Lab 7: Plotting Figures and Error Bars

Introduction and Objective:

An important part of biomedical research is presenting the results. It is very often, the results are presented as figures. R is very good at generating high-quality figures. In today's lab, we are going to learn how to use R to plot our results from statistical analyses. First, by using R built-in functions, we will generate bar charts, line charts, pie charts, as well as plot our result from the frequency distribution. And then, we will learn how to use the ggplot2 package to plot figures and add error bars to bar charts and line graphs.

There is a group-work assignment.

The following four files you should have received and are needed for today's class:

1) Midterm.csv
2) MidtermWithRank.csv
3) Mouse_DNA_methylation_transformed.csv
4) Mouse_DNA_methylation_Aging.csv

All the R command lines are in **bold**; all the notes are following a #; all the R results directly follow the R codes/command lines and are not in bold or following a #.

If you are using a PC, set the working directory like:

setwd("C:/R")

If you are using a Mac set the working directory as we learned in the first lab.

A. Use R built-in functions to plot your data:

Do you still remember how you can import data files into R? Import the csv file "Midterm" into R and deposit it in an object M:

M<-read.csv("Midterm.csv")

Check the dimensions, the header, and the first six rows to get a basic idea of how the data looks like:

dim(M)

```
[1] 37  1
```

head(M)

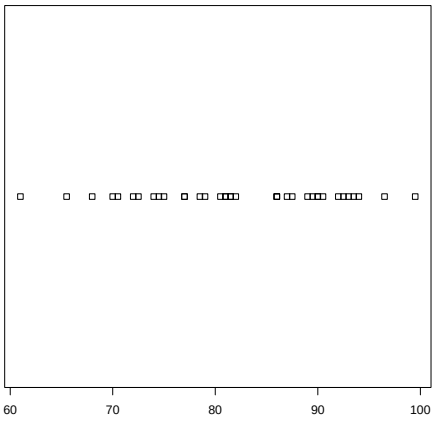
```
  Midterm
1    81.0
2    78.5
3    82.0
4    94.0
5    77.0
6    87.0
```

Once you checked the dimension and the header of the data, you know the data is not that large. So, you may take a look at the entire data table:

View(M)

Now, you are going to generate your first figure using R. There is a built-in R function called “plot()”, which allows you to plot some basic figures. Let’s start with the simplest and basic one:

plot(M)



As you can see, the first figure you generated by using R is not pretty nor delivering much helpful information.

The following steps will make the figure looks much better and more informative.

In the above figure, you may notice that all the little squares representing the scores are actually plotted in one dimension. So, for students who have the same scores, the squares representing their scores overlapped in the above figure. To separate the data points from each other, we need to make a two-dimensional graph. Thus, we have to give each score an I.D. Of course, we can use the student I.D. or the names of the students to solve this problem. But if we only want to check the scattering of the scores without having an interest in knowing who gets what score, we can simply solve this problem in this way:

write.csv(M,"Midterm1.csv")

Do you remember? As we learned previously, if we write a csv file, by default, R will add a column of row numbers to all the rows (If you don't want R to add the row names, you have to specify row.names=FALSE). So, in this way you can create a csv file with all the data (the scores) and another dimension (the row numbers added by R).

Challenge Question 1:

Can you write a text file like write.table(M,"Midterm1.txt") to solve this problem? (The answer is at the end of this note.)

M1<-read.csv("Midterm1.csv")

After you read the "Midterm1" into an object "M1", you can check and see if it has two columns now:

head(M1)

```
  X Midterm
1 1    81.0
2 2    78.5
3 3    82.0
4 4    94.0
5 5    77.0
6 6    87.0
```

Now you can plot the M1 by using the built-in R function "plot":

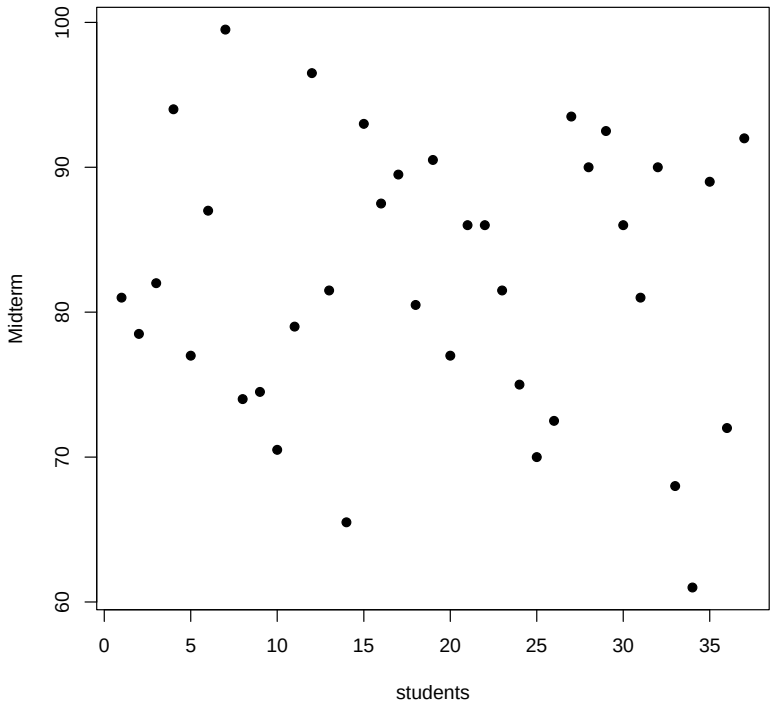
plot(M1,xlab="students",pch=19)

The “xlab” is the argument being used to label the x-axis. Similarly, if you add an argument as “ylab”, it will label the y-axis. Following the “xlab”, and after the “=”, in between the quotation marks is the name of the x-axis you desire to give.

Different numbers of pch give different sizes and shapes of the data points. You can try to change the number of pch to see the effects.

Challenge Question 2:

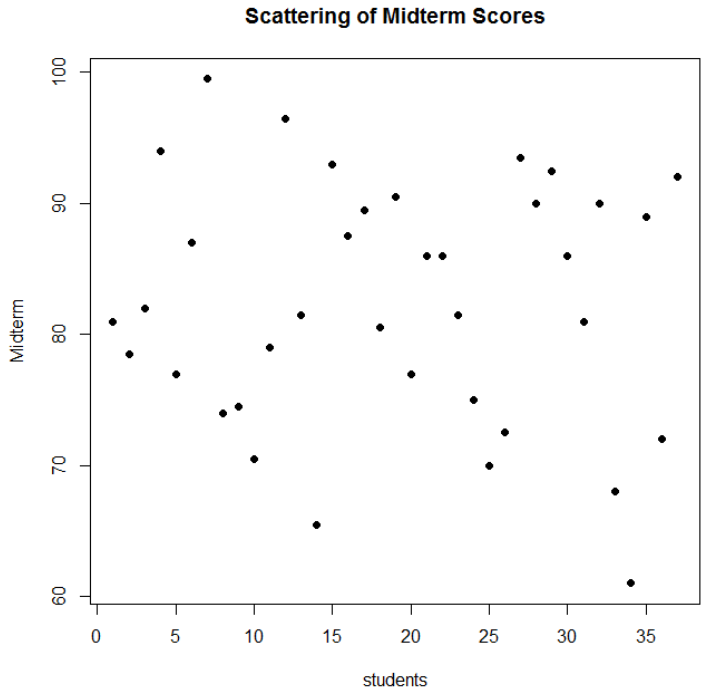
Can you figure out another way to generate the following graph without using “xlab”? (The answer is at the end of this note.)



You can also add a title to the figure:

plot(M1,xlab="students",pch=19,main="Scattering of Midterm Scores")

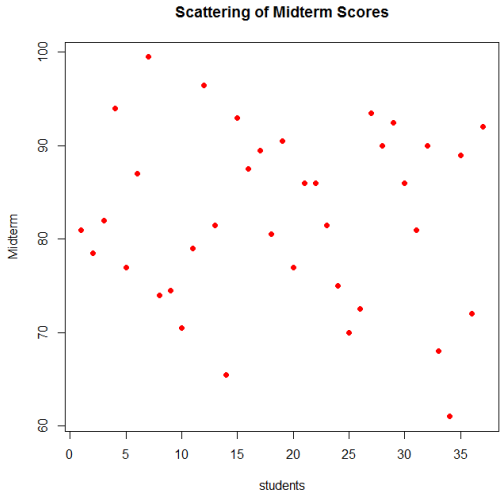
The argument “main” is for adding a title to the figure.



Also, you can change the color of the data points:

plot(M1,xlab="students",pch=19,main="Scattering of Midterm Scores",col="red")

As shown above, using “col”, we can color the data points.

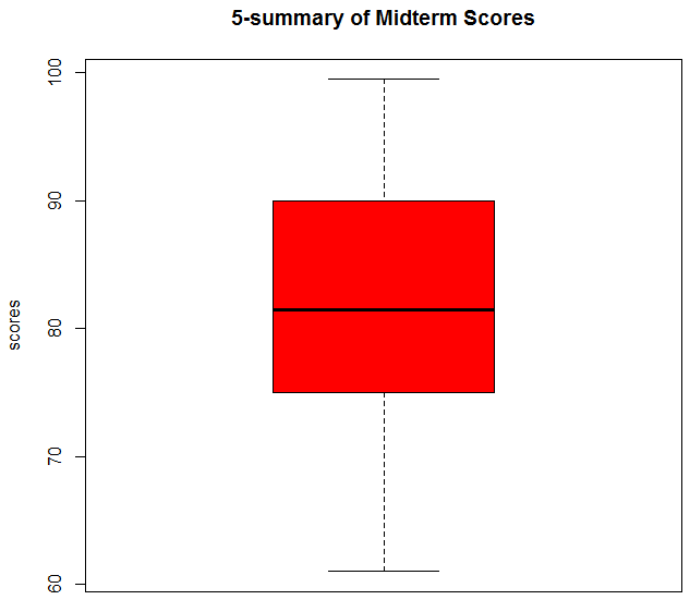


Try it yourself and see what you will get if you change the code only a little bit:

plot(M1,xlab="students",pch=19,main="Scattering of Midterm Scores",col=rainbow(6))

Of course, we can also use another R built-in function, boxplot(), to generate a box-whisker plot and show the five-summary statistics of the midterm scores:

boxplot(M,ylab="scores",main="5-summary of Midterm Scores",col="red")



Challenge Question 3:

For the box-whisker plot, what are the top cap, lower cap, the upper line of the box, the lower line of the box, and the middle line of the box represent, respectively? (The answer is at the end of this note.)

Also, we can create a bar chart and other types of charts.

B. Other R built-in functions

We are going to generate a few graphs using another data file: MidtermWithRank.csv

d<-read.csv("MidtermWithRank.csv")

View(d)

d1<-d[,3]

head(d1)

[1] 81.0 78.5 82.0 94.0 77.0 87.0

mode(d1)

[1] "numeric"

Plotting Frequency Distribution:

We talked about frequency distribution in class when we discussed error bars. Now we may generate a frequency distribution figure to show the distribution of the midterm scores.

range(d1)
[1] 61.0 99.5

cat<-seq(60,100,by=10)

The function "seq()" in the above command line generates a series of numbers from 60 to 100 with an increment of 10 (we learned this last week when we were learning how to process data using R).

cat
[1] 60 70 80 90 100

score.cut<-cut(d1,cat)

The function "cut()" in the above command line tells R to fit every number in the object "d1" in the ranges set by "cat".

score.cut
[1] (80,90] (70,80] (80,90] (90,100] (70,80] (80,90] (90,100] (70,80]
[9] (70,80] (70,80] (70,80] (90,100] (80,90] (60,70] (90,100] (80,90]
[17] (80,90] (80,90] (90,100] (70,80] (80,90] (80,90] (80,90] (70,80]
[25] (60,70] (70,80] (90,100] (80,90] (90,100] (80,90] (80,90] (80,90]
[33] (60,70] (60,70] (80,90] (70,80] (90,100]

Levels: (60,70] (70,80] (80,90] (90,100]

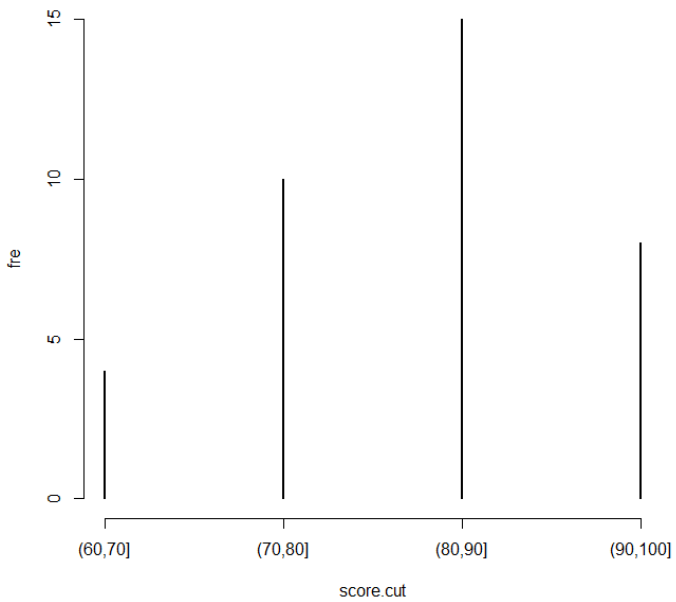
(fre<-table(score.cut))

The function table() will count how many events are at each level.

```
score.cut
 (60,70]  (70,80]  (80,90] (90,100]
       4       10       15        8
```

If in the future you are going to take an upper-level biostatistics course, you will know, indeed, we can plot a histogram to show the frequency distribution, and the function "seq()" is actually generating the "bins", and the function "cut()" is actually put the numbers in the bins, the function "table()" is actually counting the "density".

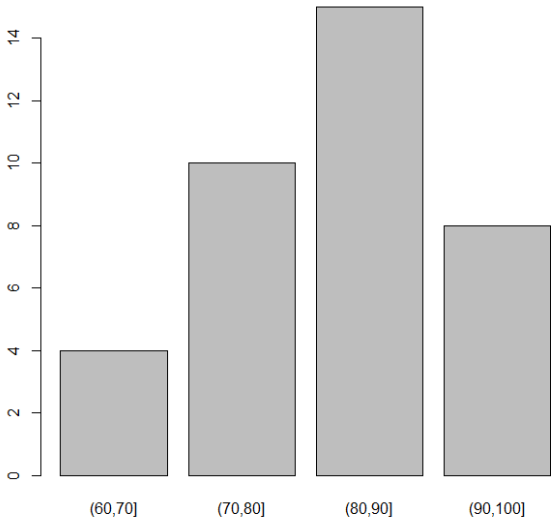
plot(fre)



As you can see, if we use the built-in function "plot()", the figure isn't pretty.

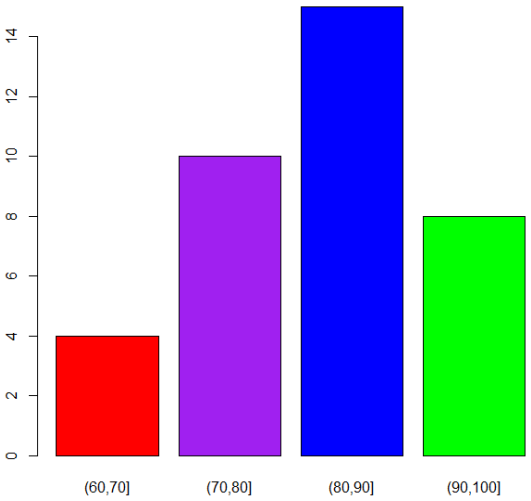
Thus, we may choose to use the following command to make a bar chart:

barplot(fre)



We can color the bars:

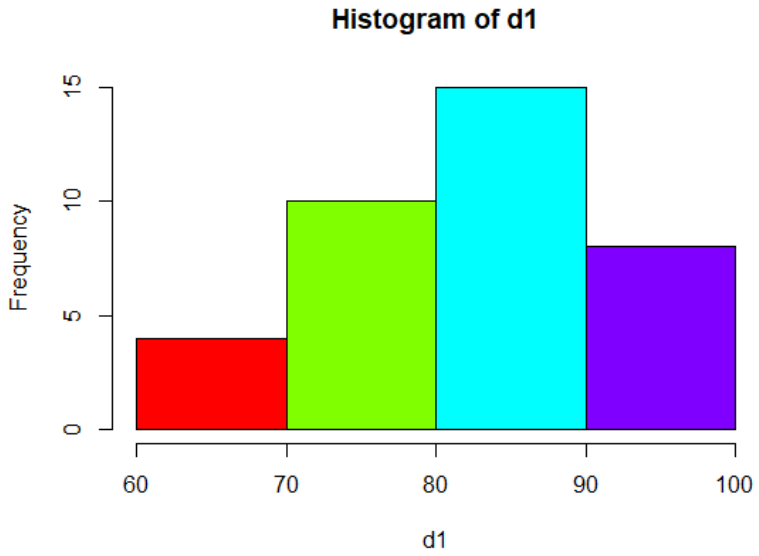
colors<-c("red","purple","blue","green")
barplot(fre,col=colors)



In fact, there is an easier way to generate the histogram by using the R built-in function hist(). You can try and see it yourself by typing the following codes:

hist(d1,breaks=5)

hist(d1,breaks=5,col=rainbow(4))



It is worthwhile to notice that you may add "right=FALSE" to exclude the upper limit scores:

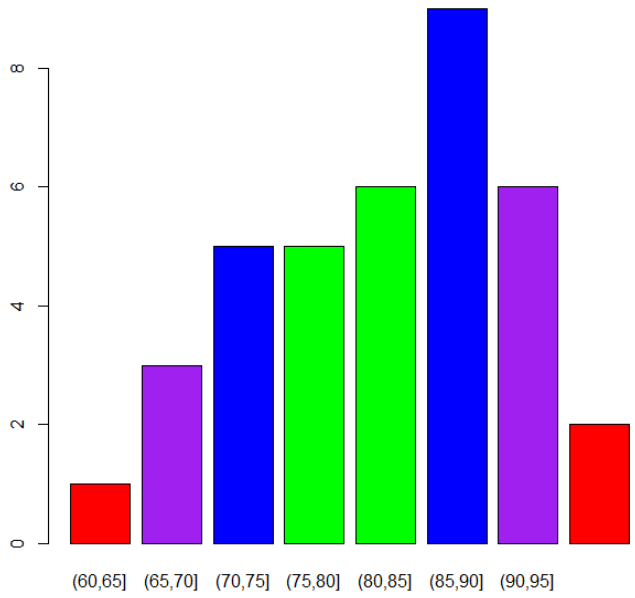
```
score.cut<-cut(d1,cat,right=FALSE)
score.cut
 [1] [80,90)  [70,80)  [80,90)  [90,100) [70,80)  [80,90)  [90,100) [70,80)
 [9] [70,80)  [70,80)  [70,80)  [90,100) [80,90)  [60,70)  [90,100) [80,90)
[17] [80,90)  [80,90)  [90,100) [70,80)  [80,90)  [80,90)  [80,90)  [70,80)
[25] [70,80)  [70,80)  [90,100) [90,100) [90,100) [80,90)  [80,90)  [90,100)
[33] [60,70)  [60,70)  [80,90)  [70,80)  [90,100)
Levels: [60,70) [70,80) [80,90) [90,100)

fre<-table(score.cut)

fre
score.cut
 [60,70)  [70,80)  [80,90) [90,100)
       3       11       13       10
```

You may also be able to change the increment:

```
cat<-seq(60,100,by=5)

cat
[1]  60  65  70  75  80  85  90  95 100
```

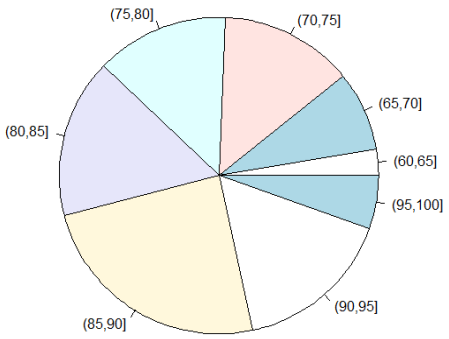
You may add “left=FALSE” to exclude the lower limit scores:

```
score<-cut(d1,cat,left=FALSE)
score
 [1] (80,85]  (75,80]  (80,85]  (90,95]  (75,80]  (85,90]  (95,100] (70,75]
 [9] (75,80]  (70,75]  (75,80]  (95,100] (80,85]  (65,70]  (90,95]  (85,90]
[17] (85,90]  (80,85]  (90,95]  (75,80]  (85,90]  (85,90]  (80,85]  (70,75]
[25] (65,70]  (70,75]  (90,95]  (85,90]  (90,95]  (85,90]  (80,85]  (85,90]
[33] (65,70]  (60,65]  (85,90]  (70,75]  (90,95]
Levels: (60,65] (65,70] (70,75] (75,80] (80,85] (85,90] (90,95] (95,100]

sf<-table(score)
sf
score
(60,65] (65,70] (70,75] (75,80] (80,85] (85,90] (90,95] (95,100]
      1       3       5       5       6       9       6       2

colors<-c("red","purple","blue","green","green","blue","purple","red")
barplot(sf,col=colors)
```



You can create a pie chart:

```
pie(sf)
```

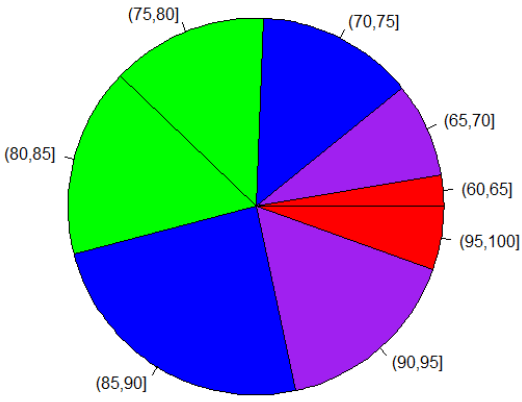


You can also change the color to a variety of colors, by using the argument col=rainbow(). See it yourself:

pie(sf,col=rainbow(40))

And you may use your own colors:

pie(sf,col=colors)



Plotting Cumulative Frequency Distribution:

(cum<-cumsum(fre))

```
[60,70)  [70,80)  [80,90) [90,100)
      3       14       27       37
```

cbind(cum)

```
         cum
[60,70)    3
[70,80)   14
[80,90)   27
[90,100)  37
```

e<-cbind(cum)

The function “cbind()” is for combining data by columns. There is another R built-in function, “rbind()”, for combining data by rows.

plot(e,col="red",pch=19)

We defined “e”. We need to define the x-axis as well. Since it ranges from 1 to 4, we define it as:

x<-1:4

Then we do the general linear regression using “glm()”:

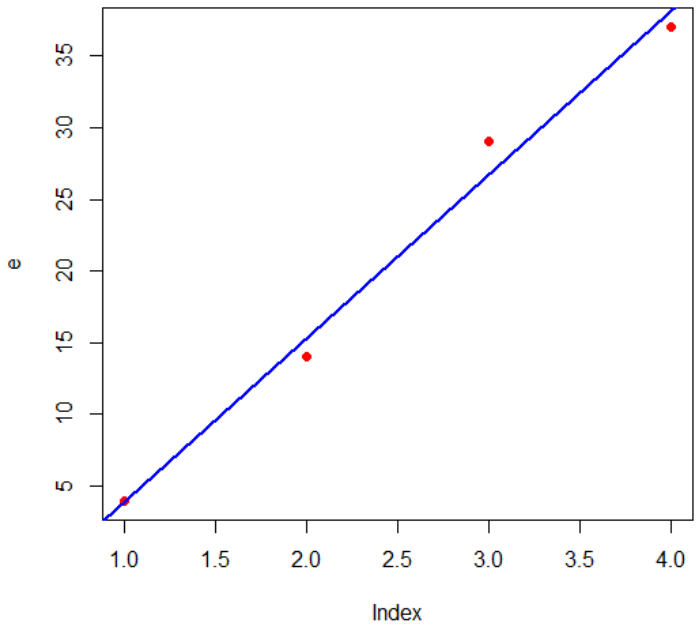
f<-glm(e~x)

After that we find the coefficient of the general linear regression and plot it by using “abline()” to add a straight line:

cf<-coef(f)

abline(cf,col="blue", lwd=2)

The argument lwd= is for line width.



C. Plotting figures using the R package "ggplot2":

We learned how to use the built-in R function plot() to generate scatter plots. Now, we are going to explore other types of plots we may generate by using a powerful R package: ggplot2.

First, we need to download and install this package on our computers (In fact, if you have installed package "tidyverse", which contains package "ggplot2", you only need to activate tidyverse). Remember? There are two different ways to install an R package. Here, we choose to type the command line:

install.packages("ggplot2")

Before we can use the package, we have to activate it:

library(ggplot2)

Now, we are going to use ggplot2 to plot figures. You will see, it is more powerful than using the built-in R functions for plotting figures.

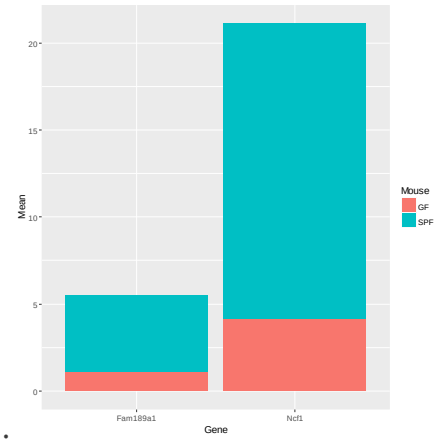
We are going to plot a bar chart using "ggplot2" to demonstrate how to use this powerful R package to plot figures.

s<-read.csv("Mouse_DNA_methylation_transformed.csv")

View(s)

Let's use "ggplot2" to generate a bar chart to plot the data in object "s":

(a<-ggplot(s,aes(x=Gene, y=Mean, fill=Mouse))+geom_bar(stat="identity"))



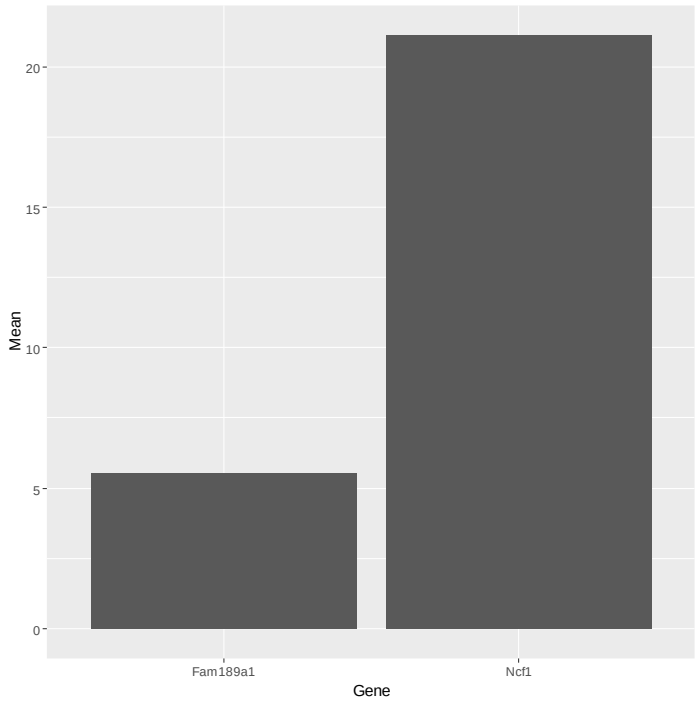
You can see although the name of the package is "ggplot2", the function actually is ggplot() not ggplot2().

Each of the commands after "+" is adding a "layer" to the graph.

The function geom_bar() tells R to plot a bar chart. There are other functions, such as geom_histogram(), geom_points(), geom_line(), geom_boxplot(), geom_density(), and geom_errorbar(), allow you to plot different types of graphs. There is a full list of these functions at http://had.co.nz/ggplot2/

The function aes() is for plotting the aesthetics (the visual properties) of the data. Then, you have to tell R, for the aesthetics, what data is for the x-axis, and what is for the y-axis. After the "fill=", you have to tell R, what the categories of the genes are. There are two types of mice, the "GF" and "SPF", both of which have the genes "Fam189a1" and "Ncf1". If you don't tell R, what the types of mice are, R will plot the data with the same color, which can't allow you to differentiate them from each other. You can try and see, if you don't type fill=Mouse, R will plot the data with the same names on the x-axis together, not side by side. And thus, the bars you get are stacked. And since these stacked bars are not "filled" with a different color, you cannot

differentiate them and will only see the tall bars. You will get something that looks like the following:



The heights of the bars either show the density / a count of cases in each group or indicate the values in a column. By default, the function geom_bar() uses the argument stat="bin". This makes the height of each bar equal to the count of cases in each group. If you want the heights of the bars to represent values in the data, use stat="identity".

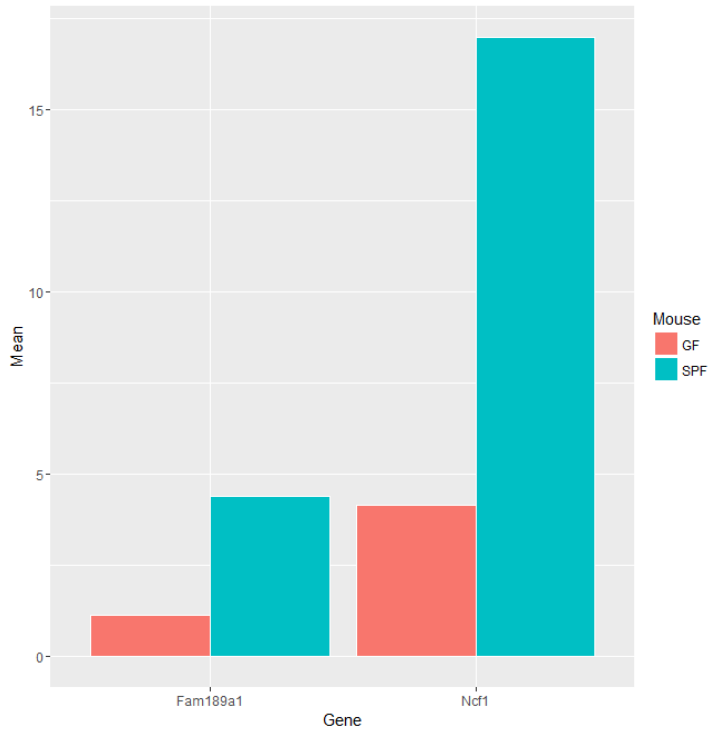
The argument “position=” tells R where the bars should be put. If you don’t tell R where to put the bars, by default, R will stack the bars with the same variable names on the x-axis. If you want the bars to be put side-by-side, you have to tell R the bars to dodge each other. If the bars completely dodge each other, meaning they don’t stack on top of each other, then the position of the bars should be “position=position_dodge()”

If you wish not to have all these bars with the same variable names on the x-axis to stack on top of one another, you can modify your code and tell the bars to dodge each other:

ggplot(s,aes(x=Gene, y=Mean,fill=Mouse))+geom_bar(stat="identity", position="dodge")

or

ggplot(s, aes(x=Gene, y=Mean, fill=Mouse)) + geom_bar(stat="identity", position=position_dodge())

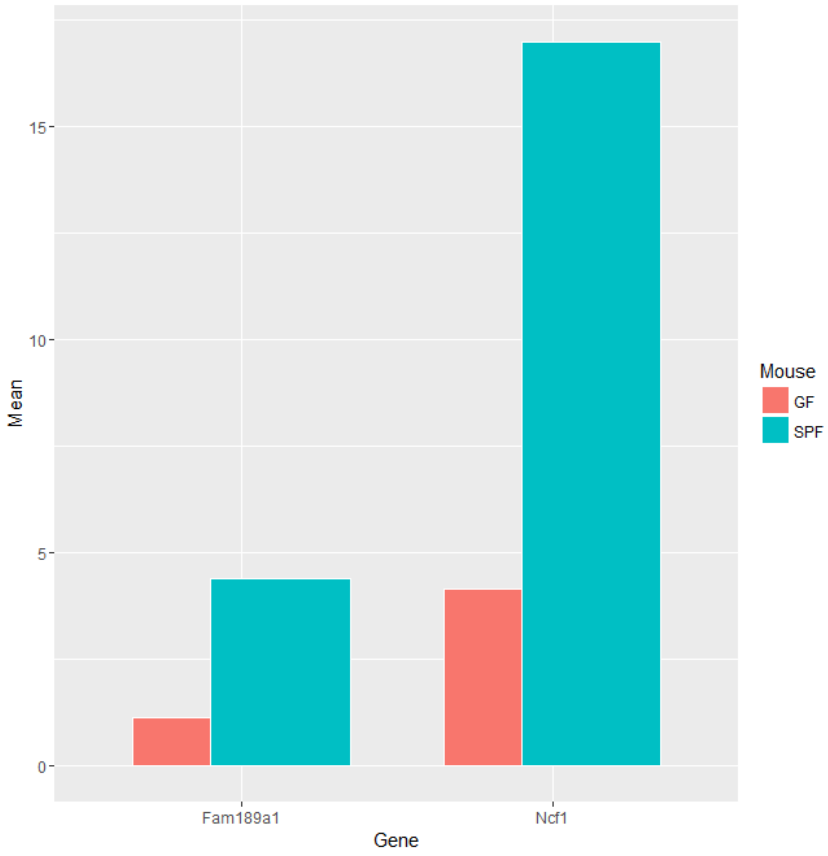


If you tell R, the bars only dodge each other in a half-way, meaning half of the bars overlap each other, then the position is "position=position_dodge(0.5)"

You can test this by typing:

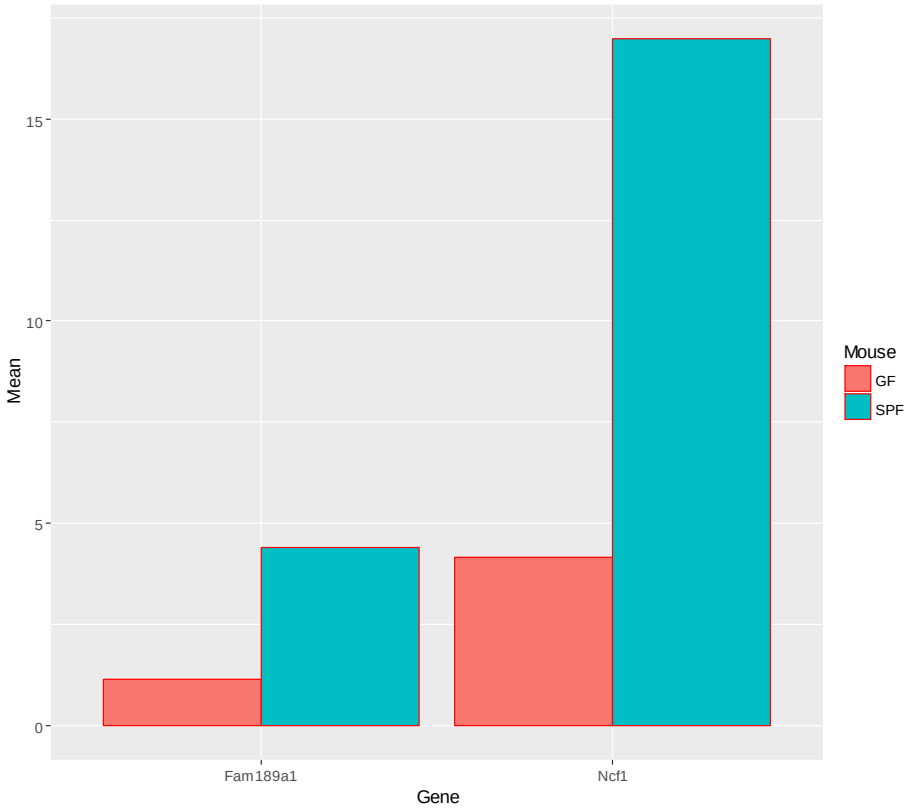
ggplot(s, aes(x=Gene, y=Mean, fill=Mouse))+ geom_bar(stat="identity", position=position_dodge(0.5),colour='white')

And you will get:



Furthermore, you can change the border of the bars to a color you want to use (here, to make it more apparent to see, I chose red):

ggplot(s,aes(x=Gene, y=Mean,fill=Mouse))+geom_bar(stat="identity", position="dodge",colour="red")

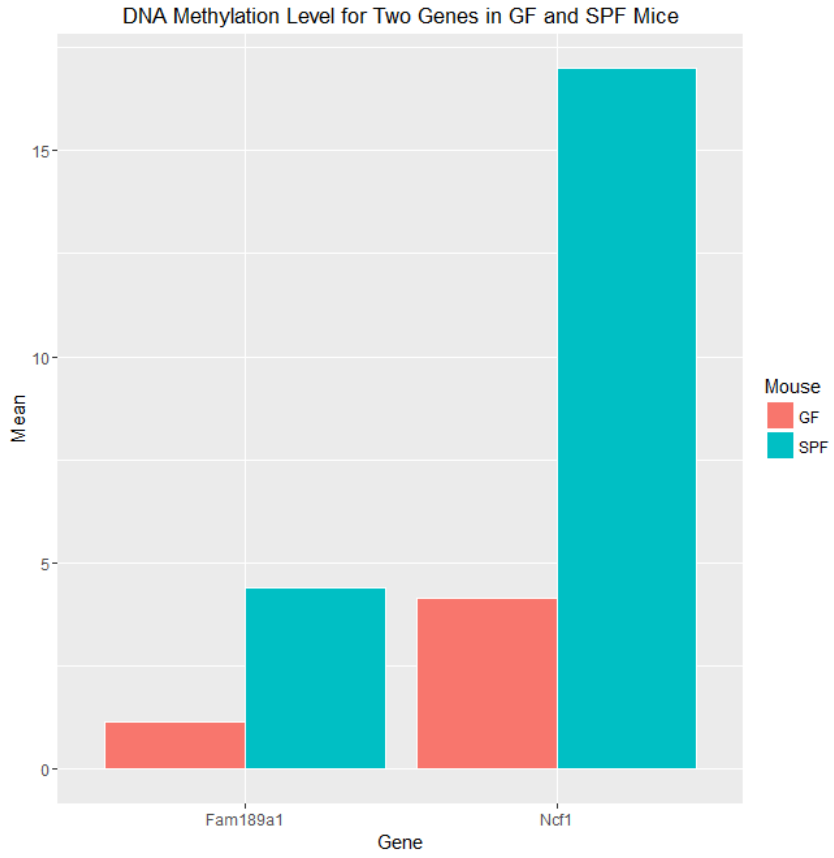


The argument stat=”identity” tells R to plot the data according to the values of the data.

The argument “colour” or “color” (both work) is telling R the desired color for outlining the bars.

If you wish to add a title to the chart, you can use the function ggtitle() and add another layer. For example, if you want to add “DNA Methylation Level for Two Genes in GF and SPF Mice” as a title to the chart. You can type:

ggplot(s, aes(x=Gene, y=Mean, fill=Mouse))+ geom_bar(stat="identity", position=position_dodge(),color="white")+ggtitle("DNA Methylation Level for Two Genes in GF and SPF Mice")



If you wish to color the bars in the colors you desire, you can add another layer to the code, and use the function scale_fill_manual(). For instance, if you desire to have the bars colored as black and green, you can type the codes as (do try it by yourself and see the difference would be made to the chart):

ggplot(s, aes(x=Gene, y=Mean, fill=Mouse))+ geom_bar(stat="identity", position=position_dodge(),color="white")+ggtitle("DNA Methylation Level for Two Genes in GF and SPF Mice")+scale_fill_manual(values=c(“black”,”green”))

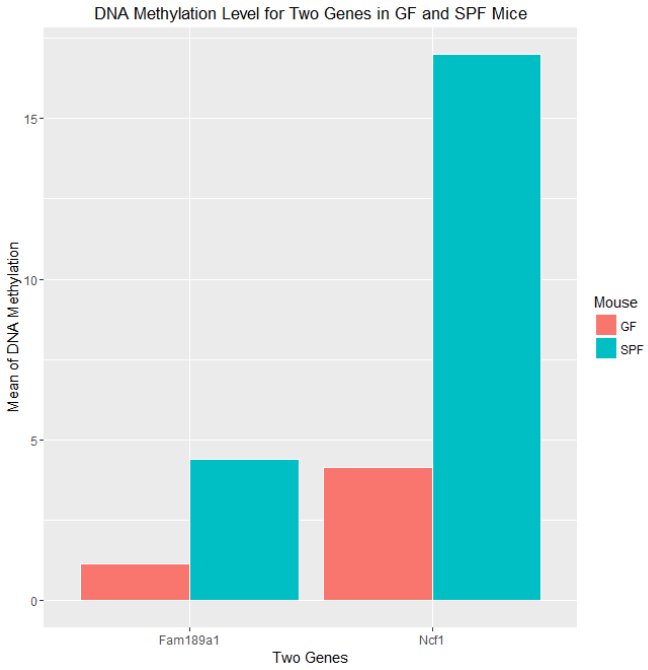
or you can type the following code to get different colors of the bars:

ggplot(s, aes(x=Gene, y=Mean, fill=Mouse))+ geom_bar(stat="identity", position=position_dodge(),color="white")+ggtitle("DNA Methylation Level for Two Genes in GF and SPF Mice")+scale_fill_manual(values=rainbow(10))

You can also change the title of the x-axis by using xlab(), and change the y-axis title by using ylab(). For instance, if you type:

ggplot(s, aes(x=Gene, y=Mean, fill=Mouse))+ geom_bar(stat="identity", position=position_dodge(),color="white")+ggtitle("DNA Methylation Level for Two Genes in GF and SPF Mice")+xlab("Two Genes")+ylab("Mean of DNA Methylation")

You will get:



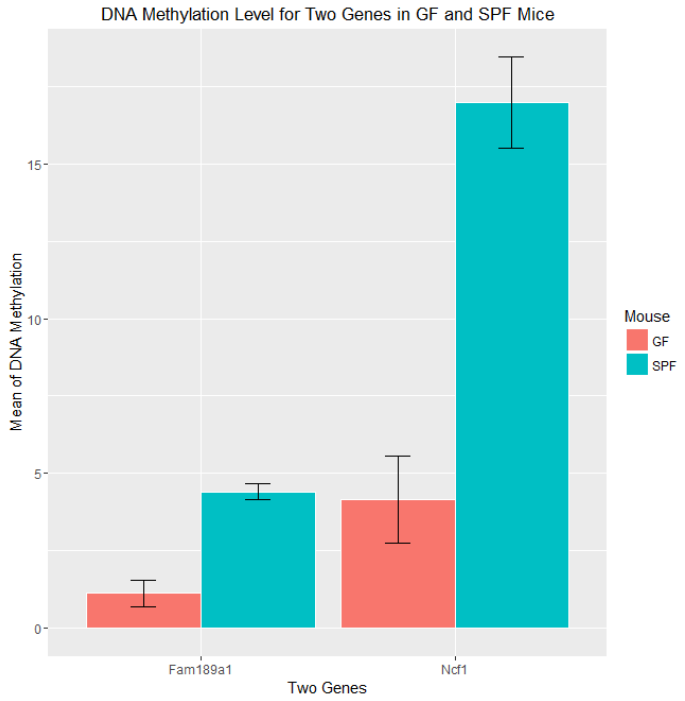
D. Adding error bars to the bar charts:

We use the function geom_errorbar() to add error bars. So, we add another layer to what we have generated.

If we want the error bars stands for the standard deviation. We can type:

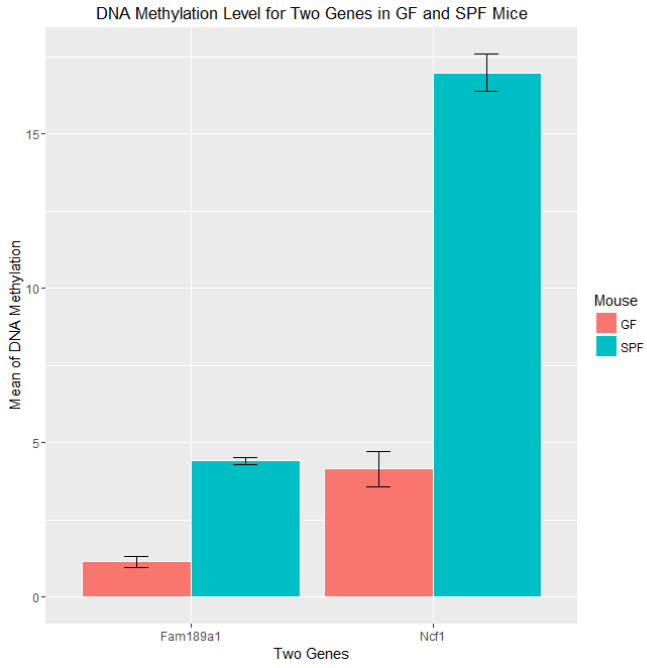
ggplot(s, aes(x=Gene, y=Mean, fill=Mouse))+ geom_bar(stat="identity", position=position_dodge(),color="white")+ggtitle("DNA Methylation Level for Two Genes in GF and SPF Mice")+xlab("Two Genes")+ylab("Mean of DNA Methylation")+geom_errorbar(aes(ymin=Mean-SD, ymax=Mean+SD), width=.2,position=position_dodge(0.9))

Then you will get:



If we want to change the error bars to mean+/- standard error of the mean, we can type:

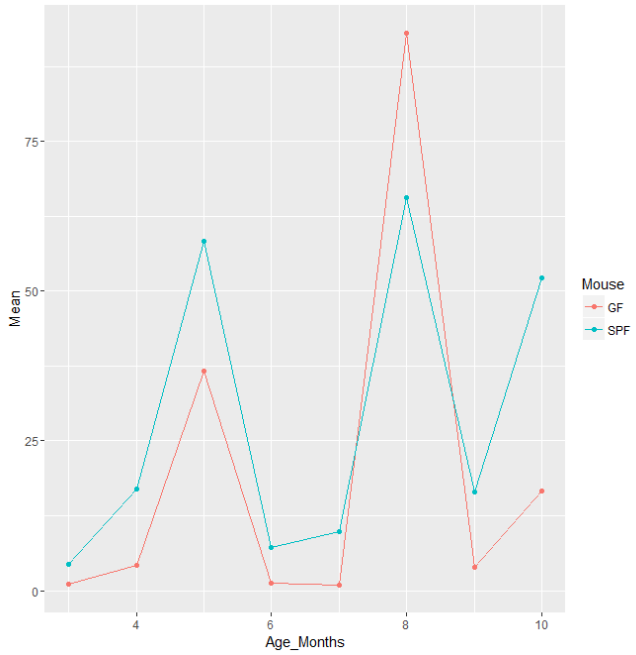
ggplot(s, aes(x=Gene, y=Mean, fill=Mouse))+ geom_bar(stat="identity", position=position_dodge(),color="white")+ggtitle("DNA Methylation Level for Two Genes in GF and SPF Mice")+xlab("Two Genes")+ylab("Mean of DNA Methylation")+ geom_errorbar(aes(ymin=Mean-SEM, ymax=Mean+SEM), width=.2,position=position_dodge(.9))



E. Adding Error Bars in Line Graphs:

m<-read.csv("Mouse_DNA_methylation_Aging.csv")

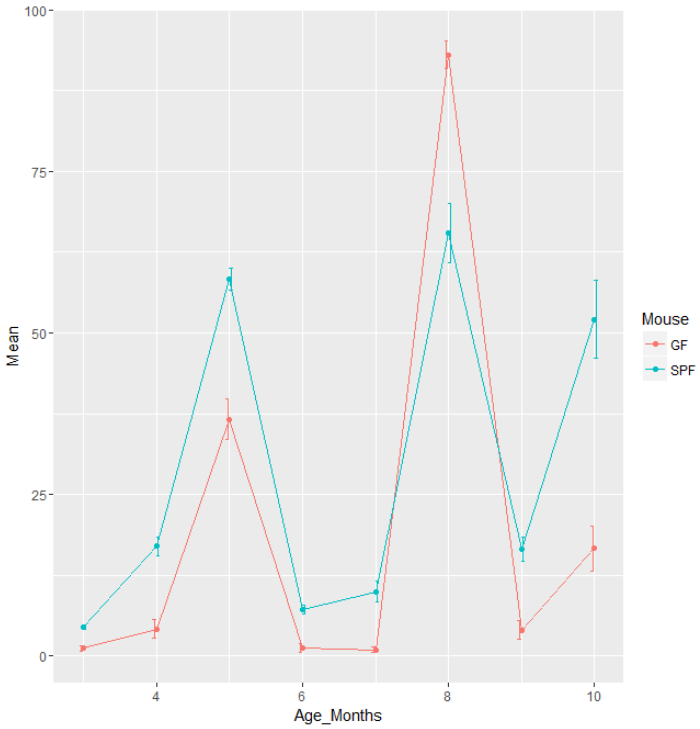
ggplot(m, aes(x=Age_Months, y=Mean,colour=Mouse)) +geom_line() + geom_point()



The function geom_point() is adding the data points to the chart. Another layer, the geom_line(), is adding lines to the chart.

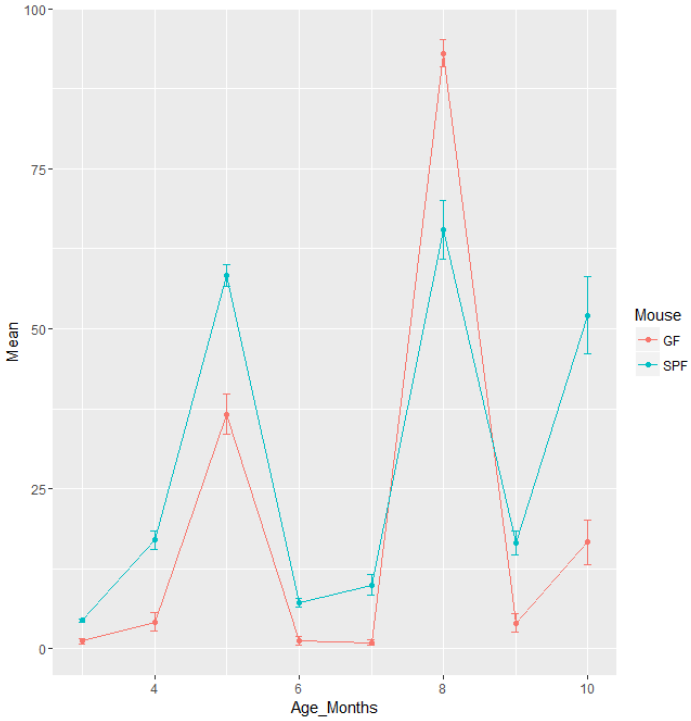
Adding and dodging the error bars:

ggplot(m, aes(x=Age_Months, y=Mean,colour=Mouse)) +geom_line() +geom_point()+geom_errorbar(aes(ymin=Mean-SD, ymax=Mean+SD), width=.1, position=position_dodge(0.1))



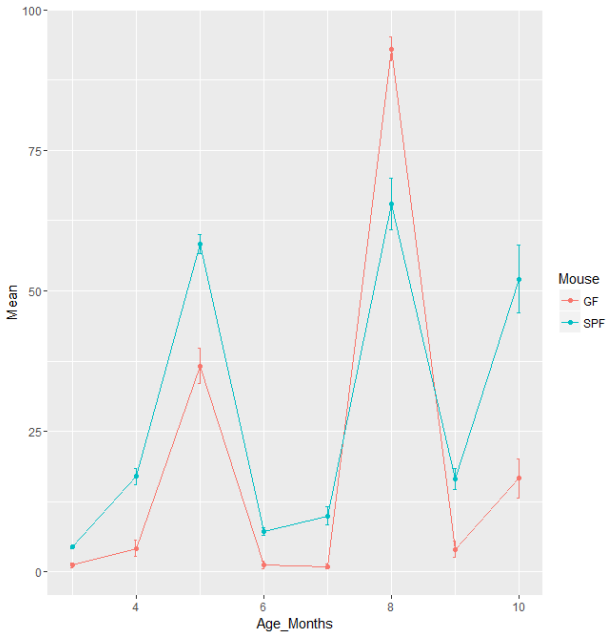
Adjusting dodging:

ggplot(m, aes(x=Age_Months, y=Mean,colour=Mouse)) +geom_line() +geom_point()+geom_errorbar(aes(ymin=Mean-SD, ymax=Mean+SD), width=.1, position=position_dodge(0.01))



Don’t dodge:

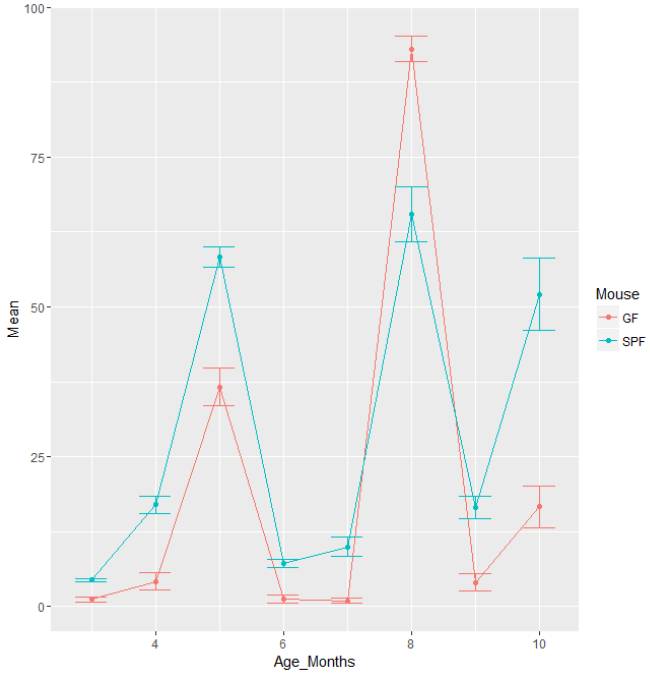
ggplot(m, aes(x=Age_Months, y=Mean,colour=Mouse)) +geom_line() +geom_point()+geom_errorbar(aes(ymin=Mean-SD, ymax=Mean+SD), width=.1)



Increasing the width of error bars:

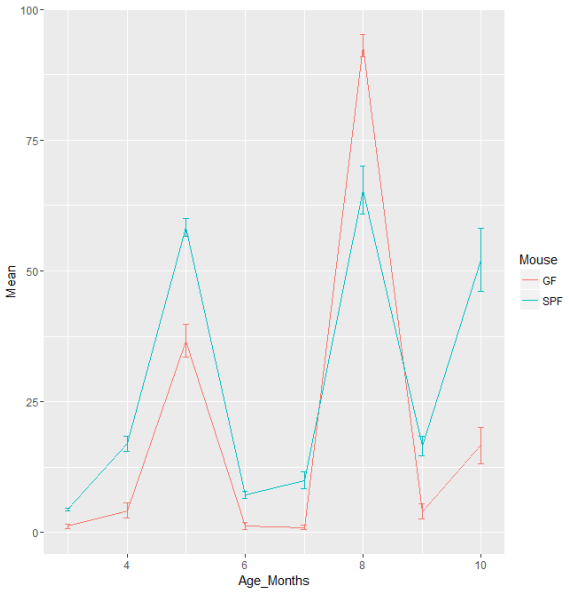
ggplot(m, aes(x=Age_Months, y=Mean,colour=Mouse)) +geom_line() +

geom_point()+geom_errorbar(aes(ymin=Mean-SD, ymax=Mean+SD), width=.5)



Without data points:

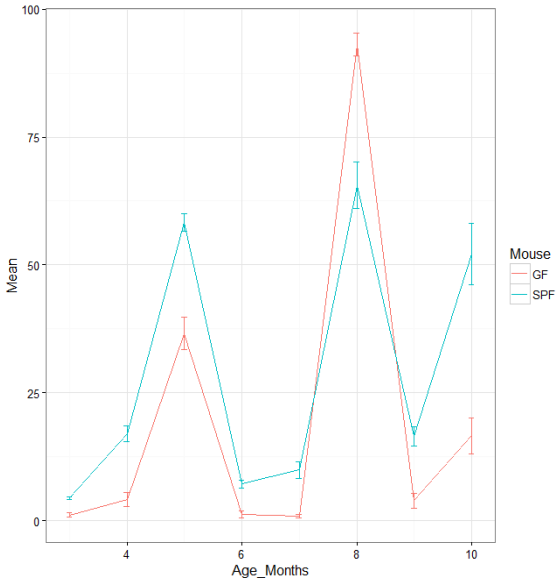
ggplot(m, aes(x=Age_Months, y=Mean,colour=Mouse))+geom_line()+geom_errorbar(aes(ymin=Mean-SD, ymax=Mean+SD), width=.1)



Clean background:

The function theme_bw() gives a clean white background:

ggplot(m, aes(x=Age_Months, y=Mean,colour=Mouse))+geom_line()+geom_errorbar(aes(ymin=Mean-SD, ymax=Mean+SD), width=.1)+theme_bw()



If you wish to remove the panel grids, you can add an additional layer.

For removing the major grid, you can add:

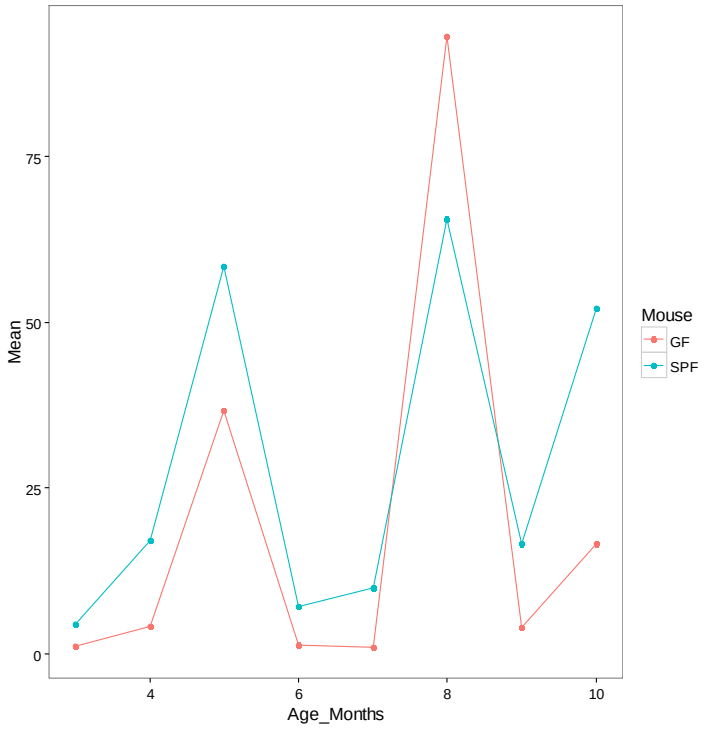
theme(panel.grid.major=element_blank())

For removing the minor grid, you can add:

theme(panel.grid.minor=element_blank())

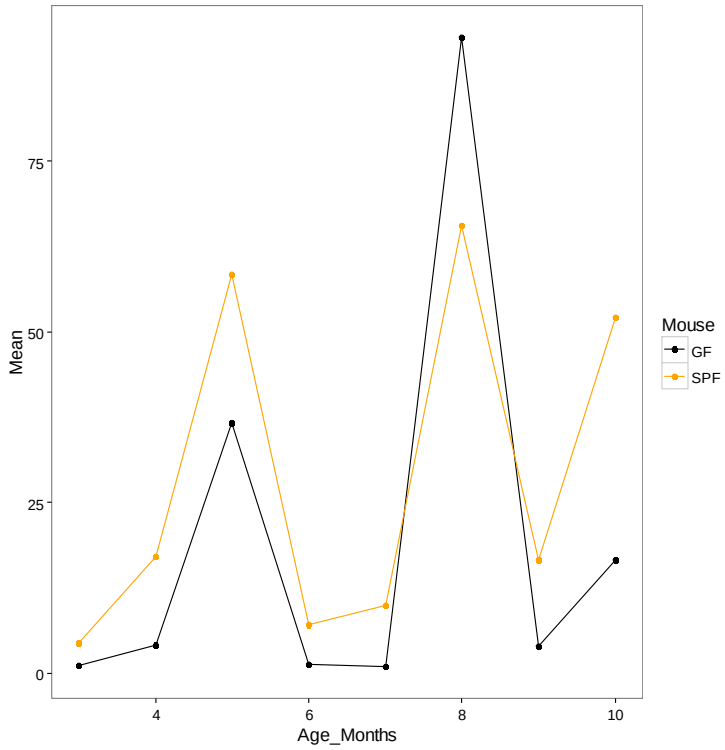
So, by typing in the following code in your R session, you can get a figure without any grids or the gray background:

ggplot(m, aes(x=Age_Months, y=Mean,colour=Mouse)) +geom_line() + geom_point()+theme_bw()+theme(panel.grid.major=element_blank())+theme(panel.grid.minor=element_blank())



You may wonder whether you are allowed to use the color not chosen by default but by you to color each of the lines. The short answer to this question is “Yes”. Now let’s use the above example to learn how to achieve this. Actually, it is just adding another layer to the command. The function that is going to be used is scale_colour_manual(). Here, we pick our own colors and use black and orange to color the lines for GF and SPF mice, respectively:

ggplot(m,aes(x=Age_Months,y=Mean,colour=Mouse))+geom_line()+geom_point()+theme_bw()+theme(panel.grid.major=element_blank())+theme(panel.grid.minor=element_blank())+scale_colour_manual(values=c("black","orange"))



We learned how to adding a title to the graph, how to change/lable both the x-axis and the y-axis. However, you may also try use labs() as in (xlab, ylab, labs(x="",y="")

F. Saving graphs:

You may wish to save the graphs you generated by R. There is a function ggsave() allows you to do so. For instance, you generated a graph by typing:

ggplot(s, aes(x=Gene, y=Mean, fill=Mouse))+ geom_bar(stat="identity", position=position_dodge(),color="white")

You want to save it as a tiff file and give it a name as "Figure1", then you can type:

ggsave("Figure1.tiff")

Challenge Question 4:

Where can you find this figure? Where was it saved?

Also, you can define the size of the graph when you save it. For instance, you can type:

ggsave("Figure2.pdf",width=3,height=3)

The figure will be saved as 3 inches wide by 3 inches tall and in a PDF file.

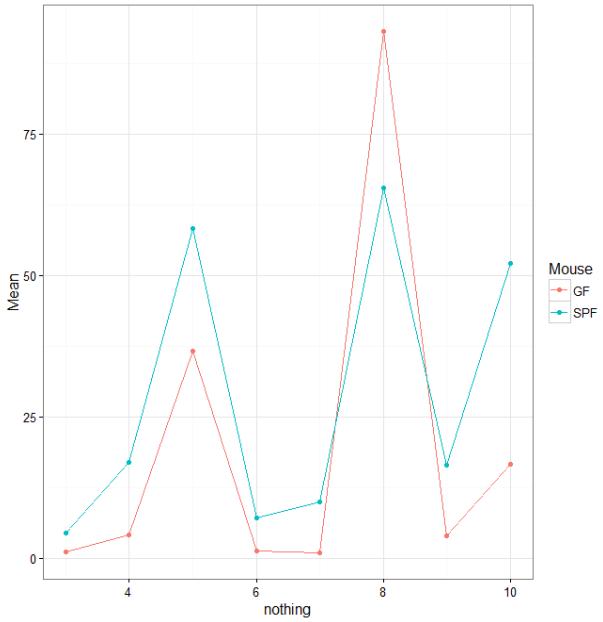
You can also save the graph into other types of files, such as png, jpeg, bmp, svg, PostScript (.eps/.ps), tex (pictex). Try and see.

G. Extra excises:

Different from barplot(), hist(), or other functions, where xlab="x-axis name", when use ggplot(), xlab or ylab has to be considered as a layer, and in the format as +xlab("x-axis name").

ggplot(m, aes(x=Age_Months, y=Mean,colour=Mouse))+xlab("nothing") +geom_line() + geom_point()+theme_bw()

Use "colour", "color", or "col", will get the same result.



ggplot(m, aes(x=Age_Months, y=Mean,colour=Mouse))+xlab("nothing") +geom_line() + geom_point()+theme_bw()+theme(panel.grid.major=element_blank())

history(4)

Answer keys to the challenge questions:

Challenge question 1:

No, as R won't automatically add the row numbers to each row if you are creating a text file.

Challenge question 2:
We can change the column name in Midterm1 from "X" to "students" by using the function "colnames()", which we learned previously.

Challenge question 3:
Maximum, minimum, 3rd quartile, 1st quartile, and the median.

Challenge question 4:
The figure was saved to the working directory.

Computer Practice Assignment 5:

Use the "Mouse_DNA_methylation_Aging.csv" to generate bar charts for all the data, the x-axis is the Age_Months with a name of "Age of the mice", the y-axis is the mean with a name of "Average DNA Methylation". The title of the chart is "DNA methylation level in GF and SPF mice at different ages". The error bars are for the mean=+/-SEM. The background is clean without gray color or any grids. Show me your R command lines and the final graph.