Dapr1: Notes on the Live R session, Week 3

This week's topic is describing continuous data. We will again use the dataset from the previous lecture. First we will load "tidyverse", and read the data into R, and name it "ex1".

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

Remember that the dataset includes information about 150 fictional students, including their degree, their year of study, and scores in two tests. Let's look at the first few lines of data to remind ourselves of what it looks like.

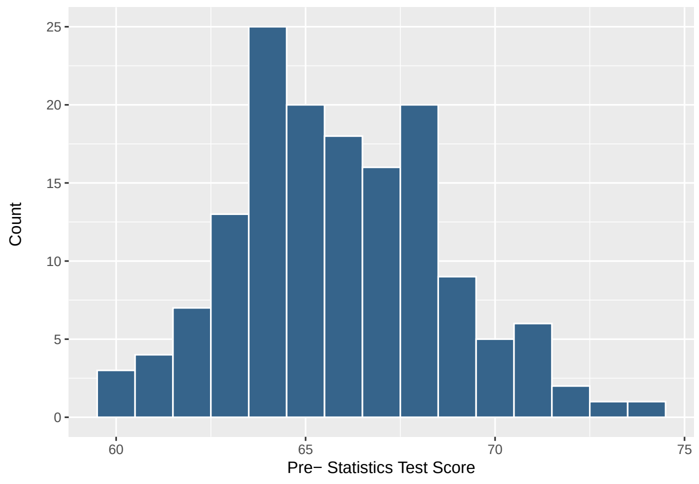
```
head(ex1)
```

```
## # 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
```

Histograms

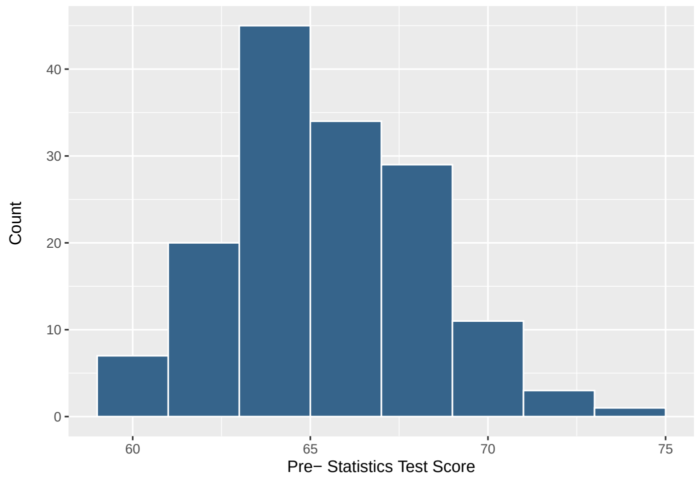
Now let's try a histogram of Score1, the students' scores in the first test:. To do this, we use the "ggplot()" command, just as we did last week for bar plots. We then use "geom_histogram()" to tell R that the type of graph should be a histogram, and here we include information about the number of bins ("bins=15"), and the colour of the bars ("fill") and their outlines ("color"). Finally, we add information about the labels of the x-axis and y-axis. Each section of the command is separate by a plus sign +

```
ggplot(data=ex1, aes(x=Score1)) +
geom_histogram(bins = 15,
               color = "white",
               fill = "steelblue4")+
xlab("Pre- Statistics Test Score") +
ylab("Count \n")
```



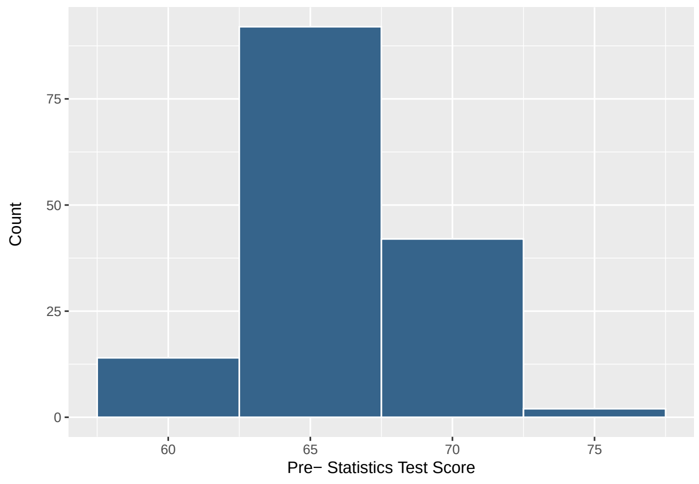
As usual, there are things that can be changed in the command. For example, we could change the number of bins. How about 8 bins?

```
ggplot(data=ex1, aes(x=Score1)) +
geom_histogram(bins = 8,
               color = "white",
               fill = "steelblue4")+
xlab("Pre- Statistics Test Score") +
ylab("Count \n")
```



Instead of specifying how many bins we want, we could also directly specify the width of the bins. For example, we could specify that each bin should cover a range of 5 percentage points, using “binwidth=5”

```
ggplot(data=ex1, aes(x=Score1)) +
geom_histogram(binwidth = 5,
               color = "white",
               fill = "steelblue4")+
xlab("Pre- Statistics Test Score") +
ylab("Count \n")
```



Now let's find the mean of Score1. We could do this by explicitly calculating the sum of the Score1 values and dividing that by the total number of values (the length of the vector):

```
sum(ex1$Score1)/length(ex1$Score1)
```

```
## [1] 65.9
```

Or we could simply use the "mean()" function in R

```
mean(ex1$Score1)
```

```
## [1] 65.9
```

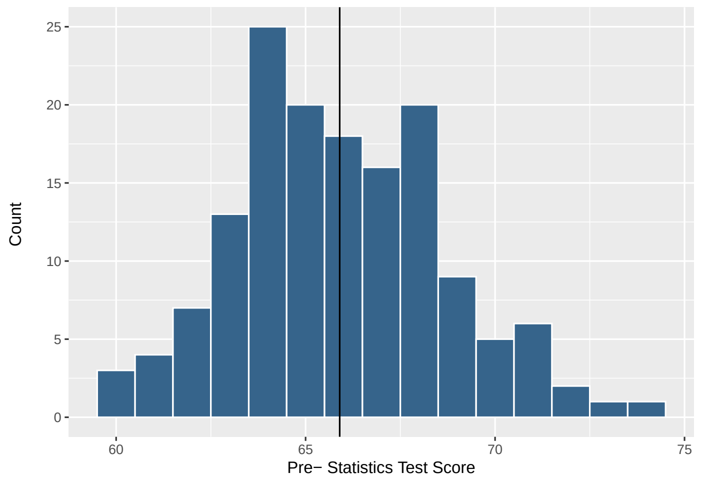
In the above calculations we used the $ sign to specify the column. However, we could also do it the tidyverse way, by piping the data to the "summarise()" function. In this case, we make a new data frame and we add the mean to a new column, which we choose to name "mean". In this case, we don't need to use the $ sign to specify the column

```
ex1 |>
summarise(
   mean = mean(Score1)
)
```

```
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  65.9
```

As a bonus, we can also add the mean to the histogram, by plotting a vertical line at the mean value on the x-axis. The vertical line can be plotted using "geom_vline()", specifying the location with "xintercept":

```
ggplot(data=ex1, aes(x=Score1)) +
geom_histogram(bins = 15,
               color = "white",
               fill = "steelblue4")+
xlab("Pre- Statistics Test Score") +
ylab("Count \n") +
geom_vline(xintercept = mean(ex1$Score1))
```



Standard Deviation

We will explain the standard deviation in stages, building up the formula as we go along. We will consider the variable Score1. At the heart of the calculation is the difference between each score and the mean. If we do this for each value, the result will be a list of 150 numbers (R will show the first 10 in the command below). A positive value will be shown if the score is greater than the mean, and a negative value if the score is less than the mean:

```
ex1 |> summarise(Score1 - mean(Score1))
```

```
## # A tibble: 150 x 1
##    `Score1 - mean(Score1)`
##                      <dbl>
##   1                   5.10
##   2                  -0.900
##   3                  -1.90
##   4                   3.10
##   5                  -3.90
##   6                   2.10
##   7                   0.100
##   8                  -1.90
##   9                  -0.900
##  10                  -1.90
## # i 140 more rows
```

The next thing we will do is square all these numbers. As the square always results in a positive number, all values will now be positive:

```
ex1 |> summarise((Score1 - mean(Score1))^2)
```

```
## # A tibble: 150 x 1
##    `(Score1 - mean(Score1))^2`
##                          <dbl>
##   1                     26.0
##   2                      0.810
##   3                      3.61
##   4                      9.61
##   5                     15.2
##   6                      4.41
##  7                      0.0100
##  8                      3.61
##  9                      0.810
## 10                      3.61
## # i 140 more rows
```

Now, we can sum up all these squared differences using "sum()":

```
ex1 |> summarise(sum((Score1 - mean(Score1))^2) )
```

```
## # A tibble: 1 x 1
##   `sum((Score1 - mean(Score1))^2)`
##                              <dbl>
## 1                            1148.
```

Then, if we divide this value by the total number of scores, we get the (population) variance. We can give it the name PopVariance in the output table:

```
ex1 |> summarise(PopVariance = sum((Score1 - mean(Score1))^2)/length(Score1))
```

```
## # A tibble: 1 x 1
##   PopVariance
##         <dbl>
## 1        7.65
```

The standard deviation is now simply the square root of the variance. The full command below will produce a table including both the population variance and the population standard deviation, rounded to 2 decimal places. Remember, these are the formulas for the population variance and standard deviation, because we divide by the number of data points (N, or "length(Score1)"). So we can label them in this way, to avoid confusion.

```
ex1 |>
summarise(
  Variable = "Statistics Test Score",
  Population_Variance = round((sum((Score1 - mean(Score1))^2))/length(Score1),2),
  Population_SD = round(sqrt((sum((Score1 - mean(Score1))^2))/length(Score1)),2)
)
```

```
## # A tibble: 1 x 3
##   Variable              Population_Variance Population_SD
##   <chr>                               <dbl>         <dbl>
## 1 Statistics Test Score                7.65          2.77
```

Finally, in most cases in statistics, we are measuring samples, and not the whole population. For this reason, it is more usual to use the sample variance and sample standard deviation, not the population variance and standard deviation. To get the sample variance and SD, we need to divide by n-1 (the total number of data points minus 1), rather than N:

```
ex1 |>
summarise(
  Variable = "Statistics Test Score",
  Sample_Variance = round(sum((Score1 - mean(Score1))^2)/(length(Score1)-1),2),
  Sample_SD = round(sqrt(sum((Score1 - mean(Score1))^2)/(length(Score1)-1)),2)
)
```

```
## # A tibble: 1 x 3
##   Variable              Sample_Variance Sample_SD
##   <chr>                           <dbl>     <dbl>
## 1 Statistics Test Score             7.7      2.78
```

In fact, we could have saved all this trouble by simply using the R functions "var()" and "sd()":

```
round(var(ex1$Score1),2)
```

```
## [1] 7.7
```

```
round(sd(ex1$Score1),2)
```

```
## [1] 2.78
```

You can see that R calculates the sample versions of variance and standard deviation by default, not the population versions.

The effect of extreme values on the mean vs. median, and the SD vs. IQR

Let's say you have a list of six numbers, comprising the set of integers from 2 to 8. We store this list of numbers as x. We also have another list of numbers that is identical to the first, except that the maximum value is 8000 instead of 8, in other words, it includes an extreme value. We store this new list as y:

```
x <- c(2,3,4,5,6,7,8)
y <- c(2,3,4,5,6,7,8000)
```

It turns out that the means of these two data sets are very different:

```
mean(x)
```

```
## [1] 5
```

```
mean(y)
```

```
## [1] 1146.714
```

The standard deviations of x and y are also very different:

```
sd(x)
```

```
## [1] 2.160247
```

```
sd(y)
```

```
## [1] 3022.015
```

However, the medians of both datasets are the same:

```
median(x)
```

```
## [1] 5
```

```
median(y)
```

```
## [1] 5
```

Also the interquartile ranges of both sets are the same

```
IQR(x)
```

```
## [1] 3
```

```
IQR(y)
```

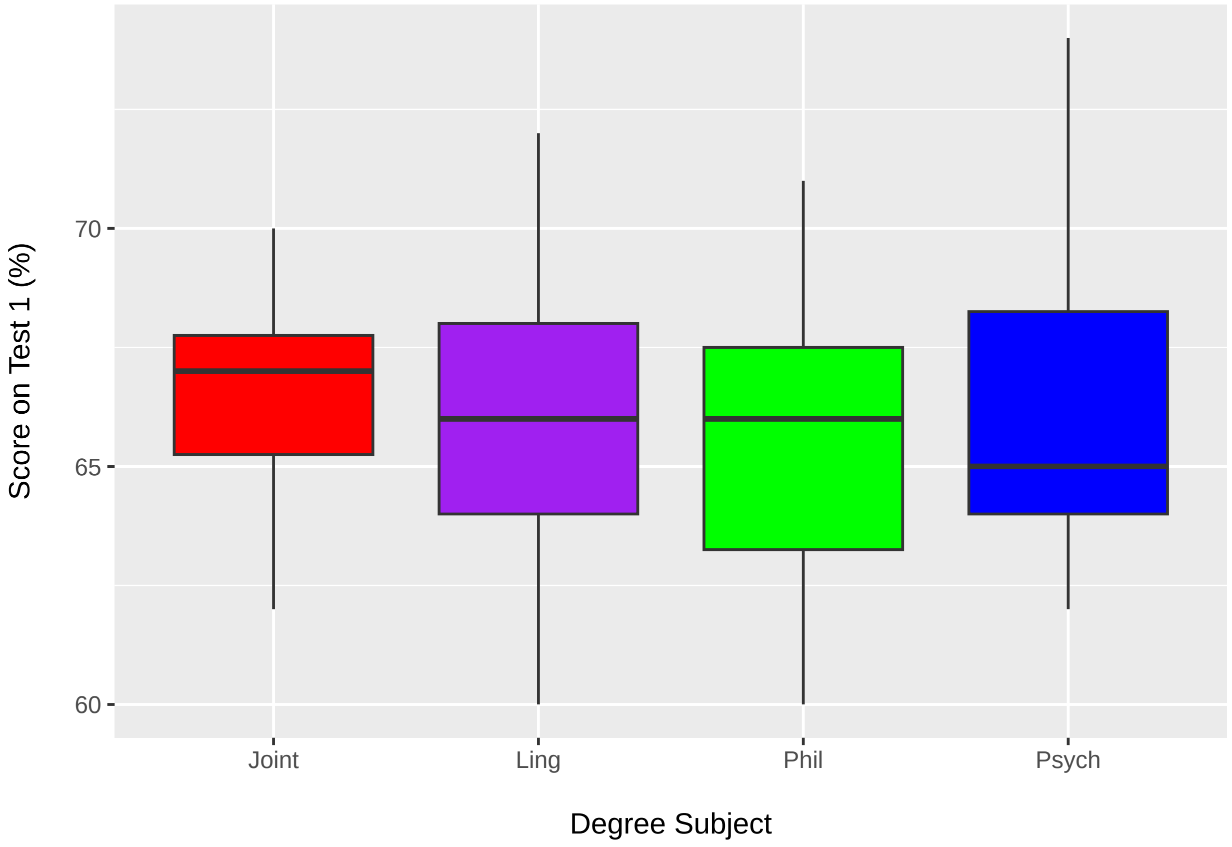
```
## [1] 3
```

In general, for commonly observed distributions, the mean and standard deviation are more sensitive to extreme values than the median and the interquartile range. Why is this so?

Box Plots

We will now introduce a new type of graph, namely, the Box Plot. Box plots are a way to visualise the central tendency and dispersion of a set of data. Let's plot the Score1 of the students in our dataset as a function of the degree that they are taking:

```r
ggplot(data=ex1,aes(x=Degree,y=Score1)) +
geom_boxplot(fill=c("red","purple","green","blue")) +
xlab("\nDegree Subject") + ylab("Score on Test 1 (%)\n")
```



In this example, in the "aes()" part of the code, we specified that "Degree" should be on the x-axis, and "Score1" should be plotted on the y-axis. This means that we will be visualising the data for Score1 separately for each degree. We then use the "geom_boxplot()" function to specify that we want boxplots, and as in previous examples, we can specify the colour using the "fill" function (and we can have different colours for each bar, if we specify a list of colours). In the example above, for each degree, we can see a box with a horizontal line inside it. The horizontal line in the box represents the median score. The upper and lower edges of the box show the 1st and 3rd quartiles (i.e. the 25% and 75% quantiles), so the height of the box represents the interquartile range. The vertical lines above and below each box connect the quartiles with the maxima and mimima. If there are no extreme values in the data, they will stretch to the value of the greatest and smallest data points in the relevant sample. For example, for the joint degree, the highest score is 70%, and there are no extreme values, so the vertical line stretches to 70 on the y-axis. If there are extreme values in the data, then these extreme values are plotted as circles above or below the vertical lines. By default, extreme values are defined as those that are more than 1.5×IQR above the 3rd quartile, or more than 1.5×IQR below the 1st quartile. In the data visualised above, there are no extreme values. If there had been extreme values, then the vertical lines would stretch up or down to the relevant cut-off value, and the extreme values would be plotted as circles above or below the vertical line.