Introduction to Data Visualization with R

Shan Ye (https://yeshancqcq.github.io)

Sample data and code can be found here: https://github.com/yeshancqcq/ggplot2_tutorial

Installing R and relevant packages

R is a free open-source programming language with various useful packages for different purposes. The language and most packages can be downloaded via CRAN (Comprehensive R Archive Network: https://cran.r-project.org/).

- For MacOS: https://cran.r-project.org/bin/macosx/
- For Windows: https://cran.r-project.org/bin/windows/base/
- For Linux: `sudo apt-get install r-base`

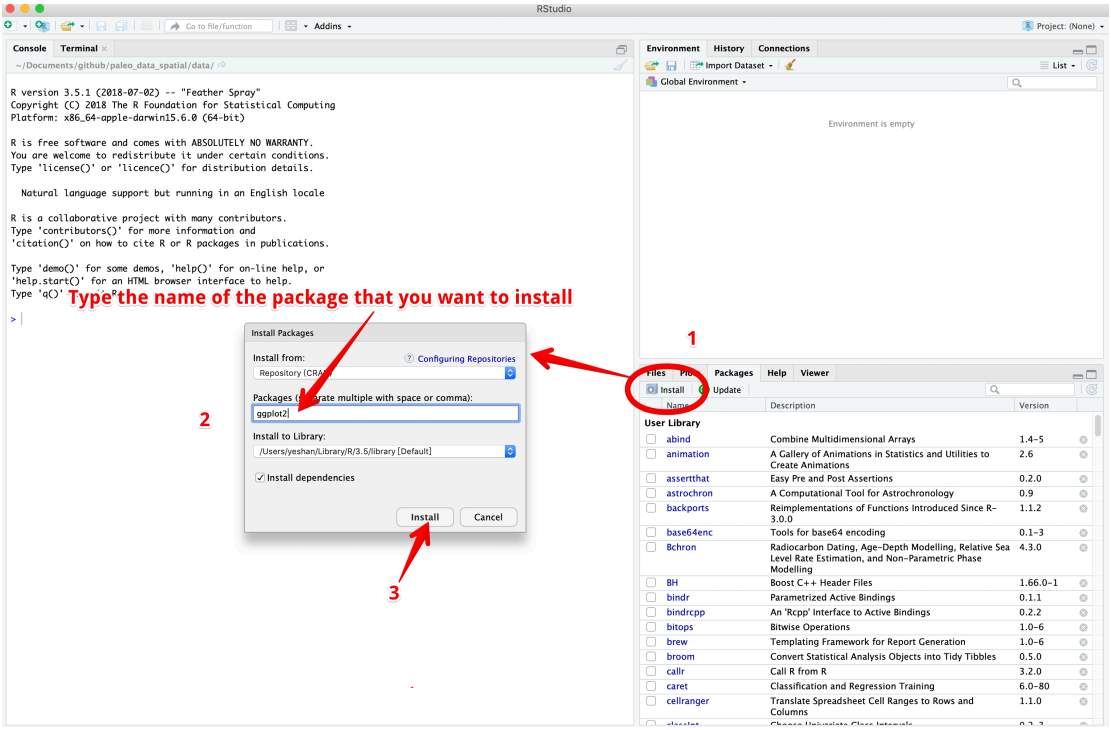
Follow instructions on these websites to download and install R on your computer.

RStudio is a good integrated development environment (IDE) for R programming. It is easier to use than the default R scripting interface. RStudio can be downloaded from: https://rstudio.com/products/rstudio/download/ (you only need the free version).

Once you finished installing R and RStudio, you can open the RStudio for some initial setup. One of most common things to do is to install some frequently-used libraries/packages, including the `ggplot2` that we are using.

There are two ways to do this.

1. Installing through the Package window:



2. Installing in the Console window

Type the following command in the Console window after the blue > mark, and hit Enter.

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

This will download the `ggplot2` package from CRAN and install it onto your system. Change the `ggplot2` in `" "` to other packages' names if you want to install something else. Some other useful packages (for this tutorial and other tasks) include `magrittr`, `multipanelfigure`, `maps`, `ggmap` and `readr`.

Set up a working directory

Create a new folder (or choose one that already exists) and create another folder named "data" within it. Put all your data files (csv, json, shp, txt, etc.) in the "data" folder. In this tutorial, we are only dealing with csv files. (For importing data via Macrostrat API, see my instruction here: https://yeshancqcq.github.io/document/Macrostrat_API.pdf)

Go back to your RStudio. You can set up your working directory to the folder one hierarchy above the "data" folder by typing the following line in the Console window and hit Enter.

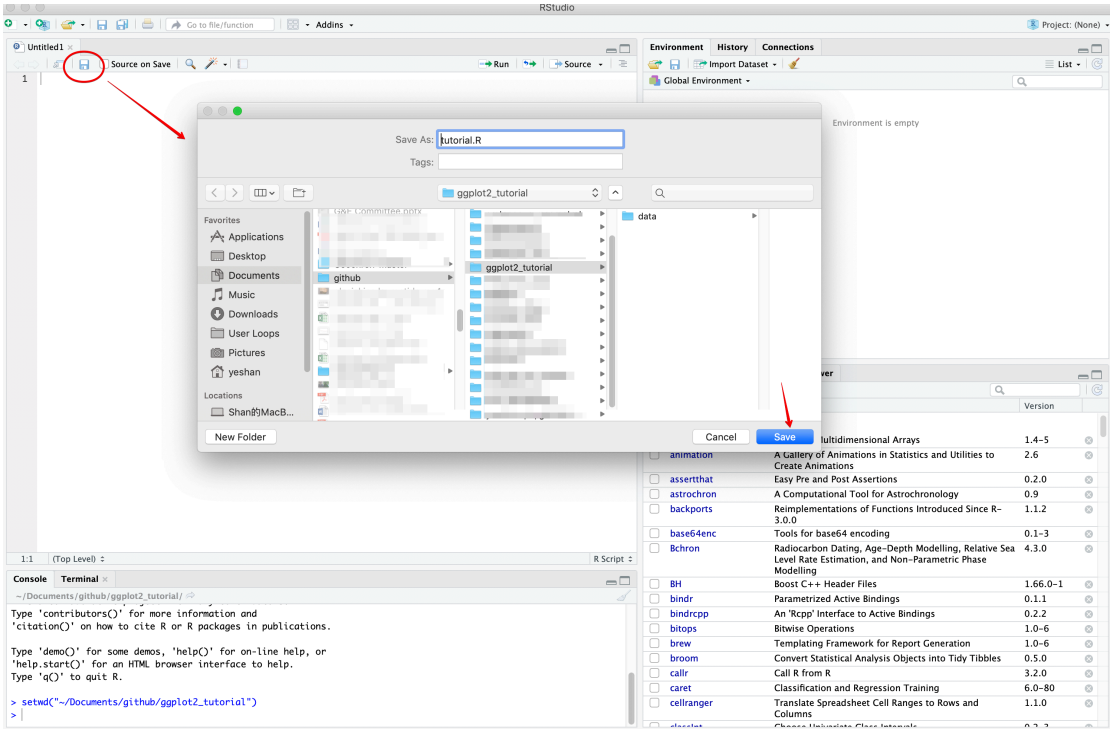
```
setwd("~/Documents/github/ggplot2_tutorial")
```

You should change the directory in `""` to your own folder. If you do not know how to write

the directory, you can open your Terminal Window (search "Terminal" in your system), and then copy and paste the folder into the Terminal Window, then the directory will be shown there. Alternatively, you can manually choose your directory in the menu bar of RStudio (Session → Set Working Directory → Choose Directory).

Now, you are ready to code. If you just want to make a one-time plot, you can just code in the Console Window. However, in most cases we want to make our code recyclable. In this case, we need to create an R script. In the menu bar of RStudio, create a new script via File → New File → R Script. Now, you can see a 4th window appearing on top of the Console Window. That's where your code will go.

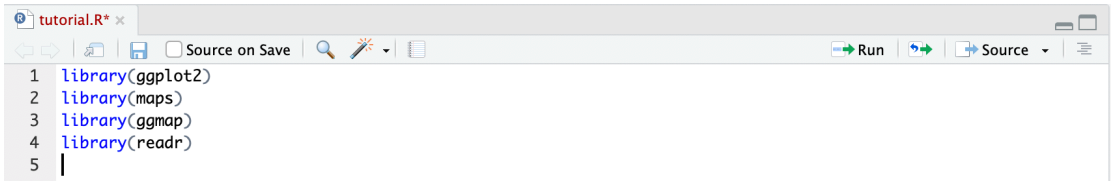
Save your R Script to your working directory.



Note: Some computers (especially with Windows system) will not automatically add .R after the file name. You do need to add it when saving the file. Otherwise the RStudio would not recognize it.

Coding for plots

First things first: import packages that you have installed on your computer.



To run each line, put your cursor at the end of the line, and press Command+Enter (for MacOS) or Ctrl+Enter (for Windows and Linux). To run multiple lines, select those lines with your mouse and hit above-mentioned shortcut keys.

Once your packages are imported and loaded, you are ready to import your data files.

```
6  # import football data of Wisconsin, Iowa and Minnesota
7  wisconsin <- read_csv("data/wisconsin_football.csv")
8  iowa <- read_csv("data/iowa_football.csv")
9  minnesota <- read_csv("data/minnesota_football.csv")
10
```

Line 6 is a comment of what I am doing next. It is always good to include some comments in your code. It will be helpful if you need to debug later, or if you want to use this code again 5 months later. To add a comment, type # at the beginning of a line, and type your comments after it.

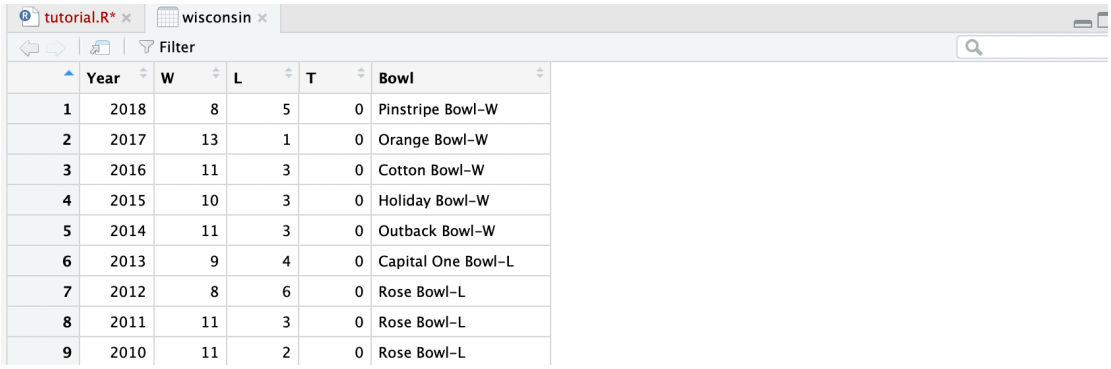
Lines 7, 8 and 9 are commands of importing csv data with `read_csv( )` function built in the `readr` package. Let's take a look at what are in Line 7:

```
wisconsin <- read_csv("data/wisconsin_football.csv")
```

The first "wisconsin" is a variable name. This means that you want to turn the data file imported at this line into a variable named `wisconsin`. Everytime you refer to this data file later in the code, you only need to call this `wisconsin` name. Then, in the ( ) after the function, you need to tell RStudio how to find the Wisconsin football data. Remember that we have set our working directory to one hierarchy above the data folder. Therefore, you need to tell RStudio that "in the working directory, find a folder named data, and then find a file named wisconsin_football.csv. That's how you get `"data/wisconsin_football.csv"`.

Now, data files of Wisconsin, Iowa and Minnesota have all been imported to RStudio. You can

see them in the upper-right window. You can view them by double click on each of them. Alternatively, you can type `View(wisconsin)` in the Console window and hit Enter to view the Wisconsin data file.



tutorial.R* × wisconsin ×

Filter

	Year	W	L	T	Bowl
1	2018	8	5	0	Pinstripe Bowl-W
2	2017	13	1	0	Orange Bowl-W
3	2016	11	3	0	Cotton Bowl-W
4	2015	10	3	0	Holiday Bowl-W
5	2014	11	3	0	Outback Bowl-W
6	2013	9	4	0	Capital One Bowl-L
7	2012	8	6	0	Rose Bowl-L
8	2011	11	3	0	Rose Bowl-L
9	2010	11	2	0	Rose Bowl-L

There are 5 columns in this data frame, with column names of Year, W, L, T and Bowl respectively. Iowa and Minnesota data frames have the same structure.

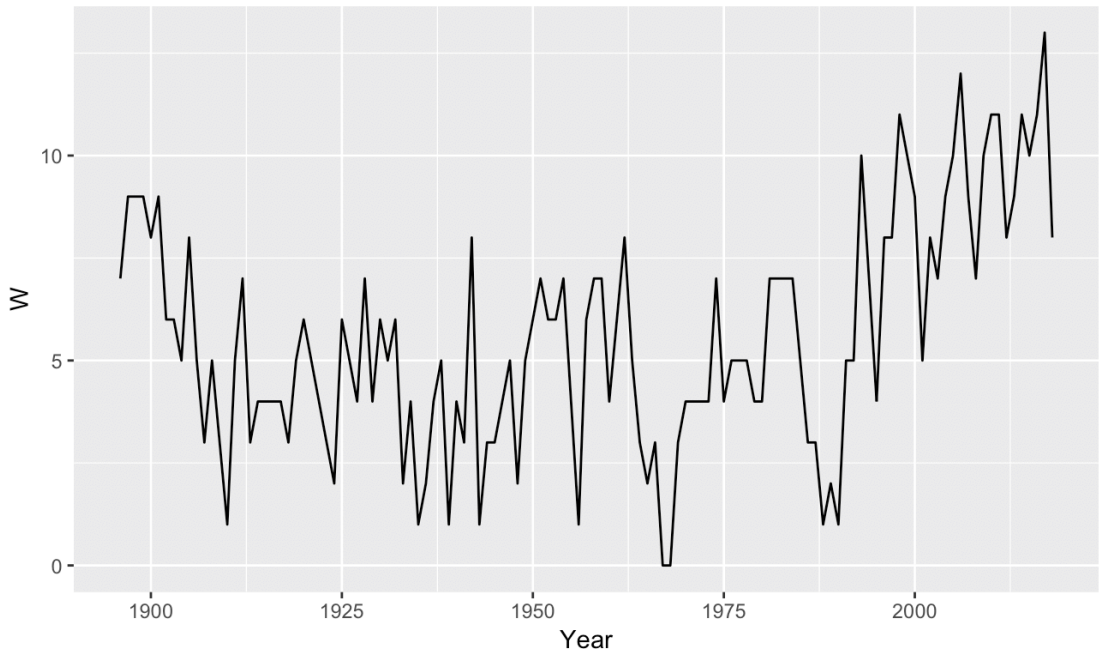
Now, let's make our first plot with `ggplot2`: number of wins of the Wisconsin football team in each year. The code is shown below:

```
11  # plot Wisconsin data: number of wins each year
12  plot1 <- ggplot()+
13    geom_line(data=wisconsin,aes(Year, W))
14
15  plot1
```

Here, `plot1` is the name of the plot. Everything after `<-` is command to make plot1. The `ggplot()` tells R to use ggplot function for this plot. The `geom_line` (refers to geometry: line) defines what type of plot you are making. In this case, it is a line plot. Within the `geom_line` function, you need to define what data you are using (`wisconsin` in this case) and what are on your x axis and y axis (under `aes` function, in this case, the column Year in `wisconsin` data is on the x axis and column W is on the y axis).

After running this block, no plot will be generated. To get the plot, you need to run the next `plot1` (Line 15). The earlier `plot1` is where you define this plot, while the later `plot1` is where you tell RStudio to show you the plot.

Now, you should get a plot like this:

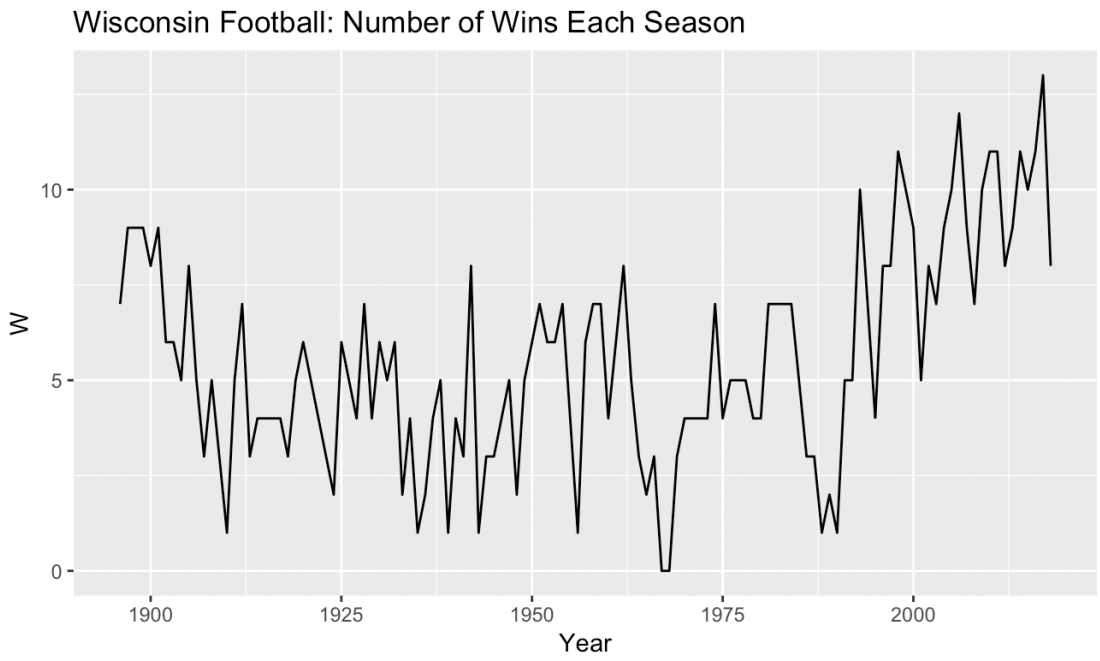


This plot is kind of ugly because it is the default output of ggplot2. We haven't add our other options yet. To add more options, you simply need to add more commands after the ggplot(). The ggplot function will add each line of your command as a layer to the plot (i.e. overlaying all things generated from your different lines of commands).

For example, let's add a title to this plot:

```
# plot Wisconsin data: number of wins each year
plot1 <- ggplot()+
  geom_line(data=wisconsin,aes(Year, W)) +
  ggtitle("Wisconsin Football: Number of Wins Each Season")

plot1
```

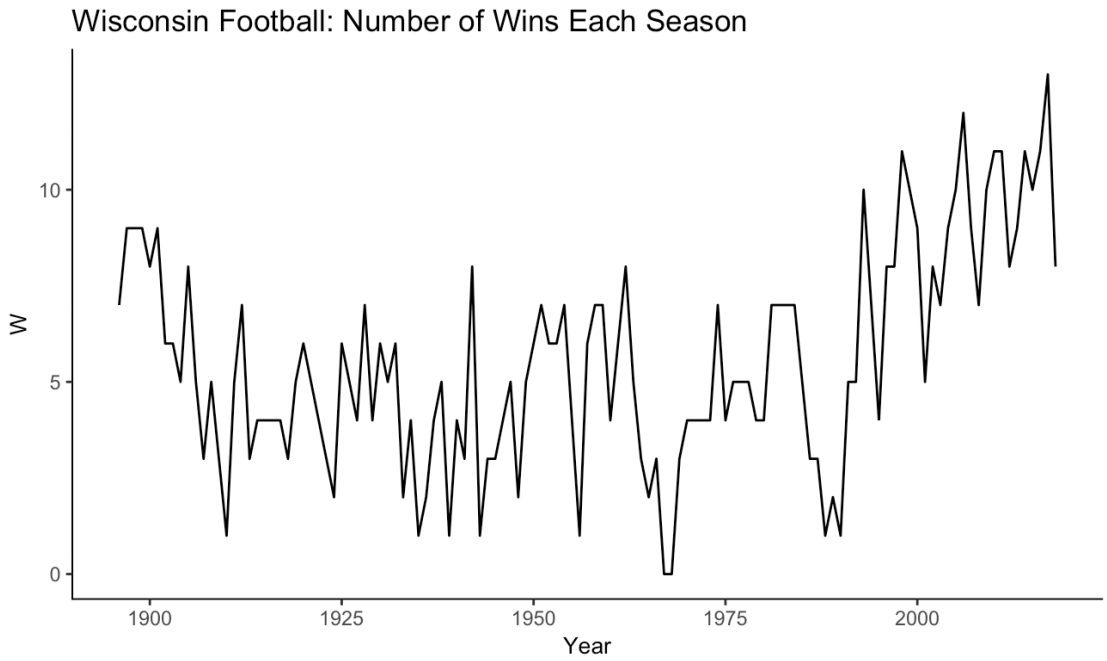
Then you'll get:



Then, let's get rid of the gray-colored backgrounds, and add solid axis to this plot. This kind of styling things can be done by adding `theme()` function commands to the plot. For example:

```
# plot Wisconsin data: number of wins each year
plot1 <- ggplot()+
  geom_line(data=wisconsin,aes(Year, W)) +
  ggtitle("Wisconsin Football: Number of Wins Each Season") +
  theme(panel.grid.major = element_blank(), # Get rid of major grid lines
        panel.grid.minor = element_blank(), # Get rid of minor grid lines
        panel.background = element_blank(), # Get rid of gray backgrounds
        axis.line.x = element_line(colour = "black"), # Have a solid x axis in black color
        axis.title.x=element_text(size=10), # Set the size of texts on x axis to be 10 points
        axis.line.y= element_line(colour = "black"), # Have a solid y axis in black color
        axis.title.y=element_text(size=10) # Set the size of texts on y axis to be 10 points
        # Note: no "," at the end of the last line in the theme() function
  )

plot1
```

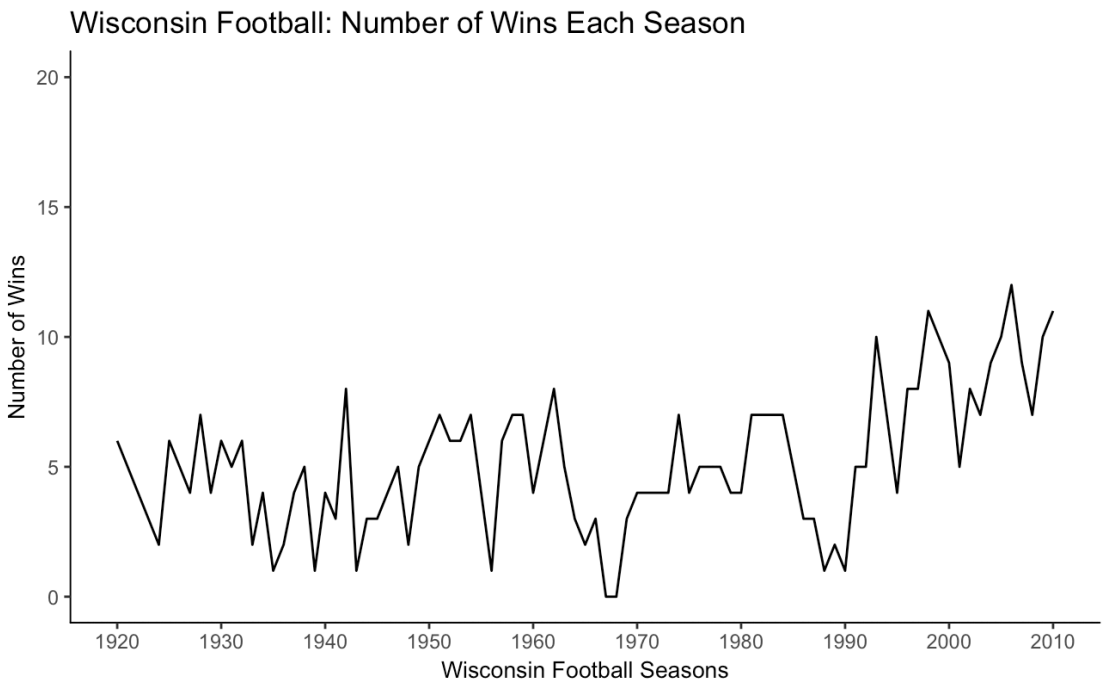
One thing to notice is that in `ggplot2` functions, "color" is spelled in British way ("colour"). The `element_blank()` means add a blank element (nothing) to corresponding components in the theme. Other common elements include `element_text()`, `element_rect()` and `element_line()`. There will be some examples later. Now you should have this plot:



Now, let's adjust the range and breaks of axis. In default, ggplot sets ranges and breaks of axis accordingly to your data distributions. However, sometimes we need to adjust it for various reasons (e.g. only show a certain portion of the data). Say, if we only care about the number of wins between 1920 and 2010, we can do it like this:

```
# plot Wisconsin data: number of wins each year
plot1 <- ggplot()+
  geom_line(data=wisconsin,aes(Year, W)) +
  ggtitle("Wisconsin Football: Number of Wins Each Season") +
  theme(panel.grid.major = element_blank(), # Get rid of major grid lines
        panel.grid.minor = element_blank(), # Get rid of minor grid lines
        panel.background = element_blank(), # Get rid of gray backgrounds
        axis.line.x = element_line(colour = "black"), # Have a solid x axis in black color
        axis.title.x=element_text(size=10), # Set the size of texts on x axis to be 10 points
        axis.line.y= element_line(colour = "black"), # Have a solid y axis in black color
        axis.title.y=element_text(size=10) # Set the size of texts on y axis to be 10 points
        # Note: no "," at the end of the last line in the theme() function
  ) +
  scale_x_continuous(name = expression("Wisconsin Football Seasons"), # Label on x axis
                     limits = c(1920, 2010), # Lower and upper limits of x axis
                     breaks = scales::pretty_breaks(n = 10) # Number of tick marks on x axis
                     ) +
  scale_y_continuous(name = expression("Number of Wins"), # Label on y axis
                     limits = c(0, 20), # Lower and upper limits of y axis
                     breaks = scales::pretty_breaks(n = 5) # Number of tick marks on y axis
  )

plot1
```

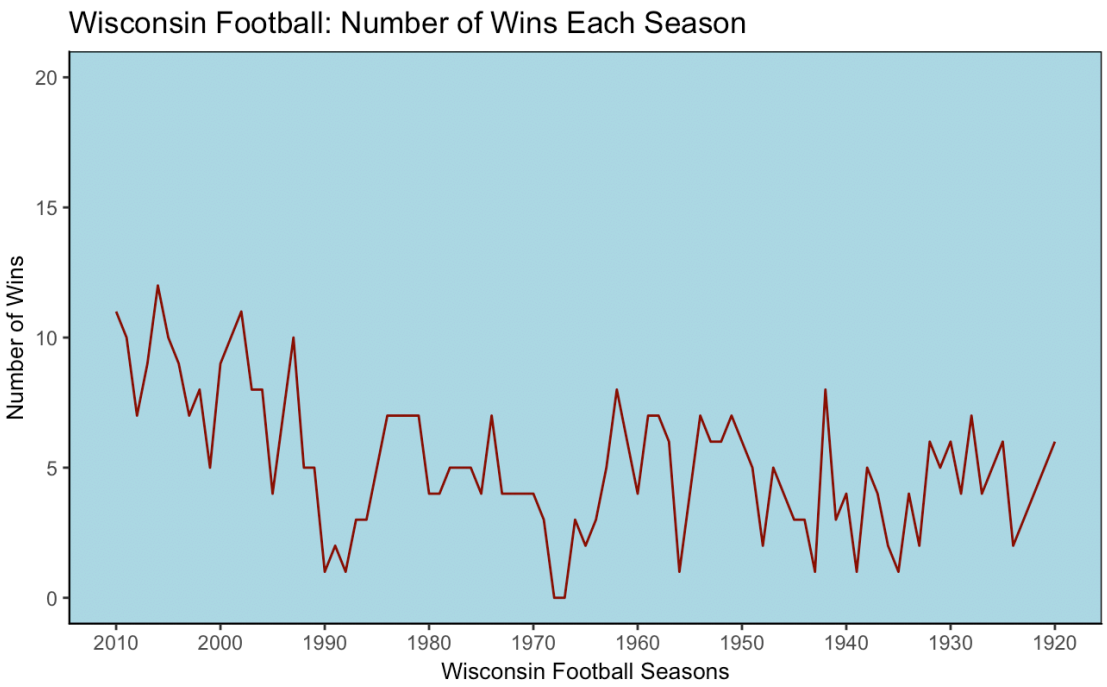
And this is what we get:



Sometimes, people need to get axis reversed. Also, some people might prefer to have solid borders all around the figure, and maybe with some background colors. We are also turning the curve into a dark red color. Let's try this:

```
# plot Wisconsin data: number of wins each year
plot1 <- ggplot()+
  geom_line(data=wisconsin,aes(Year, W), colour = "dark red") +
  ggtitle("Wisconsin Football: Number of Wins Each Season") +
  theme(panel.grid.major = element_blank(), # Get rid of major grid lines
        panel.grid.minor = element_blank(), # Get rid of minor grid lines
        panel.background = element_rect(fill = "light blue"),  # Set a light blue backgorund
        panel.border = element_rect(colour = "black", fill=NA), # Add solid borders
        axis.line.x = element_line(colour = "black"), # Have a solid x axis in black color
        axis.title.x=element_text(size=10), # Set the size of texts on x axis to be 10 points
        axis.line.y= element_line(colour = "black"), # Have a solid y axis in black color
        axis.title.y=element_text(size=10) # Set the size of texts on y axis to be 10 points
        # Note: no "," at the end of the last line in the theme() function
  ) +
  #Let's reverse the x axis:
  scale_x_reverse(name = expression("Wisconsin Football Seasons"), # Label on x axis
                  # Remember to change the following line:
                  limits = c(2010, 1920), # Lower and upper limits of x axis
                  breaks = scales::pretty_breaks(n = 10) # Number of tick marks on x axis
                  ) +
  scale_y_continuous(name = expression("Number of Wins"), # Label on y axis
                     limits = c(0, 20), # Lower and upper limits of y axis
                     breaks = scales::pretty_breaks(n = 5) # Number of tick marks on y axis
  )

plot1
```

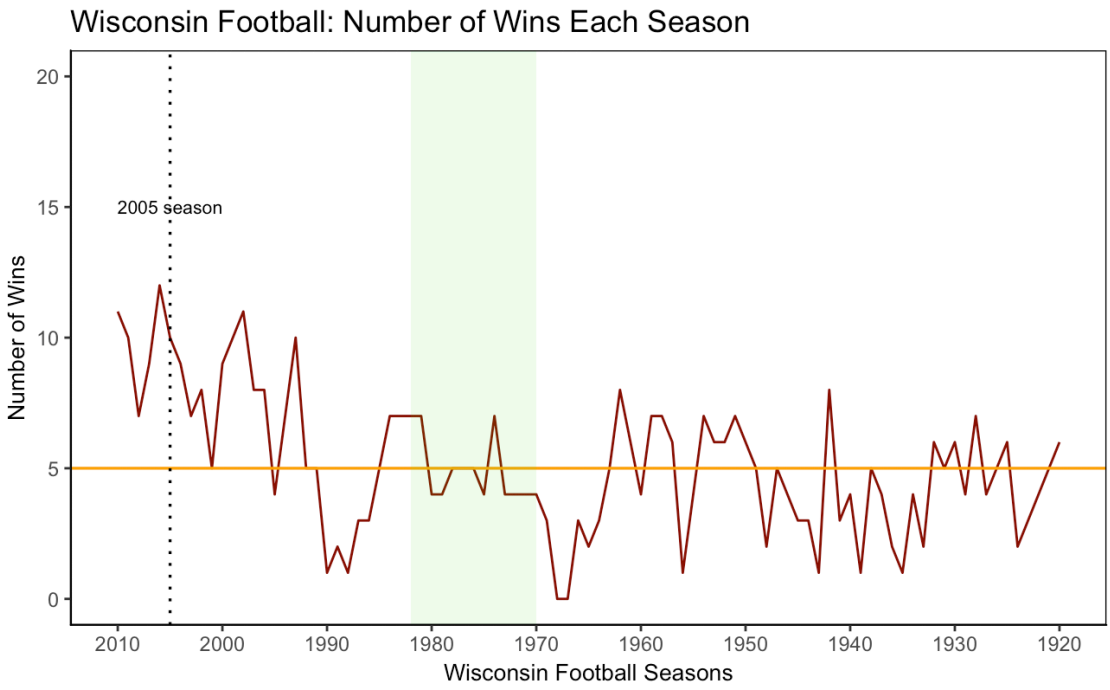
Here is the plot:



We can add some more elements. For example, we want to set the background color back to blank, and then we want to highlight the period between 1970 and 1982 with a green-shaded box (we need to initialize a new "data frame" for this; see the code below), and we also want to add a reference dash line for the 2005 season with a label "2005 season", and also a solid reference line of orange color for 5-win seasons. Here is what we need to do:

```r
# Add a new data frame for the reference box
ref <- data.frame(xmin = 1970, xmax = 1982, ymin = -Inf, ymax = Inf)

# plot Wisconsin data: number of wins each year
plot1 <- ggplot()+
  geom_line(data=wisconsin,aes(Year, W), colour = "dark red") +
  geom_hline(yintercept = 5, size = 0.6, colour = "orange") + # Add 5-win reference
  geom_vline(xintercept = 2005, size = 0.6, linetype = 'dotted') + # Add 2005 reference
  geom_rect(data=ref, aes(xmin=xmin, xmax=xmax, ymin=ymin, ymax=ymax), fill="green",
            alpha=0.1, inherit.aes = FALSE)+ # Add the green reference box w/ 10% transparency (alpha)
  annotate("text", x = 2005, y = 15,
           label = "2005 season", size = 2.8, colour = "black")+ # Add label for 2005 season
  ggtitle("Wisconsin Football: Number of Wins Each Season") +
  theme(panel.grid.major = element_blank(), # Get rid of major grid lines
        panel.grid.minor = element_blank(), # Get rid of minor grid lines
        panel.background = element_blank(),  # Set a blank backgorund
        panel.border = element_rect(colour = "black", fill=NA), # Add solid borders
        axis.line.x = element_line(colour = "black"), # Have a solid x axis in black color
        axis.title.x=element_text(size=10), # Set the size of texts on x axis to be 10 points
        axis.line.y= element_line(colour = "black"), # Have a solid y axis in black color
        axis.title.y=element_text(size=10) # Set the size of texts on y axis to be 10 points
        # Note: no "," at the end of the last line in the theme() function
  ) +
  #Let's reverse the x axis:
  scale_x_reverse(name = expression("Wisconsin Football Seasons"), # Label on x axis
                  # Remember to change the following line:
                  limits = c(2010, 1920), # Lower and upper limits of x axis
                  breaks = scales::pretty_breaks(n = 10) # Number of tick marks on x axis
                  ) +
  scale_y_continuous(name = expression("Number of Wins"), # Label on y axis
                     limits = c(0, 20), # Lower and upper limits of y axis
                     breaks = scales::pretty_breaks(n = 5) # Number of tick marks on y axis
  )

plot1
```

And here is what we get:

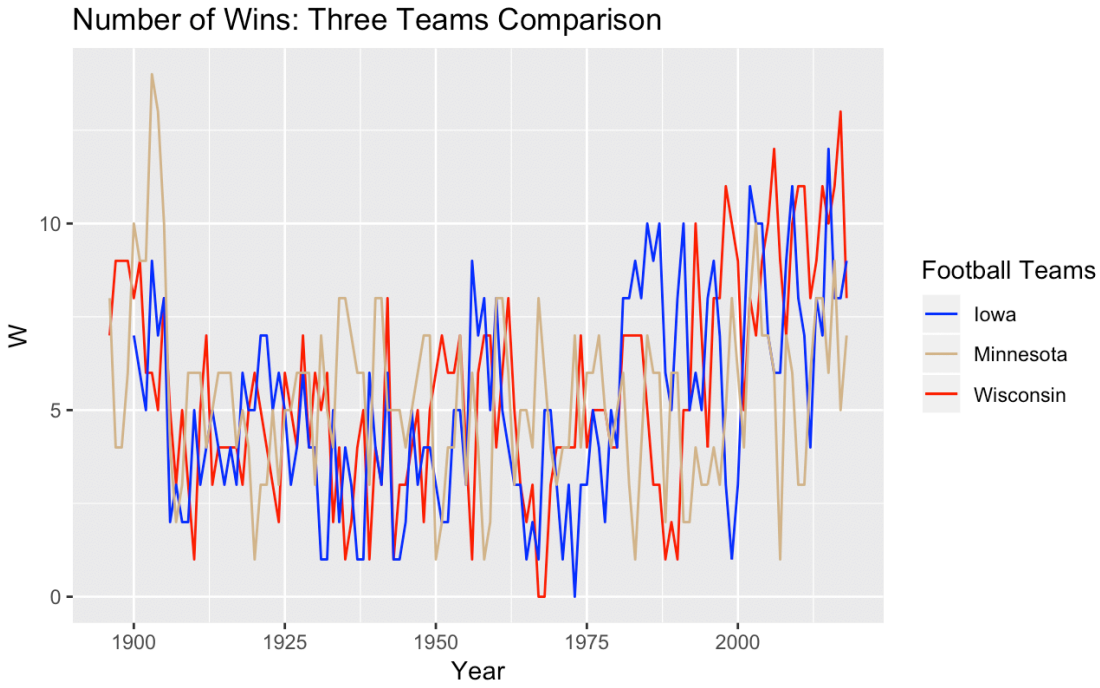


Let's now make another plot. This time, we want to compare number of wins each season between Wisconsin, Iowa and Minnesota, but those data are stored in different csv files or data

frames. This does not matter, we can still plot them on the same graph by calling out different data frames for each layer (just like what we did in adding the green reference box in the plot1). However, since now we have 3 different teams, we need to assign different colors to them as we adding their layers, and then we need to set up a legend for those teams. Here is the code:

```
plot2 <- ggplot()+
  geom_line(data=wisconsin,aes(Year, W, colour = "Wisconsin")) +
  geom_line(data=iowa,aes(Year, W, colour = "Iowa")) +
  geom_line(data=minnesota,aes(Year, W, colour = "Minnesota")) +
  # Pair teams with colors
  scale_colour_manual(values = c('Wisconsin' = 'red', 'Iowa' = 'blue', 'Minnesota' = 'tan'))+
  labs(colour = 'Football Teams') + # Title of the Legend
  ggtitle("Number of Wins: Three Teams Comparison")

plot2
```

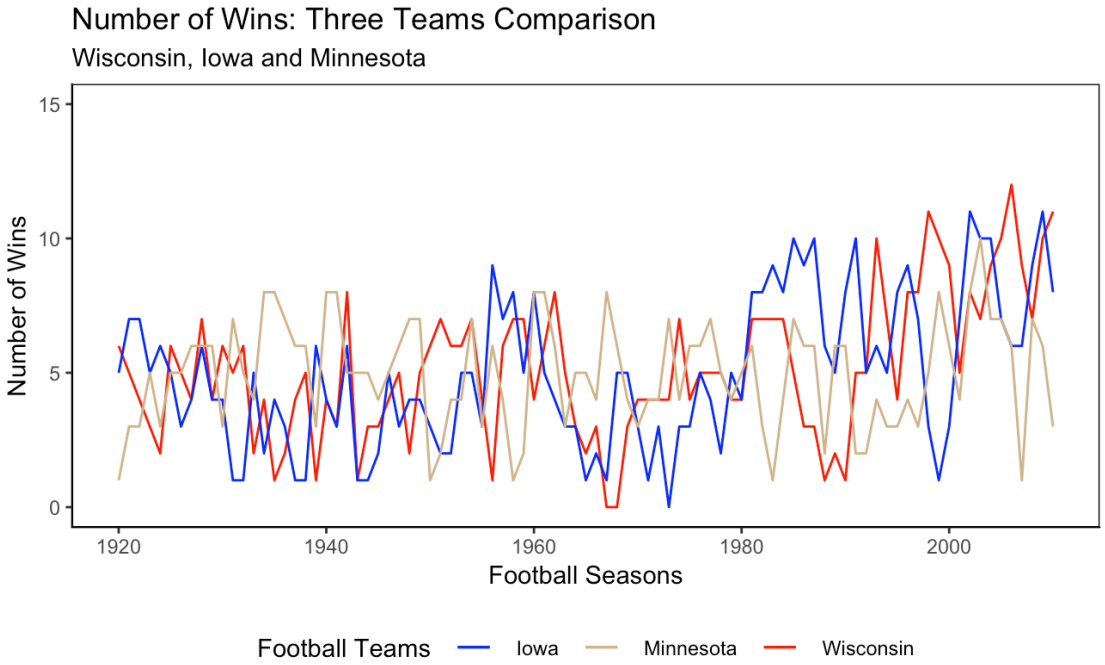
And we will see this:



Now let's set up styles:

```
plot2 <- ggplot()+
  geom_line(data=wisconsin,aes(Year, W, colour = "Wisconsin")) +
  geom_line(data=iowa,aes(Year, W, colour = "Iowa")) +
  geom_line(data=minnesota,aes(Year, W, colour = "Minnesota")) +
  # Pair teams with colors
  scale_colour_manual(values = c('Wisconsin' = 'red', 'Iowa' = 'blue', 'Minnesota' = 'tan'))+
  labs(colour = 'Football Teams') + # Title of the Legend
  ggtitle("Number of Wins: Three Teams Comparison", "Wisconsin, Iowa and Minnesota") + # Subtitle
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), panel.border = element_rect(colour = "black", fill=NA),
        axis.line.x = element_line(colour = "black"), axis.line.y= element_line(colour = "black"),
        legend.position = "bottom", # Move the legend to the bottom
        legend.spacing.x = unit(10, 'pt'), # 10 point spacing between legend elements
        legend.background = element_rect(colour = NA), # Remove the gray background of legend
        legend.key = element_rect(colour = NA, fill = NA) # Do not add boxes around legend
  ) +
  scale_x_continuous(name = expression("Football Seasons"), limits = c(1920, 2010)) +
  scale_y_continuous(name = expression("Number of Wins"), limits = c(0, 15))

plot2
```

And now we get this:



The legend can be put at "top", "bottom", "left" or "right". In some cases, if you want to add a box around the legend, you can change it in the `legend.background` function (set the color to be black or whatever you want). Also, if you want to use colors other than default colors in R, you can use an online color picker (Google "HTML color picker" and you will find a lot of them) or some other color guides (such as the ColorBrewer) to get the Hex number of the color (6 digits following a #), and directly use them in R.

Now, let's do something more challenging. Let's make a plot3 which illustrates the average number of wins of these three teams in each season. Since there is no available data of average wins for us to directly use, we need to process our data a little bit.

The first step is to make a new data frame. Before doing this, let's think about how many rows and columns that we need to use. Coverages of these football data files are from 1896 (the year that Big Ten was established) to 2018, so they should have 123 rows. This is easy to count. If your data is very large or confusing, then you can use `nrow()` function to check out the number of rows. For example, you can type the command of `nrow(wisconsin)` in the Console Window and hit Enter to see how many rows there are in the `wisconsin` data frame. As for columns, we need 5, with names of "Year", "Wisconsin_Wins", "Iowa_Wins", "Minnesota_Wins" and "Average_Wins" respectively.

This can be done by:

```
# Initializing a new data frame named average_data
average_data <- data.frame(
  (matrix(vector(), 123, 5,
          dimnames=list(
            c(), # This is for row names. Since we don't have row names, leave it blank
            c("Year","Wisconsin_Wins", "Iowa_Wins", "Minnesota_Wins", "Average_Wins") # Column names
            )
          )
   ))
```

If we view it, we can see it is an empty data frame (every element is NA) with a name of `average_data`.

	Year	Wisconsin_Wins	Iowa_Wins	Minnesota_Wins	Average_Wins
1	NA	NA	NA	NA	NA
2	NA	NA	NA	NA	NA
3	NA	NA	NA	NA	NA
4	NA	NA	NA	NA	NA
5	NA	NA	NA	NA	NA
6	NA	NA	NA	NA	NA
7	NA	NA	NA	NA	NA
8	NA	NA	NA	NA	NA
9	NA	NA	NA	NA	NA
10	NA	NA	NA	NA	NA
11	NA	NA	NA	NA	NA
12	NA	NA	NA	NA	NA
13	NA	NA	NA	NA	NA

Now, let's assign known data to corresponding columns.

```
average_data$Year <- wisconsin$Year
average_data$Wisconsin_Wins <- wisconsin$W
average_data$Iowa_Wins <- iowa$W
average_data$Minnesota_Wins <- minnesota$W
```

Let's take a look at the Line 78:

`average_data$Year <- wisconsin$Year`

This line of code means that assign data in the column with a name of "Year" in data frame `wisconsin` to the column with a name of "Year" in data frame `average_data`.

Now, let's view this `average_data` data frame:

	Year	Wisconsin_Wins	Iowa_Wins	Minnesota_Wins	Average_Wins
1	2018	8	9	7	NA
2	2017	13	8	5	NA
3	2016	11	8	9	NA
4	2015	10	12	6	NA
5	2014	11	7	8	NA
6	2013	9	8	8	NA
7	2012	8	4	6	NA
8	2011	11	7	3	NA
9	2010	11	8	3	NA

Now, we got data in columns other than Average_Wins.

Sometimes when we are dealing with big datasets, we only want to process the portion of data that we are interested in. In this case, say, we are only interested in data after 1920. Then, we can subset this data frame and let it only contains data after 1920.

The first step is that we need to locate which row that 1920 is in. Type the following command in the Console Window and hit Enter:

`which(average_data$Year==1920)`

This will return a value of 99, which means that 1920 is located in row 99. Then, let's subset this data frame and let it only contains rows 1 to 99.

```
# Subsetting the data frame
average_data <- average_data[1:99,1:5] # Keep rows 1 to 99 and columns 1 to 5; drop everything else
```

Now, our new `average_data` only contains data after 1920.

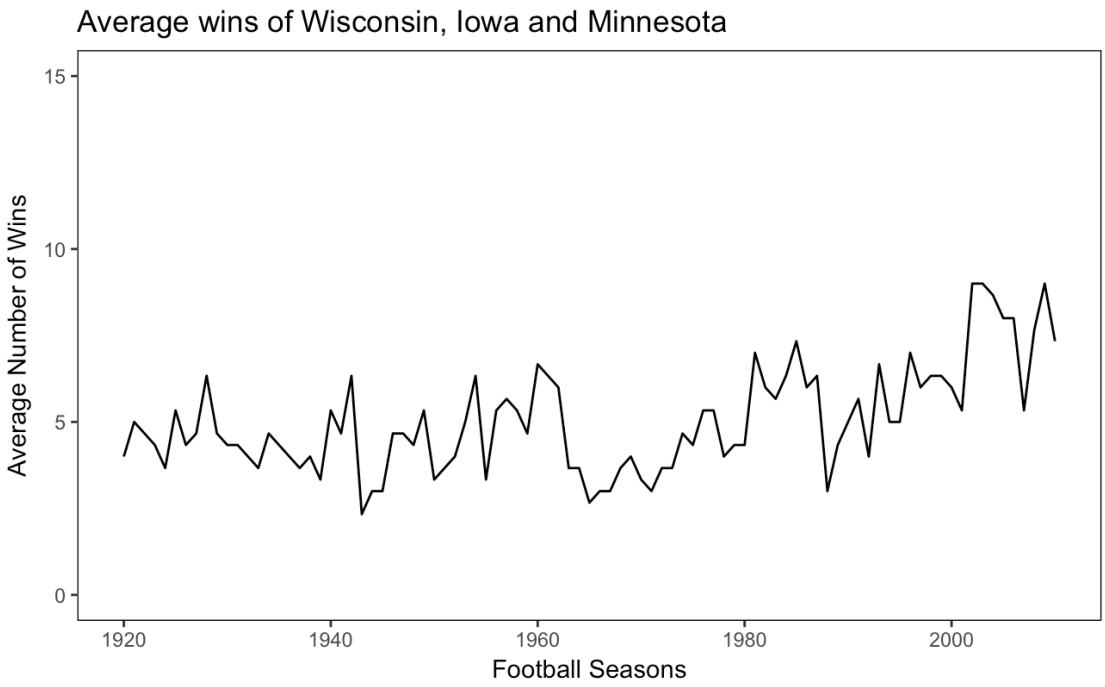
Next, let's calculate the average wins at each year.

```
# Average wins calculation
average_data$Average_Wins <- rowMeans(average_data[,2:4], # Only average columns 2 to 4 in average_data
                                      na.rm = TRUE, # Ignoring missing values
                                      dims = 1)
```

The `rowMeans()` is one of existing functions in R. You do not need to import any other packages to use them. The first parameter in the `()` is the data frame to use; the second parameter is whether to omit missing values; the third parameter is set the dimension of calculation (usually 1). Note that in the `[]`, nothing is entered before the comma. This is a common syntax in R, which means "to include all rows". Similarly, there are `rowSums()`, `colMeans()` and `colSums()` functions which are all very useful functions. With these functions, you no longer have to write for-loops (potentially with if statements for missing values) to do these types of simple calculations.

Now, the Average_Wins column is filled with data. We can then make our plot3:

```
plot3 <- ggplot()+
  geom_line(data=average_data,aes(Year, Average_Wins)) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), panel.border = element_rect(colour = "black", fill=NA)) +
  ggtitle("Average wins of Wisconsin, Iowa and Minnesota") +
  scale_x_continuous(name = expression("Football Seasons"), limits = c(1920, 2010)) +
  scale_y_continuous(name = expression("Average Number of Wins"), limits = c(0, 15))

plot3
```



In plot4, we will plot the winning percentage of Wisconsin in each season. Again, we do not have existing data available to directly plot. We need to extend the `wisconsin` data frame and make some calculations.

To calculate the winning percentage, we also need to know the total numbers of games in each season. Therefore, we need to add 2 more columns to the `wisconsin` data frame: "Total_Games" and "Percentage".

```
# Preparing data for plot4: calculating winning percentages of Wisconsin
# initializing 2 empty columns for wisconsin
wisconsin$Total_Games <- NA
wisconsin$Percentage <- NA
```

Now, if we view the `wisconsin` data frame, we could see those 2 new empty columns.

	Year	W	L	T	Bowl	Total_Games	Percentage
1	2018	8	5	0	Pinstripe Bowl-W	*NA*	*NA*
2	2017	13	1	0	Orange Bowl-W	*NA*	*NA*
3	2016	11	3	0	Cotton Bowl-W	*NA*	*NA*
4	2015	10	3	0	Holiday Bowl-W	*NA*	*NA*
5	2014	11	3	0	Outback Bowl-W	*NA*	*NA*
6	2013	9	4	0	Capital One Bowl-L	*NA*	*NA*
7	2012	8	6	0	Rose Bowl-L	*NA*	*NA*

Now, let's use the `rowSums()` function to sum up columns W, L and T (2, 3 and 4) and add these results to column Total_Games. Then, calculate the Percentage.

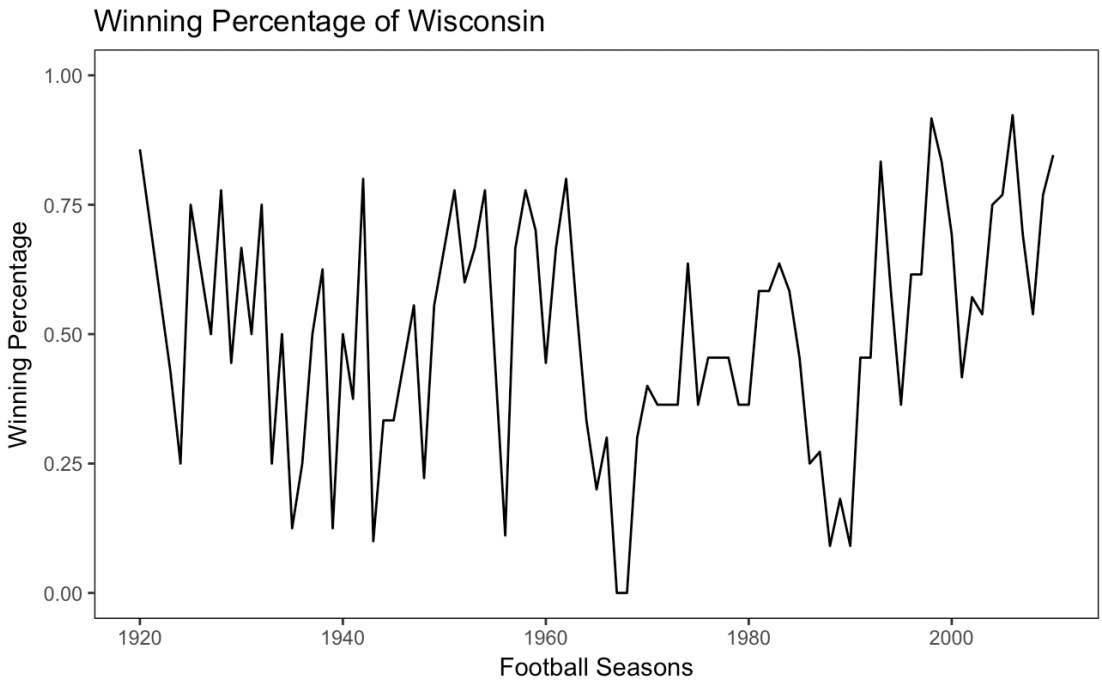
```
106  # calculate total games and percentages
107  wisconsin$Total_Games <- rowSums (wisconsin[,2:4], na.rm = FALSE, dims = 1)
108  wisconsin$Percentage <- wisconsin$W / wisconsin$Total_Games
```

	Year	W	L	T	Bowl	Total_Games	Percentage
1	2018	8	5	0	Pinstripe Bowl-W	13	0.61538462
2	2017	13	1	0	Orange Bowl-W	14	0.92857143
3	2016	11	3	0	Cotton Bowl-W	14	0.78571429
4	2015	10	3	0	Holiday Bowl-W	13	0.76923077
5	2014	11	3	0	Outback Bowl-W	14	0.78571429
6	2013	9	4	0	Capital One Bowl-L	13	0.69230769
7	2012	8	6	0	Rose Bowl-L	14	0.57142857
8	2011	11	3	0	Rose Bowl-L	14	0.78571429
9	2010	11	2	0	Rose Bowl-L	13	0.84615385
10	2009	10	3	0	Champs Sports Bowl-W	13	0.76923077
11	2008	7	6	0	Champs Sports Bowl-L	13	0.53846154
12	2007	9	4	0	Outback Bowl-L	13	0.69230769

Now, let's make plot4:

```
110  plot4 <- ggplot()+
111    geom_line(data=wisconsin,aes(Year, Percentage)) +
112    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
113          panel.background = element_blank(), panel.border = element_rect(colour = "black", fill=NA)) +
114    ggtitle("Winning Percentage of Wisconsin") +
115    scale_x_continuous(name = expression("Football Seasons"), limits = c(1920, 2010)) +
116    scale_y_continuous(name = expression("Winning Percentage"), limits = c(0, 1))
117
118  plot4
```

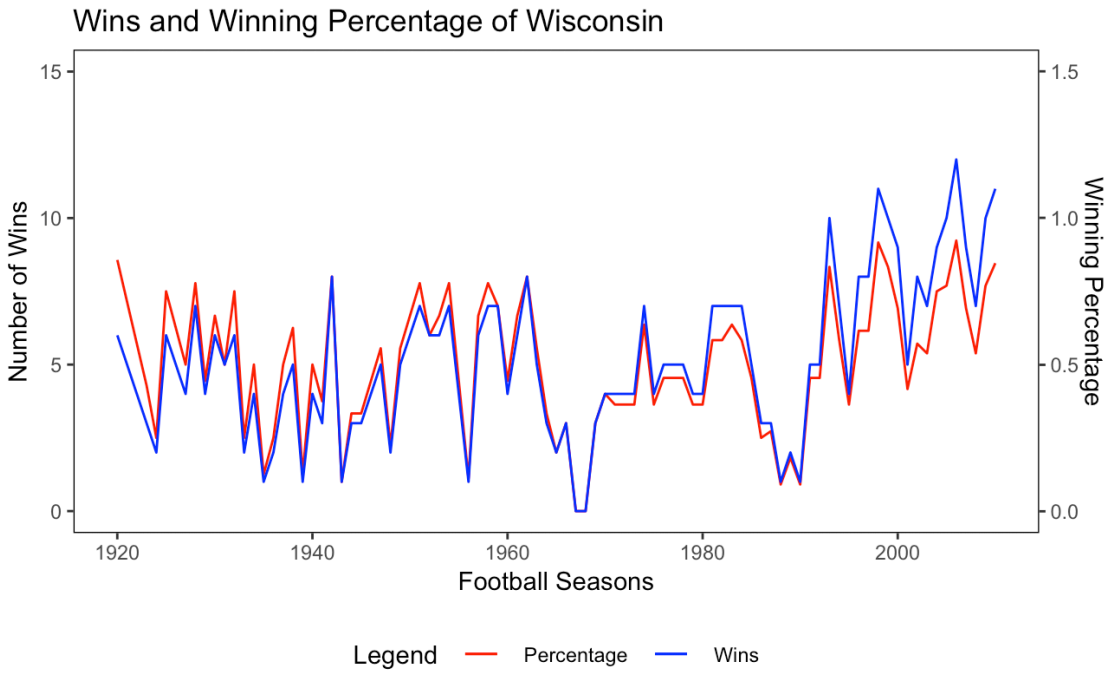
And the plot is



Then, what if we want to plot the number of wins and winning percentages on the same graph? If we use the same y axis scale, the winning percentages might be too small to be clearly shown (percentages are at most 1, while number of wins could be higher than 10). In this case, we need to use a secondary y axis so that different data can be plotted at different scales.

```
plot4 <- ggplot()+
  geom_line(data=wisconsin,aes(Year, Percentage*10, colour = "Percentage")) + # Plot percentage * 10
  geom_line(data=wisconsin,aes(Year, W, colour = "Wins")) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), panel.border = element_rect(colour = "black", fill=NA),
        legend.position = "bottom", legend.spacing.x = unit(10, 'pt'),
        legend.background = element_rect(colour = NA), legend.key = element_rect(colour = NA, fill = NA)
        ) +
  ggtitle("Wins and Winning Percentage of Wisconsin") +
  labs(colour = "Legend")+
  scale_colour_manual(values = c('Percentage' = 'red', 'Wins' = 'blue'))+
  scale_x_continuous(name = expression("Football Seasons"), limits = c(1920, 2010)) +
  scale_y_continuous(name = expression("Number of Wins"), limits = c(0, 15),
                     sec.axis = sec_axis(~./10, name = "Winning Percentage")) # scale down the 2nd axis

plot4
```

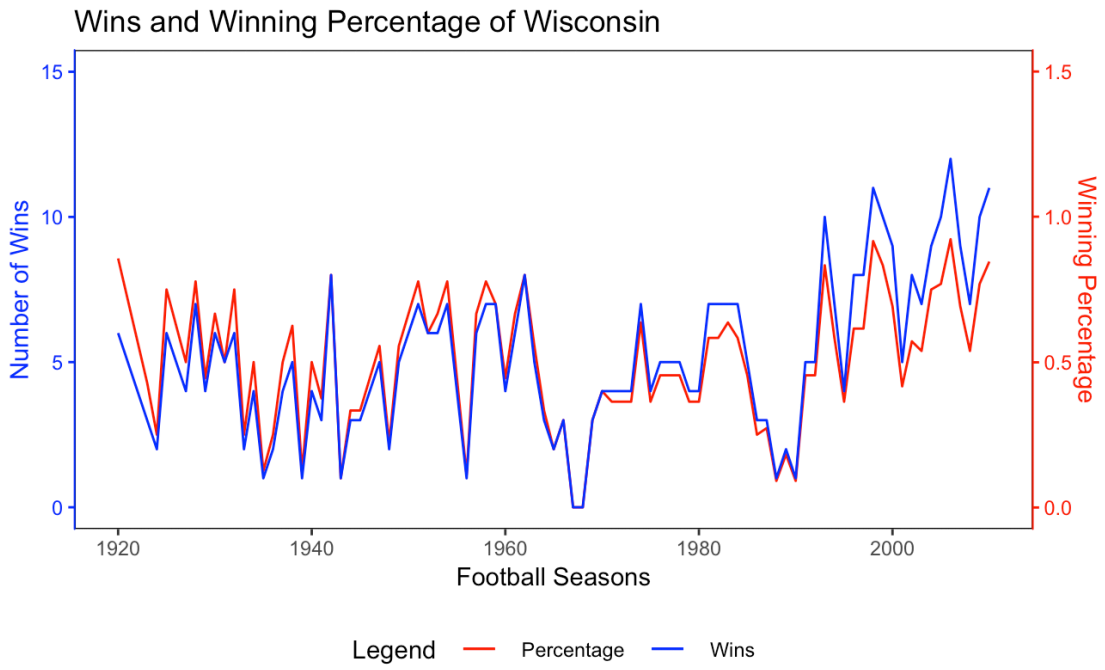
And the plot is like this:



We can also change colors of y axis to be the same as data curves:

```r
plot4 <- ggplot()+
  geom_line(data=wisconsin,aes(Year, Percentage*10, colour = "Percentage")) + # Plot percentage * 10
  geom_line(data=wisconsin,aes(Year, W, colour = "Wins")) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), panel.border = element_rect(colour = "black", fill=NA),
        axis.line.y.left = element_line(colour = "blue"), # set left axis elements to be blue
        axis.title.y.left = element_text(colour = "blue"),
        axis.text.y.left = element_text(colour = "blue"),
        axis.ticks.y.left = element_line(colour = "blue"),
        axis.line.y.right = element_line(colour = "red"), # set right axis elements to be red
        axis.title.y.right = element_text(colour = "red"),
        axis.text.y.right = element_text(colour = "red"),
        axis.ticks.y.right = element_line(colour = "red"),
        legend.position = "bottom", legend.spacing.x = unit(10, 'pt'),
        legend.background = element_rect(colour = NA), legend.key = element_rect(colour = NA, fill = NA)
        ) +
  ggtitle("Wins and Winning Percentage of Wisconsin") +
  labs(colour = "Legend")+
  scale_colour_manual(values = c('Percentage' = 'red', 'Wins' = 'blue'))+
  scale_x_continuous(name = expression("Football Seasons"), limits = c(1920, 2010)) +
  scale_y_continuous(name = expression("Number of Wins"), limits = c(0, 15),
                     sec.axis = sec_axis(~./10, name = "Winning Percentage")) # scale down the 2nd axis

plot4
```

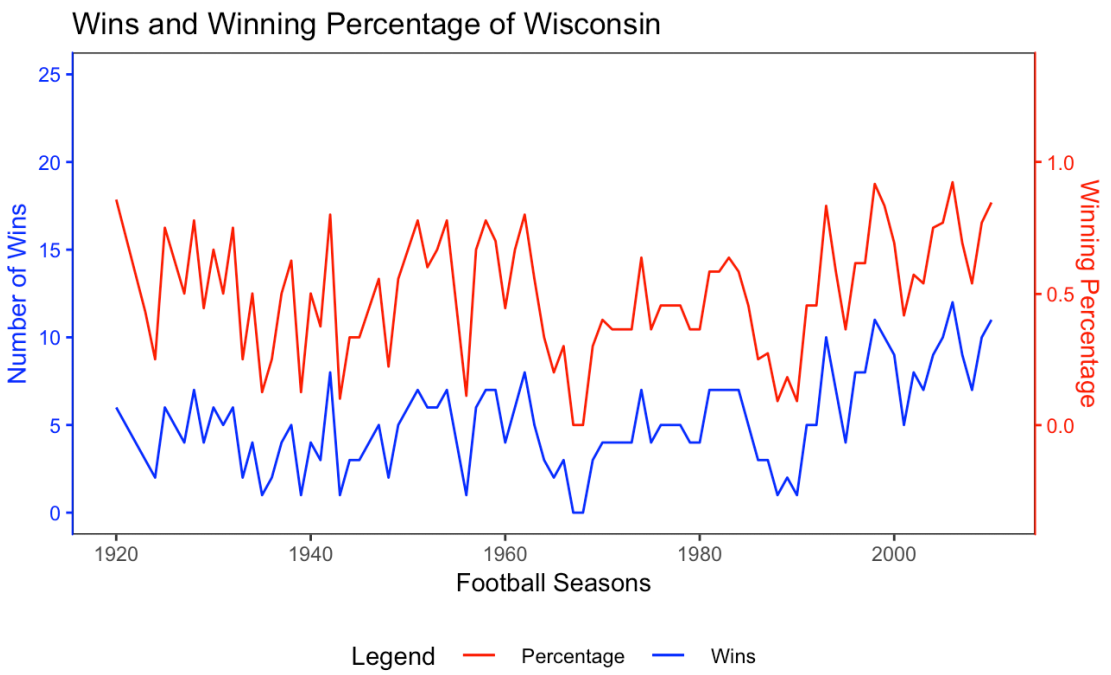
The plot:



We can also adjust scaling of data curves and the secondary y axis to avoid most of the overlapping:

```r
plot4 <- ggplot()+
  geom_line(data=wisconsin,aes(Year, Percentage*15+5, colour = "Percentage")) + # Plot percentage * 10
  geom_line(data=wisconsin,aes(Year, W, colour = "Wins")) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), panel.border = element_rect(colour = "black", fill=NA),
        axis.line.y.left = element_line(colour = "blue"), # set left axis elements to be blue
        axis.title.y.left = element_text(colour = "blue"),
        axis.text.y.left = element_text(colour = "blue"),
        axis.ticks.y.left = element_line(colour = "blue"),
        axis.line.y.right = element_line(colour = "red"), # set right axis elements to be red
        axis.title.y.right = element_text(colour = "red"),
        axis.text.y.right = element_text(colour = "red"),
        axis.ticks.y.right = element_line(colour = "red"),
        legend.position = "bottom", legend.spacing.x = unit(10, 'pt'),
        legend.background = element_rect(colour = NA), legend.key = element_rect(colour = NA, fill = NA)
        ) +
  ggtitle("Wins and Winning Percentage of Wisconsin") +
  labs(colour = "Legend")+
  scale_colour_manual(values = c('Percentage' = 'red', 'Wins' = 'blue'))+
  scale_x_continuous(name = expression("Football Seasons"), limits = c(1920, 2010)) +
  scale_y_continuous(name = expression("Number of Wins"), limits = c(0, 25),
                     sec.axis = sec_axis(~(.-5)/15, name = "Winning Percentage")) # scale down the 2nd a

plot4
```

The plot:



Then, how to save this plot?

You need to create a new folder within your directory named img (or whatever you want). The, you can use `ggsave()` function to save your **latest** plot that was made by `ggplot()` function. Type this command in the Console Window:

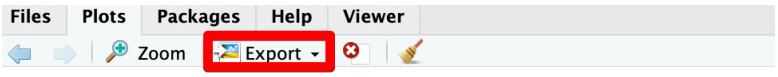
```
ggsave("img/plot4.jpg", plot = last_plot(), width = 50, height = 40, units = "cm", dpi = 300)
```

Or

```
ggsave("plot4.jpg", plot = last_plot(), path = "img", width = 50, height = 40, units = "cm", dpi = 300)
```

Units could be “cm”, “mm” or “in”, and the format could be svg, tiff, jpg, png, gif and pdf.

If a plot is not made by `ggplot()`, such as the next plot we are making below, or made by base plot functions in R, then you can save it by clicking the “export” button at the lower right window of RStudio, and manually adjust the size in the GUI and save it to your folder.



Multi-panel figures

Now, what if we want to combine plots 1 ~ 4 into a single figure? There are multiple ways to do this. Here, I am only showing one simpler way.

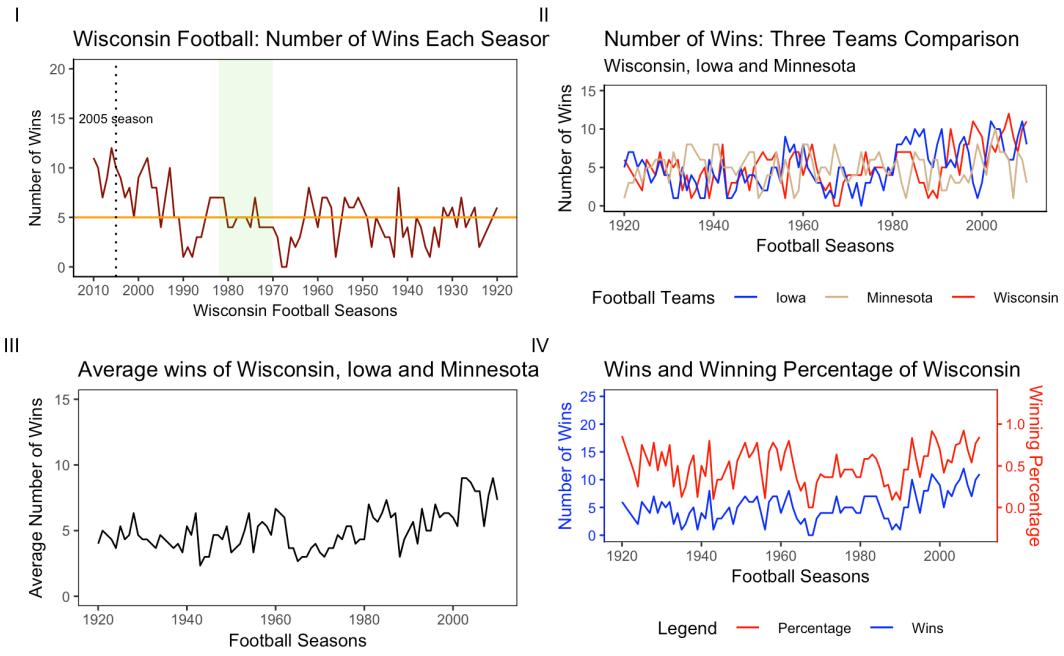
Firstly, we need to import 2 more packages:

```
library(ggplot2)
library(maps)
library(ggmap)
library(readr)
library(magrittr)
library(multipanelfigure)
```

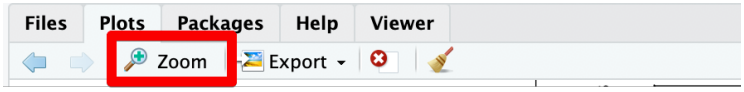
Then, we can construct a multi-panel figure:

```
# set up the layout of plot 5: 2 rows and 2 columns (for plots 1 ~ 4)
plot5 <- multi_panel_figure(columns = 2, rows = 2, panel_label_type = "upper-roman")

plot5 %<>%
  fill_panel(plot1, column = 1, row = 1) %<>%
  fill_panel(plot2, column = 2, row = 1) %<>%
  fill_panel(plot3, column = 1, row = 2) %<>%
  fill_panel(plot4, column = 2, row = 2)
plot5
```

The result is this:



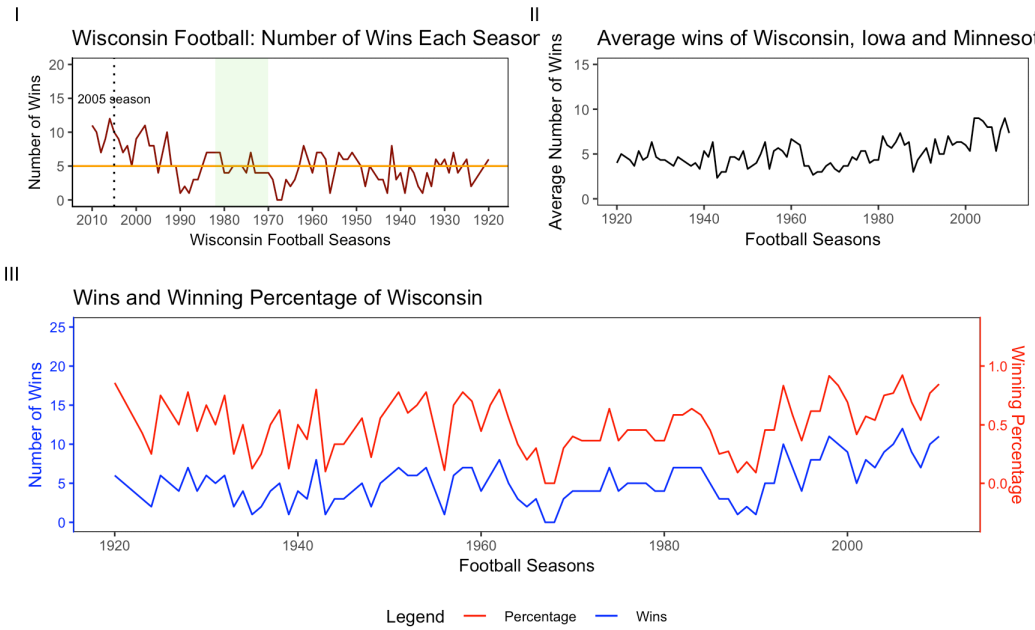
Sometimes, this multi-panel figure cannot be correctly shown in the default lower-right window of RStudio. In this case, you only need to click on "zoom" button to view it correctly.



We can also re-arrange multi-panel figures in more details. For example:

```
# set up a new layout with 2 columns and 3 rows
plot6 <- multi_panel_figure(columns = 2, rows = 5, panel_label_type = "upper-roman")

plot6 %<>%
  fill_panel(plot1, column = 1, row = 1:2) %<>% # plot 1 occupies 2 upper left panels
  fill_panel(plot3, column = 2, row = 1:2) %<>% # plot 3 occupies 2 upper right panels
  fill_panel(plot4, column = 1:2, row = 3:5) # plot 4 occupies the 4 panels in rows 3, 4 and 5
plot6
```

And the result is:



Creating repeating plots automatically

Next, what if I have a data frame with hundreds of columns, and I want to make a plot for each of those columns? In this case, we need to write a for-loop to let RStudio plot them for us. Say, I want to plot Wisconsin_Win, Iowa_Win and Minnesota_Win in the `average_data` data frame (columns 2 to 4) automatically, then this is what I should do.

This is the logic (pseudo code) in this process:

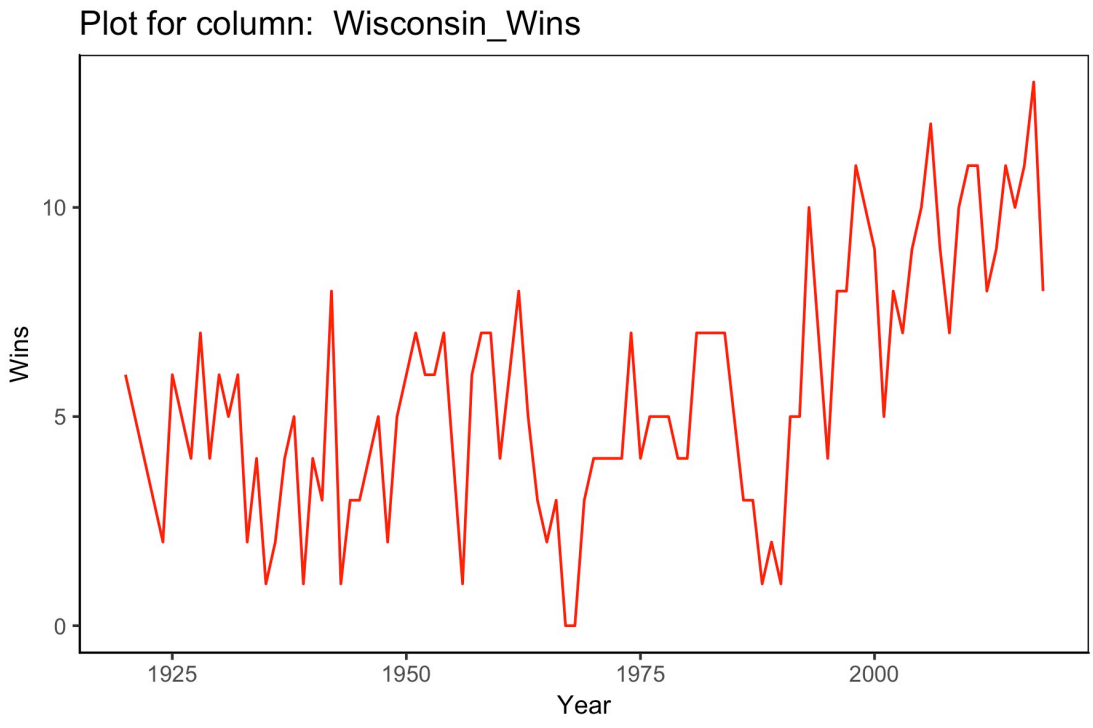
```
for col in (all columns to plot){
    construct a temporary data frame for this col;
    use ggplot() to plot the temporary data frame's data;
    save this plot to your directory;
    }
```

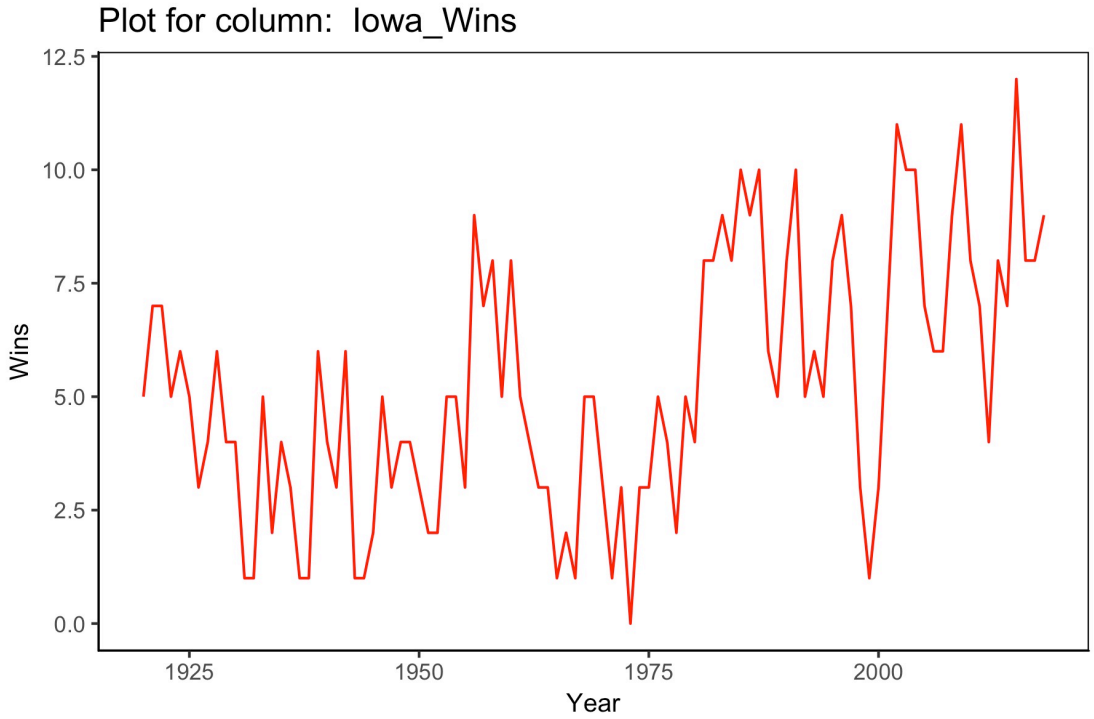
The complete code is shown below:

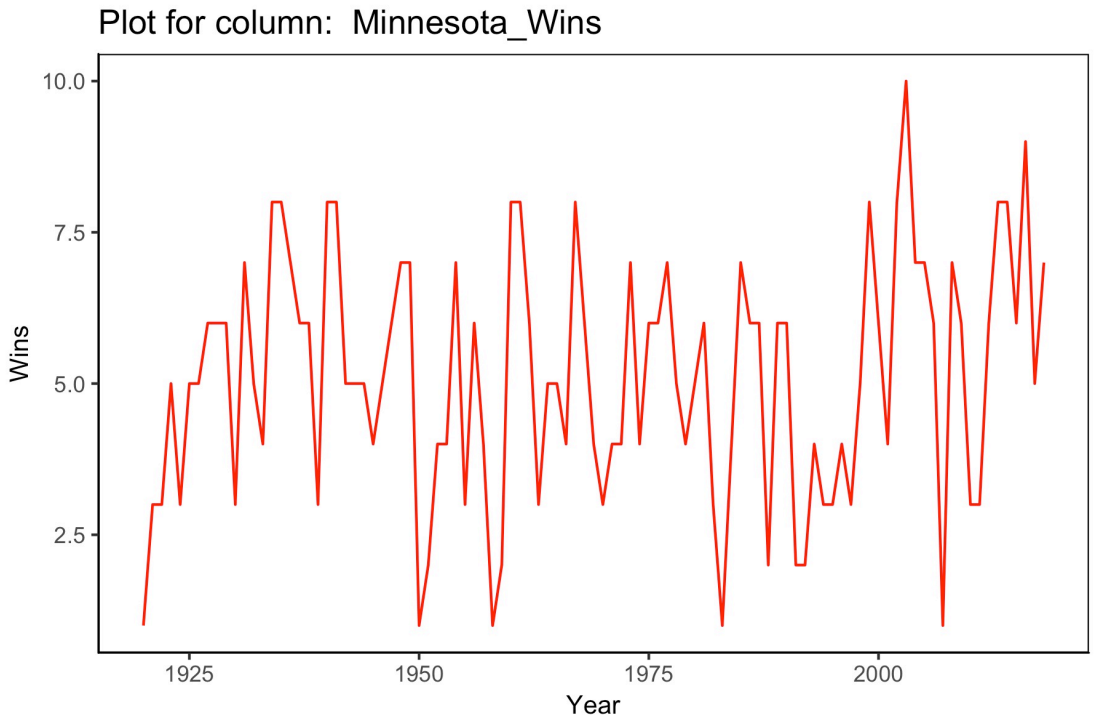
```r
# automatically plot
for(col in 2:4){
  # Get the column name (for the plot title)
  this_name <- names(average_data)[col]
  # construct a temporary data frame
  temp_df <-data.frame(Year=average_data$Year, Wins=average_data[,col])
  # plot the data
  this_plot <- ggplot()+
    geom_line(data=temp_df,aes(Year, Wins), colour = "red") +
    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
          panel.background = element_blank(), panel.border = element_rect(colour = "black", fill=NA),
          axis.line.x = element_line(colour = "black"), axis.title.x=element_text(size=10),
          axis.line.y= element_line(colour = "black"), axis.title.y=element_text(size=10)
    ) +
    ggtitle(paste("Plot for column: ", as.character(this_name)))

  this_plot

  # save the plot to the directory
  ggsave(paste(as.character(this_name), ".jpg"), plot = last_plot(), path = "img",
         width = 6, height = 4, units = "in",
         dpi = 300)
}
```

Note the `paste()` function used in Line 170 and Line 175. It can combine 2 different strings into one single string. The `as.character()` function in these two lines can turn other types of variables into characters so that they can be pasted into other strings. Other lines should be clear.

Then, we will get these figures (already in the directory):







With this method, you can ask R to make hundreds of repeating plots for you.

Creating a map in R

Now, let's move to maps. I am only showing how to make simple maps with point features and labels in this tutorial since these maps are most widely used in geological papers. Making

choropleth maps and some other maps requires additional data processing skills (dealing with shapefile, geojson or other geospatial databases) which is not relevant to this tutorial.

In the data folder, there is another csv file containing geographic information regarding Big Ten Conference schools. Load it to RStudio.

```
180 # import big ten data
181 bigten <- read_csv("data/bigten.csv")
```

	University	Division	lon	lat	National_Championship
1	Northwestern	W	-87.75	41.98	0
2	Wisconsin	W	-89.40	43.04	0
3	Michigan State	E	-84.50	42.67	6
4	Michigan	E	-83.72	42.28	11
5	Ohio State	E	-82.96	39.99	8
6	Penn State	E	-77.85	40.81	2

We can see that it has 5 fields (columns): school name, division, longitude, latitude and national championships. Now, let's plot them onto a map in R.

Create a base map layer:

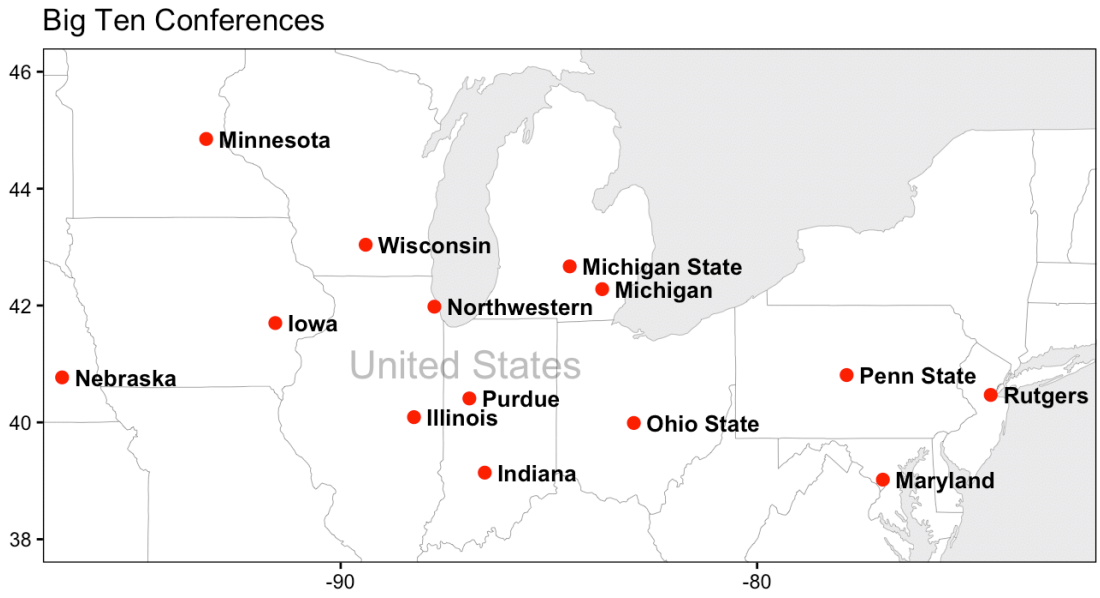
```
182 # add a base map layer
183 ##(this is provided by the "maps" package; commonly used base maps are "usa", "state" and "world")
184 base <- map_data("state")
```

Plot data on the base map:

```
188 map <- ggplot() +
189   geom_polygon(data = base, aes(x=long, y = lat, group = group),
190                fill = "white", color = "black", size = 0.05) +
191   coord_fixed(xlim = c(-96, -73),  ylim = c(38, 46), ratio = 1.4)+
192   geom_point(data = bigten, aes(x = lon, y = lat), colour = "red", size = 2.2) +
193   geom_text(data = bigten, aes(x = lon, y = lat, label = paste("  ", as.character(University), sep="")),
194   theme(
195     axis.line = element_blank(),
196     axis.text = element_text(colour = "black"),
197     axis.ticks = element_line(colour = "black"),
198     panel.border = element_rect(colour = "black", fill = NA),
199     panel.grid = element_blank(),
200     axis.title = element_blank()
201   )+
202   ggtitle("Big Ten Conferences") +
203   annotate("text", x = -87, y = 41, label = "United States", size = 6, colour = "black", alpha = 0.3)
204
205 map
```

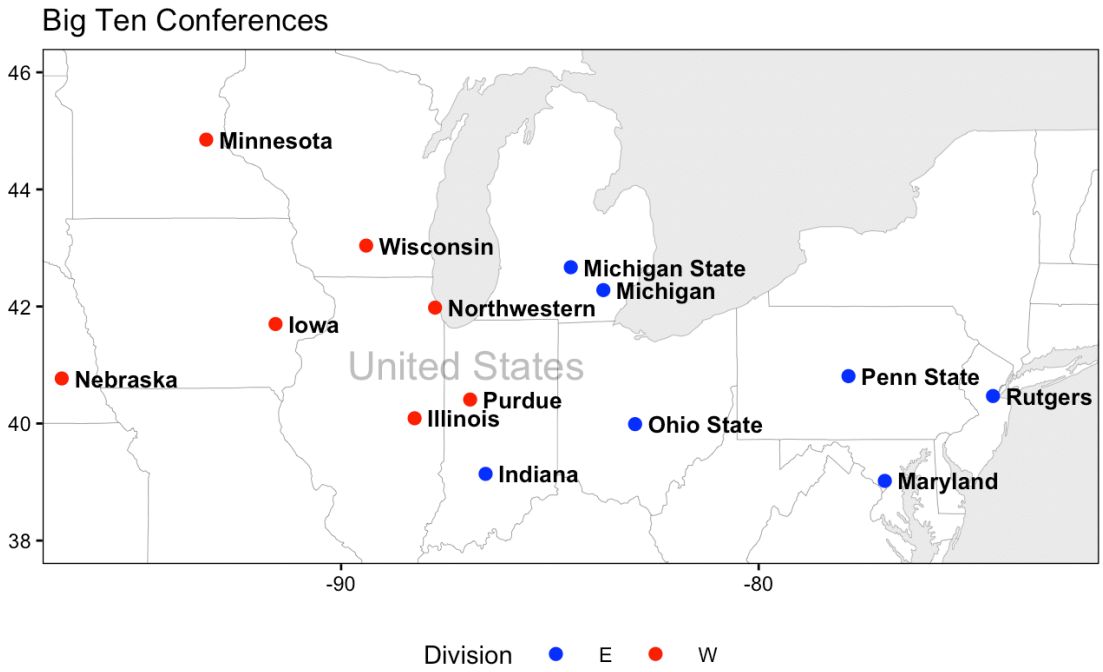
Most functions are very similar to what we have done previously. In Line 191, the `xlim` sets the range of longitude shown on the map, and the `ylim` sets the range of latitude shown on the map. The `ratio` sets the map scale ratio between 1 longitude and 1 latitude.

The map is shown below:



Also, we can plot universities in different colors based on their divisions:

```
map <- ggplot() +
  geom_polygon(data = base, aes(x=long, y = lat, group = group),
               fill = "white", color = "black", size = 0.05) +
  coord_fixed(xlim = c(-96, -73),  ylim = c(38, 46), ratio = 1.4)+
  geom_point(data = bigten, aes(x = lon, y = lat, colour=Division), size = 2.2) +
  scale_colour_manual(values = c('W' = 'red', 'E' = 'blue'))+
  geom_text(data = bigten, aes(x = lon, y = lat, label = paste("  ", as.character(University), sep="")),
  theme(
    axis.line = element_blank(),
    axis.text = element_text(colour = "black"),
    axis.ticks = element_line(colour = "black"),
    panel.border = element_rect(colour = "black", fill = NA),
    panel.grid = element_blank(),
    axis.title = element_blank(),
    legend.position = "bottom",
    legend.spacing.x = unit(10, 'pt'),
    legend.background = element_rect(colour = NA),
    legend.key = element_rect(colour = NA, fill = NA)
  )+
  labs(colour = "Division")+
  ggtitle("Big Ten Conference") +
  annotate("text", x = -87, y = 41, label = "United States", size = 6, colour = "black", alpha = 0.3)

map
```

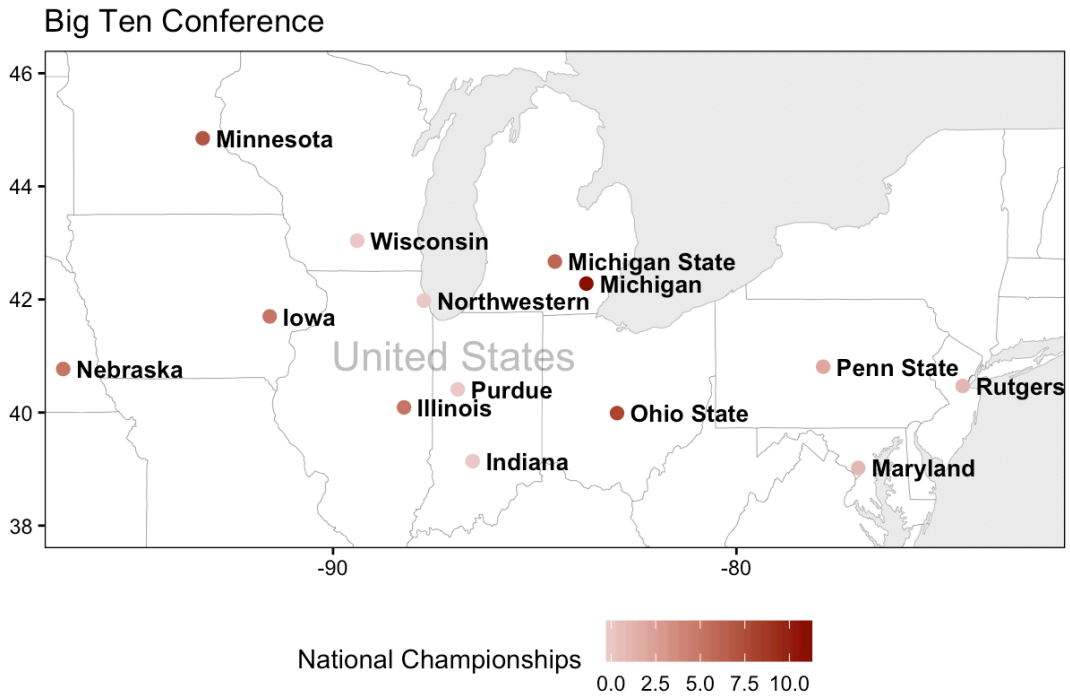


We can also color those points based on the number of football national championships. We can use a gradient color ramp to represent this data field:

```
188 map <- ggplot() +
189   geom_polygon(data = base, aes(x=long, y = lat, group = group),
190                fill = "white", color = "black", size = 0.05) +
191   coord_fixed(xlim = c(-96, -73),  ylim = c(38, 46), ratio = 1.4)+
192   geom_point(data = bigten, aes(x = lon, y = lat, colour=National_Championship), size = 2.2) +
193   scale_colour_gradient2(low=NA, mid="#edcac7",high="dark red",limits=c(0,11), na.value = NA)+
194   geom_text(data = bigten, aes(x = lon, y = lat, label = paste("  ", as.character(University), sep="")),
195   theme(
196     axis.line = element_blank(),
197     axis.text = element_text(colour = "black"),
198     axis.ticks = element_line(colour = "black"),
199     panel.border = element_rect(colour = "black", fill = NA),
200     panel.grid = element_blank(),
201     axis.title = element_blank(),
202     legend.position = "bottom",
203     legend.spacing.x = unit(10, 'pt'),
204     legend.background = element_rect(colour = NA),
205     legend.key = element_rect(colour = NA, fill = NA)
206   )+
207   labs(colour = "National Championships")+
208   ggtitle("Big Ten Conference") +
209   annotate("text", x = -87, y = 41, label = "United States", size = 6, colour = "black", alpha = 0.3)
210
211 map
```

Note that the `mid` parameter in Line 193 represents `0` in the dataset. The `low` parameter goes negative, which is not applicable here.

The map is shown below:



Frequency plot and histogram

Another useful type of plot is frequency/histogram plot, which groups data into bins to visualize the temporal distribution of your data points. Making those plots require a little bit data processing.

For example, we want to visualize the bowl game appearances of Wisconsin every 5 years. To achieve this, we need firstly to process data in Bowl column of the `wisconsin` data frame.

```
213  # frequency plot
214  # In which years that Wisconsin made it to a bowl game?
215  ## Intitiating an empty vector that will contain every year that Wisconsin was at a bowl game
216  bowl <- vector()
```

Then, loop through the Bowl column of the `wisconsin` data frame. Record the value in the Year column if there is a bowl game in the Bowl column in the same row.

```
218  # Loop through the Bowl column of wisconsin data frame
219  for(row in 1:nrow(wisconsin)){
220    #check if there is a bowl game in this row
221    if(!is.na(wisconsin$Bowl[row])){
222      #record the value in the Year columne
223      bowl_year <- wisconsin$Year[row]
224      #add it to the bowl vector
225      bowl <- c(bowl, bowl_year)
226    }
227  }
```

In Line 221, the `is.na()` function will return `TRUE` if the value at `wisconsin$Bowl[row]` is NA or `FALSE` if the value at `wisconsin$Bowl[row]` is not NA. Adding a `!` in front of this function will return an opposite Boolean result. In this case, every year that Wisconsin went to a bowl game has been recorded in the vector `bowl`.

Now, we can turn the `bowl` into a data frame so that we can plot the frequency or histogram.

```
229  # make bowl into a data frame
230  bowl_df <- data.frame(Appearance = bowl)
```

Filter

	Appearance
1	2018
2	2017
3	2016
4	2015
5	2014
6	2013
7	2012
8	2011

Then, we can plot the frequency or histogram in the `ggplot()` function.

```
234  plot7 <- ggplot()+
235    geom_freqpoly(data=bowl_df, aes(x=Appearance), bins=20, colour="purple", size = 1.5)+
236    geom_histogram(data=bowl_df, aes(x=Appearance), binwidth = 5,
237                   fill="orange", alpha=0.5, colour="black")+
238    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
239          panel.background = element_blank(), panel.border = element_rect(colour = "black", fill=NA),
240          axis.line.x = element_line(colour = "black"), axis.title.x=element_text(size=10),
241          axis.line.y= element_line(colour = "black"), axis.title.y=element_text(size=10)
242    ) +
243    ggtitle("Wisconsin Bowl Appearances Every 5 Years")+
244    scale_x_continuous(name = expression("Football Seasons"), limits = c(1920, 2015)) +
245    scale_y_continuous(name = expression("Bowl Appearances"), limits = c(0, 5))
246
247  plot7
```

Note that the `bins` in Line 235 indicates how many bins you want to have in the figure, while the `binwidth` in Line 236 indicates your step size of each bin. The plot is shown below: