Dapr1: Notes on the live R session, Week 4

2022-10-06

This week's topic is "describing relationships". We will start by loading "tidyverse"

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.2      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

We will also need to use another package called "ggmosaic". You may or may not have this package installed. You can check whether you have it installed if you select the "Packages" tab in the bottom right panel of R studio. If "ggmosaic" is listed, then you have it installed. If it is not installed, you need to install it, which you can do by clicking the "Install" tab on that panel, and typing "ggmosaic" in the "Packages" box. Once it is installed, you need to load it into your R studio session, like this:

```
library(ggmosaic)
```

Now we will load the dataset. This is the "ex1" dataset. It is the same as the one we have been using in weeks 1-3:

```
ex1 <- read_csv("https://uoepsy.github.io/data/ex1.csv")
```

```
## Rows: 150 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): ID, Degree
## dbl (3): Year, Score1, Score2
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Categorical-Categorical relationships

Let's start by making a contingency table of "Degree" by "Year". This will show the number of students in each year of study, for each subject:

```
ex1 %>%
  select(.,Degree, Year) %>%
  table()
```

```
##        Year
## Degree   1  2  3  4
##   Joint  3  2  3  2
##   Ling  17 42 17  6
##   Phil   6 10  9  1
##   Psych  7 17  7  1
```

In the code, we pipe the "ex1" dataset to the "select" command, which outputs the "Degree" and "Year" columns. Another pipe is then used to input these two columns to the "table()" command, which makes the 2-dimensional contingency table. Note that we type "select(.,Degree,Year)". The dot "." is used in the first argument slot because we have already specified the dataset before the pipe symbol. We could also have reached the same result with the following code, which explicitly states that the dataset is "ex1", within the "select" command. In this case, we don't use the pipe at the start of the command.

```
select(ex1,Degree,Year) %>% table()
```

```
##        Year
## Degree   1  2  3  4
##   Joint  3  2  3  2
##   Ling  17 42 17  6
##   Phil   6 10  9  1
##   Psych  7 17  7  1
```

If we reverse the order of "Degree" and "Year", we will get the same result, but with each "Degree" as a column, and each "Year" as a row:

```
ex1 %>%
  select(.,Year, Degree) %>%
  table()
```

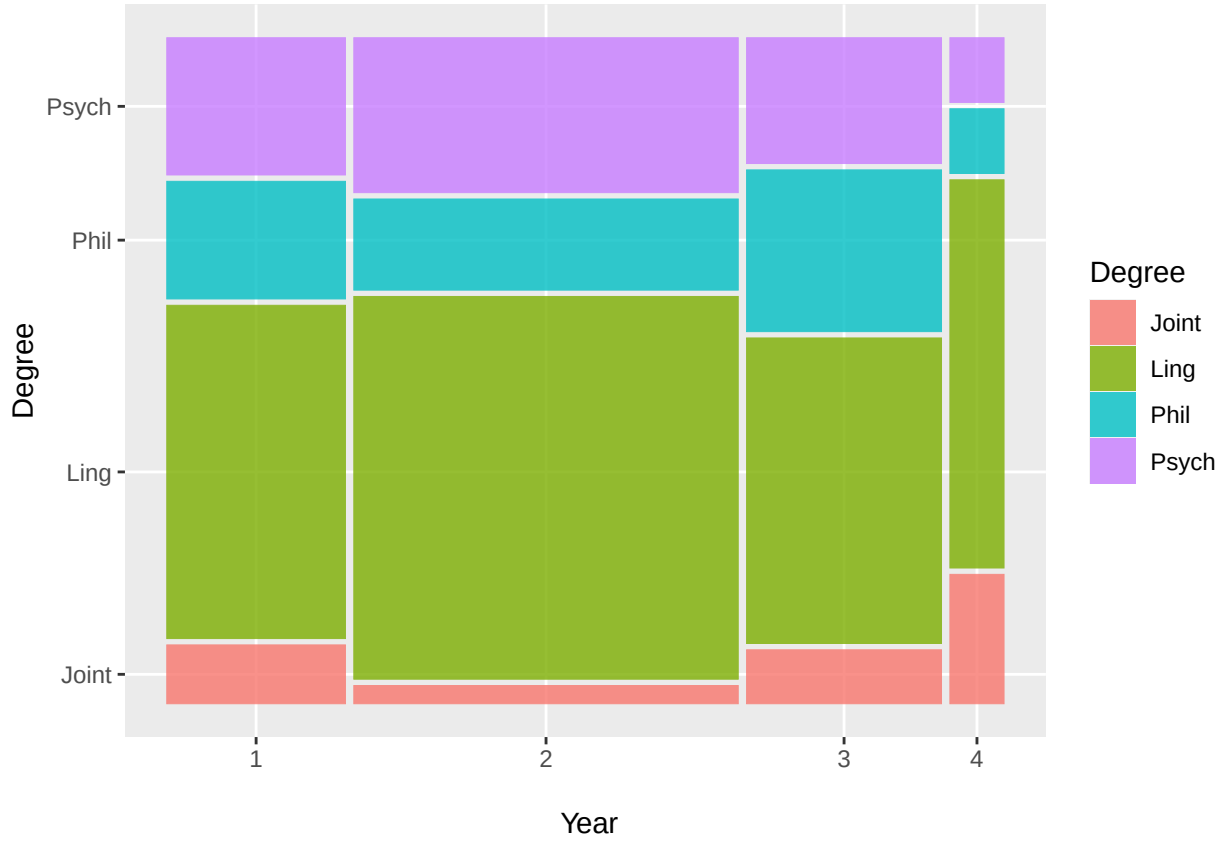
```
##     Degree
## Year Joint Ling Phil Psych
##    1     3   17    6     7
##    2     2   42   10    17
##    3     3   17    9     7
##    4     2    6    1     1
```

Mosaic plots

We can display the same information using a mosaic plot, with the following command. This uses "geom_mosaic()", which requires the "ggmosaic" package to be loaded.

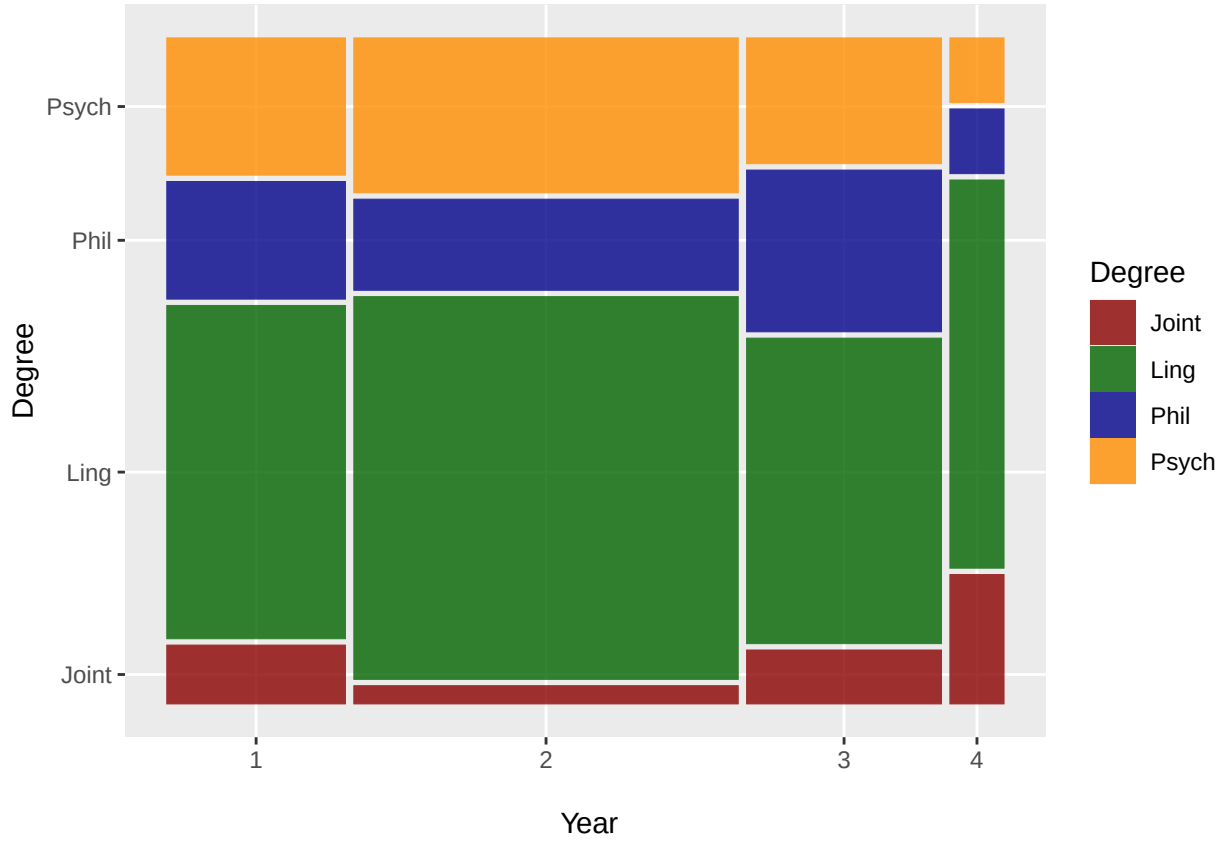
```
ex1 %>%
  ggplot(.) +
  geom_mosaic(aes(x = product(Degree, Year), fill=Degree)) +
  labs(x = "\n Year")
```

```
## Warning: `unite_()` was deprecated in tidyr 1.2.0.
## Please use `unite()` instead.
```



Notice that we specify that we want to plot "Degree" and "Year" using the expression "product(Degree, Year)". Notice also that "fill=Degree" means that we are asking R to use a different colour for each degree, and we allow R to use its own defaults. We can change the colours by adding "scale_fill_manual()", and specifying a list of four colours to go with the four degrees:

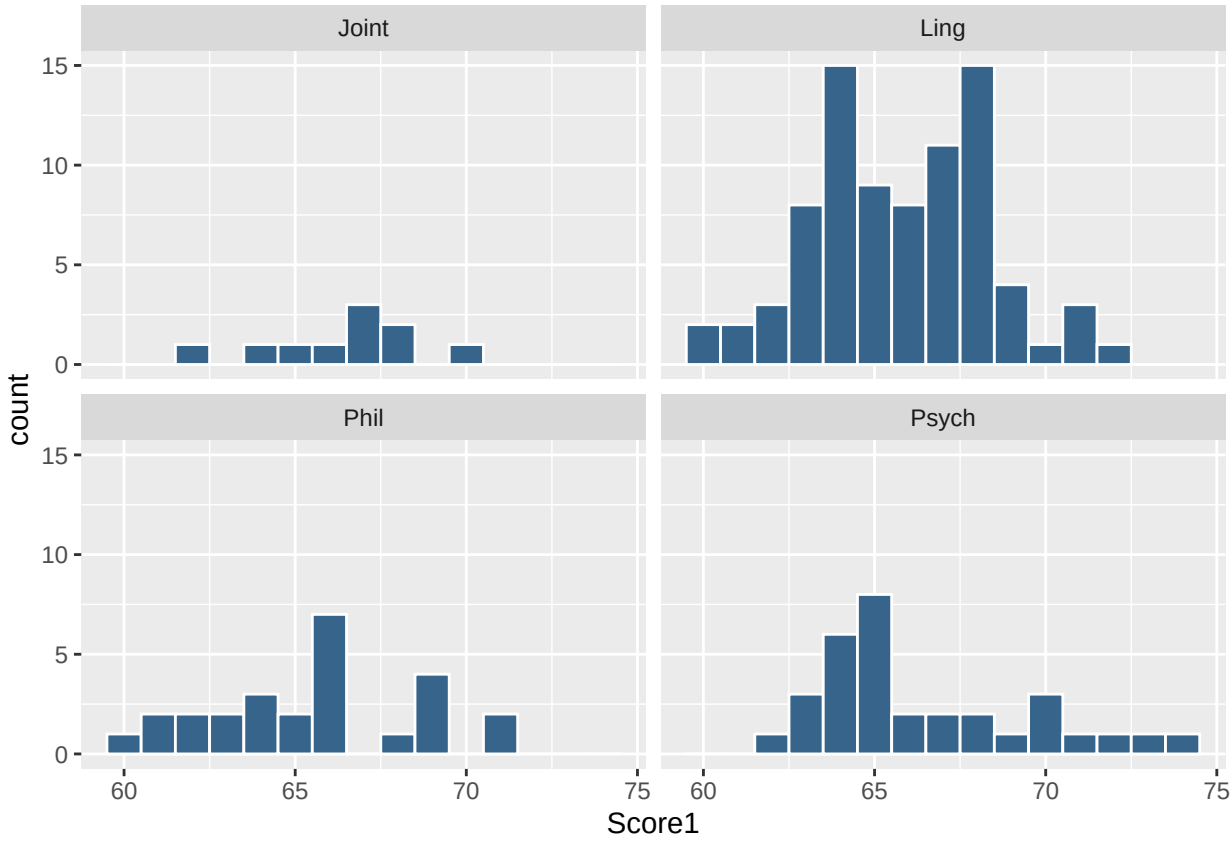
```
ex1 %>%
  ggplot(.) +
  geom_mosaic(aes(x = product(Degree, Year), fill=Degree)) +
  labs(x = "\n Year") +
  scale_fill_manual(values=c("darkred","darkgreen","darkblue","darkorange"))
```



Grouped histograms

We may want to visualise the frequency distributions of the scores for each degree. In this case, we can use grouped histograms. This is done with the following code, which is very similar to the histogram code that we have seen before:

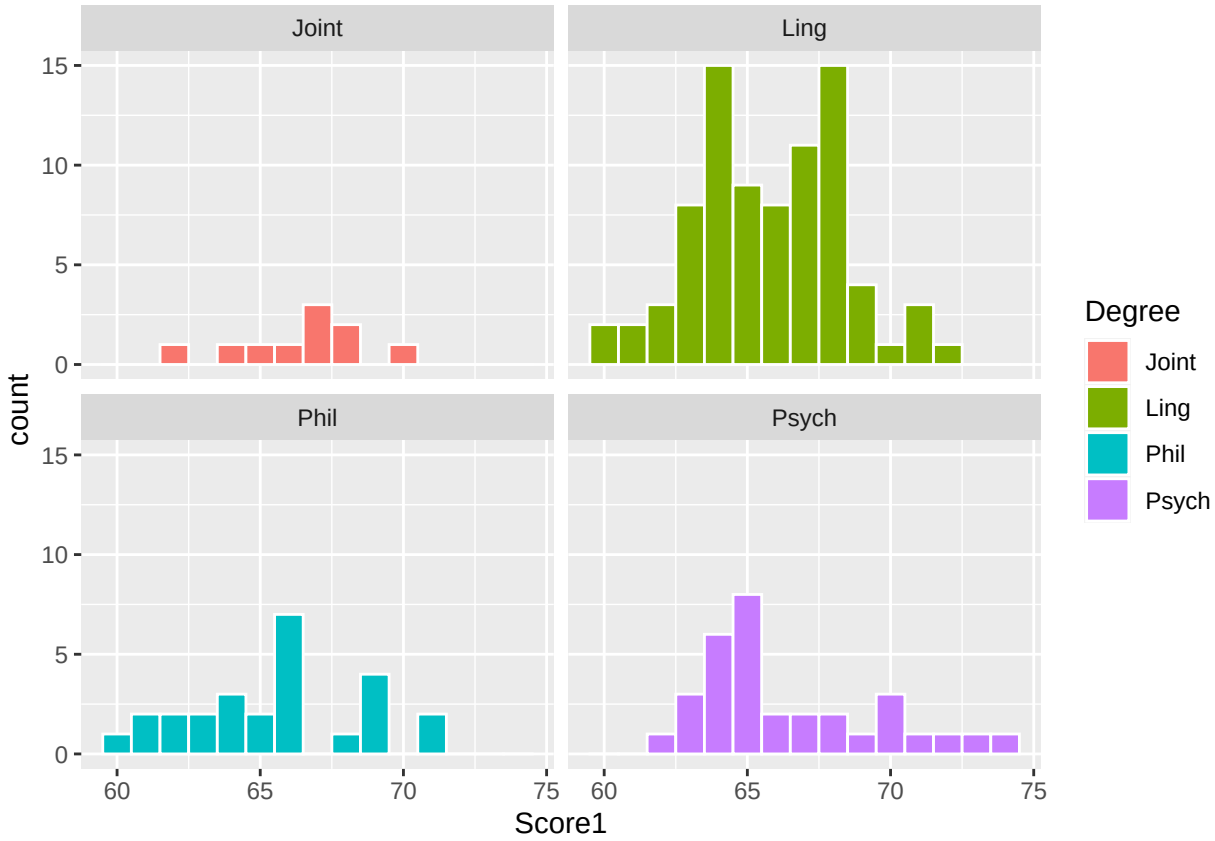
```
ex1 %>%
  ggplot(., aes(x=Score1)) +
  geom_histogram(bins = 15,
                 color = "white",
                 fill = "steelblue4") +
  facet_wrap(~Degree)
```



The part of the code that deals with plotting separate histograms is "facet_wrap(~Degree)", which tells R to plot a separate histogram for each degree. Notice the use of the tilde character "~" in "~Degree".

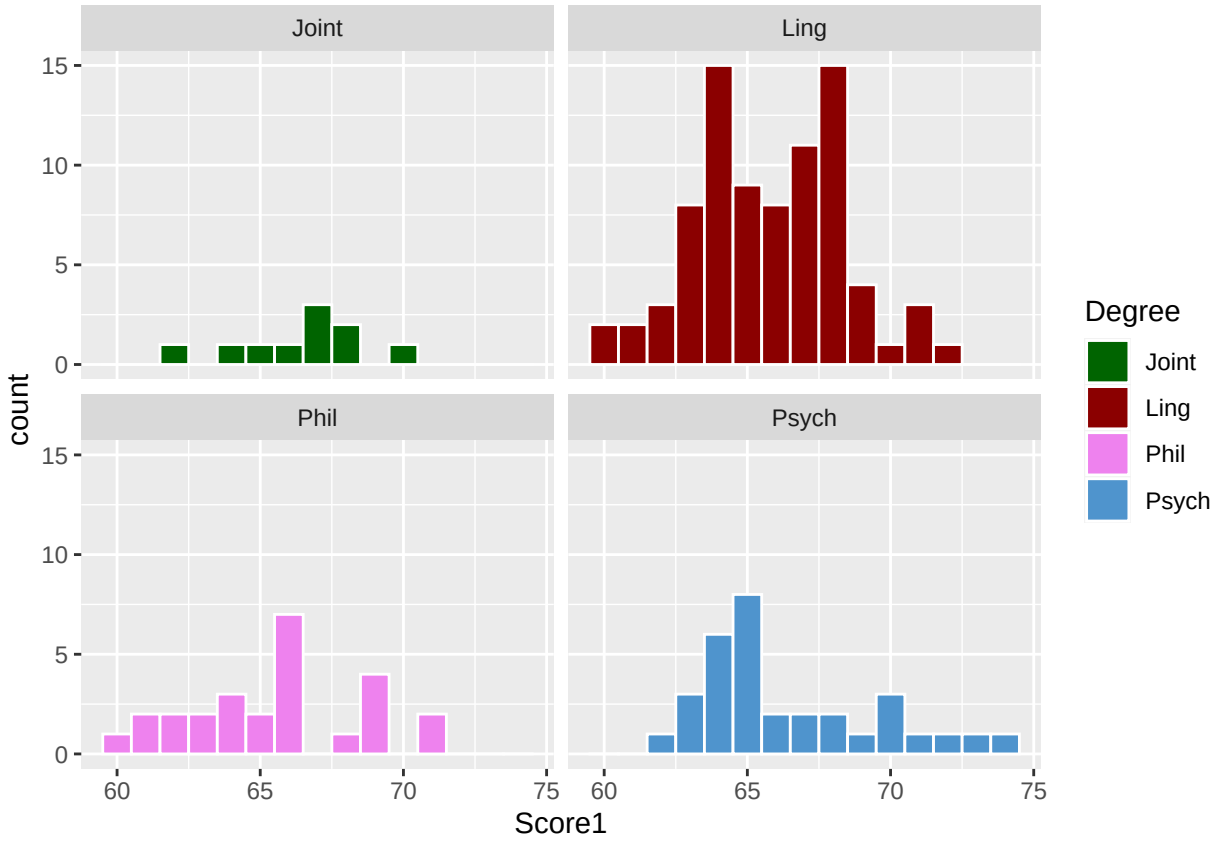
If we want a different fill color for each degree, we need to use "aes(fill=Degree)" inside the "geom_histogram()" command. This will tell R to choose colours based on the degree:

```
ex1 %>%
  ggplot(., aes(x=Score1)) +
  geom_histogram(bins = 15,
                 color = "white",
                 aes(fill=Degree)) +
  facet_wrap(~Degree)
```



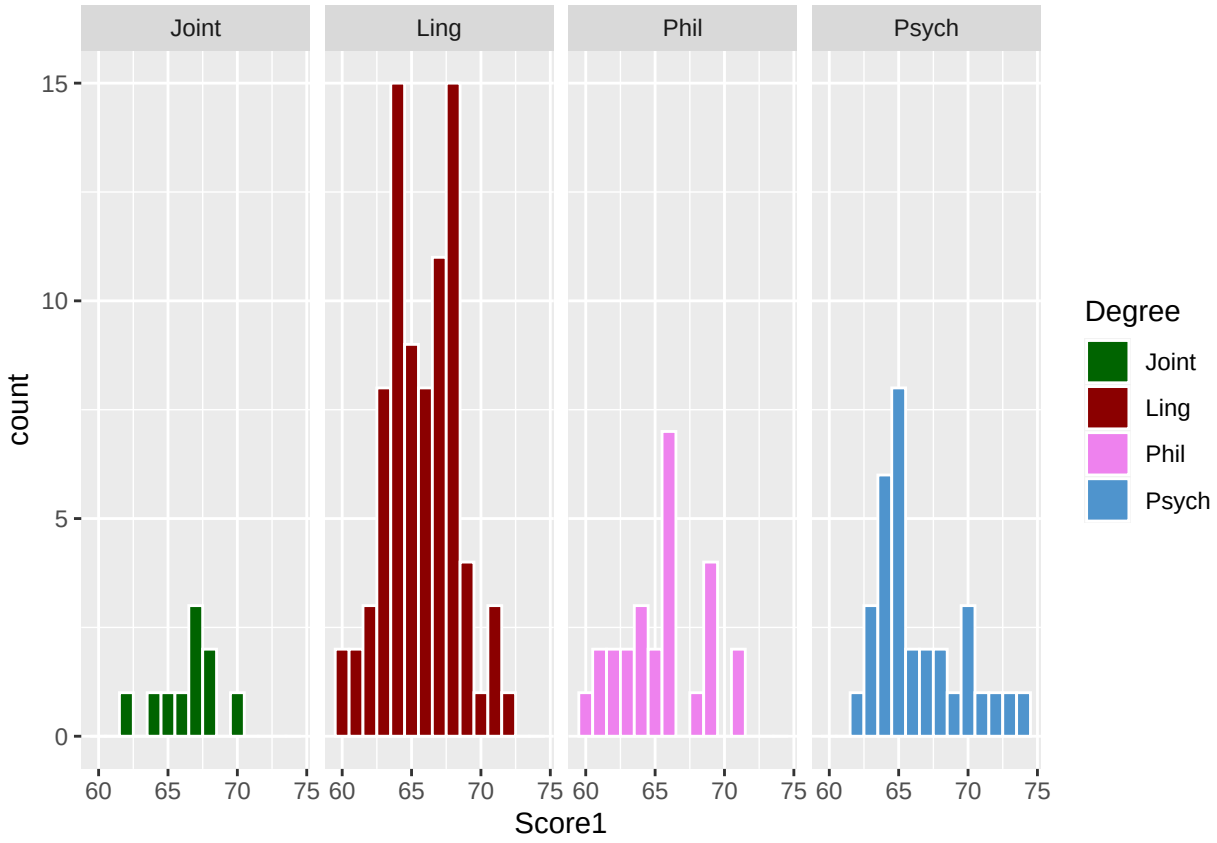
And, if we want to choose our own colours, again, we need to add "scale_fill_manual()", like this:

```
ex1 %>%
  ggplot(., aes(x=Score1)) +
  geom_histogram(bins = 15,
                 color = "white",
                 aes(fill=Degree)) +
  facet_wrap(~Degree) +
  scale_fill_manual(values=c("darkgreen","darkred","violet","steelblue3"))
```



We can also change how the histogram is arranged. For example, we can change the number of rows or columns using "nrow" or "ncol" in the "facet_wrap()" command. For example, in the following, we use "nrow=1", so there will only be one row.

```
ex1 %>%
  ggplot(., aes(x=Score1)) +
  geom_histogram(bins = 15,
                 color = "white",
                 aes(fill=Degree)) +
  facet_wrap(~Degree, nrow=1) +
  scale_fill_manual(values=c("darkgreen","darkred","violet","steelblue3"))
```



Continuous - Continuous relationships and Covariance

The covariance between two variables "x" and "y" describes how "x" and "y" vary together.

Recall the formula for covariance:

$$Cov_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

To give a very intuitive idea of how the covariance formula captures a relationship, we will use a simplified (and unrealistic) example. We define "x" to be a list containing integers 2 to 8, and "y" to be a list containing integers 12 to 18

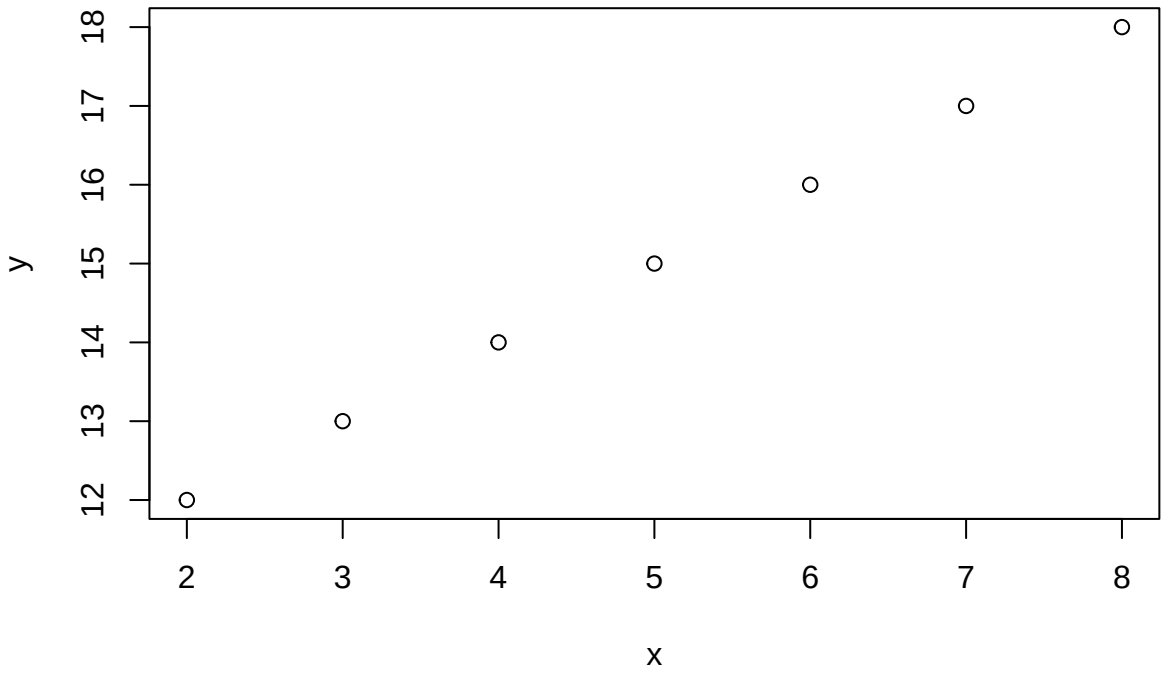
```
x <- c(2,3,4,5,6,7,8)
y <- c(12,13,14,15,16,17,18)
```

(by the way, we could also have done this with the following simpler code):

```
x <- 2:8
y <- 12:18
```

Before we start, we will be able to see that the relationship between "x" and "y" is positive: "x" goes up when "y" goes up, and "x" goes down when "y" goes down. We can confirm this by doing a quick and easy plot, which shows dots following a perfectly linear positive slope:

```
plot(x,y)
```



A positive relationship between "x" and "y" will correspond to a positive value for covariance.

As a first step in explaining the covariance formula, let's take each item in the list "x" and subtract the mean of "x". And also, let's take each item in the list "y" and subtract the mean of "y"

```
x-mean(x)
```

```
## [1] -3 -2 -1  0  1  2  3
```

```
y-mean(y)
```

```
## [1] -3 -2 -1  0  1  2  3
```

For both lists, this results in a negative value for items on the list that are below the mean. For example, in list "x", the first item on the list is 2, and as the mean of the list is 5, the subtraction results in $2 - 5 = -3$. The result is positive for items that are higher than the mean (e.g. $8 - 5 = 3$), and the result is zero for the item that is the same as the mean. Now, let's multiply these two sets of differences together:

```
(x-mean(x)) * (y-mean(y))
```

```
## [1] 9 4 1 0 1 4 9
```

The multiplication results mostly in positive numbers. All differences that came out as negative for list "x" correspond to items that were also negative for list "y". All differences that were positive for "x" were also positive for "y". In both cases, multiplying the relevant values together results in a positive number. Therefore, if we sum these products together, we will get a positive number:

```
sum((x-mean(x)) * (y-mean(y)))
```

```
## [1] 28
```

To calculate covariance, we divide this sum by $n - 1$, and this will also result in a positive number:

```
sum((x-mean(x)) * (y-mean(y)))/(length(x)-1)
```

```
## [1] 4.666667
```

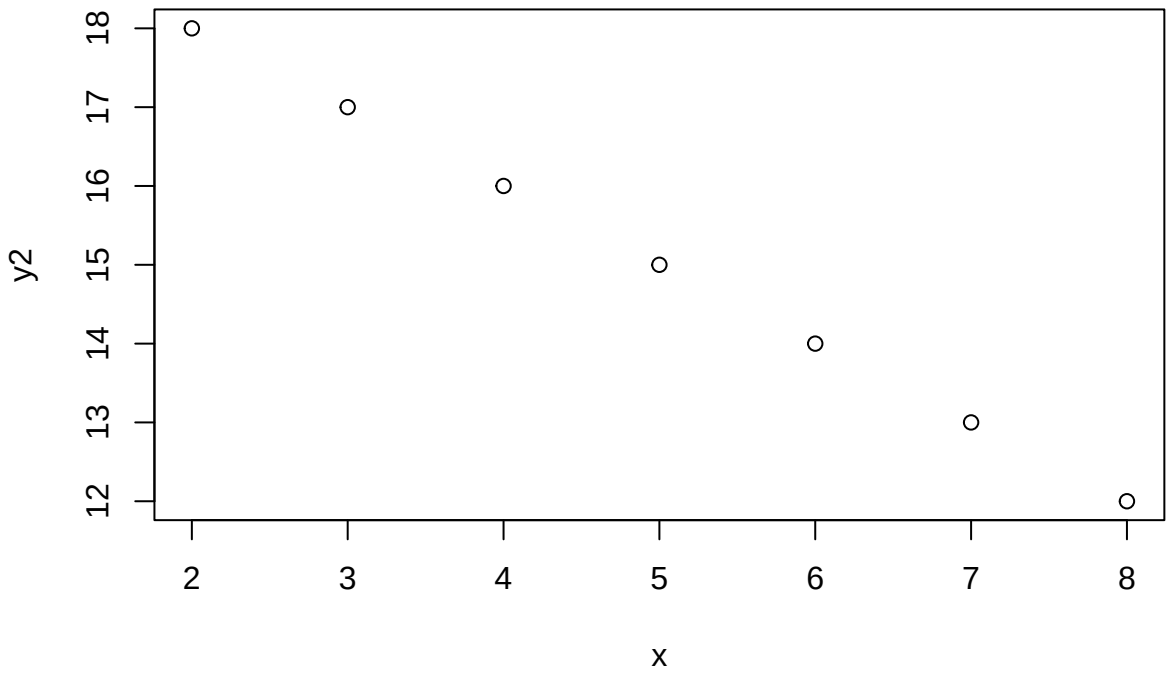
This positive covariance reflects the fact that the relationship between x and y is positive: the positions on list "x" that have higher values (relative to the mean of "x") tend to correspond to positions on list "y" that also have higher values (relative to the mean of "y").

How about if we have a negative relationship? Let's say that we keep the "x" list, define a new list "y2". This time, the higher numbers in "x" correspond to lower numbers in "y":

```
x  <- c(2,3,4,5,6,7,8)
y2 <- c(18,17,16,15,14,13,12)
```

We can get a visual impression of the negative relationship with a plot, and we see that the dots follow a perfectly linear negative slope:

```
plot(x,y2)
```



Now, what happens if we work out the differences between each value and its mean, for "x" and "y2"?

```
x-mean(x)
```

```
## [1] -3 -2 -1  0  1  2  3
```

```
y2-mean(y2)
```

```
## [1]  3  2  1  0 -1 -2 -3
```

Here, the negative differences for list "x" correspond to positive differences for list "y", and vice versa. Multiplying each pair of (positive and negative) values together will result in a negative product.

```
(x-mean(x))*(y2-mean(y2))
```

```
## [1] -9 -4 -1  0 -1 -4 -9
```

Finally, if we take the sum of all these differences, we will end up with a negative value, and that means that the variance (after dividing by "n-1") will also be negative:

```
sum((x-mean(x))*(y2-mean(y2)))/(length(x)-1)
```

```
## [1] -4.666667
```

As usual, there is an easier way to do it. We could have done the variance calculations with the command "cov()":

```
cov(x,y)
```

```
## [1] 4.666667
```

```
cov(x,y2)
```

```
## [1] -4.666667
```

We can also see that variance depends on the scale. Let's say "x" represents measurements in metres, but now we want to represent it as measurements in centimetres, so we need to multiply each value of "x" by 100. If we calculate the covariance between this and the original version of "y", the result will now be different from the previous calculation:

```
cov(x*100,y)
```

```
## [1] 466.6667
```

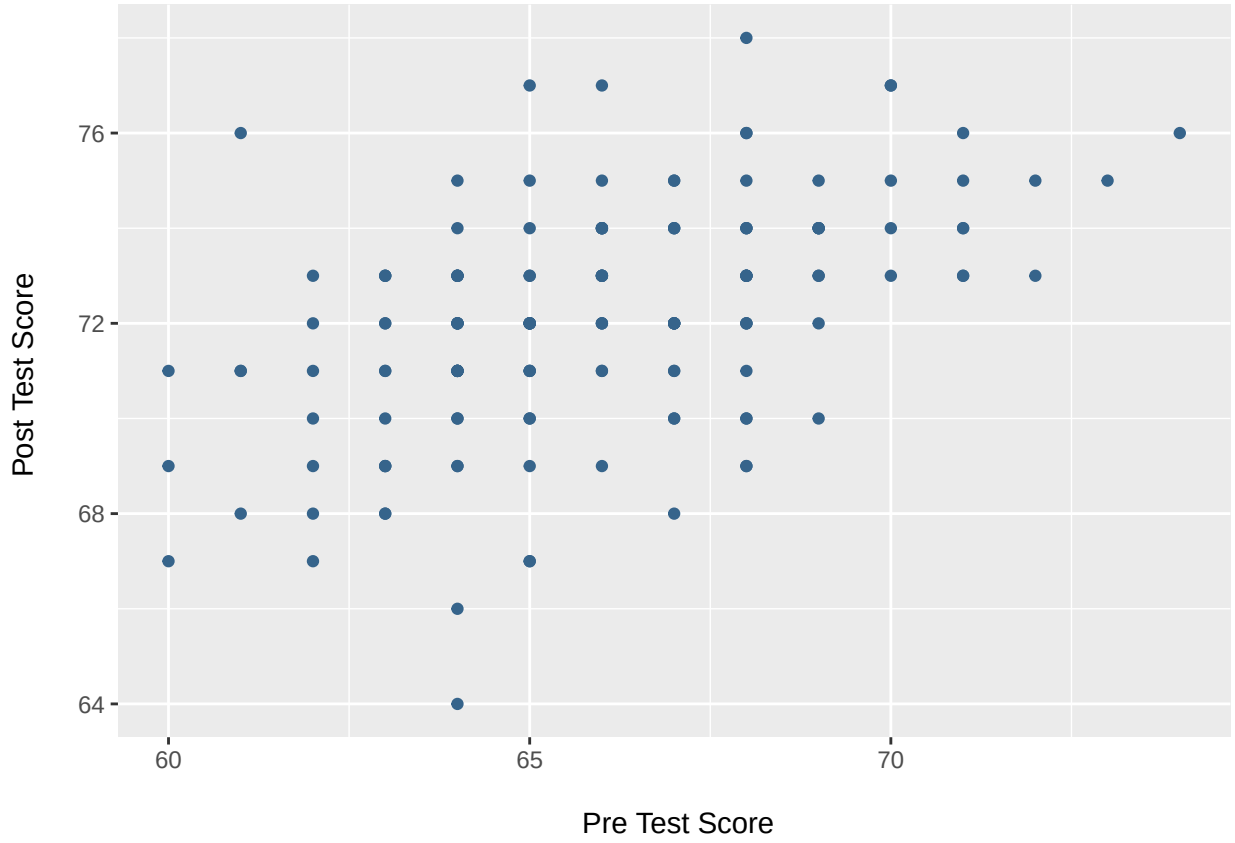
Later in the course, you will learn how to standardize covariance, so that it is not sensitive to measurement scale. The standardized covariance is known as the correlation coefficient. The correlation coefficient always has a value between -1 and 1, regardless of the measurement scale of the variables. A value of -1 for the coefficeint indicates a perfect negative correlation, and a value of 1 indicates a perfect positive correlation.

A slightly more realistic example

Of course, in real life, the relationships between variables are not as neat as the simlistic example above. Let's have a look at the relationship between "Score1" and "Score2" in the ex1 dataset.

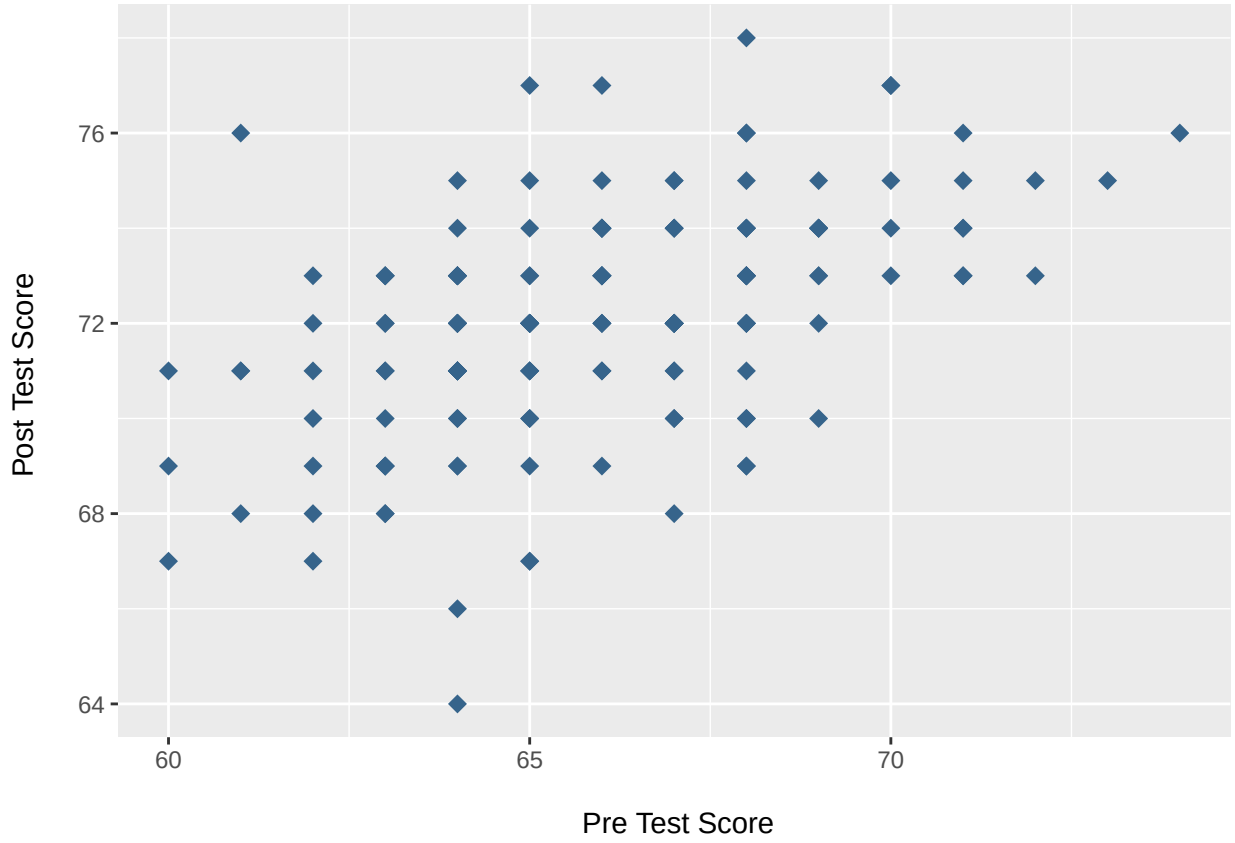
First, we can plot the relationship. In this case, rather than using "plot()", we will use "gglot()" and "geom_point()", giving more options for how the plot will be displayed:

```
ex1 %>%
  ggplot(., aes(x=Score1, y=Score2)) + #<<
  geom_point(colour = "steelblue4") +
  labs(x = "\n Pre Test Score",
       y = "Post Test Score \n")
```



Here, we can see what looks like a positive relationship: the general pattern is for the dots to follow a positive slope, from the bottom left to the top right of the plot, although the pattern is not as neat as the simple case that we looked at above. Before going on to work out the covariance, we could have a look at some of the display options. For example, we can change the size of the points (using "size"), and also the shape (using "shape"):

```
ex1 %>%
  ggplot(., aes(x=Score1, y=Score2)) + #<<
  geom_point(colour = "steelblue4", size=3, shape="diamond") +
  labs(x = "\n Pre Test Score",
       y = "Post Test Score \n")
```



Now, let's calculate the covariance. Using "summarise()" and the pipe, we could explicitly code the formula like this:

```
ex1 %>% summarise(Covariance =
                    round(sum((Score1-mean(Score1))*(Score2-mean(Score2)))/
                                                (length(Score1)-1),digits=2))
```

```
## # A tibble: 1 x 1
##   Covariance
##        <dbl>
## 1       3.36
```

We can confirm that the result is the same as using "cov()":

```
ex1 %>% summarise(Covariance =
                    round(cov(Score1,Score2),digits=2))
```

```
## # A tibble: 1 x 1
##   Covariance
##        <dbl>
## 1       3.36
```

The covariance is positive, as visualised on the plot.