Eagle I.O's Course Take-aways: Psychometrics

This course is taught in MSU's I/O Psychology program

Course Objectives: Understand the fundamental concepts of psychological measurement & learn how to practically apply them through the R programming language.

Validity: The extent to which assessment inferences are considered appropriate.	**Reliability:** The stability of assessment scores across repeated testing occasions.
• **Face Validity** = The degree to which the instrument *appears* to measure what it purports to measure.	• **Split-Half Reliability** = The degree to which participants' scores on one half of the instrument correlate with those same participants' scores on the other half of the instrument (always accompanied by Spearman-Brown).
• **Construct Validity** = How well the instrument actually measures the construct it purports to measure.	• **Parallel Forms Reliability** = The degree to which a set of participants' scores on one form of the instrument correlate with the same participants' scores on an alternative form of the instrument.
• **Criterion Validity** = The degree to which the scores on the instrument correlate with an outcome of interest.	• **Test-Retest Reliability** = The degree to which participants' scores on the instrument correlate with their scores on the same instrument completed at a later time.
• **Content Validity** = The degree to which the instrument's items adequately sample from your construct's content domain.	• **Internal Consistency Reliability** = The degree to which the instrument's items that are intended to measure the same construct are responded to in a similar manner. This is usually denoted by Cronbach's Alpha.

Key Terms

- **Raw Score:** A test score that does not reflect relative performance (e.g., there is no acknowledgement or consideration given to how other respondents perform).
- **Z-score:** A standard score with a mean of zero and a standard deviation of 1. These scores can range from -4 to 4.
- **Standard Score:** A test score that is expressed as the number of standard deviations either above or below the mean, expressed in standard deviation units and dependent on a normative distribution (aka "norm").
- **Percentiles:** The percent of scores in a distribution that "lie below" a value. Percentile ranks range from 1st (meaning a respondent only scored higher than 1% of participants), to 99th (meaning a respondent scored higher than 99% of participants).

Eagle I.O
Student Led Consulting Group

Types of Bias

Test Bias:

When one group of respondents' true scores on a construct are misrepresented by the instrument, compared to those of another group. There are two types of test bias.

- **Construct Bias** = When a test has different meanings for different groups based on the construct that the test purports to measure.
- **Predictive Bias** = When a test predicts certain outcome criteria at differing levels of accuracy for different groups.

Response Bias:

When the responses are at least partially attributable to systematic tendencies of the respondent or response context rather than (or in addition to) the substantive construct.

- **Acquiescence** = Consistently endorsing or rejecting items without regarding the items' content.
- **Extremity** = Consistently overusing "extreme" response options, regardless of the respondent's true levels of the construct being measured.
- **Social desirability** = Responding in a way that presents the respondent as socially desirable, regardless of the respondent's true levels of the construct of interest.
- **Malingering** = Responding in a way that exaggerates a respondent's problem or disability.
- **Careless responding** = Responding to items randomly or without regard for the items' content.

More Key Terms

- **Standard Deviation:** The average distance of scores from the mean. It is the square root of the variance.
- **Variance:** A measure of the degree of spread in a data set computed as the average of squared deviations from the mean.
- **Covariance:** A measure of the degree to which two variables in a data set change together. It is an unstandardized measure, as it is computed from raw scores. The standardized version is known as...
- **Correlation:** The direction and magnitude of the association between two variables in a data set. It is a standardized measure, as it uses z-scores rather than raw scores, thus correlation statistics range from -1 to 1 and are usually denoted by Pearson's Correlation Coefficient.

	x	y
x	cor(x,x)	cor(x,y)
y	cor(y,x)	cor(y,y)

Correlation Matrix

	x	y
x	var(x)	cov(x,y)
y	cov(y,x)	var(y)

Covariance Matrix

Eagle I.O
Student Led Consulting Group

Contributors: Pasquale Tosto Paulina Wiedmann Ian Lee

Sources

Frey, B. (2018). Split-Half Reliability. *The SAGE Encyclopedia of Educational Research, Measurement, And Evaluation*. https://doi.org/10.4135/9781506326139.n653

Furr, R. Michael, and Verne R. Bacharach. *Psychometrics: An Introduction*. Second edition, SAGE, 2014.

Middleton, F. (2021a, July 16). *Types of reliability and how to measure them*. Scribbr. https://www.scribbr.com/methodology/types-of-reliability/

Middleton, F. (2021b, October 15). *The four types of validity*. Scribbr. https://www.scribbr.com/methodology/types-of-validity/

Stephanie Glen. "Inter-rater Reliability IRR: Definition, Calculation" From StatisticsHowTo.com: Elementary Statistics for the rest of us! https://www.statisticshowto.com/inter-rater-reliability/

Stephanie Glen. "Parallel Forms Reliability (Equivalent Forms)" From StatisticsHowTo.com: Elementary Statistics for the rest of us! https://www.statisticshowto.com/parallel-forms-reliability/

Psychometrics Takeaway R Codes

Pasquale Tosto, Paulina Wiedmann, & Ian Lee

Creating a Data Frame

In order to create a data frame, you must first create either your columns, or your rows. For the sake of this tutorial, we will create columns first, using the `concatenate` function like so:

```
col_1 <- c(1, 2, 3, 4, 5)
col_2 <- c(5, 4, 3, 2, 1)
col_3 <- c(3, 2, 1, 4, 5)
col_4 <- c(4, 5, 1, 2, 3)
```

Now that our four columns have been created, it's time to piece them together in a data frame using the `as.data.frame` function, as well as the `cbind` function, and we will title our data frame, "frame":

```
frame <- as.data.frame(cbind(col_1, col_2, col_3, col_4))
```

And the resulting data frame will look like this:

Table 1: Our very basic data frame!

col_1	col_2	col_3	col_4
1	5	3	4
2	4	2	5
3	3	1	1
4	2	4	2
5	1	5	3

Now that our data frame has been created, we can call individual columns in order to use them for statistical analyses. For example, if we wanted to look at the correlation between column 2 and column 4, we would use the `cor` function for correlation, and we would call the columns using a dollar sign, like so:

```
cor(frame$col_2, frame$col_4)
```

And the resulting output will look like this; a nicely sized correlation of 0.5!

```
## [1] 0.5
```

Let's say you have items that are negatively worded, that is, items that need to be reverse coded in your data frame. You can reverse code them by simply adding 1 to the number of numeric response options, and subtracting the name of the column within your frame from that sum. Let's try it for column 4 of your data frame:

```
frame$col_4 <- 6 - frame$col_4
```

Now when we take a look at our data frame, you'll notice that the values for column 4 have all been reverse scored.

Table 2: Look at column 4's values!

col_1	col_2	col_3	col_4
1	5	3	2
2	4	2	1
3	3	1	5
4	2	4	4
5	1	5	3

Alpha

In order to view important statistical qualities of a data frame, you use the `alpha` function within the `psych` package. You start by installing the `psych` package using the following code:

```
install.packages("psych")
```

Next, you load the package in from your library using the following code:

```
library(psych)
```

Finally, you use the `alpha` function and enclose the name of your data frame within parentheses. For the sake of this tutorial, we will use the `mtcars` data frame, columns 2 through 5, denoted with brackets. Table 3 shows item statistics, which are one of many types of vital statistics the `alpha` function provides.

```
alpha(mtcars[2:5])
```

Table 3: These are vital item statistics that the Alpha function provides

	n	raw.r	std.r	r.cor	r.drop	mean	sd
cyl	32	0.9237324	0.8810072	0.9092715	0.9222917	6.187500	1.7859216
disp	32	0.9731847	0.8585875	0.8694455	0.7944121	230.721875	123.9386938
hp	32	0.9104707	0.9416717	0.9179976	0.7931131	146.687500	68.5628685
drat	32	-0.6482843	-0.3719298	-0.6009034	-0.6499610	3.596563	0.5346787

Class Conversions

Sometimes you need to directly tell R how to classify your values. Most commonly, you will have to convert character values into numeric values, or vice versa, so that R knows they are numbers and not characters.

Let's use the following vector, "vecky", as an example, and convert a numeric vector into a character vector. We will start by creating our vector.

```
vecky <- c(1, 1, 3, 5, 8, 13, 21)
```

Now, we will convert `vecky` into a character vector using the `as.character()` function:

```
vecky <- as.character(vecky)
```

In order to ensure that the vector has been converted to character, we use the `class()` or `typeof()` function and enclose the name of the vector within parentheses:

```
class(vecky)
```

```
## [1] "character"
```

Now that we have confirmed that our vector is a character vector, we will use the `as.numeric()` function to convert `vecky` into a numeric vector:

```
vecky <- as.numeric(vecky)
```

Finally, we will use the `class()` function to confirm that our vector is indeed a numeric vector.

```
class(vecky)
```

```
## [1] "numeric"
```

Oh, also, if you want to clear your entire global environment, i.e. remove all objects (vectors, data frames, etc.) from your workspace, you can use the simple code below!

```
rm(list = ls())
```

Quick Views!

Let's say we have a very large data frame, for example, `mtcars`, and we only want to see a snippet of the beginning of the data. Here, we can use the `head` function, like so:

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Now, let's say we only want to see the last parts of the dataframe. Here, we can use the `tail` function for just the end:

```
tail(mtcars)
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

There is also a function you can use to compactly display the internal structure of a data frame or other object in `R`, and that function is `str` .

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Data Visualization

In order to better understand your data, it is often helpful to visualize it with a graphic representation! We can use the extremely handy `ggplot2` package for data visualization. As an example, let's create a basic scatterplot for two columns of the `iris` data set. We will start by installing the package onto our machines:

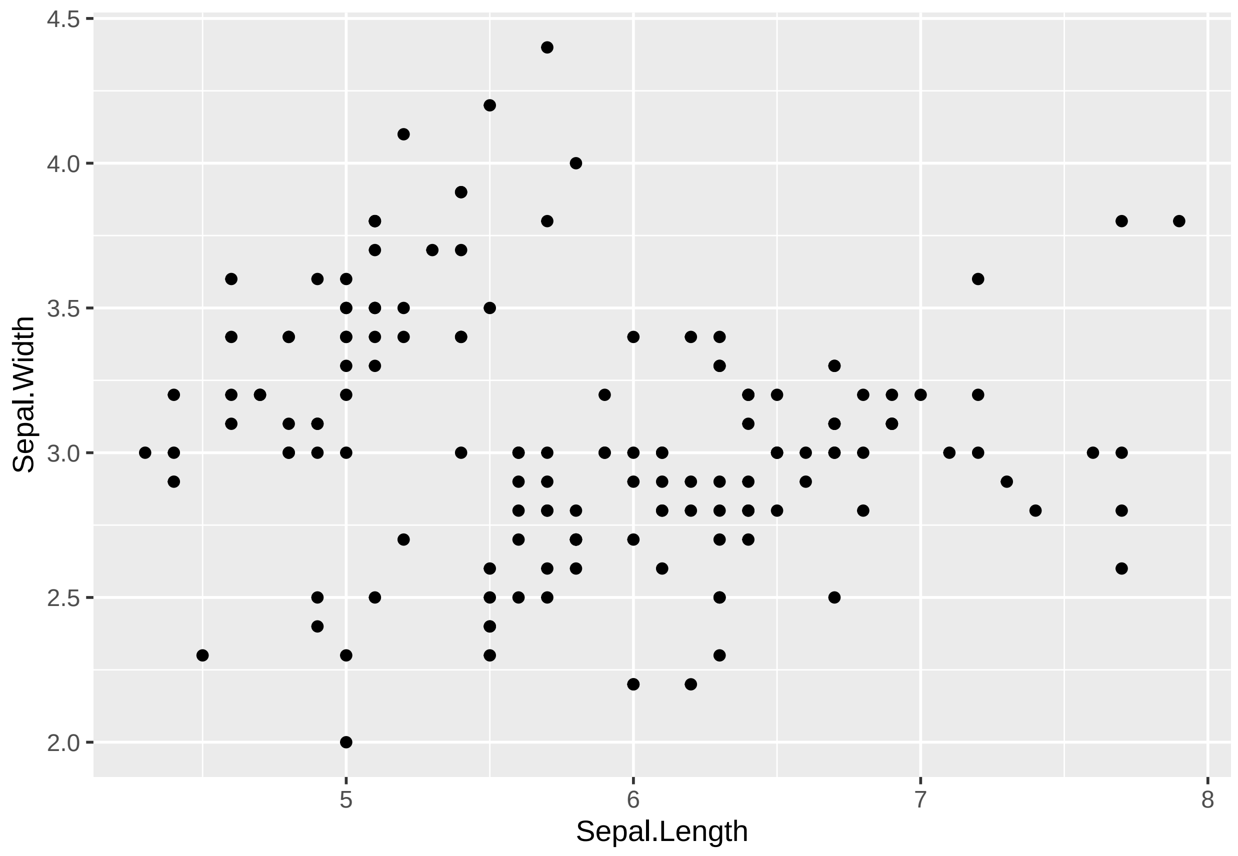
```
install.packages("ggplot2")
```

Next, you load the package in from your library using the following code:

```
library(ggplot2)
```

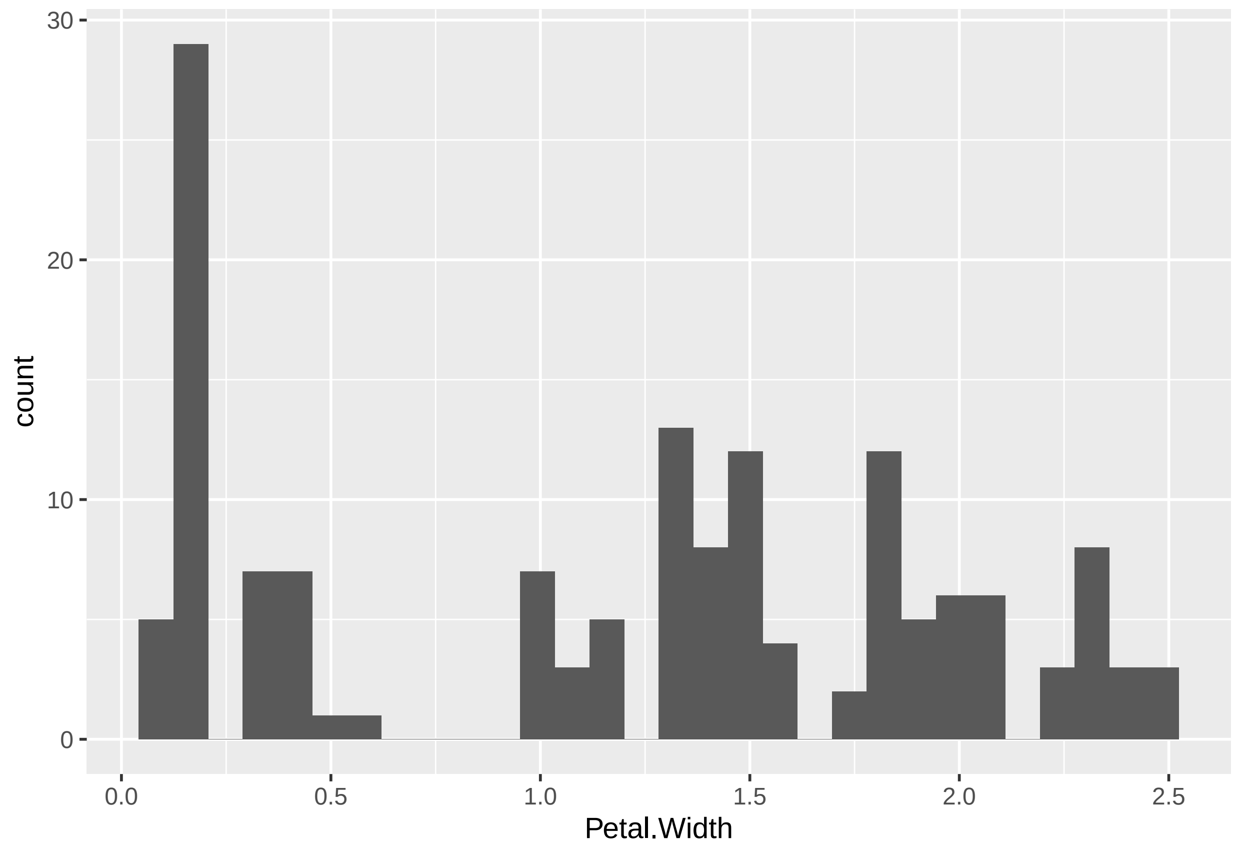
Next, we will use the following code to create a basic scatterplot of the `Sepal.Length` and `Sepal.Width` columns. Note that adding a `geom_point()` call is what we use to specify that our data will be graphed as a scatterplot.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point()
```



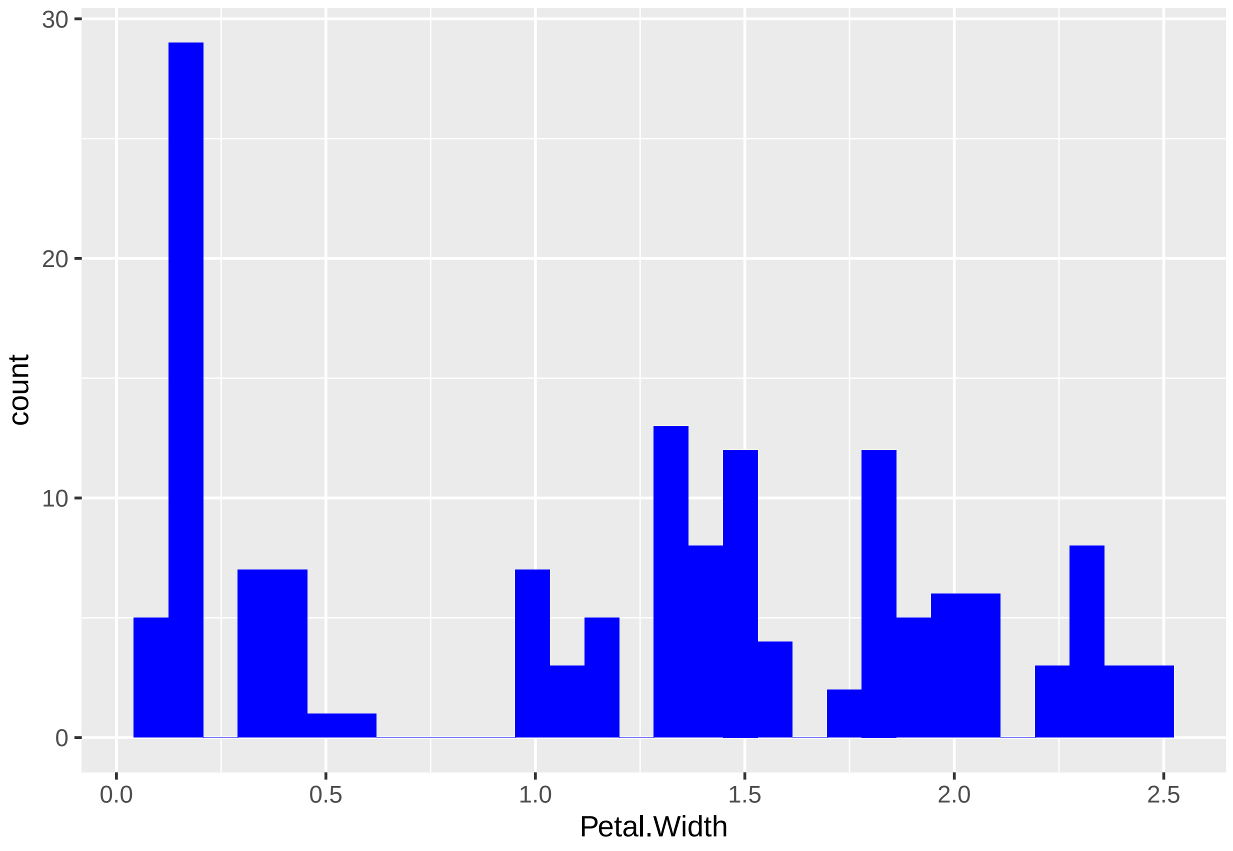
It should be noted that within the `aes` function inside the `ggplot` call, x denotes the column whose data you want graphed on the x-axis, and y denotes the column whose data you want graphed on the y-axis. You do not actually need to write "x" and "y", as the arguments are always in this order! Let's use the `Petal.Width` column as an example of this with a histogram rather than a scatterplot. Note that adding a `geom_histogram()` call is what we use to specify that our data will be graphed as a histogram. Additionally, histograms typically only take one column and are used to graph the frequency of each score.

```
ggplot(iris, aes(Petal.Width)) +
  geom_histogram()
```



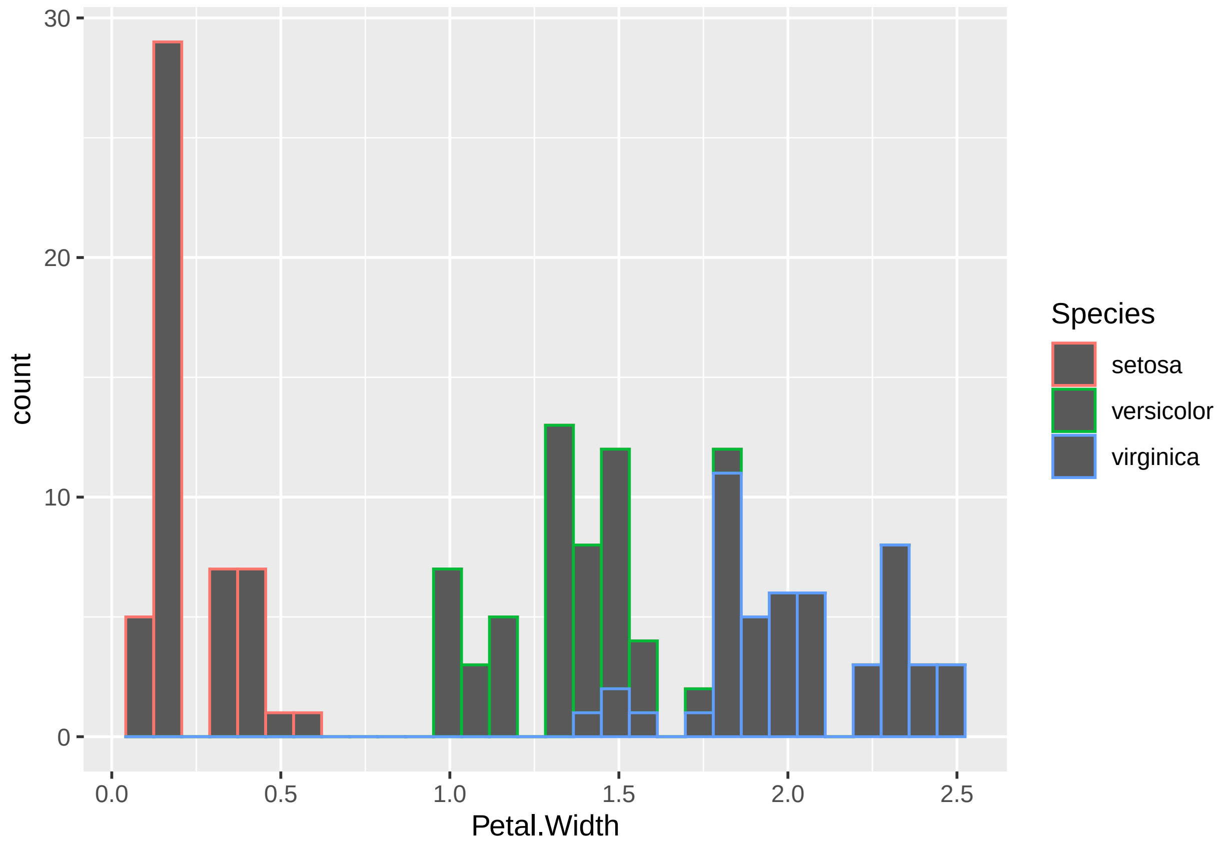
We will now demonstrate how to change the color of ggplot2 graphics, which is one of MANY different aesthetic changes you can add to your data graphics. In order to change the color of our histogram bars from the previous plot, we will use the fill function within the call of geom_histogram() to turn the bars blue:

```
ggplot(iris, aes(Petal.Width)) +
  geom_histogram(fill = "blue")
```



Finally, we will demonstrate how to plot your histogram with your data separated by a categorical variable, which in the case of the `iris` dataset, will be the `Species` column. To do this, we simply add the `color` argument within the call of `aes`, and set it equal to the categorical column of interest like so:

```r
ggplot(iris, aes(Petal.Width, color = Species)) +
  geom_histogram()
```



Z-scores

Because scale scores are typically simple means across a number of items, we can use the `rowMeans` function to create a simple, averaged standard score column, and we will use our `frame` dataframe as an example:

```
frame$averages <- rowMeans(frame)
```

If we look at our frame, we will see it has an additional column that includes averaged scores for all cases:

Table 4: Our frame now has an averages column

col_1	col_2	col_3	col_4	averages
1	5	3	2	2.75
2	4	2	1	2.25
3	3	1	5	3.00
4	2	4	4	3.50
5	1	5	3	3.50

In order to create our z-scores, we make a new column and run `scale` on the column we just created.

```
frame$zscores <- scale(frame$averages)
```

Now, when we look at our frame, we will see that the newest column has our z-scores!

Table 5: Our frame now has a column for z-scores

col_1	col_2	col_3	col_4	averages	zscores
1	5	3	2	2.75	-0.4714045
2	4	2	1	2.25	-1.4142136
3	3	1	5	3.00	0.0000000
4	2	4	4	3.50	0.9428090
5	1	5	3	3.50	0.9428090

Finally, we are going to create standard scale scores for a mean of 100 and a standard deviation of 15 using the `round` function on the z-score column we just created like so:

```
frame$standard <- round((15 * frame$zscores) + 100)
```

Now, when we look at our frame, we will see that the newest column has our standard scale scores!

Table 6: Our frame now has a column for standard scale scores

col_1	col_2	col_3	col_4	averages	zscores	standard
1	5	3	2	2.75	-0.4714045	93
2	4	2	1	2.25	-1.4142136	79
3	3	1	5	3.00	0.0000000	100
4	2	4	4	3.50	0.9428090	114
5	1	5	3	3.50	0.9428090	114

Dealing with NA Values

Finally, we will show you how to deal with NA, or not applicable values from your R objects. First, we will create a simple vector that has NA values already in it. Ordinarily we don't want such values in our data, but for the sake of this tutorial, we will create the following vector:

```
messed_up_vector <- c(NA, 3,4,5,6,7)
```

Uh-oh! That vector is all messed up! It has an NA value in it! What do we do?! In order to remove this value from our vector, we use the following code:

```
messed_up_vector <- na.omit(messed_up_vector)
```

Now, if we inspect our vector, we will find that it is no longer messed up! So, we are going to rename it after we inspect it:

```
head(messed_up_vector)
```

```
## [1] 3 4 5 6 7
```

```
not_messed_up_vector <- messed_up_vector
```

More frequently, however, we will find that we need to remove NA values from dataframes while creating scale score columns, as functions such as rowMeans will yield NA if there are NA values in the columns on which the function is operating. Let's construct the same dataframe that we made at the beginning of this tutorial, but replace each 1 with an NA .

```
col_1 <- c(NA, 2, 3, 4, 5)
col_2 <- c(5, 4, 3, 2, NA)
col_3 <- c(3, 2, NA, 4, 5)
col_4 <- c(4, 5, NA, 2, 3)

messed_up_frame <- as.data.frame(cbind(col_1, col_2, col_3, col_4))
```

If we try to create a scale score column using rowMeans, we will have NA values in our scale score column.

```
messed_up_frame$scale <- rowMeans(messed_up_frame)
```

Table 7: Our scale column has NA values in it

col_1	col_2	col_3	col_4	scale
NA	5	3	4	NA
2	4	2	5	3.25
3	3	NA	NA	NA
4	2	4	2	3.00
5	NA	5	3	NA

In order to get around this issue, we can simply include the argument, `na.rm = TRUE`, within our call of `rowMeans`, after inputting our dataframe like so:

```
messed_up_frame$scale <- rowMeans(messed_up_frame, na.rm = TRUE)
```

Now when we look at our dataframe, we will see that the scale score column is populated with numeric values because they were computed with `NA` values exluded from calculations!

Table 8: Our scale column looks nice now!

col_1	col_2	col_3	col_4	scale
NA	5	3	4	4.00
2	4	2	5	3.25
3	3	NA	NA	3.00
4	2	4	2	3.00
5	NA	5	3	4.33