Dapr1: Notes on the Live R session, Week 2

2022-09-26

This week's topic is describing categorical data. We will illustrate this using the dataset that was discussed in the Week 2 lecture. The dataset includes information about 150 students who attended a fictional summer school. Before we start, let's load tidyverse, which we will need for some of the functions in the session. We can do this with the "library(tidyverse)" command, and then R will print some information on the screen, for example, showing the attached packages.

```
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()
```

What is a pipe?

In the "tidyverse" package in R, the symbol %>% is called the "pipe" symbol. The pipe takes the output of the expression to its left, and gives this as the input to the expression on its right. Here is an example to explain how the pipe works. Let's say we start off by making a list of the integers 1 to 10. We can do that in R like this:

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Now, let's say we want to find the log of each of those numbers. One way to do that is to use the pipe symbol, and direct the output of "1:10" to the "log()" function:

```
1:10 %>% log()
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851
```

R has taken the output of "1:10", which is a list of the integers 1 to 10. Then, this output is given as the input of log(), and the output of that computation is the log of the integers 1 to 10. Now, let's say we don't want to waste so much space, and we want to round the numbers to only two decimal places. We can pipe the output of "log()" to another function, "round()", which will round the number to a spcified number of decimal places, in this case 2.

```
1:10 %>% log() %>% round(digits=2)
```

```
##  [1] 0.00 0.69 1.10 1.39 1.61 1.79 1.95 2.08 2.20 2.30
```

The pipe symbol will be used a lot in this course, so thinking of this example will help you to understand how it works. Incidentally, there is another way to do this same calculation, without using the pipe:

```
round(log(1:10),digits=2)
```

```
##  [1] 0.00 0.69 1.10 1.39 1.61 1.79 1.95 2.08 2.20 2.30
```

In this second way, R evaluates the expressions from the most deeply embedded and moves outward. This means that the order in which the function names are written from left to right is not the same as the order that they are evaluated. This is why some people prefer to use the pipe method.

Reading in the data and defining a factor

Now, back to the main topic. Let's start by reading in the data file. The data file is on the departmental github website, and it is in csv format, so first we read it into R using read_csv.

```
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.
```

We have saved the data with the name ex1. Let's have a look at the first few lines:

```
head(ex1)
```

```
## # A tibble: 6 x 5
##   ID    Degree  Year Score1 Score2
##   <chr> <chr>  <dbl>  <dbl>  <dbl>
## 1 ID101 Psych      2     71     74
## 2 ID102 Ling       2     65     72
## 3 ID103 Ling       2     64     72
## 4 ID104 Phil       1     69     74
## 5 ID105 Ling       3     62     69
## 6 ID106 Ling       1     68     72
```

This command will show the first six lines of data, and it will report the type of each variable column. Now, let's say we will be doing an analysis where "Degree" is used a factor. If this is the case, we would need to tell R to treat this variable as a factor. One way to do this is as follows:

```
ex1$Degree <- factor(ex1$Degree)
```

There is another way to do this, using the “pipe”. The “mutate()” function is also used. This function is used to manipulate columns in a data frame. It could be used to add a column, or it could be used to change an existing column in some way. Here, we are using “mutate()” to declare that the variable represented by the column is to be treated as a factor. We take the data that we saved in “ex1”, and we pass that data to “mutate()” using the pipe sybol “%>%”. Within the “mutate()” function, we specify that we want to change the type of the column “Degree” so that it is treated as a factor.

```
ex1 %>% mutate(Degree = factor(Degree))
```

```
## # A tibble: 150 x 5
##    ID    Degree  Year Score1 Score2
##    <chr> <fct>  <dbl>  <dbl>  <dbl>
##  1 ID101 Psych      2     71     74
##  2 ID102 Ling       2     65     72
##  3 ID103 Ling       2     64     72
##  4 ID104 Phil       1     69     74
##  5 ID105 Ling       3     62     69
##  6 ID106 Ling       1     68     72
##  7 ID107 Phil       3     66     75
##  8 ID108 Psych      1     64     71
##  9 ID109 Psych      3     65     73
## 10 ID110 Ling       1     64     72
## # ... with 140 more rows
```

Calculating and displaying the frequency distributions

We can now count how many students are doing each of the degrees:

```
ex1 %>% count(Degree)
```

```
## # A tibble: 4 x 2
##   Degree     n
##   <fct>  <int>
## 1 Joint     10
## 2 Ling      82
## 3 Phil      26
## 4 Psych     32
```

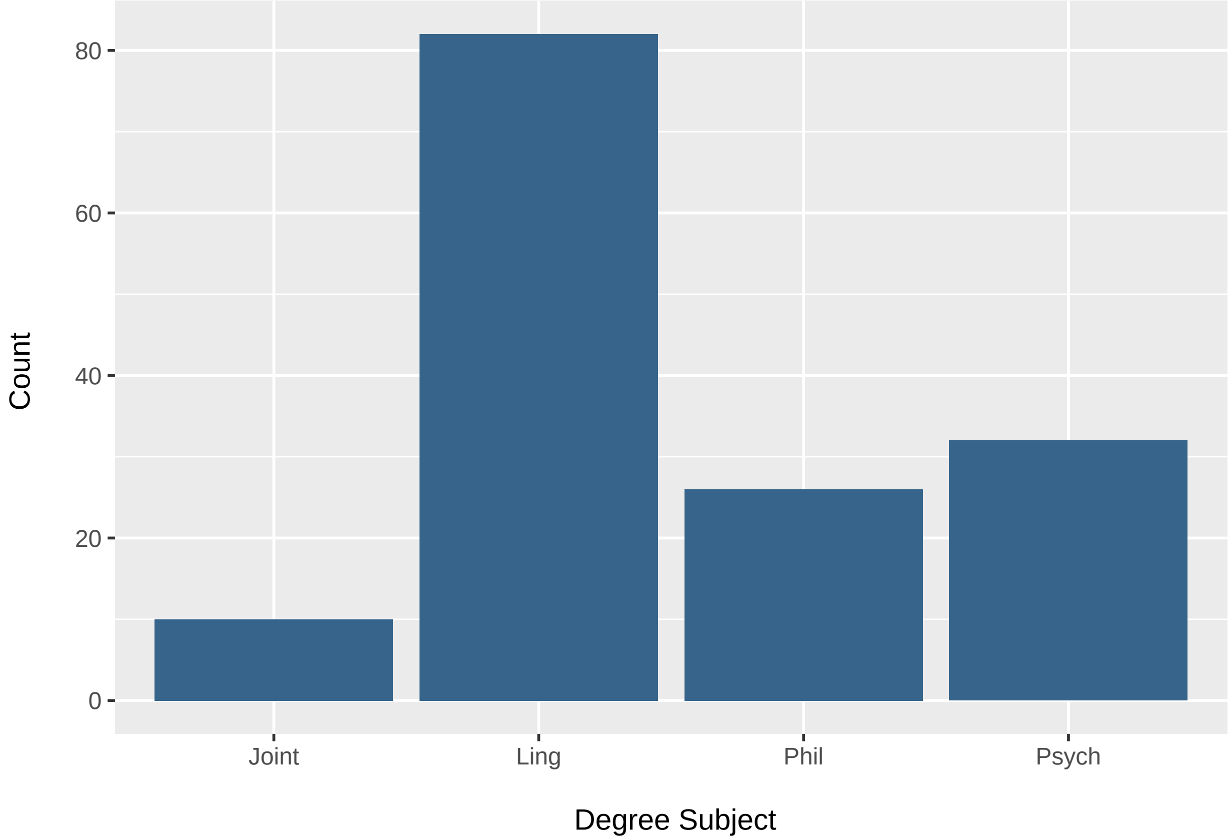
We can also go further and print out “Percent” column with the percentages. Here, we first apply “count()”, then we pipe the output of that to a “mutate()” command that adds the new column. The column takes each value of the “n” column (which shows the number of students in the category), and divides it by the sum of all values in the “n” column (the total of 150 students) to get a proportion. The proportion is then multiplied by 100, and rounded to 2 decimal places:

```
ex1 %>%
  count(Degree) %>%
  mutate(
    Percent = round((n/sum(n))*100,2))
```

```
## # A tibble: 4 x 3
##   Degree     n Percent
##   <fct>  <int>   <dbl>
## 1 Joint     10    6.67
## 2 Ling      82   54.7
## 3 Phil      26   17.3
## 4 Psych     32   21.3
```

Now we will do a graph of the frequency distribution:

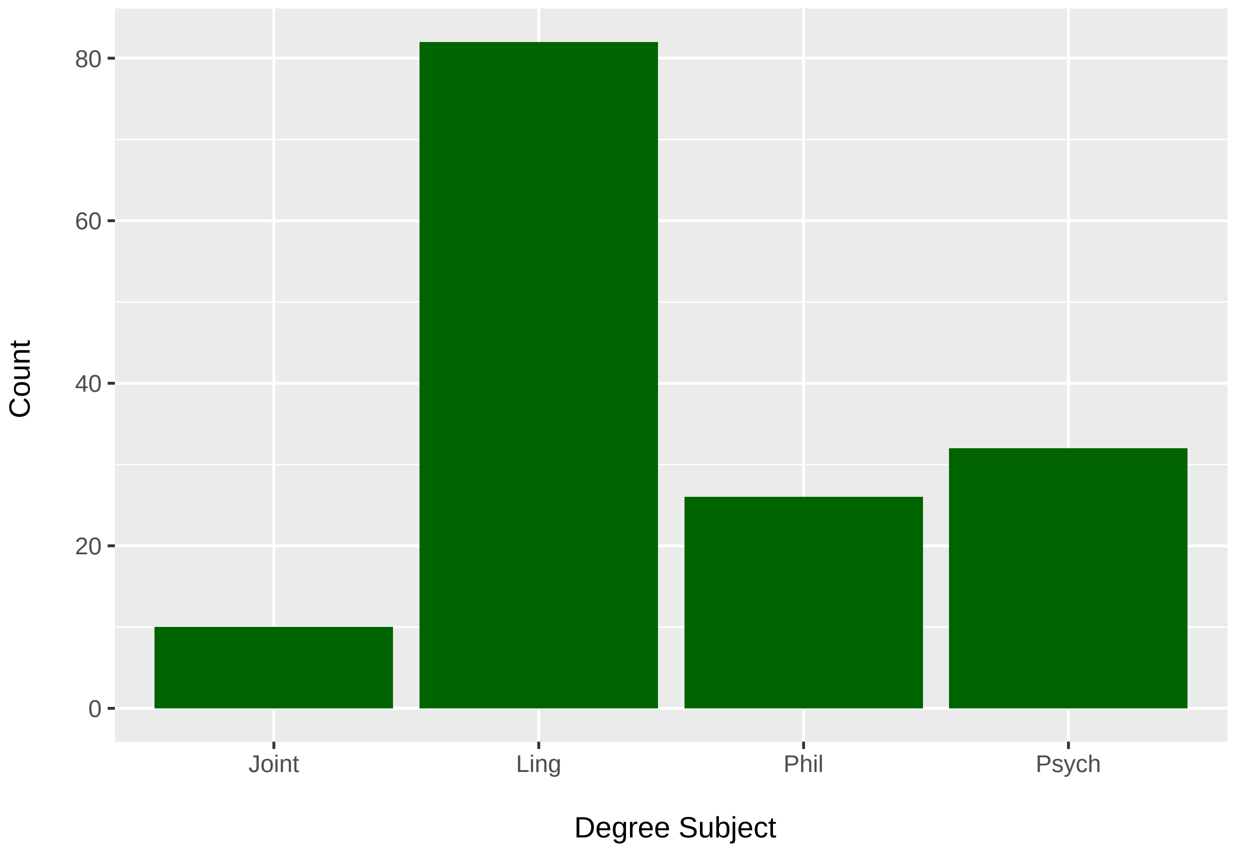
```
ex1 %>%
    ggplot(., aes(x=Degree)) +
    geom_bar(fill="steelblue4") +
    labs(x = "\n Degree Subject", y = "Count \n")
```



Here, we are specifying that the x-axis should include the value of “Degree”, the bars should be blue, and the labels are “Degree Subject” and “Count”. Note that the “\n” charater adds a new line, and this can be used to put extra space between the graph and the axis label.

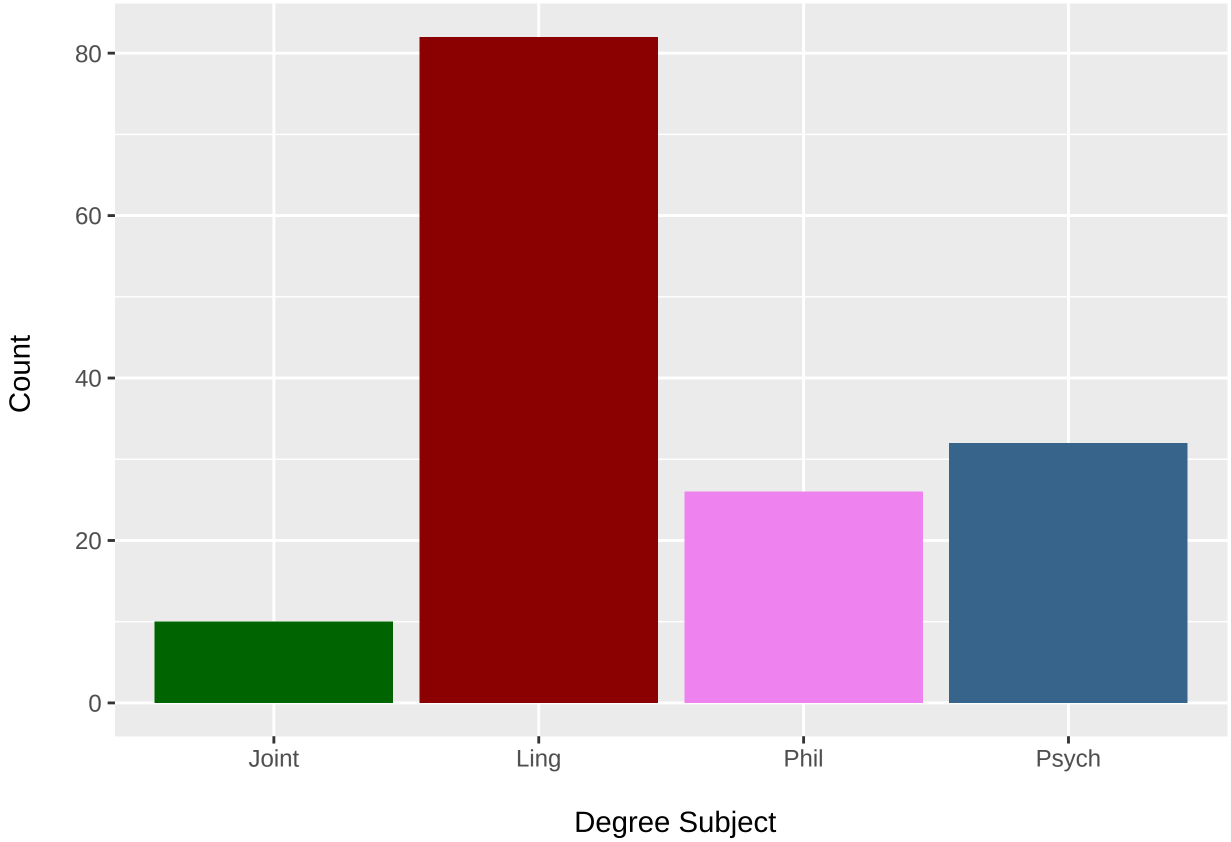
What happens if we want the bars to be a different colour? How about dark green?

```
ex1 %>%
    ggplot(., aes(x=Degree)) +
    geom_bar(fill="darkgreen") +
    labs(x = "\n Degree Subject", y = "Count \n")
```



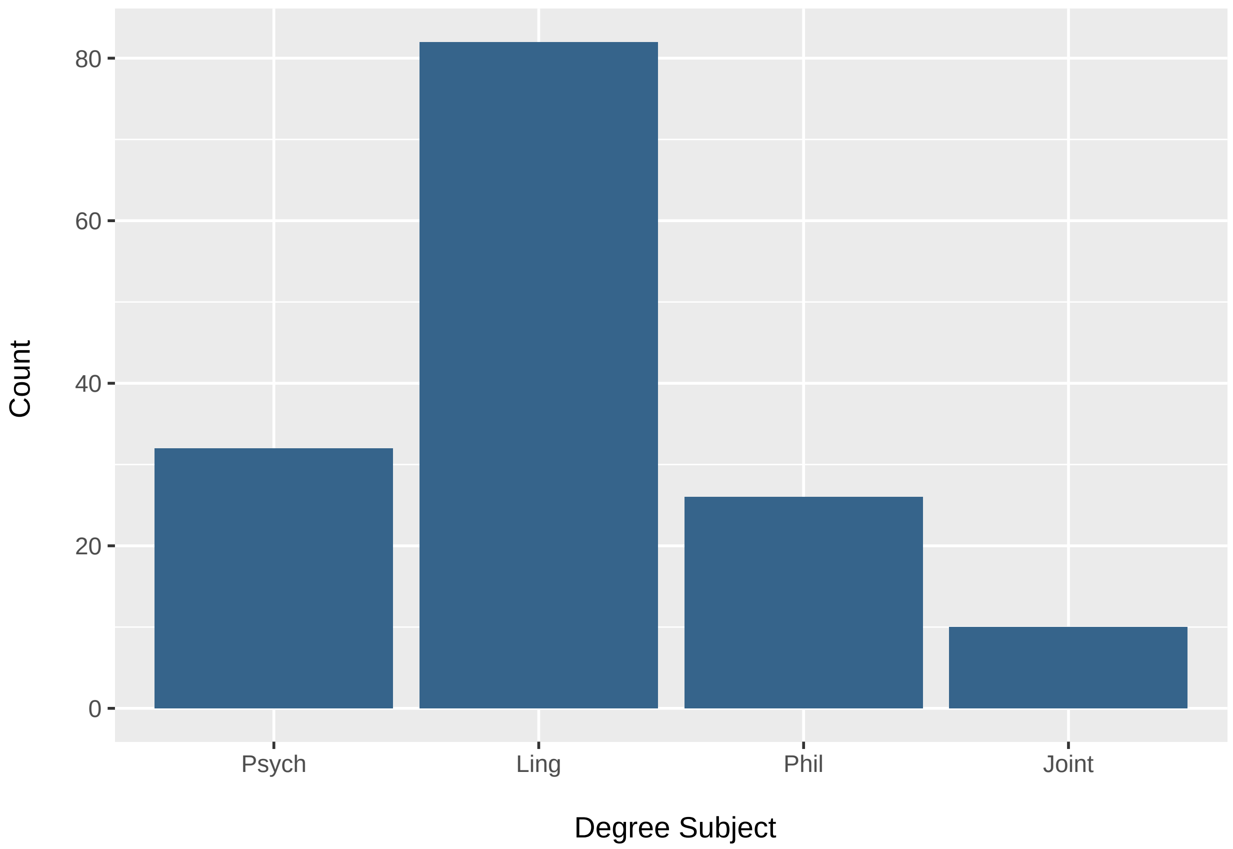
How about if we want each bar to be a different colour? Then we could use a list of colours:

```
ex1 %>%
    ggplot(., aes(x=Degree)) +
    geom_bar(fill=c("darkgreen","darkred","violet","steelblue4")) +
    labs(x = "\n Degree Subject", y = "Count \n")
```



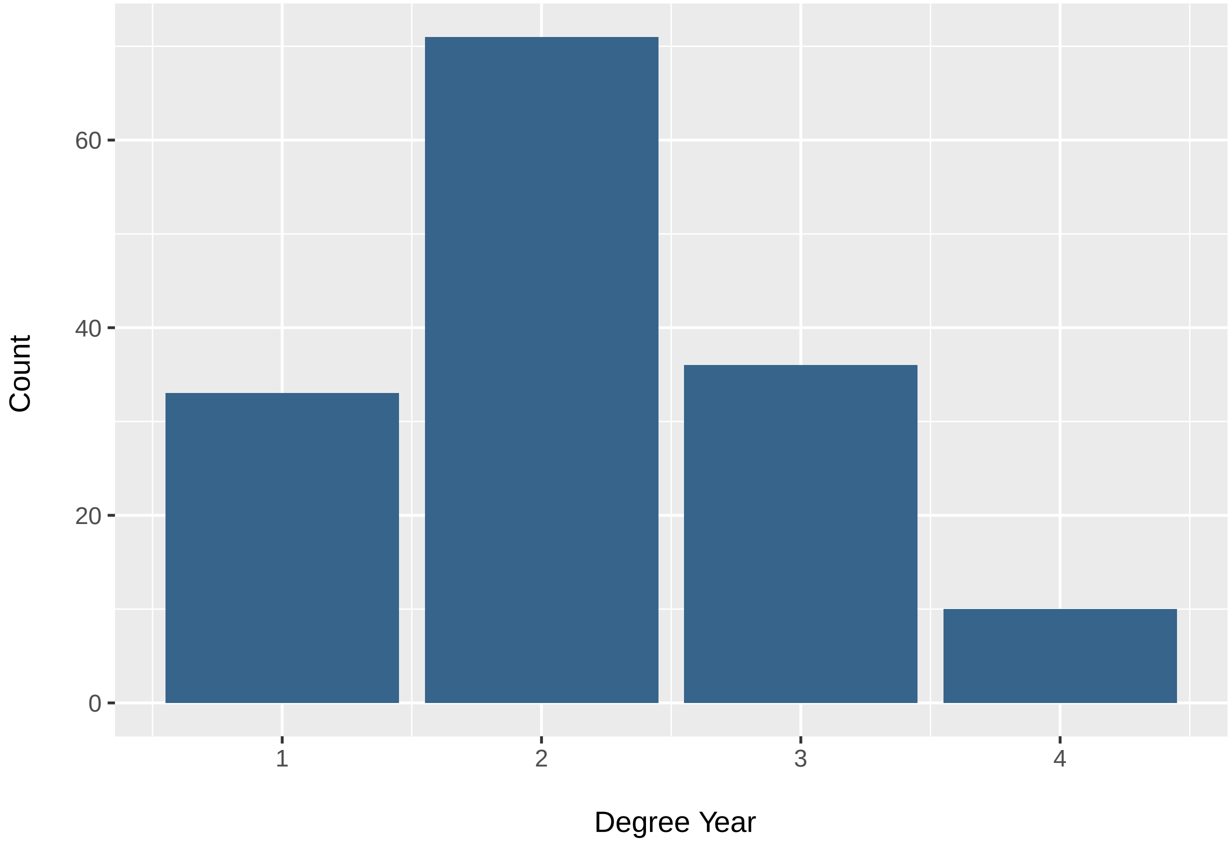
How about changing the order of the bars? We have already declared “Degree” to be a factor, but now we could specify an order:

```
ex1 %>% mutate(Degree = factor(Degree,levels=c("Psych","Ling","Phil","Joint"))) %>%
  ggplot(., aes(x=Degree)) +
    geom_bar(fill="steelblue4") +
    labs(x = "\n Degree Subject", y = "Count \n")
```



To plot the year instead of the degree subject, we can simply write “x=Year” in the aesthetics section, and use a different label for the x-axis. R will use the information in the “Year” column to make the counts.

```
ex1 %>%
    ggplot(., aes(x=Year)) +
    geom_bar(fill="steelblue4") +
    labs(x = "\n Degree Year",
         y = "Count \n")
```



Cumulative percentages, median and interquartile range

OK, now let's look at the percentages of students in each year group, and put them in a column called "Percent". Then we will also calculate the cumulative percentages, making a new column "Cumulative".

```
ex1 %>% count(Year) %>%
  mutate(Percent = round((n/sum(n)*100),digits=2), Cumulative = cumsum(Percent))
```

```
## # A tibble: 4 x 4
##    Year     n Percent Cumulative
##   <dbl> <int>   <dbl>      <dbl>
## 1     1    33   22          22
## 2     2    71   47.3        69.3
## 3     3    36   24          93.3
## 4     4    10    6.67      100
```

From this, we can see that Year 2 is the first category where the cumulative percentage reaches 25%, and Year 3 is the first category where the cumulative percentage reaches 75%. Therefore Year 2 is quartile 1 (25% quartile), and Year 3 is quartile 3 (75% quartile). We can confirm this with the quantile() function:

```
quantile(ex1$Year, c(.25, .75))
```

```
## 25% 75%
##   2   3
```

To find the interquartile range, we take the difference: 3-2=1. We can confirm this with the IQR function:

```
IQR(ex1$Year)
```

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

We can also see how the cumulative percentages relate to the median. Here, Year 2 is the point where the cumulative percentage reaches 50% (year 2 is half-way through the ordered list of year categories of the 150 students), so 2 is the median. We can confirm this with the "median()" function:.

```
median(ex1$Year)
```

```
## [1] 2
```