Exploring data with graphs

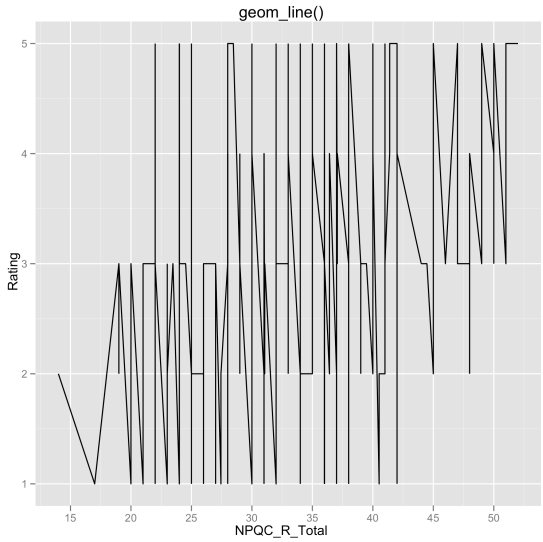
Self-test answers

- Go back to the Facebook narcissism data from the earlier tutorial. Plot a graph that shows the pattern in the data using only a line.
- Plot different coloured lines for the different types of rating (cool, fashionable, attractive, glamorous).
- Add a layer displaying the raw data as points.
- Add labels to the axes.

We can add a line using the *geom_line()* command. You might have, therefore, executed the following command:

```
graph + geom_line()
```

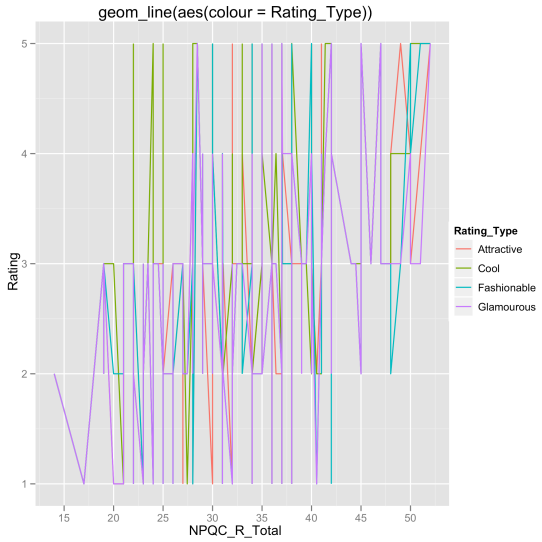
The resulting graph is a mess, because it has connected every single data point.



This is not what we wanted, we wanted a line to summarize the data. Perhaps it will help to have different lines for each type of rating? We can specify this in the same way as for *geom_point()*, by executing:

```
graph + geom_line(aes(colour = Rating_Type))
```

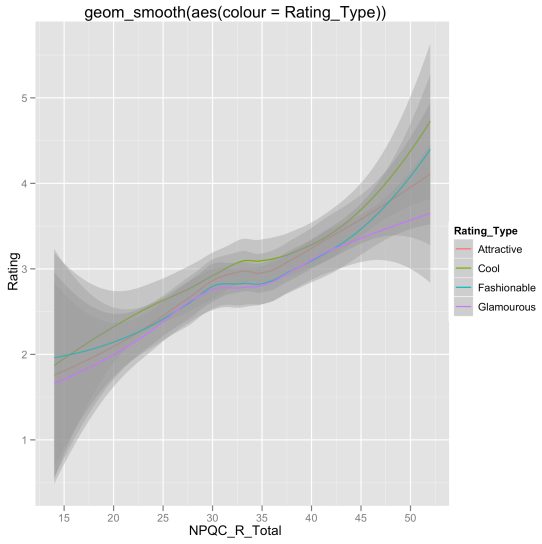
You should see this graph:



This hasn't helped much either. What we actually need is a summary line, or smoother. We can use *geom_smooth*() to do this:

```
graph + geom_smooth(aes(colour = Rating_Type))
```

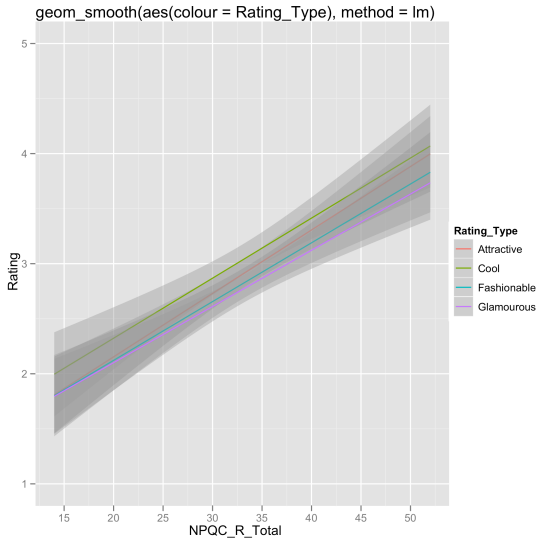
Note that we have continued to specify that we want different colours to represent different types of rating. The resulting plot should look like this:



Note that by default *ggplot2* fits a curved line. The shaded areas are the 95% confidence intervals around these lines. If we want a straight rather than curved line, we simply use the *method* command to specify a linear model or *lm* for short:

```
graph + geom_smooth(aes(colour = Rating_Type), method = lm)
```

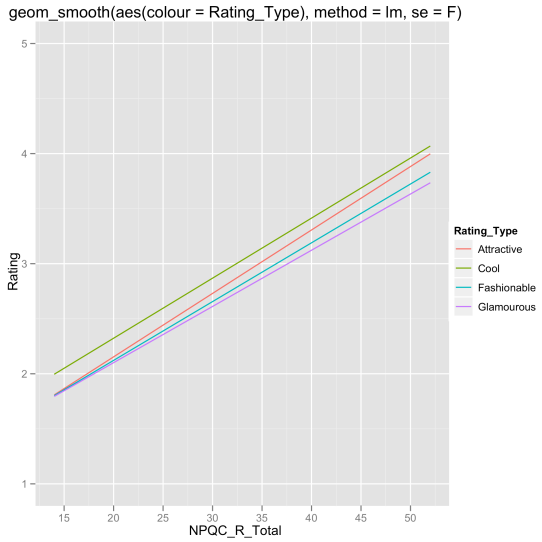
The resulting plot looks like this:



Note that again the shaded areas are the 95% confidence intervals. If we want to remove these intervals, we can do so by specifying *se = F* (short for standard error = false):

```
graph + geom_smooth(aes(colour = Rating_Type), method = lm, se = F)
```

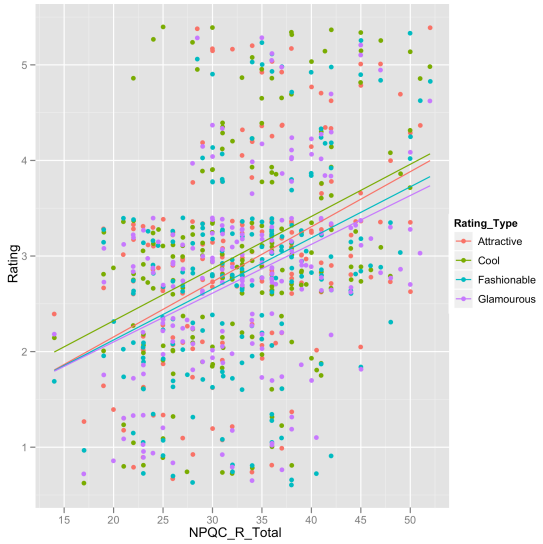
The resulting plot is:



We now have nice summary lines for each type of question. We can also include the raw data by using our earlier *geom_point()* command (note I've kept the jitter so that there is no overplotting):

```
graph + geom_point(aes(colour = Rating_Type), position = "jitter") +
geom_smooth(aes(colour = Rating_Type), method = lm, se = F)
```

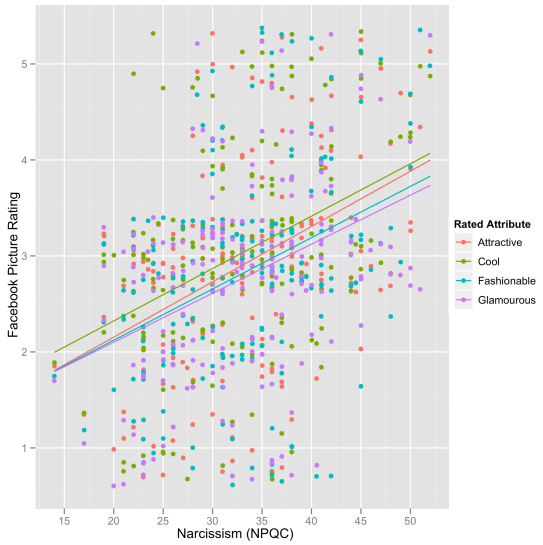
You should see this plot:



As a finishing touch we could add some labels using the *labs()* command to the axes that give a more helpful description of the variables:

```
graph + geom_point(aes(colour = Rating_Type), position = "jitter") +
geom_smooth(aes(colour = Rating_Type), method = lm, se = F) + labs(x = "Narcissism
(NPQC)", y = "Facebook Picture Rating", colour = "Rated Attribute")
```

We have a finished graph:



- What does a histogram show?

A histogram is a graph in which values of observations are plotted on the horizontal axis, and the frequency with which each value occurs in the data set is plotted on the vertical axis.

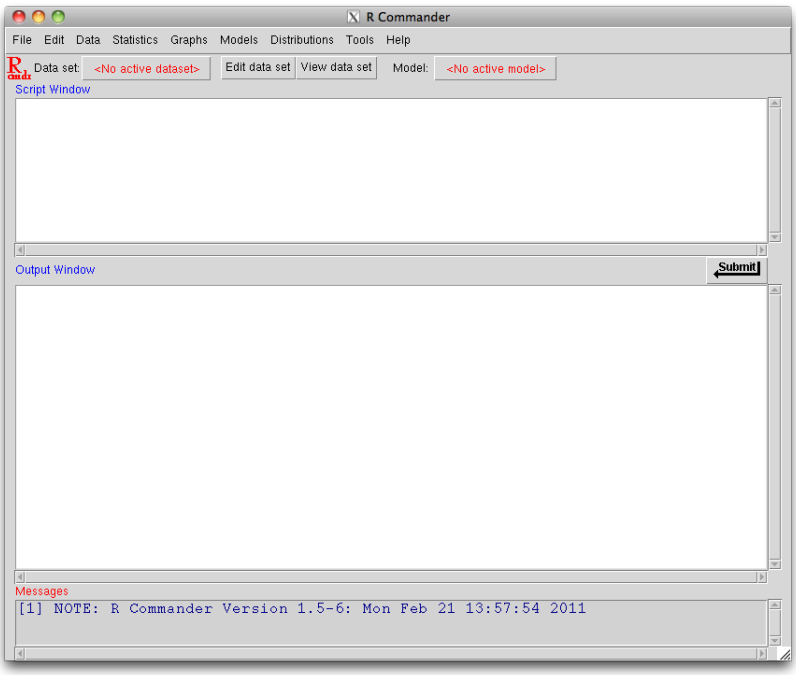
- Remove the outlier and replot the histogram.

An outlier is an extreme score, so the easiest way to find it is to sort the data:

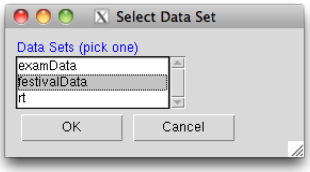
```
festivalData<-festivalData[order(festivalData$day1),]
```

This command takes *festivalData* and sorts it by the variable **day1**. All we have to do now is to look at the last case (i.e. the largest value of **day1**) and change it. The easiest way to edit the case is to start R Commander:

```
library(Rcmdr)
```



Click on <No active dataset> to open the select the *Select Data Set* dialog box. In this box, click on *festivalData* and then OK to open the data.



Click on Edit data set to open the data editor window:

R Data Editor

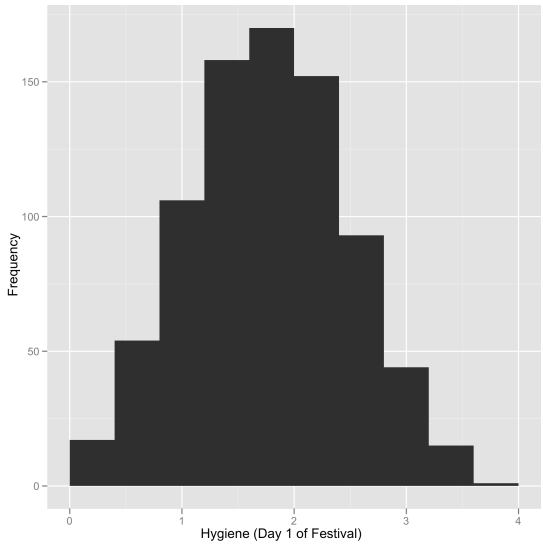
row.names	ticknumb	gender	day1	day2	day3
393	3599	Female	3.09	NA	NA
748	4470	Male	3.11	NA	NA
527	3919	Female	3.12	2.2	NA
213	3116	Female	3.14	NA	NA
476	3787	Female	3.15	3	NA
706	4383	Female	3.15	NA	NA
775	4569	Male	3.17	1	1.73
456	3726	Female	3.2	NA	NA
594	4106	Female	3.2	NA	NA
315	3395	Female	3.21	NA	NA
446	3716	Female	3.21	NA	NA
779	4590	Female	3.21	2.85	NA
29	2601	Male	3.23	NA	NA
163	2989	Female	3.26	1.97	1.67
417	3646	Female	3.29	NA	NA
744	4459	Female	3.29	0.26	NA
488	3828	Female	3.32	3.21	NA
623	4172	Female	3.32	2.91	3.02
624	4193	Male	3.32	NA	NA
285	3338	Female	3.38	NA	NA
774	4564	Female	3.38	3.44	3.41
300	3371	Female	3.41	NA	NA
657	4264	Male	3.44	NA	NA
303	3374	Male	3.58	3.35	NA
574	4016	Female	3.69	NA	NA
611	4158	Female	20.02	2.44	NA

Notice that the last case (case 611) is 20.02. This is clearly wrong. Perhaps we might go back to the original data and find that it has been mistyped and should be 2.02. Double click on the cell and change 20.02 to 2.02. Close the window and the new value will be saved in the dataframe. To replot the histogram, execute the same commands as in the book:

```
festivalHistogram <- ggplot(festivalData, aes(day1)) + opts(legend.position="none")
```

```
festivalHistogram + geom_histogram(binwidth = 0.4) + labs(x = "Hygiene (Day 1 of Festival)", y = "Frequency")
```

The resulting histogram looks like this:

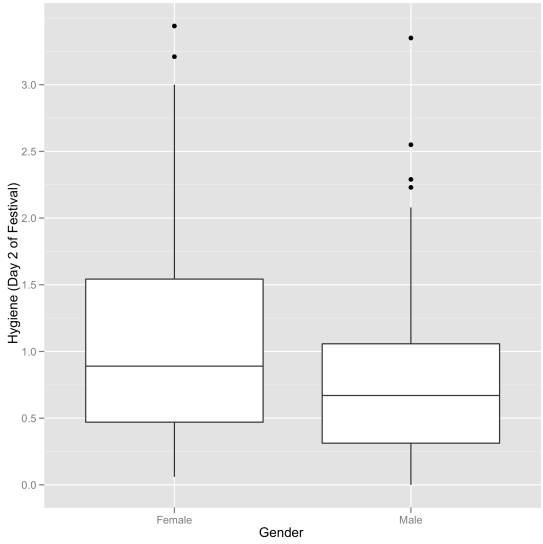


- Produce boxplots for the day 2 and day 3 hygiene scores and interpret them.

The code you need is simply:

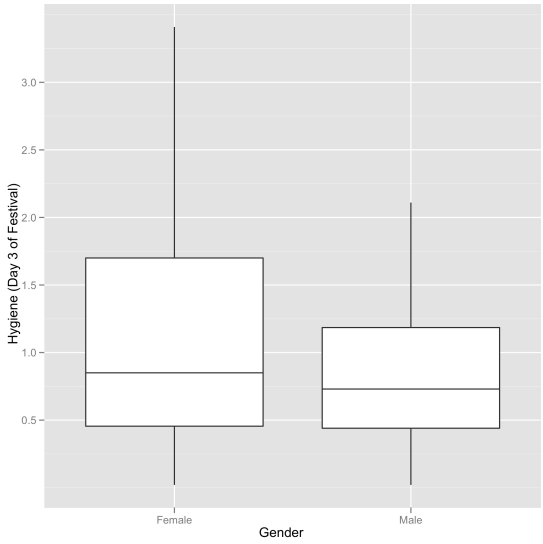
```
festivalBoxplot <- ggplot(festivalData, aes(gender, day2))
festivalBoxplot + geom_boxplot() + labs(x = "Gender", y = "Hygiene (Day 2 of
Festival)")

festivalBoxplot <- ggplot(festivalData, aes(gender, day3))
festivalBoxplot + geom_boxplot() + labs(x = "Gender", y = "Hygiene (Day 3 of
Festival)")
```

The boxplot for the day 2 data should look like this:



Note that, as for day 1, the females are slightly more fragrant than males (look at the median line). However, if you compare these to the day 1 boxplots (in the book), scores are getting lower (i.e. people are getting less hygienic). In the males there are now more outliers (i.e. a rebellious few who have maintained their sanitary standards).

The boxplot for the day 3 data should look like this:



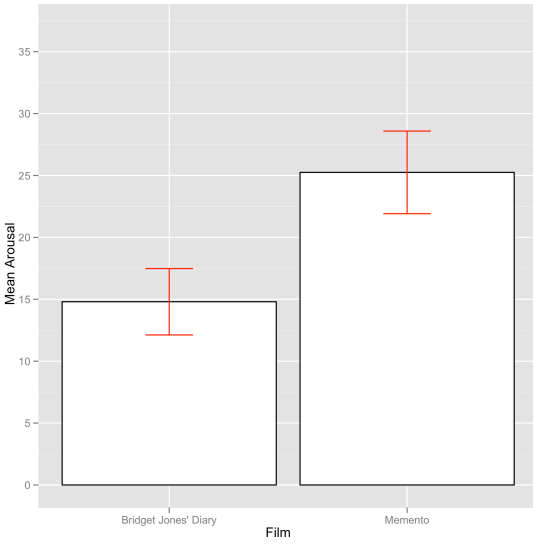
Note that, compared to day 1 and day 2, the females are getting more like the males (i.e. smelly). However, if you look at the top whisker, this is much longer for the females. In other words, the top 25% of females are more variable in how smelly they are compared to males. Also, the top score is higher than for males. So, at the top end females are better at maintaining their hygiene at the festival compared to males. Also, the box is longer for

females, and although both boxes start at the same score, the top edge of the box is higher in females, again suggesting that above the median score more women are achieving higher levels of hygiene than men. Finally, note that for both days 1 and 2, the boxplots have become less symmetrical (the top whiskers are longer than the bottom whiskers). On day 1 (see the book chapter), which is symmetrical, the whiskers on either side of the box are of equal length (the range of the top and bottom 25% of scores is the same); however, on days 2 and 3 the whisker coming out of the top of the box is longer than that at the bottom, which shows that the distribution is skewed (i.e. the top 25% of scores is spread out over a wider range than the bottom 25%).

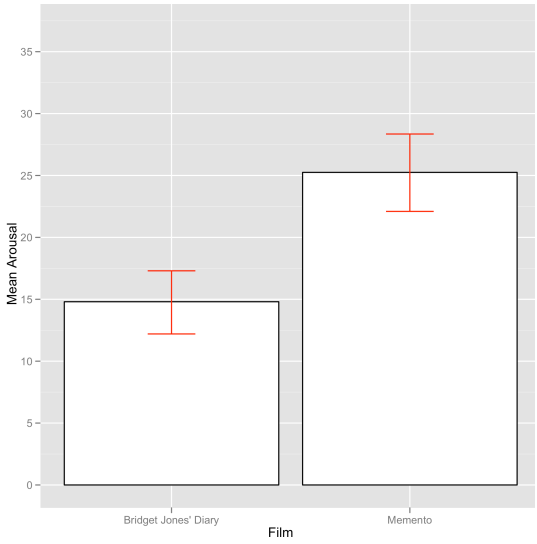
- Change the geom for the error bar to 'errorbar' and change its colour to red. Replot the graph.
- Plot the graph again but with bootstrapped confidence intervals.

```
bar <- ggplot(chickFlick, aes(film, arousal))

bar + stat_summary(fun.y = mean, geom = "bar", fill = "White", colour = "Black") +
stat_summary(fun.data = mean_cl_normal, geom = "errorbar", colour = "Red",  width =
0.2) + labs(x = "Film", y = "Mean Arousal")
```

```
bar <- ggplot(chickFlick, aes(film, arousal))

bar + stat_summary(fun.y = mean, geom = "bar", fill = "White", colour = "Black") +
stat_summary(fun.data = mean_cl_boot, geom = "errorbar", colour = "Red", width = 0.2)
+ labs(x = "Film", y = "Mean Arousal")
```



Normal Confidence Interval



Bootstrap Confidence Interval

- Restructure the data to a new dataframe called *textMessages* that is in long format. Use the *factor()* command to convert the 'Time' variable to a factor with levels called 'Baseline' and '6 Months'.

First we need to create a variable that gives each case an id so that we have a unique identifier for the *reshape* command:

```
textData$id = row(textData[1])
```

To create the new dataframe called *textMessage*, we execute the following command:

```
textMessages<-melt(textData, id = c("id", "Group"), measured = c("Baseline",
"Six_months"))

names(textMessages)<-c("id", "Group", "Time", "Grammar_Score")
```

Within this command, we specify the data to be reshaped (*textData*), the variables that remain constant for a participant (*id = c("id", "Group")*), and the variables that vary over time for a participant (*measured = c("Baseline", "Six_months")*).[1] The *names()* function then just gives the new columns some more helpful names: it names the variable that now contains the scores for each participant *"Grammar_Score"*, and the variable that represents the different points in time *"Time"*.

We can now change the newly created variable **Time** so that it is treated as a factor, and provide labels for the two levels of this variable:

```
textMessages$Time<-factor(textMessages$Time, labels = c("Baseline", "6 Months"))
```

The data before and after reshaping are presented below:

textData

	Group	Baseline	Six_months
1	Text Messagers	52	32
2	Text Messagers	68	48
3	Text Messagers	85	62
4	Text Messagers	47	16
5	Text Messagers	73	63
6	Text Messagers	57	53
7	Text Messagers	63	59
8	Text Messagers	50	58
9	Text Messagers	66	59
10	Text Messagers	60	57
11	Text Messagers	51	60
12	Text Messagers	72	56
13	Text Messagers	77	61
14	Text Messagers	57	52
15	Text Messagers	79	9
16	Text Messagers	75	76
17	Text Messagers	53	38
18	Text Messagers	72	63
19	Text Messagers	62	53
20	Text Messagers	71	61
21	Text Messagers	53	50
22	Text Messagers	64	78
23	Text Messagers	79	33
24	Text Messagers	75	68
25	Text Messagers	60	59
26	Controls	65	62
27	Controls	57	50
28	Controls	66	62
29	Controls	71	61
30	Controls	75	70
31	Controls	61	64
32	Controls	80	64
33	Controls	66	55
34	Controls	53	47
35	Controls	62	61
36	Controls	61	56
37	Controls	77	64
38	Controls	66	62

Before reshaping (wide format)

textMessages

	Group	id	Time	Grammar_Score
1.Text Messagers.0	Text Messagers	1	Baseline	52
2.Text Messagers.0	Text Messagers	2	Baseline	68
3.Text Messagers.0	Text Messagers	3	Baseline	85
4.Text Messagers.0	Text Messagers	4	Baseline	47
5.Text Messagers.0	Text Messagers	5	Baseline	73
6.Text Messagers.0	Text Messagers	6	Baseline	57
7.Text Messagers.0	Text Messagers	7	Baseline	63
8.Text Messagers.0	Text Messagers	8	Baseline	50
9.Text Messagers.0	Text Messagers	9	Baseline	66
10.Text Messagers.0	Text Messagers	10	Baseline	60
11.Text Messagers.0	Text Messagers	11	Baseline	51
12.Text Messagers.0	Text Messagers	12	Baseline	72
13.Text Messagers.0	Text Messagers	13	Baseline	77
14.Text Messagers.0	Text Messagers	14	Baseline	57
15.Text Messagers.0	Text Messagers	15	Baseline	79
16.Text Messagers.0	Text Messagers	16	Baseline	75
17.Text Messagers.0	Text Messagers	17	Baseline	53
18.Text Messagers.0	Text Messagers	18	Baseline	72
19.Text Messagers.0	Text Messagers	19	Baseline	62
20.Text Messagers.0	Text Messagers	20	Baseline	71
21.Text Messagers.0	Text Messagers	21	Baseline	53
22.Text Messagers.0	Text Messagers	22	Baseline	64
23.Text Messagers.0	Text Messagers	23	Baseline	79
24.Text Messagers.0	Text Messagers	24	Baseline	75
25.Text Messagers.0	Text Messagers	25	Baseline	60
26.Controls.0	Controls	26	Baseline	65
27.Controls.0	Controls	27	Baseline	57
28.Controls.0	Controls	28	Baseline	66
29.Controls.0	Controls	29	Baseline	71
30.Controls.0	Controls	30	Baseline	75
31.Controls.0	Controls	31	Baseline	61
32.Controls.0	Controls	32	Baseline	80
33.Controls.0	Controls	33	Baseline	66
34.Controls.0	Controls	34	Baseline	53

After reshaping (long format)

- Use what you have learnt to repeat the text message data plot but to also have different symbols for text messagers and controls and different types of lines.

The commands to execute are:

```
line <- ggplot(textMessages, aes(Time, Grammar_Score, colour = Group))

line + stat_summary(fun.y = mean, geom = "point", aes(shape = Group), size = 4) +
stat_summary(fun.y = mean, geom = "line", aes(group= Group, linetype = Group)) +
stat_summary(fun.data = mean_cl_boot, geom = "errorbar", width = 0.2) + labs(x =
"Time", y = "Mean Grammar Score", colour = "Group")
```

[1] You can also use the *reshape* command as follows:

```
textMessages = reshape(textData, idvar = c("id", "Group"), varying = c("Baseline",
"Six_months"), v.names = "Grammar_Score", timevar = "Time", times = c(0:1), direction
= "long")
```

The main differences are that when I specified the means using the point geom, I added an aesthetic that sets the shape to be equal to **Group** (which means that a different shape will be used for each group). So,

```
stat_summary(fun.y = mean, geom = "point")
```

from the example in the book, became

```
stat_summary(fun.y = mean, geom = "point", aes(shape = Group))
```

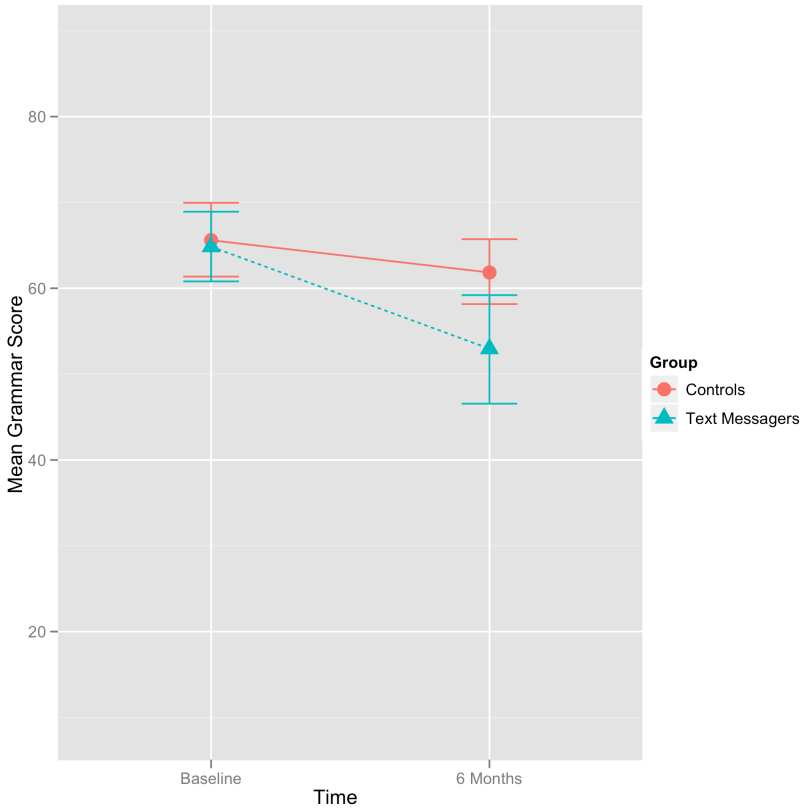
(To create the diagram below I also added *size = 4* to make the symbols bigger.) Similarly, I added a *linestyle* aesthetic when I defined the line geom. So,

```
stat_summary(fun.y = mean, geom = "line", aes(group= Group))
```

from the example in the book, became

```
stat_summary(fun.y = mean, geom = "line", aes(group= Group, linetype = Group))
```

The end result is:



Error bar graph of the mean grammar score over six months in children who were allowed to text message versus those who were forbidden

Note that both the symbol and the line style changes as a function of the variable **Group** (i.e. text messagers and controls have different symbols representing their means, different types of lines, and different coloured lines).

Oliver Twisted

Please Sir, can I have some more … complicated stuff?

To check for outliers we can look at *z*-scores. In the book chapter we wrote a nice little function to display the percentage of *z*-scores that fell outside of the standard limits. This section explains how that function works.

Z-scores are simply a way of standardizing a data set by expressing the scores in terms of a distribution with a mean of 0 and a standard deviation of 1. In doing so we can use benchmarks that we can apply to any data set (regardless of what its original mean and standard deviation were). To convert a score into a *z*-score we simply take each score (*X*) and subtract the mean of all scores ($\bar{X}$) from it and divide by the standard deviation of all scores (*s*).

We can get **R** to do this conversion for us using the *mean()* and *sd()* functions to compute the mean and *sd* of the data:

```
zvariable<-(variable-mean(variable, na.rm = TRUE))/sd(variable, na.rm = TRUE)
```

This command creates a variable called **zvariable**, which is the original variable with the mean subtracted, computed using *mean(variable, na.rm = TRUE)*, and divided by the standard deviation, computed using *sd(variable, na.rm = TRUE)*. It's important to include *na.rm = TRUE* to eliminate missing values before the mean and standard deviation are computed.

To look for outliers we could use these *z*-scores and count how many fall within certain important limits. If we take the absolute value (i.e., we ignore whether the *z*-score is positive or negative) then in a normal distribution we'd expect about 5% to have absolute values greater than 1.96 (we often use 2 for convenience), 1% to have absolute values greater than 2.58, and none to be greater than about 3.29. We can create variables that will return the value TRUE if the *z*-score is greater than these cut-off points as follows:

```
outlier95<-abs(zvariable) >= 1.96
```

```
outlier99<-abs(zvariable) >= 2.58
```

```
outlier999<-abs(zvariable) >= 3.29
```

The first command creates a variable called **outlier95** that will be TRUE if the absolute value of a *z*-score (which we computed earlier and stored in the variable **zvariable**) is greater than or equal to 1.96. We use the *abs()* function to give us the absolute value. To be clear, *abs(zvariable) >= 1.96* means 'the absolute value of the score in the variable called **zvariable** is greater than or equal to 1.96'. The next two commands do a similar thing, creating variables **outlier99** and **outlier999** that will return the value TRUE if the absolute value of the *z*-score is greater or equal to 2.58 and 3.29, respectively.

We can now use these values to calculate the percentage of outliers we have. First we need to know the number of cases that we have. We can use the *length()* function to calculate this value:

```
ncases<-length(na.omit(zvariable))
```

This command creates a variable called **ncases**, which is the value of the length of the variable **zvariable**. Note that we enclose **zvariable** in the function *na.omit()*; this has the effect of deleting any missing cases (and we don't want to include missing cases when we calculate the length of the variable). The *length()* function returns the number of items within the variable – in other words, the number of cases. Therefore, this command gives us the total number of *z*-scores.

We now need to know how many of the *z*-scores were greater than our cut-off values. This information is stored in the variables **outlier95**, **outlier99** and **outlier999**. Essentially, we want to count the number of TRUE responses within these variables (remember that when the response is TRUE it means that the absolute value of the *z*-score was greater than the cut-off). A simple way to do this, is to use the *subset()* function to filter out the TRUE responses and then count them using the *length()* function:

```
nTrue95<-length(subset(outlier95, outlier95 == TRUE))
```

```
nTrue99<-length(subset(outlier99, outlier99 == TRUE))
```

```
nTrue999<-length(subset(outlier999, outlier999 == TRUE))
```

These commands first create a subset of the variables (**outlier95**, **outlier99** and **outlier999**, respectively) by specifying that cases be retained if their value is equal to TRUE. The *length()* function then counts the number of cases. Therefore, the new variables **nTrue95**, **nTrue99**, and **nTrue999** tell us the number of TRUE responses (i.e., the number of cases with absolute *z*-scores greater than the cut off) for each cut-off.

It's fairly easy then to compute the number of true responses as a percentage of the number of total cases:

```
percent95<-round(100* nTrue95)/ncases, 2)
percent99<-round(100* nTrue99)/ncases, 2)
percent999<-round(100* nTrue999)/ncases, 2)
```

In each case we create a variable that is the number of TRUE responses (i.e., the number of cases with absolute *z*-scores greater than the cut-off), divided by the total number of cases (**ncases**) multiplied by 100 to get the percentage. In all cases we use the *round()* function to round the result to 2 decimal places (which is sufficient)

The final bit of the function specifies some text to be printed around each of these percentages (so that the output looks pretty):

```
cat("Absolute z-score greater than 1.96 = ", percent95, "%", "\n")
cat("Absolute z-score greater than 2.58 = ",  percent99, "%", "\n")
cat("Absolute z-score greater than 3.29 = ",  percent999, "%", "\n")
```

The function in its completed form looks like this:

```
outlierSummary<-function(variable, digits = 2){
        zvariable<-(variable-mean(variable, na.rm = TRUE))/sd(variable, na.rm = TRUE)
        outlier95<-abs(zvariable) >= 1.96
        outlier99<-abs(zvariable) >= 2.58
        outlier999<-abs(zvariable) >= 3.29
        ncases<-length(na.omit(zvariable))
        percent95<-round(100*length(subset(outlier95, outlier95 == TRUE))/ncases,
digits)
        percent99<-round(100*length(subset(outlier99, outlier99 == TRUE))/ncases,
digits)
        percent999<-round(100*length(subset(outlier999, outlier999 == TRUE))/ncases,
digits)
        cat("Absolute z-score greater than 1.96 = ", percent95, "%", "\n")
        cat("Absolute z-score greater than 2.58 = ",  percent99, "%", "\n")
        cat("Absolute z-score greater than 3.29 = ",  percent999, "%", "\n")
}
```

The function doesn't go through as many steps as I have just explained (it computes the percentages more directly), but I wanted to take you through each step systematically. The other thing to note is that we have allowed users to specify how many decimal places they want in the output the default is 2, but if the user inputs a value then they'll get a different amount. For example, executing:

```
outlierSummary(festivalData$day2, 3)
```

returns 3 decimal places and:

```
outlierSummary(festivalData$day2, 1)
```

would return 1 decimal place.

Smart Alex's solutions

Task 1

- Using the data from Chapter 3 (which you should have saved, but if you didn't, re-enter it) plot and interpret the following graphs:
  - An error bar chart showing the mean number of friends for students and lecturers.
  - An error bar chart showing the mean alcohol consumption for students and lecturers.
  - An error line chart showing the mean income for students and lecturers.
  - An error line chart showing the mean neuroticism for students and lecturers.
  - A scatterplot (with regression lines) of alcohol consumption and neuroticism grouped by lecturer/student.

An error bar chart showing the mean number of friends for students and lecturers

I have called this data file **Lecturer Data.dat**; if you have called the data file something different you need to adapt the code in this section accordingly, or, alternatively, you could rename your data file.

First of all, set your working directory to be the location of the data file. Then create a dataframe called *lecturerData* by executing the following command:

```
lecturerData = read.delim("Lecturer Data.dat", header = TRUE)
```

Tell R that the variable 'job' is a factor:

```
lecturerData$job<-factor(lecturerData$job, levels=c(1:2), labels = c("lecturer",
"student"))
```

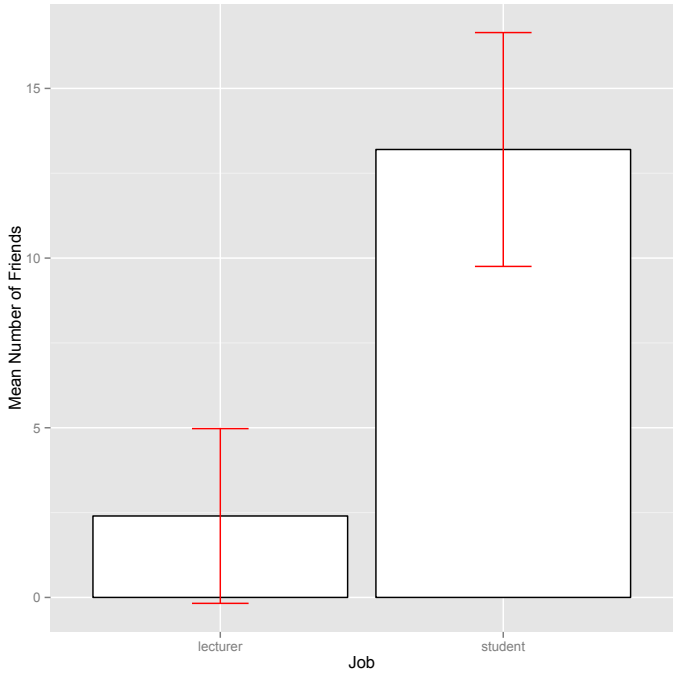
To plot the mean number of friends (*y*-axis – as this is the thing we have measured) for students and lecturers (*x*-axis), we need to create a plot object by executing the following command:

```
bar < - ggplot(lecturerData, aes(job, friends))
```

Executing the following command will create an error bar chart (I have put in red error bars, but you can use whichever colour you prefer) displaying the mean number of friends (and the 95% confidence interval of those means) for students and lecturers as white bars with a black outline with nicely labeled axes:

```
bar + stat_summary(fun.y = mean, geom = "bar", fill = "White", colour = "Black") +
stat_summary(fun.data = mean_cl_normal, geom = "errorbar", colour = "Red", width =
0.2) + labs(x = "Group", y = "Mean Number of Friends")
```

The error bar chart will look like this:



We can conclude that, on average, students had more friends than lecturers.

An error bar chart showing the mean alcohol consumption for students and lecturers

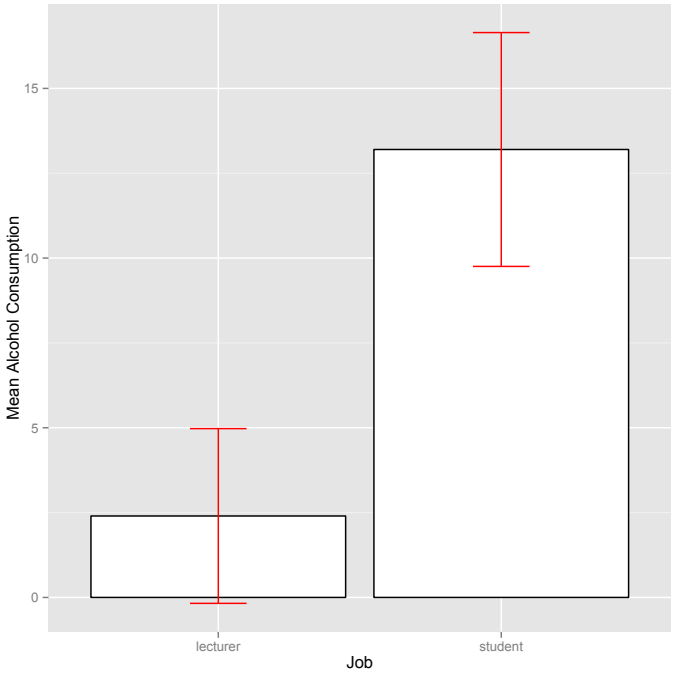
To plot the mean alcohol consumption (*y*-axis) for students and lecturers (*x*-axis), we first need to create a plot object by executing the following command:

```
bar < - ggplot(lecturerData, aes(job, alcohol))
```

To create an error bar chart displaying the mean alcohol consumption (and the 95% confidence interval of those means) for students and lecturers as white bars with a black outline with nicely labeled, axes execute the following command:

```
bar + stat_summary(fun.y = "mean", geom = "bar", fill = "White", colour = "Black") +
stat_summary(fun.data = "mean_cl_normal", geom = "errorbar", colour = "Red", width =
0.2) + labs(x = "Job", y = "Mean Alcohol Consumption")
```

The error bar chart will look like this:



We can conclude that, on average, students and lecturers drank similar amounts, but the error bars tell us that the mean is a better representation of the population for students than for lecturers (there is more variability in lecturers' drinking habits compared to students').

An error line chart showing the mean income for students and lecturers

First create the plot object and define the variables that we want to plot as aesthetics:

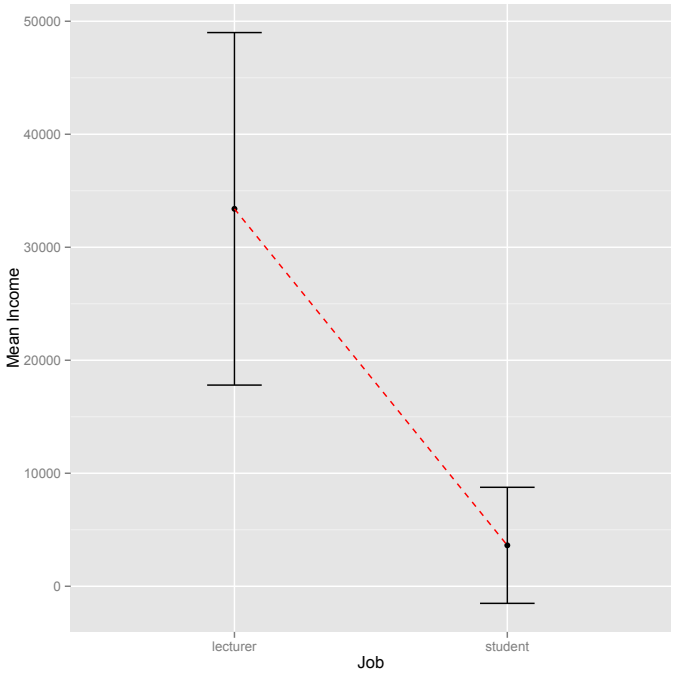
```
line <- ggplot(lecturerData, aes(job, income))
```

I have called the object *line*. The contents of this command specify the dataframe to be used (*lecturerData*) and set **job** to be plotted on the *x*-axis, and **income** to be plotted on the *y*-axis.

Executing the command below will create an error line graph of the mean income for students and lecturers. The command uses *stat_summary()* to create the mean and the aesthetics are set so that the group means are connected with a red dashed line and error bars are set at a width of 0.2. Labels will be added to the *x* (Group) and *y axis* (Income):

```
line + stat_summary(fun.y = "mean", geom = "point") + stat_summary(fun.data =
"mean_cl_normal", geom= "errorbar", width = 0.2) + labs(x = "Job", y = "Mean Income")+
stat_summary(fun.y = "mean", geom = "line", aes(group=1),colour = "Red", linetype =
"dashed")
```

The error line chart will look like this:



We can conclude that, on average, students earn less than lecturers, but the error bars tells us that the mean is a better representation of the population for students than for lecturers (there is more variability in lecturers' income compared to students').

An error line chart showing the mean neuroticism for students and lecturers

First create the plot object and define the variables that we want to plot as aesthetics:

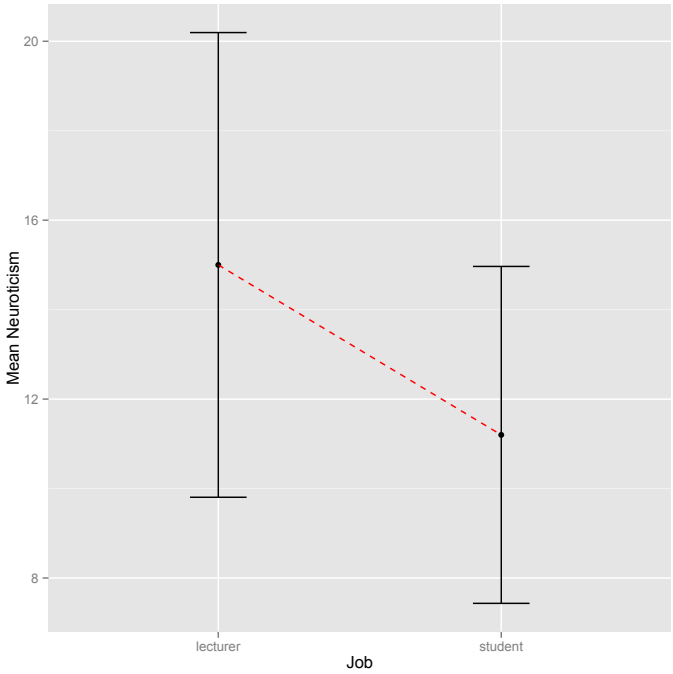
```
line <- ggplot(lecturerData, aes(job, neurotic))
```

As in the previous example, I have called the object *line*. The contents of this command specify the dataframe to be used (*lecturerData*) and set **neuroticism** to be plotted on the *y*-axis (as it is the thing we have measured) and **job** to be plotted on the *x*-axis.

Executing the command below will produce an error line graph of the mean neuroticism for students and lecturers. As before, the command uses *stat_summary()* to create the mean and another layer of *stat_summary()* to add an error bar to each group mean. The aesthetics are set so that the group means are connected with a red dashed line and error bars are set at a width of 0.2. Labels will be added to the *x*-axis (Job) and *y*-axis (Neuroticism):

```
line + stat_summary(fun.y = "mean", geom =  "point") + stat_summary(fun.y = "mean",
geom = "line", aes(group=1),colour = "Red", linetype = "dashed")+
stat_summary(fun.data = "mean_cl_normal", geom = "errorbar", width = 0.2) + labs(x =
"Job", y = "Mean Neuroticism")
```

The error line chart will look like this:



We can conclude that, on average, students are slightly less neurotic than lecturers.

A scatterplot with regression lines of alcohol consumption and neuroticism grouped by lecturer/student.

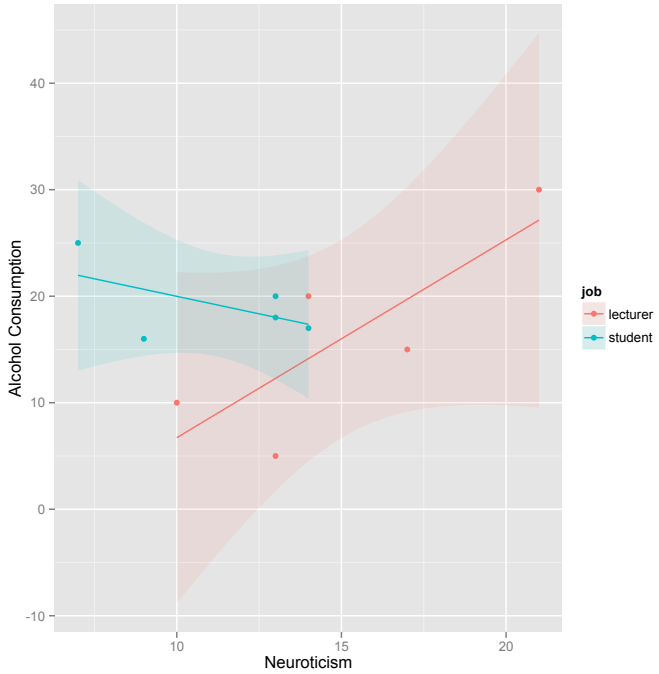
So, we want to see whether the relationship between neuroticism and alcohol was different for students and lecturers. To do this, we need to set group as an aesthetic. This is fairly straightforward. First, we define group as a colour aesthetic when we initiate the plot object:

```
scatter <- ggplot(lecturerData, aes(neurotic, alcohol, colour = job))
```

The command below will produce a scatterplot with different coloured dots and regression lines for students and lecturers and confidence intervals on the regression lines coloured according to the **job** variable, complete with labels:

```
scatter + geom_point() + geom_smooth(method = "lm", aes(fill = job), alpha = 0.1) +
labs(x = "Neuroticism", y = "Alcohol Consumption", colour = "job")
```

The scatterplot will look like this:



We can conclude that for lecturers, as neuroticism increases so does alcohol consumption (a positive relationship), but that for students the opposite is true, as neuroticism increases alcohol consumption decreases.

Task 2

- Using the **infidelity.dat** data from Chapter 3 (see Smart Alex's Task 3), plot a clustered error bar chart of the mean number of bullets used against self and partner for males and females.

Set your working directory to be the location of the data file. Then create a dataframe called *Infidelity* by executing the following command:

```
infidelityData = read.delim("Infidelity Data.csv", header = TRUE)
```

We need to create a variable that gives each case an id so that we have a unique identifier for the reshape command:

```
infidelityData$id = row(infidelityData[1])
```

To create the new dataframe called *Bullets*, execute the following command:

```
Bullets = reshape(infidelityData, idvar = c("id", "Gender"), varying = c("Partner",
"Self"), v.names = "Number_of_Bullets", timevar = "Recipient", times = c(0:1),
direction = "long")
```

Within this command, we specify the data to be reshaped (*infidelityData*), the variables that remain constant for a participant (*idvar = c("id", "Gender")*), and the variables that vary for a participant (*varying = c("Partner", "Self")*). We need to name the variable that now contains the scores for each participant (*v.names = "Number_of_Bullets"*) and the variable that represents the different recipient of the bullets (*timevar = "Recipient"*). Note that I have used *timevar* here because even though the number of bullets does not vary over time (they vary across recipient) the underlying principle is the same, i.e. using *timevar* will tell **R** that the number of bullets varies across recipient (partner and self). Finally, we specify the codes to be used to represent levels of the variable that represents the recipient of the bullets (*recipients = c(0:1)*) and we specify the direction of the reshape (*direction = "long"*).

We can now change the newly created variable **Recipient** so that it is treated as a factor, and provide labels for the two levels of this variable:

```
Bullets$Recipient<-factor(Bullets$Recipient, labels = c("Partner","Self"))
```

We are now ready to plot the graph. As always, we first create the plot object and define the variables that we want to plot as aesthetics (remember that the new dataframe is called *Bullets*):

```
bar <- ggplot(Bullets, aes(Recipient, Number_of_Bullets, fill = Gender))
```

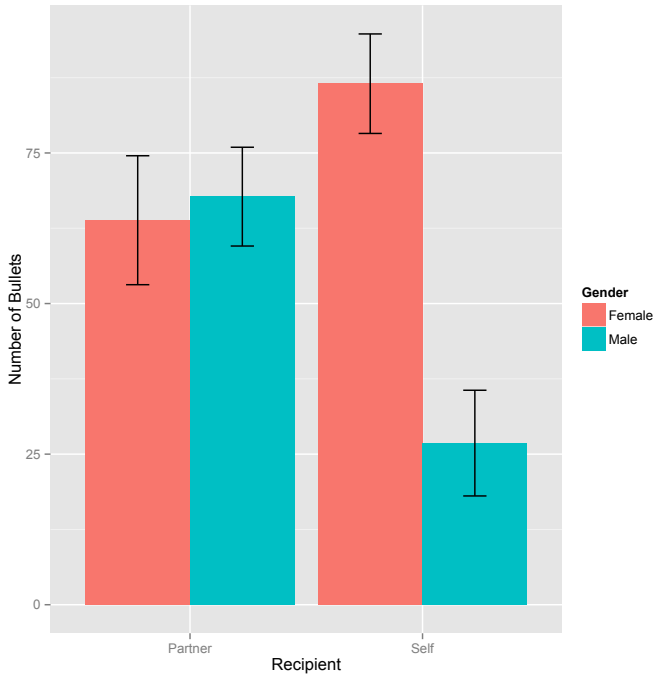
As before, we can simply add the mean displayed as bars by adding a layer to 'bar' using *stat_summary()* command. However, there is one important difference: we have to specify *position = "dodge"* (see section 4.4.6 in the book) so that the male and female bars are forced to stand side-by-side, rather than behind each other.

```
bar + stat_summary(fun.y = "mean", geom = "bar", position="dodge")
```

As you probably know by now, *fun.y = mean* computes the mean for us and *geom = "bar"* displays these values as bars. If we want to add error bars we can again add these as a layer using *stat_summary()*. We can also specify labels for the axes using the *labs()* command and we can add a title for the legend by using *fill = "Gender"*:

```
bar + stat_summary(fun.y = "mean", geom = "bar", position="dodge") +
stat_summary(fun.data = "mean_cl_normal", geom = "errorbar",
position=position_dodge(width=0.90), width = 0.2) + labs(x = "Recipient", y = "Number
of Bullets", fill = "Gender")
```

The resulting graph looks like this:



The graph shows that, on average, males and females did not differ much in the number of bullets that they shot at the target when it had their partner's face on it. However, men used fewer bullets than women when the target had their own face on it.