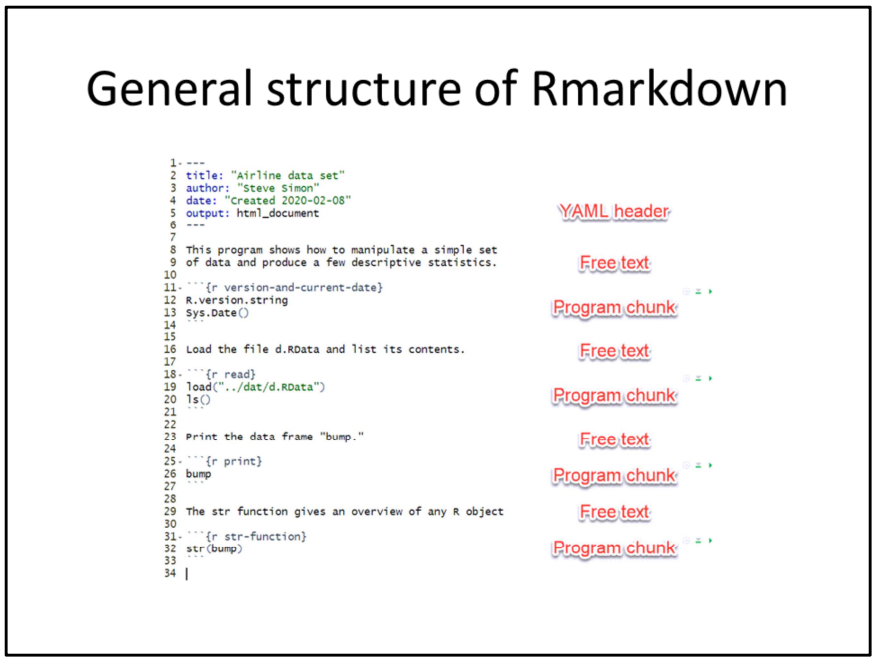
Module02: Data with mostly continuous variables

Steve Simon

Created 2020-02-08

General structure of Rmarkdown



Recall the general structure of an Rmarkdown file. It starts with a header, in a simple format known as YAML. Then you alternate between free text commentary and chunks of R programming statements. The chunks are surrounded by lines with three backticks and the opening set of three backticks is followed by the programming language (R) and an optional label for the chunk.

Documentation (1/3)

```
---
title: "Airline data set"
author: "Steve Simon"
date: "Created 2020-02-08"
output: html_document
---
```

You should use the beginning of the program to provide some brief documentation about the program.

The header in an Rmarkdown file, provides the name of the program, the author, and when the program was created. I always modify the date in the header so that it includes the word “Created” so people don’t get this confused with the date that the program was last modified.

Documentation (2/3)

```
This program shows how to manipulate a simple set
of data and produce a few descriptive statistics.
```

Next, a section of free text briefly describes the purpose of the program.

Documentation (3/3)

```
```{r version-and-current-date}
R.version.string
Sys.Date()
```
[1] "R version 3.6.1 (2019-07-05)"
[1] "2020-02-14"
```

You should always print out the version of R that you are using and the date that the program was run. This is optional, but is one of the documentation practices that you should get in the habit of using.

Reading existing data

```
load("../dat/module02-datasets.RData")
ls()
## [1] "bump"  "fd"    "sleep"
```

After loading data, use the ls function to review what information was loaded.

From airline-data-dictionary.txt

This file has 12 rows and 5 variables. The variables measures the number of times passengers were “bumped” from an airline flight. Bumping occurs when an airline overbooks reservations for a flight. Normally, the airline will ask for volunteers who will agree to take a later flight in exchange for cash or travel vouchers. But if no one volunteers, airlines will involuntarily bump passengers from the flight.

The first dataset that we will use in this video is “bump.”

Here is the first paragraph from the data dictionary.

Displaying data

```
bump
##       airline b2017 b2016 r2017 r2016
## 1       DELTA   679   912  0.07  0.09
## 2      VIRGIN   165    77  0.27  0.13
## 3     JETBLUE  1475  2140  0.54  0.82
## 4      UNITED  2067  2874  0.30  0.45
## 5    HAWAIIAN    92    30  0.11  0.04
## 6  EXPRESSJET   785  2541  0.67  1.58
## 7     SKYWEST   917  2177  0.37  0.96
## 8    AMERICAN  4517  6598  0.46  0.66
## 9      ALASKA   658   734  0.35  0.41
## 10  SOUTHWEST  6678 11907  0.58  1.06
## 11   FRONTIER   540   688  0.45  0.63
## 12     SPIRIT  1502  1418  0.88  0.93
```

R has a print function, but if you just list the name of a data frame, it will, by default, display itself.

Showing the structure

```
str(bump)
## 'data.frame':    12 obs. of  5 variables:
##  $ airline: chr  "DELTA" "VIRGIN" "JETBLUE"
"UNITED" ...
##  $ b2017  : int  679 165 1475 2067 92 785 917
4517 658 6678 ...
##  $ b2016  : int  912 77 2140 2874 30 2541 2177
6598 734 11907 ...
##  $ r2017  : num  0.07 0.27 0.54 0.3 0.11 0.67
0.37 0.46 0.35 0.58 ...
##  $ r2016  : num  0.09 0.13 0.82 0.45 0.04 1.58
0.96 0.66 0.41 1.06 ...
```

The “str” function provides a description of “bump.” It is a data frame. A data frame is a rectangular grid of data. The data values in a single column of data has to be consistent: either a set of numbers, a set of dates, a set of character strings, etc. But the type of data in one column can differ from the next column.

Data frames are the workhorse of R. There are other ways of storing data: lists, arrays, vectors. But most of the data you will analyze in R will be stored in a data frame.

Displaying a single value

```
bump[1, 1]
## [1] "DELTA"
bump[12, 5]
## [1] 0.93
```

You can display individual data points by specifying the row and column inside square brackets.

Displaying a single row

```
bump[1, ]
##   airline b2017 b2016 r2017 r2016
## 1   DELTA   679   912  0.07  0.09
bump[12, ]
##    airline b2017 b2016 r2017 r2016
## 12  SPIRIT  1502  1418  0.88  0.93
```

You can display an entire row by specifying it and leaving the column entry blank.

Displaying a single column (1/2)

```
bump[ , 1]
##  [1] "DELTA"      "VIRGIN"
##  [3] "JETBLUE"    "UNITED"
##  [5] "HAWAIIAN"   "EXPRESSJET"
##  [7] "SKYWEST"    "AMERICAN"
##  [9] "ALASKA"     "SOUTHWEST"
## [11] "FRONTIER"   "SPIRIT"
```

You can display an entire column similarly.

Displaying a single column (2/2)

```
bump[ , 5]
##  [1] 0.09 0.13 0.82 0.45 0.04 1.58 0.96
##  [8] 0.66 0.41 1.06 0.63 0.93
```

There is an important difference in the display here. When you display a single column, it will fit more than one data point in the same row to save space. This may seem trivial and but once in a while it is not. The distinction may come up later in this class.

Using column names

```
bump$airline
##  [1] "DELTA"      "VIRGIN"
##  [3] "JETBLUE"    "UNITED"
##  [5] "HAWAIIAN"   "EXPRESSJET"
##  [7] "SKYWEST"    "AMERICAN"
##  [9] "ALASKA"     "SOUTHWEST"
## [11] "FRONTIER"   "SPIRIT"
```

Each column has a name and you can use that name instead of the column number.

Displaying multiple rows or columns

```
bump[1:5, ]
##    airline b2017 b2016 r2017 r2016
## 1    DELTA   679   912  0.07  0.09
## 2   VIRGIN   165    77  0.27  0.13
## 3  JETBLUE  1475  2140  0.54  0.82
## 4   UNITED  2067  2874  0.30  0.45
## 5 HAWAIIAN    92    30  0.11  0.04
```

The code “1:5” means 1, 2, 3, 4, 5.

Displaying multiple rows or columns

```
bump[ , c(2,4)]
##    b2017 r2017
## 1    679  0.07
## 2    165  0.27
## 3   1475  0.54
## 4   2067  0.30
## 5     92  0.11
## 6    785  0.67
## 7    917  0.37
## 8   4517  0.46
## 9    658  0.35
## 10  6678  0.58
## 11   540  0.45
## 12  1502  0.88
```

The “c” function allows you to select values that are not in a perfect sequence.

Skipping a row or column

```
bump[ , -1]
##    b2017 b2016 r2017 r2016
## 1    679   912  0.07  0.09
## 2    165    77  0.27  0.13
## 3   1475  2140  0.54  0.82
## 4   2067  2874  0.30  0.45
## 5     92    30  0.11  0.04
## 6    785  2541  0.67  1.58
## 7    917  2177  0.37  0.96
## 8   4517  6598  0.46  0.66
## 9    658   734  0.35  0.41
## 10  6678 11907  0.58  1.06
## 11   540   688  0.45  0.63
## 12  1502  1418  0.88  0.93
```

A negative number means that you want to exclude that row or column. In this example, the -1 allows you to print everything except the airline name.

The head function

```
head(bump)
##      airline b2017 b2016 r2017 r2016
## 1      DELTA   679   912  0.07  0.09
## 2     VIRGIN   165    77  0.27  0.13
## 3    JETBLUE  1475  2140  0.54  0.82
## 4     UNITED  2067  2874  0.30  0.45
## 5   HAWAIIAN    92    30  0.11  0.04
## 6 EXPRESSJET   785  2541  0.67  1.58
```

The “head” function displays the first five rows of data.

The tail function

```
tail(bump)
##      airline b2017 b2016 r2017 r2016
## 7    SKYWEST   917  2177  0.37  0.96
## 8   AMERICAN  4517  6598  0.46  0.66
## 9     ALASKA   658   734  0.35  0.41
## 10 SOUTHWEST  6678 11907  0.58  1.06
## 11  FRONTIER   540   688  0.45  0.63
## 12    SPIRIT  1502  1418  0.88  0.93
```

The “tail” function displays the last five rows of data. If there is a problem with a data set, it often shows up more obviously at the bottom of a data frame than the top.

Break #1

- What have you learned
  - Rmarkdown structure
  - Printing a data frame
  - Printing pieces of a data frame
- What is coming next
  - Definitions of categorical and continuous data
  - The “fat” data set
  - Outliers and missing values

Some definitions

- Categorical = small number of possible values
- Examples
  - Sex (Male or Female),
  - Race/ethnicity (Caucasian, African American, Hispanic, etc.),
  - Cancer stage (I, II, III, or IV),
  - Birth delivery type (Vaginal, C-section).

A **categorical variable** is a variable that can only take on a small number of values. Each value is usually associated with a particular category.

Some definitions

- Continuous variable = large number of possible values
- Examples of continuous variables are
  - Birth weight in grams,
  - Gestational age,
  - Fasting LDL level.

A **continuous variable** is a variable that can take on a large number of possible values, potentially any value in some interval.

There are some variables that are on the boundary between categorical and continuous, but it is not worth quibbling about here.

The point to remember is that the types of graphs that you use and the types of statistics that you compute are dependent on many things, but first and foremost on whether the variables are categorical, continuous, or a mixture.

Today, you will see examples involving mostly continuous variables.

From fat-data-dictionary.txt

This file has 252 rows and 19 variables. The two key variables are measures of body fat, a rather complex assessment that requires weighing a person underwater. At the same time, researchers collected simpler measures of circumference at the neck, chest, abdomen, hip, thigh, knee, ankle, biceps, forearm, and wrist to see if the complex measure of body fat could be approximated by some combination of circumference measures.

Here is the first paragraph from the data dictionary.

First few lines of data

```
head(fd, 2)
##   case fat_b fat_s density age     wt
## 1    1  12.6  12.3  1.0708  23 154.25
## 2    2   6.9   6.1  1.0853  22 173.25
##      ht  bmi   ffw neck chest abdomen
## 1 67.75 23.7 134.9 36.2  93.1    85.2
## 2 72.25 23.4 161.3 38.5  93.6    83.0
##    hip thigh knee ankle biceps forearm
## 1 94.5  59.0 37.3  21.9   32.0    27.4
## 2 98.7  58.7 37.3  23.4   30.5    28.9
##   wrist
## 1  17.1
## 2  18.2
```

Here are the first few lines of the data set.

Some simple rules for data frames

- Rectangular grid
- Different types across columns
- Single type within a column
- Alternative ways to store data
  - Vector
  - Matrix
  - Array
  - List

R has many of the features of an object-oriented language, but it is not a true object-oriented programming language. There are a variety of objects in R like vectors, lists, matrices, and arrays, that are useful for storing, manipulating, and analyzing research data. We will spend most of this class using a particular object, the data frame.

The object, fd, that you just created with the read.table function is a data frame. Data frames are rectangular grids of data. Each column in the data frame has the same length. A data frame can store data of various types (numeric, character, and dates are the most common types of data). The data within a column has to have the same type, but the different columns can have different data types.

There are times when the rectangular grid of a data frame is too restrictive for your data, and R has other ways of storing this data (most notably, lists), but you will find that for most data analyses, a data frame will work just fine.

The head function shows the first few rows of the data set and the tail function shows the last few rows of the data set.

Always get in the habit of checking out the very bottom of your data frame. It's a

common location for glitches.

Variable names

- Short but descriptive
- Mix of letters and numbers
  - Must start with a letter
  - Avoid most symbols
- No blanks
  - CamelCase
  - dot.delimited.names
  - underscore_delimited_names

This data set did not have a header, a line at the very top of the file that lists variable names. R uses the default names V1, V2, etc. As a general rule, you should use brief (but descriptive) names for every variable in your data set. The names should be around 8 characters long. Longer variable names make your typing tedious and much shorter variable names makes your code terse and cryptic.

You should avoid special symbols in your variable names especially symbols that are likely to cause confusion (the dash symbol, for example, which is also the symbol for subtraction). You should also avoid blanks. These rules are pretty much universal across most statistical software packages. If you violate these rules, you will find out that, at a minimum, you will always have to surround your variable name by quotes to avoid problems.

There are times when you'd like to have a blank in your variable name and you can use two special symbols that you can use in R (and most other statistical pacakges), the underscore symbol (above the minus key on most keyboards) and the dot (period). These symbols create some artificial spacing that mimics the blanks. Another approach is "CamelCase" which is the mixture of upper and lower case within a variable name with each uppercase designating the beginning of a new

“word”.

Unusual data value

```
summary(fd$ht)
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##   29.50   68.25   70.00   70.15   72.25
##    Max.
##   77.75
```

There is an unusual data value, which you might not notice right away, but one of the heights is 29.5 inches. We'll talk in more detail about the summary function later, but right now I wanted to show you function because if you have an outlier in your data, you are most likely to discover it by using the summary function.

A height this small is not totally out of the realm of possibility. See, for example,

–> http://en.wikipedia.org/wiki/List_of_shortest_people

You can use the which function to identify the row with this unusual value for further investigation. Note the use of the double equals sign and how you display a single row of a data frame.

Which function

```
short_row <- which(fd$ht==29.5)
short_row
## [1] 42
fd[short_row, ]
##    case fat_b fat_s density age  wt
## 42   42  31.7  32.9   1.025  44 205
##      ht  bmi   ffw neck chest abdomen
## 42 29.5 29.9 140.1 36.6   106   104.3
##      hip thigh knee ankle biceps
## 42 115.5  70.6 42.5  23.7   33.6
##    forearm wrist
## 42    28.7  17.4
```

The other values look quite normal. You have to make a careful choice here. One possibility is to do nothing. If you leave the abnormal height in your data set, it may distort some of your graphs and skew some of your statistics. Still, it is often BETTER than some of the alternatives.

Remove the outlier

```
fd1 <- fd[-short_row, ]
summary(fd1$ht)
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##   64.00   68.25   70.00   70.31   72.25
##    Max.
##   77.75
```

A second choice is to remove the entire row from the data frame. The - means everything EXCEPT that row.

Set to missing

```
fd2 <- fd
fd2[short_row,"ht"] <- NA
summary(fd2$ht)
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##   64.00   68.25   70.00   70.31   72.25
##    Max.    NA's
##   77.75       1
```

A third possibility is to designate the abnormal value as missing. In R, a missing value is represented by NA.

Notice that the summary function for the ht variable notes that one of the values is missing. You should watch these missing values obsessively. This can get a bit tricky.

There is no one method that is preferred in this setting. If you encounter an unusual value, you should discuss it with your research team, investigate the original data sources, if possible, and review any procedures for handling unusual data values that might be specified in your research protocol.

Your data set may arrive with missing values in it already. Data might be designated as missing for a variety of reasons (lab result lost, value below the limit of detection, patient refused to answer this question) and how you handle missing values is way beyond the scope of this class. Just remember to tread cautiously around missing values as they are a minefield.

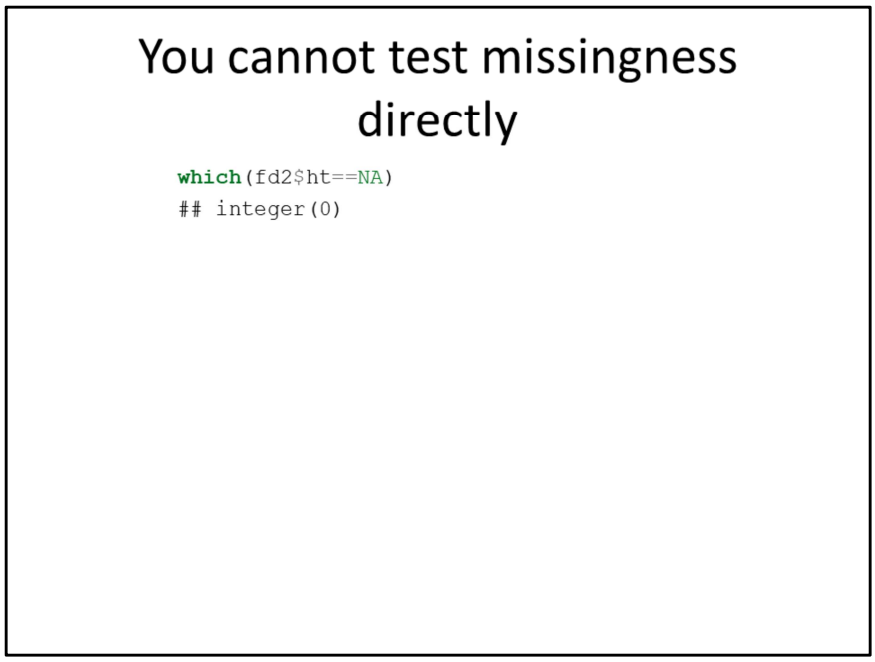
Notice that I store the revised data sets with the row removed and with the 29.5 replaced by a missing value in different data frames. This is good programming

practice. If you ever have to make a destructive change to your data set (a change that wipes out one or more values or a change that is difficult to undo), it is good form to store the new results in a fresh spot. That way, if you get cold feet, you can easily backtrack.

We'll use the data set with the 29.5 changed to a missing value for all of the remaining analyses of this data set.

Break #2

- What have you learned
  - Definitions of categorical and continuous data
  - The “fat” data set
  - Outliers and missing values
- What is coming next
  - Tracking missing values
  - Histograms



Logic involving missing values is tricky. If you checking for equality and one of the things in the comparison is missing, then the result is neither TRUE, not FALSE, but rather missing.

Fair enough, but R takes it a bit further, and if both sides when you are checking for equality are missing, then they might both be 5 is they weren't missing or maybe one might be 5 and the other one 10. So it might be TRUE or it might be FALSE, so we're better off calling the logical result as missing.

This is called a three valued logic system and it has advantages and disadvantages. I won't get into any technical details, except to say that you should never make assumptions. Check what you do when you are working with missing values to make sure that the three valued logic system doesn't produce results that you didn't expect.

Use is.na to test missingness

```
which(is.na(fd2$ht))
## [1] 42
mean(fd2$ht)
## [1] NA
sd(fd2$ht)
## [1] NA
```

The short term solution to the above problem is to use a special function, is.na.

The summary function will trap and remove missing values, but most other functions in R will, by default, report a result as missing if any values going into that function are missing. The philosophy in R, I suppose, is that you need to actively select an approach for handling missing values rather than relying on a lazy default.

R is also erring on the side of caution most of the time. You may not be aware that there are missing values lurking in your data, and R is going to go out of its way to remind you, unless you tell it otherwise.

This is different from SAS and SPSS, where the default options choose perfectly reasonable approaches, but approaches that don't raise concern about missingness to the degree that R does.

Read the help file for these functions (enter ?mean or ?sd at the command prompt).

Look carefully and note that the na.rm option allows you to compute the statistic after missing values are removed.

na.rm

```
mean(fd2$ht,na.rm=TRUE)
## [1] 70.31076
sd(fd2$ht,na.rm=TRUE)
## [1] 2.614296
```

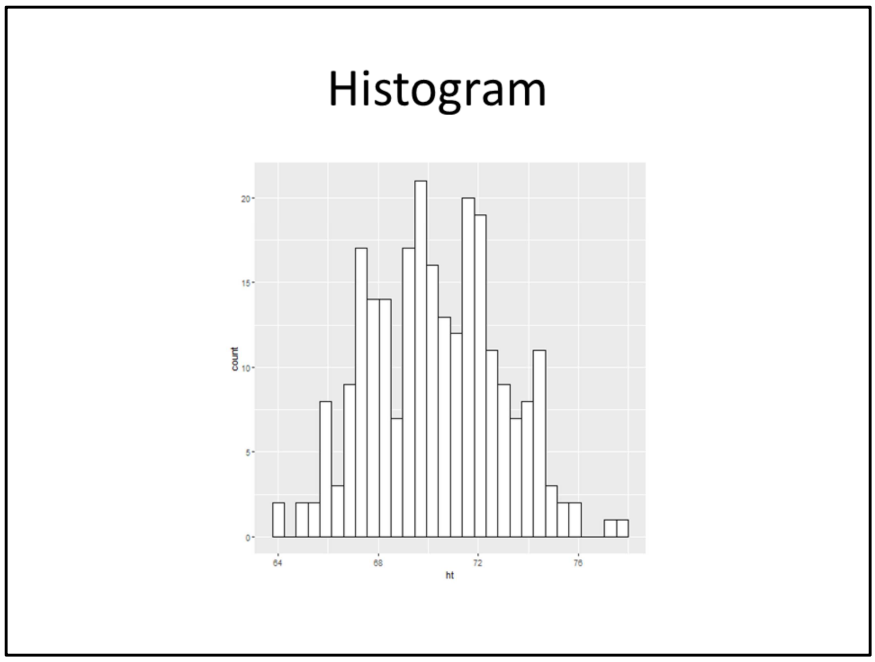
For univariate functions, there are only two realistic ways to handle missing values, but for bivariate and multivariate function, there are a multitude of approaches, such as pairwise deletion, listwise deletion, last observation carried forward, single imputation, and multiple imputation. There is a lot of controversy over various methods for handling missing values.

Histogram

```
library(ggplot2)
histogram_default <- ggplot(fd2, aes(x=ht)) +
  geom_histogram(fill="white", color="black")
png("../images/histogram01.png")
histogram_default
## `stat_bin()` using `bins = 30`. Pick
## better value with `binwidth`.
## Warning: Removed 1 rows containing non-finite
## values (stat_bin).
dev.off()
## png
##   2
```

A histogram is useful for displaying a continuous variable graphically.

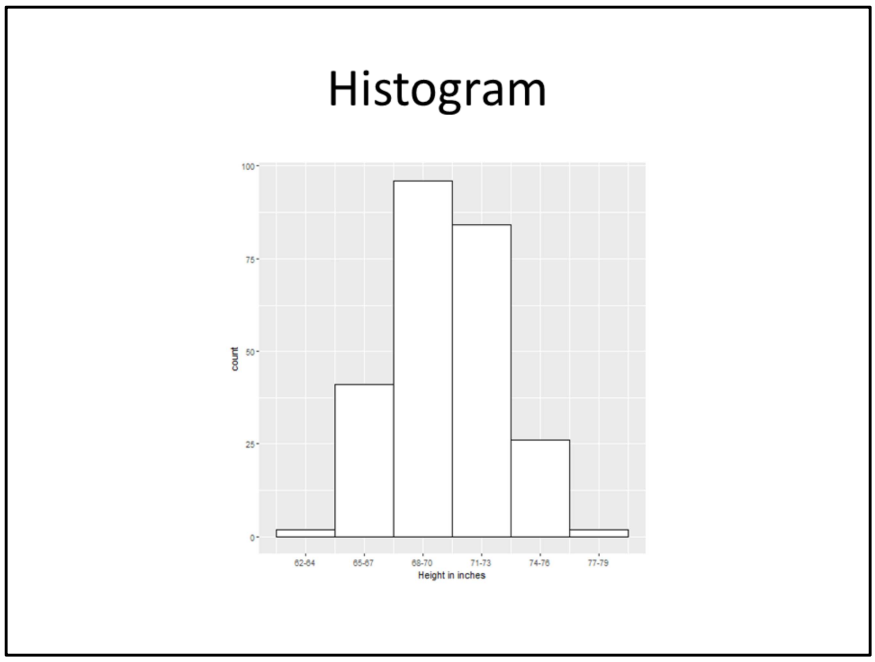
Always start with the default option, but don't settle for it.



Here is what the graph looks like.

```
histogram_enhanced <- ggplot(fd2, aes(x=ht)) +
  geom_histogram(
    fill="white", color="black",
    binwidth=3, na.rm=TRUE) +
  xlab("Height in inches") +
  scale_x_continuous(
    breaks=c(63, 66, 69, 72, 75, 78),
    labels=c("62-64", "65-67", "68-70",
             "71-73", "74-76", "77-79"))
png("../images/histogram02.png")
histogram_enhanced
dev.off()
## png
##   2
```

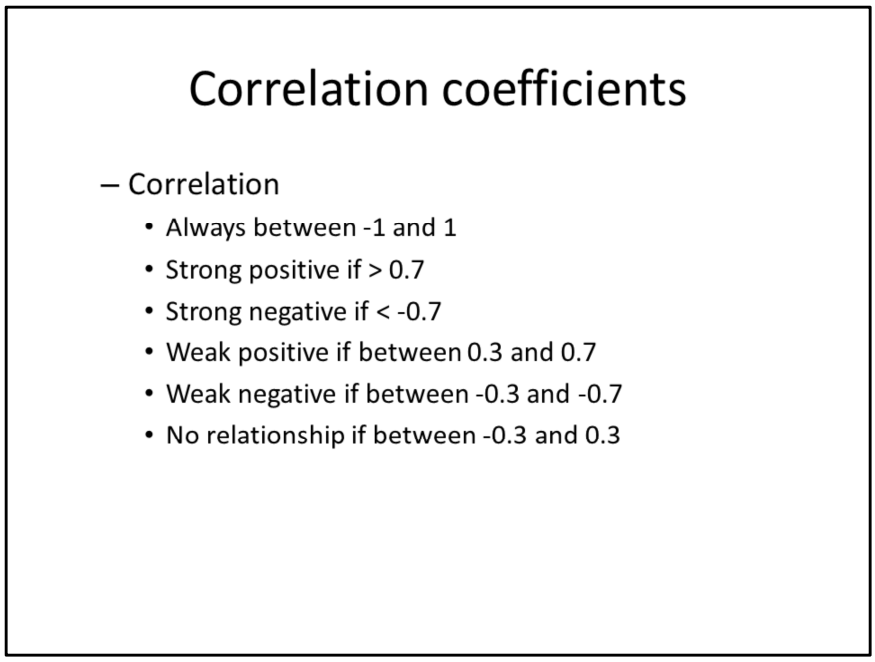
I like to have the border color of the bars and the fill color of the bars to be different. It took a bit of trial and error to get the binwidth, breaks, and labels to work properly.



Here is what the graph looks like.

Break #3

- What have you learned
  - Tracking missing values
  - Histograms
- What is coming next
  - Correlations
  - Scatterplots



The correlation coefficient is a single number between -1 and +1 that quantifies the strength and direction of a relationship between two continuous variables. As a rough rule of thumb, a correlation larger than +0.7 indicates a strong positive association and a correlation smaller than -0.7 indicates a strong negative association. A correlation between +0.3 and +0.7 (-0.3 and -0.7) indicates a weak positive (negative) association. A correlation between -0.3 and +0.3 indicates little or no association.

Don't take these rules too literally. You're not trying to make definitive statements about your data set. You are just trying to get comfortable with some general patterns that occur in your data set. A complex and definitive statistical analysis will almost certainly not agree with at least some of the preliminary correlations noted here.

Correlation coefficient

```
cor(fd2$fat_b,fd2$age)
## [1] 0.2891735
```

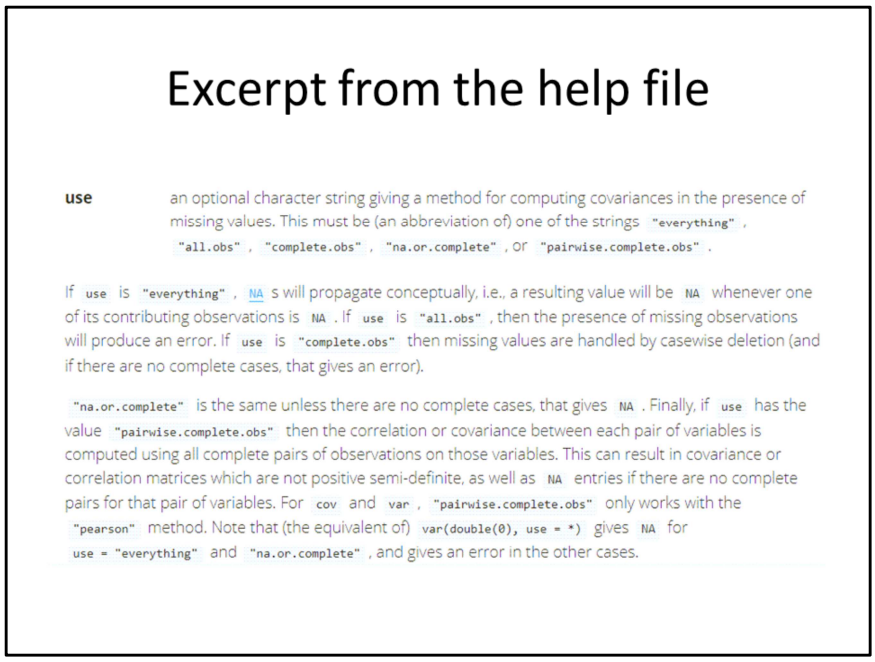
You can get a matrix of correlations for every possible pair of variables. This command becomes a bit more complicated if there are some categorical variables in your data set, as you need to exclude these prior to calculating the correlation matrix. Since you are just trying to get a general feel for your data, a bit of rounding will help you. Also, you should remove the case number before calculating the correlation matrix.

Correlation matrix

```
correlation_matrix <- cor(fd2[ ,-1])
dim(correlation_matrix)
## [1] 18 18
correlation_matrix[1:8, 1:3]
##                fat_b       fat_s
## fat_b    1.00000000  0.99974434
## fat_s    0.99974434  1.00000000
## density -0.98808673 -0.98778240
## age      0.28917352  0.29145844
## wt       0.61315611  0.61241400
## ht               NA          NA
## bmi      0.72799418  0.72748388
## ffw      0.02013209  0.01937491
##             density
## fat_b   -0.98808673
## fat_s   -0.98778240
```

Notice that the -1 in fd2 omits the first column, which is the case number, a variable that should not be incorporated in any statistical analyses.

I don't want to display the full correlation matrix because it would go off the margins of the slide. But looking at just the first eigth rows and three columns, you can see something interesting. R does not know how to compute a correlation coefficient when there are missing values.



This is two excerpts from the help file. The documentation is quite confusing, but that is because the options here are quite confusing. The two interesting options are casewise deletion (complete.obs) and pairwise deletion (pairwise.complete.obs). For the former, the row with missing values is tossed out for every correlation calculation, so that each and every correlation uses 251 rows of data rather than 252. For the latter, most correlations use all 252 rows of data, but any correlation involving height uses only 251 rows of data.

Which choice is best is beyond the scope of this class.

Correlation with pairwise deletion

```
correlation_matrix <- cor(fd2[ ,-1],
use="pairwise")
dim(correlation_matrix)
## [1] 18 18
correlation_matrix[1:8, 1:3]
##                fat_b       fat_s
## fat_b    1.00000000  0.99974434
## fat_s    0.99974434  1.00000000
## density -0.98808673 -0.98778240
## age      0.28917352  0.29145844
## wt       0.61315611  0.61241400
## ht      -0.02261395 -0.02338427
## bmi      0.72799418  0.72748388
## ffw      0.02013209  0.01937491
##              density
## fat_b   -0.98808673
```

Here is what the correlation matrix looks like with pairwise deletion. Notice that you can abbreviate “pairwise-complete-obs” to just “pairwise”. Actually, any abbreviation that is not ambiguous would work, so you could say “pair” or even just “p” here since none of the other choices for the use option begins with the letter “p.”

Rounding a correlation matrix

```
round(100*correlation_matrix[9:18, 1:2])
##         fat_b fat_s
## neck       49    49
## chest      70    70
## abdomen    81    81
## hip        63    63
## thigh      56    56
## knee       51    51
## ankle      27    27
## biceps     49    49
## forearm    36    36
## wrist      35    35
```

Rounding the correlations makes them easier to read. Here is a different part of the correlation matrix, showing the circumference measurements versus the fat measurements. The interesting pattern is that the strongest correlations are measurements near the middle (abdomen, chest, hip) and the correlation goes down as you move to the extremities. The lowest correlations are at the forearm, wrist, and ankle.

Another interesting set of correlations

```
leg.measures <- c("hip","thigh","knee","ankle")
arm.measures <- c("biceps","forearm","wrist")
round(100*cor(fd2[,leg.measures],fd2[,arm.measure
s]))
##       biceps forearm wrist
## hip       74      55    63
## thigh     76      57    56
## knee      68      56    66
## ankle     48      42    57
```

Here is another interesting set of correlations.

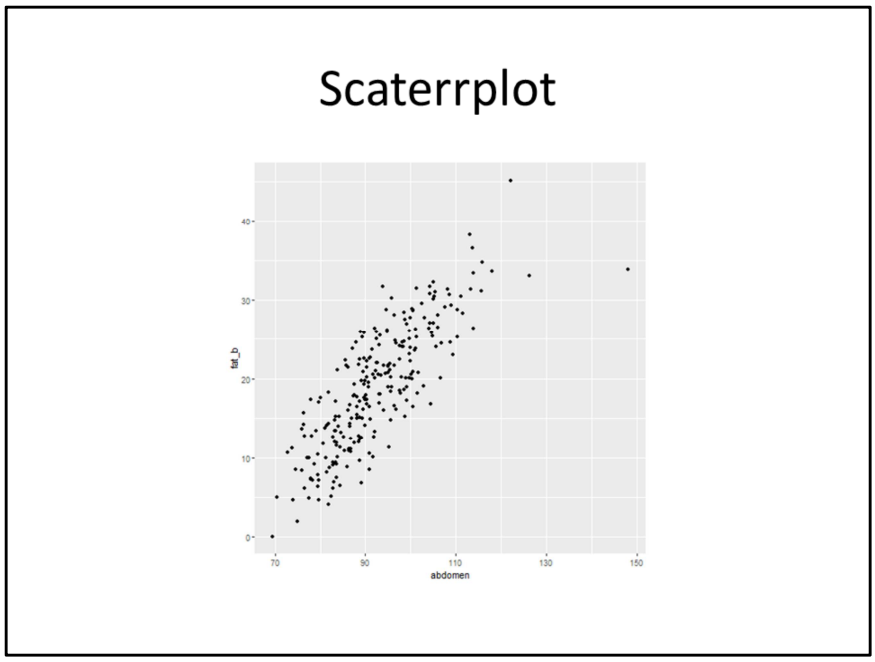
We created a few new objects, so you might want to save things here before proceeding to the next section.

The best graphical way to display a relationship between two continuous variables is a scatterplot.

Scatterplot

```
scatterplot_default <- ggplot(fd2,
    aes(x=abdomen, y=fat_b)) +
  geom_point()
png(filename="../images/scatterplot01.png")
scatterplot_default
dev.off()
## png
##   2
```

A simple and commonly used graphical approach to showing a relationship between two continuous variables is a scatterplot.



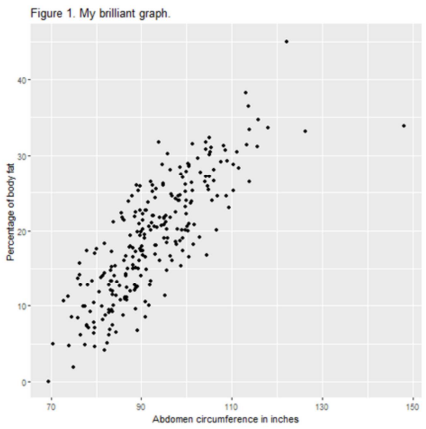
There are lots of options available to customize your graph. Here are just a few. The xlab and ylab arguments in the plot function control what is displayed on the horizontal (x) and vertical (y) axes. The pch argument control what is used as the plotting character.

Change from the defaults

```
scatterplot_enhanced <- scatterplot_default +
  xlab("Abdomen circumference in inches") +
  ylab("Percentage of body fat") +
  ggtitle("Figure 1. My brilliant graph.")
png(file="../images/scatterplot02.png")
scatterplot_enhanced
dev.off()
## png
##   2
```

Here are some adaptations, including better labels.

Scatterplot02



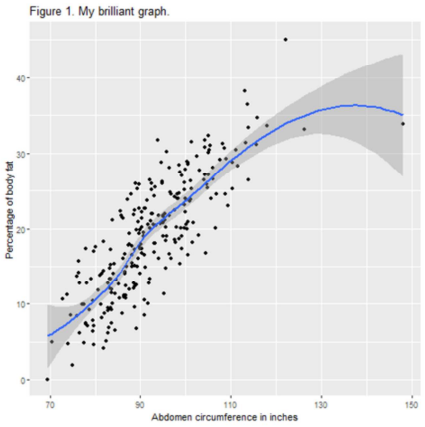
Here is what the graph looks like.

Adding trend lines

```
scatterplot_smoothed <- scatterplot_enhanced +
  geom_smooth()
png("../images/scatterplot03.png")
scatterplot_smoothed
## `geom_smooth()` using method = 'loess' and
formula 'y ~ x'
dev.off()
## png
##   2
```

Here's the code to produce a smooth curve on top of the data points.

Adding trend lines



Here is what the graph looks like.

Break #4

- What have you learned
  - Correlations
  - Scatterplots
- What is coming next
  - Description of sleep data

New slide

This file has 62 rows and 11 variables. The study wanted to see what factors affected sleep patterns (dreaming, non-dreaming, and total sleep) in mammals. The independent variables include Likert scale measurements (1-5) for predation, exposure, and danger. Other variables relate to the weight, brain size, lifespan and gestational period.

There is another data set on sleep in mammals.

There's a technicality here in that the last few variables are categorical, but for the purposes of this class, you can treat them as if they were continuous.

A quick peek at the data

```
##                  Species   BodyWt
## 1        Africanelephant 6654.000
## 2 Africangiantpouchedrat    1.000
## 3              ArcticFox    3.385
##   BrainWt NonDreaming Dreaming
## 1  5712.0          NA       NA
## 2     6.6         6.3        2
## 3    44.5          NA       NA
##   TotalSleep LifeSpan Gestation
## 1        3.3     38.6       645
## 2        8.3      4.5        42
## 3       12.5     14.0        60
##   Predation Exposure Danger
## 1         3        5      3
## 2         3        1      3
## 3         1        1      1
```

Here are the first two rows of data.