make a boxplot of `lifeExp` by `continent`:

```
# create box plot
ggplot(summary_life_exp,
```

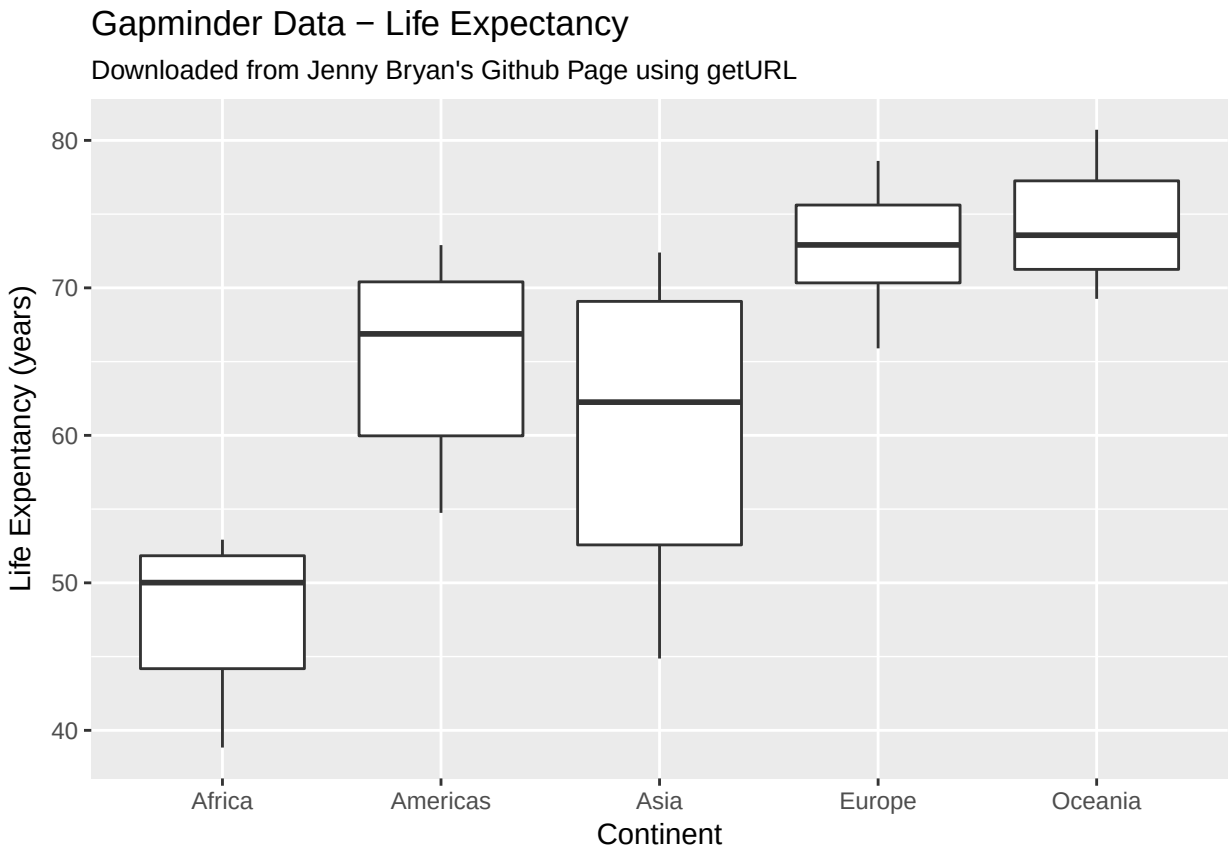

Figure 5:

```
       aes(continent, median_life)) +
      geom_boxplot()+
      labs(x="Continent",
           y="Life Expentancy (years)",
          title="Gapminder Data - Life Expectancy",
          subtitle = "Downloaded from Jenny Bryan's Github Page using getURL")
```

We can also create a more advanced plot - overlaying the data points on top of our box plot. See the ggplot documentation to learn more advanced ggplot() plotting approaches.

```
ggplot(gap_data, aes(x=continent, y=lifeExp)) +
  geom_boxplot(outlier.colour="hotpink") +
  geom_jitter(position=position_jitter(width=0.1, height=0), alpha=0.25)+
      labs(x="Continent",
           y="Life Expentancy (years)",
          title="Gapminder Data - Life Expectancy",
          subtitle = "Downloaded from Jenny Bryan's Github Page using getURL")
```

LOOK FOR ANOTHER CSV to downnload using this function – good practice for them

And maybe this becomes an assignment.

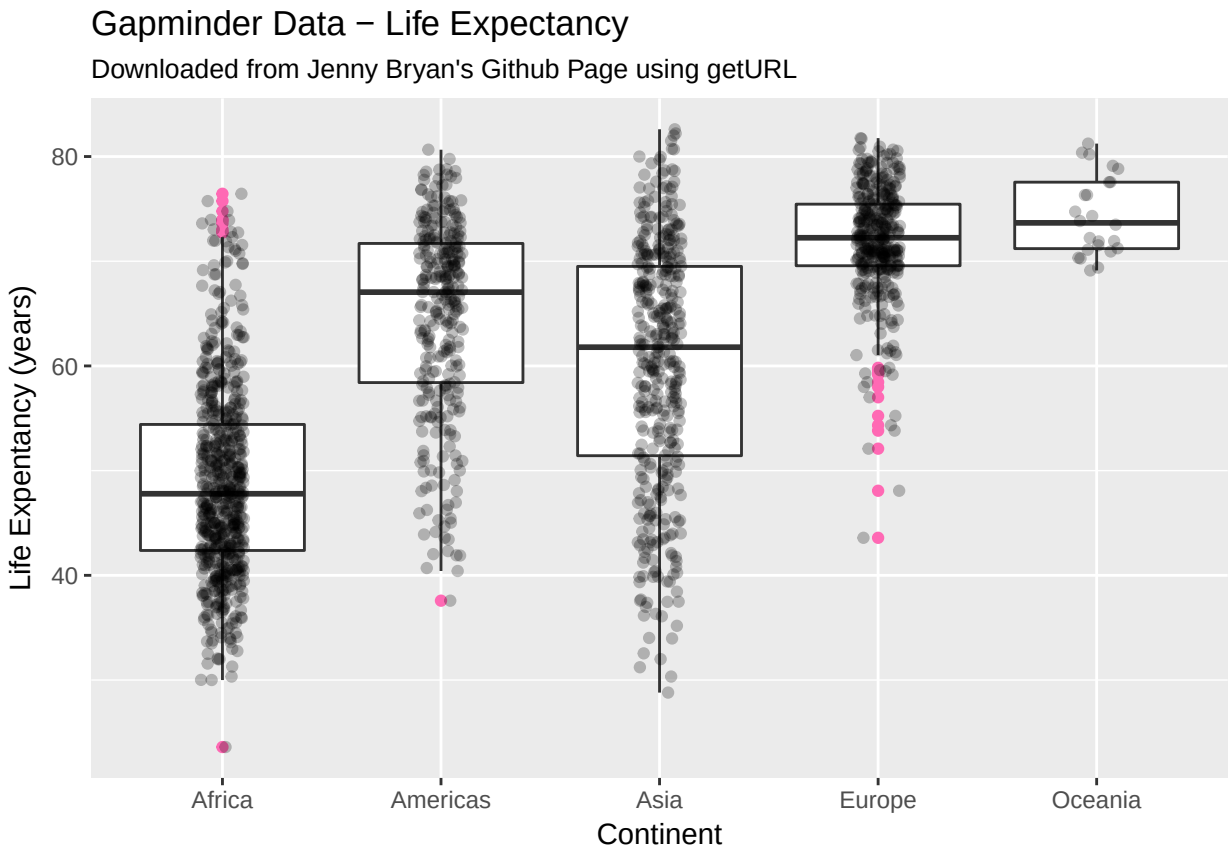
Gapminder Data − Life Expectancy
Downloaded from Jenny Bryan's Github Page using getURL
Life Expentancy (years)
80
60
40
Africa
Americas
Asia
Europe
Oceania
Continent

Figure 6:

- If you are going to be grabbing a lot of `csv` files from secure `urls`, you might want to turn the previous code into a function:

```
read.csv.https = function(url) {
  url = getURL(url)
  return(read.csv(textConnection(url)))
}
```

- On Windows you *might* be able to skip the `getURL` part...

Data Tip: The web changes constantly! Data available via a *particular* API at a *particular* point in time may not be available indefinitely... {: .notice–warning}