Chapter 1 R Basics

This chapter covers the basics: installing and using packages and loading data.

Most of the recipes in this book require the ggplot2, dplyr, and gcookbook packages to be installed on your computer. (The gcookbook package contains data sets used in some of the examples, but is not necessary for doing your real work.) If you want to get started quickly, run:

**install.packages**("tidyverse")**install.packages**("gcookbook")

Then, in each R session, before running the examples in this book, you can load them with:

**library**(tidyverse)**library**(gcookbook)

Running library(tidyverse) will load ggplot2, dplyr, and a number of other packages. If you want to keep your R session more streamlined and load only the packages that are strictly needed, you can load ggplot2 and dplyr packages individually:

**library**(ggplot2)**library**(dplyr)**library**(gcookbook)

**Note**

If you want a deeper understanding of how ggplot2 works, see Appendix [A](https://r-graphics.org/CHAPTER-GGPLOT2.html#CHAPTER-GGPLOT2), which explains the concepts behind ggplot2.

Packages in R are collections of functions and/or data that are bundled up for easy distribution, and installing a package will extend the functionality of R on your computer. If an R user creates a package and thinks that it might be useful for others, that user can distribute it through a package repository. The primary repository for distributing R packages is called CRAN (the Comprehensive R Archive Network), but there are others, such as Bioconductor, which specializes in packages related to genomic data.

If you have spent much time learning R, you may have heard of the *tidyverse*, which is a collection of R packages that share common ideas of how data should be structured and manipulated. This is in contrast to *base R*, which is the set of packages that are included when you just download and install R. The tidyverse is a set of add-ons for R, which make it easier to do many operations related to data manipulation and visualization. This book mostly uses the tidyverse, as I believe that it provides a quicker and simpler (but not less powerful!) way to work with data.

If you haven’t used the tidyverse before, there is one recipe in particular that you should read that will help you understand a foreign-looking bit of syntax: %>%, also known as the pipe operator. This is Recipe [1.7](https://r-graphics.org/RECIPE-R-BASICS-PIPE.html#RECIPE-R-BASICS-PIPE) in this chapter.

1.1 Installing a Package

1.1.1 Problem

You want to install a package from CRAN.

1.1.2 Solution

Use install.packages() and give it the name of the package you want to install. To install ggplot2, run:

**install.packages**("ggplot2")

At this point you may be prompted to select a download mirror. It’s usually best to use the first choice, <https://cloud.r-project.org/>, as it is a cloud-based mirror with endpoints all over the world.

1.1.3 Discussion

If you want to install multiple packages at once, you can pass it a vector of package names. For example, this will install most of the packages used in this book:

**install.packages**(**c**("ggplot2", "gcookbook", "MASS", "dplyr"))

When you tell R to install a package, it will automatically install any other packages that the first package depends on.

CRAN (the Comprehensive R Archive Network) is a repository of packages for R, and it is mirrored on many servers around the world. It is the default repository system used by R. There are other package repositories; Bioconductor, for example, is a repository of packages related to analyzing genomic data.

1.2 Loading a Package

1.2.1 Problem

You want to load an installed package.

1.2.2 Solution

Use library() and give it the name of the package you want to install. To load ggplot2, run:

**library**(ggplot2)

The package must already be installed on the computer.

1.2.3 Discussion

Most of the recipes in this book require loading a package before running the code, either for the graphing capabilities (as in the ggplot2 package) or for example data sets (as in the MASS and gcookbook packages).

One of R’s quirks is the package/library terminology. Although you use the library() function to load a package, a package is not a library, and some longtime R users will get irate if you call it that.

A *library* is a directory that contains a set of packages. You might, for example, have a system-wide library as well as a library for each user.

1.3 Upgrading Packages

1.3.1 Problem

You want to upgrade a package that is already installed.

1.3.2 Solution

Run update.packages():

**update.packages**()

It will prompt you for each package that can be upgraded. If you want it to upgrade all packages without asking, use ask = FALSE:

**update.packages**(ask = FALSE)

1.3.3 Discussion

Over time, package authors will release new versions of packages with bug fixes and new features, and it’s usually a good idea to keep up-to-date. However, keep in mind that occasionally new versions of packages will introduce bugs or have slightly changed behavior.

1.4 Loading a Delimited Text Data File

1.4.1 Problem

You want to load data from a delimited text file.

1.4.2 Solution

The most common way to read in a file is to use comma-separated values (CSV) data:

data <- **read.csv**("datafile.csv")

Alternatively, you can use the read\_csv() function (note the underscore instead of period) from the readr package. This function is significantly faster than read.csv(), and

1.4.3 Discussion

Since data files have many different formats, there are many options for loading them. For example, if the data file does *not* have headers in the first row:

data <- **read.csv**("datafile.csv", header = FALSE)

The resulting data frame will have columns named V1, V2, and so on, and you will probably want to rename them manually:

*# Manually assign the header names***names**(data) <- **c**("Column1", "Column2", "Column3")

You can set the delimiter with sep. If it is space-delimited, use sep = " ". If it is tab-delimited, use \t, as in:

data <- **read.csv**("datafile.csv", sep = "\t")

By default, strings in the data are treated as factors. Suppose this is your data file, and you read it in using read.csv():

"First","Last","Sex","Number""Currer","Bell","F",2"Dr.","Seuss","M",49"","Student",NA,21

The resulting data frame will store First and Last as *factors*, though it makes more sense in this case to treat them as strings (or *character vectors* in R terminology). To differentiate this, use stringsAsFactors = FALSE. If there are any columns that should be treated as factors, you can then convert them individually:

data <- **read.csv**("datafile.csv", stringsAsFactors = FALSE)*# Convert to factor*

data$Sex <- **factor**(data$Sex)**str**(data)*#> 'data.frame': 3 obs. of 4 variables:#> $ First : chr "Currer" "Dr." ""#> $ Last : chr "Bell" "Seuss" "Student"#> $ Sex : Factor w/ 2 levels "F","M": 1 2 NA#> $ Number: int 2 49 21*

Alternatively, you could load the file with strings as factors, and then convert individual columns from factors to characters.

1.4.4 See Also

read.csv() is a convenience wrapper function around read.table(). If you need more control over the input, see ?read.table.

1.5 Loading Data from an Excel File

1.5.1 Problem

You want to load data from an Excel file.

1.5.2 Solution

The readxl package has the function read\_excel() for reading .xls and .xlsx files from Excel. This will read the first sheet of an Excel spreadsheet:

*# Only need to instal once*

**install.packages**("readxl")**library**(readxl)data <- **read\_excel**("datafile.xlsx", 1)

1.5.3 Discussion

With read\_excel(), you can load from other sheets by specifying a number for sheetIndex or a name for sheetName:

data <- **read\_excel**("datafile.xls", sheet = 2)data <- **read\_excel**("datafile.xls", sheet = "Revenues")

read\_excel() uses the first row of the spreadsheet for column names. If you don’t want to use that row for column names, use col\_names = FALSE. The columns will instead be named X1, X2, and so on.

By default, read\_excel() will infer the type of each column, but if you want to specify the type of each column, you can use the col\_types argument. You can also drop columns if you specify the type as "blank".

*# Drop the first column, and specify the types of the next three columns*

data <- **read\_excel**("datafile.xls", col\_types = **c**("blank", "text", "date", "numeric"))

1.5.4 See Also

See ?read\_excel for more options controlling the reading of these files.

There are other packages for reading Excel files. The gdata package has a function read.xls() for reading in .xls files, and the xlsx package has a function read.xlsx() for reading in .xlsx files. They require external software to be installed on your computer: read.xls() requires Java, and read.xlsx() requires Perl.

**2.1 Creating a Scatter Plot**

2.1.1 Problem

You want to create a scatter plot.

2.1.2 Solution

To make a scatter plot (Figure [2.1](https://r-graphics.org/RECIPE-QUICK-SCATTER.html#fig:FIG-QUICK-SCATTER-BASE)), use plot() and pass it a vector of *x* values followed by a vector of *y* values:

**plot**(mtcars$wt, mtcars$mpg)

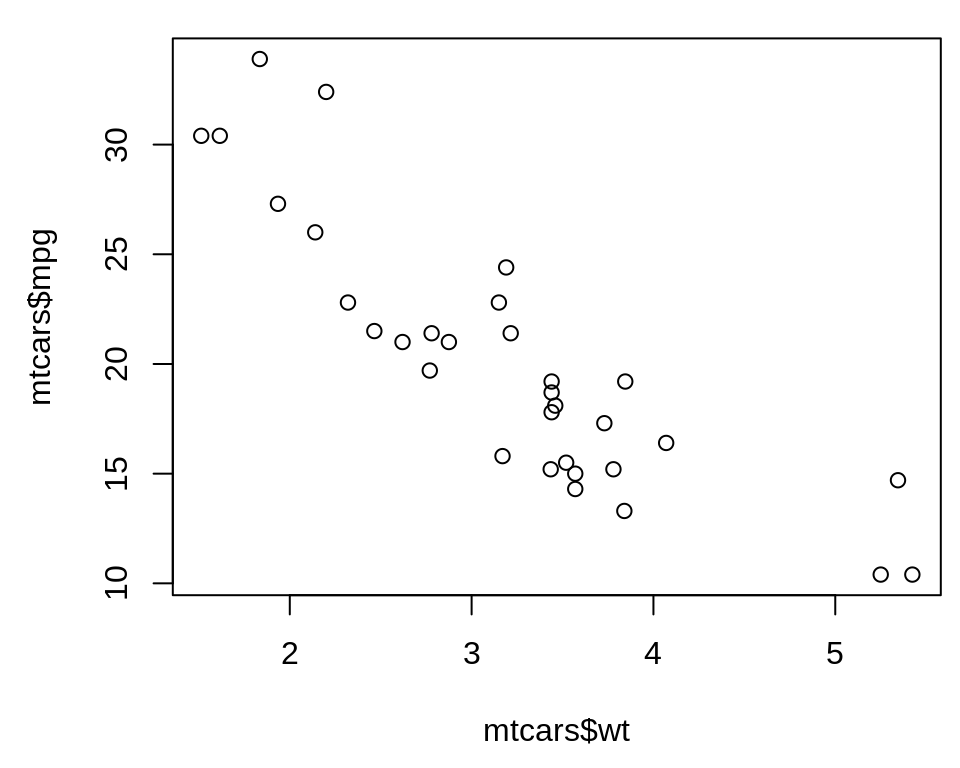


Figure 2.1: Scatter plot with base graphics

The mtcars$wt returns the column named wt from the mtcars data frame, and mtcars$mpg is the mpg column.

With ggplot2, you can get a similar result using the ggplot() function (Figure Figure [2.2](https://r-graphics.org/RECIPE-QUICK-SCATTER.html#fig:FIG-QUICK-SCATTER-GGPLOT)):

**library**(ggplot2)**ggplot**(mtcars, **aes**(x = wt, y = mpg)) + **geom\_point**()

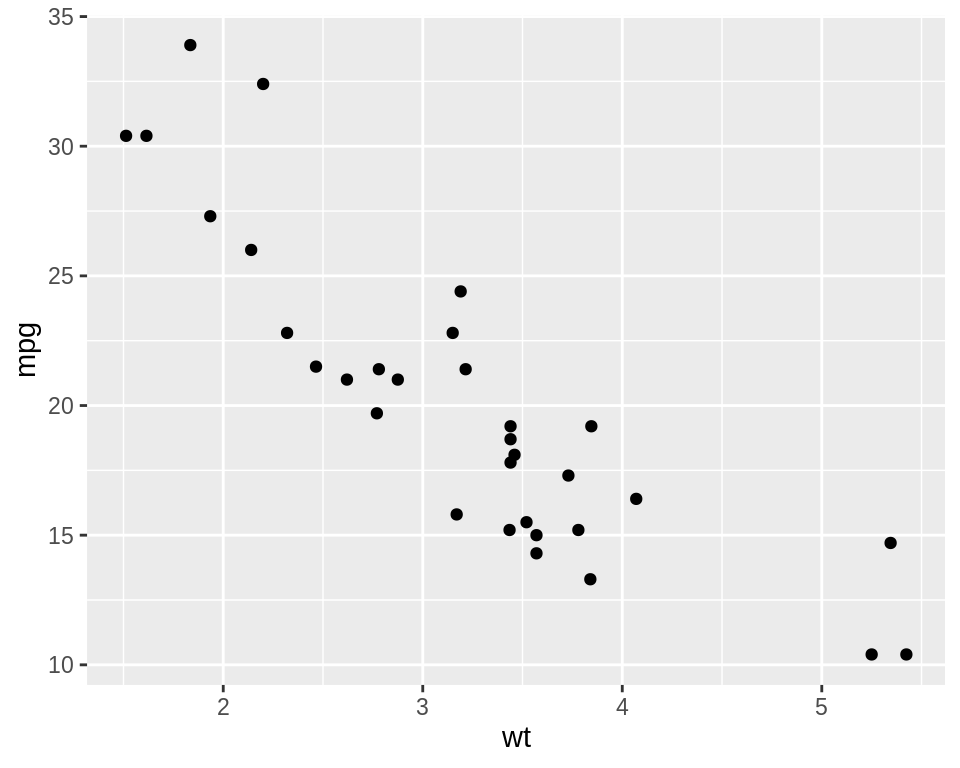


Figure 2.2: Scatter plot with ggplot2

The first part, ggplot(), tell it to create a plot object, and the second part, geom\_point(), tells it to add a layer of points to the plot.

The usual way to use ggplot() is to pass it a data frame (mtcars) and then tell it which columns to use for the x and y values. If you want to pass it two vectors for x and y values, you can use data = NULL, and then pass it the vectors. Keep in mind that ggplot2 is designed to work with data frames as the data source, not individual vectors, and that using it this way will only allow you to use a limited part of its capabilities.

**ggplot**(data = NULL, **aes**(x = mtcars$wt, y = mtcars$mpg)) + **geom\_point**()

It is common to see ggplot() commands spread across multiple lines, so you may see the above code also written like this:

2.2 Creating a Line Graph

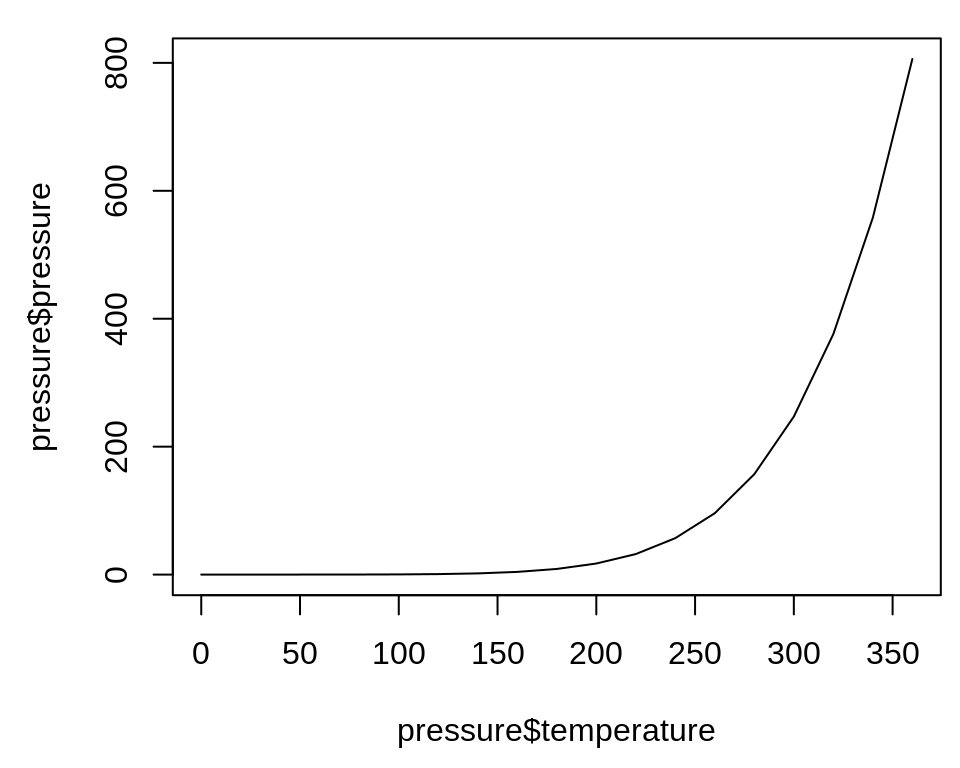
2.2.1 Problem

You want to create a line graph.

2.2.2 Solution

To make a line graph using plot() (Figure [2.3](https://r-graphics.org/RECIPE-QUICK-LINE.html#fig:FIG-QUICK-LINE-BASE), left), pass it a vector of x values and a vector of y values, and use type = "l":

**plot**(pressure$temperature, pressure$pressure, type = "l")



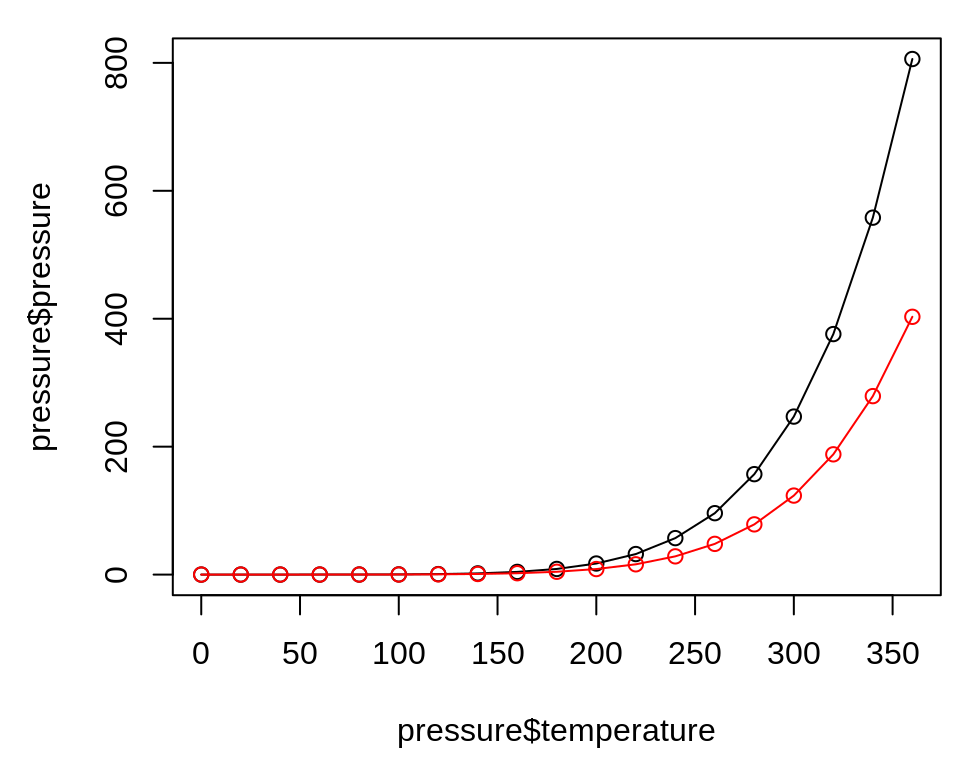


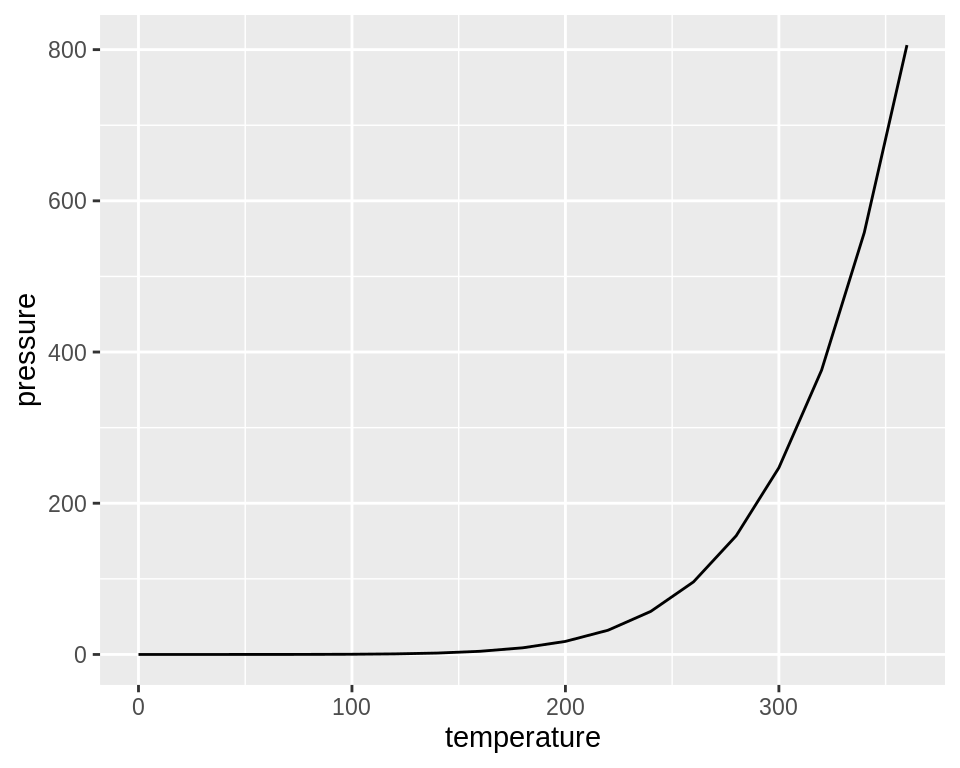
Figure 2.3: Line graph with base graphics (left); With points and another line (right)

To add points and/or multiple lines (Figure [2.3](https://r-graphics.org/RECIPE-QUICK-LINE.html#fig:FIG-QUICK-LINE-BASE), right), first call plot() for the first line, then add points with points() and additional lines with lines():

**plot**(pressure$temperature, pressure$pressure, type = "l")**points**(pressure$temperature, pressure$pressure)**lines**(pressure$temperature, pressure$pressure/2, col = "red")**points**(pressure$temperature, pressure$pressure/2, col = "red")

With ggplot2, you can get a similar result using geom\_line() (Figure [2.4](https://r-graphics.org/RECIPE-QUICK-LINE.html#fig:FIG-QUICK-LINE-GGPLOT)):

**library**(ggplot2)**ggplot**(pressure, **aes**(x = temperature, y = pressure)) + **geom\_line**()



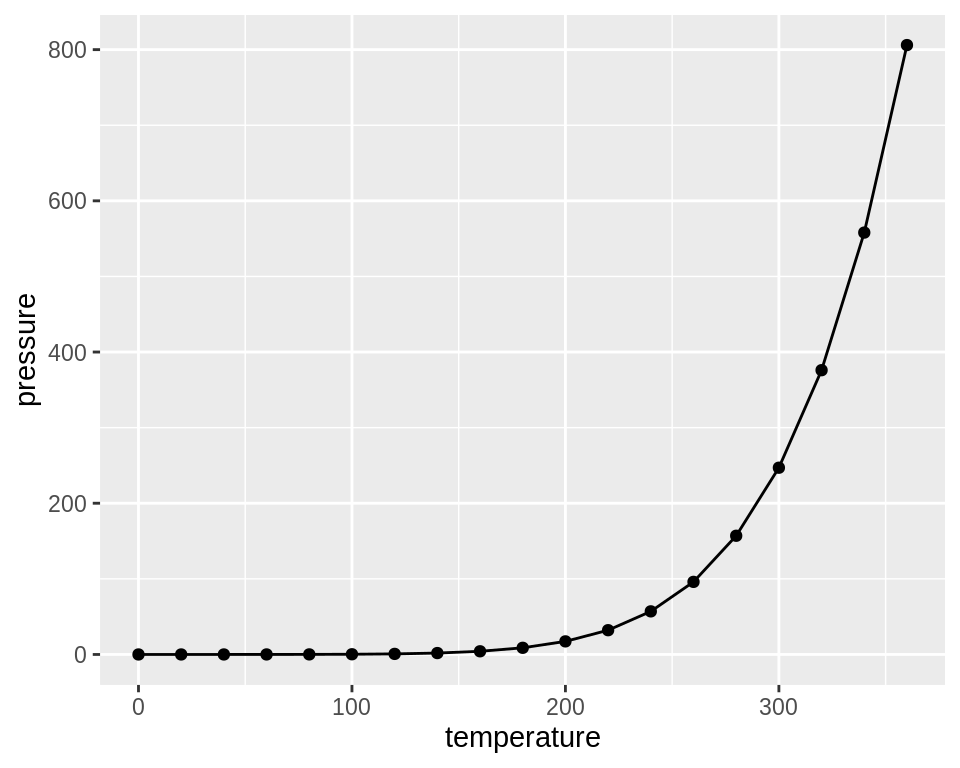


Figure 2.4: Line graph with ggplot() (left); With points added (right)

As with scatter plots, you can pass you data in vectors instead of in a data frame (but this will limit the things you can do later with the plot):

**ggplot**(pressure, **aes**(x = temperature, y = pressure)) + **geom\_line**() + **geom\_point**()

**Note**

It’s common with ggplot() to split the command on multiple lines, ending each line with a + so that R knows that the command will continue on the next line.

2.2.3 See Also

See Chapter [4](https://r-graphics.org/CHAPTER-LINE-GRAPH.html#CHAPTER-LINE-GRAPH) for more in-depth information about creating line graphs.

2.3 Creating a Bar Graph

2.3.1 Problem

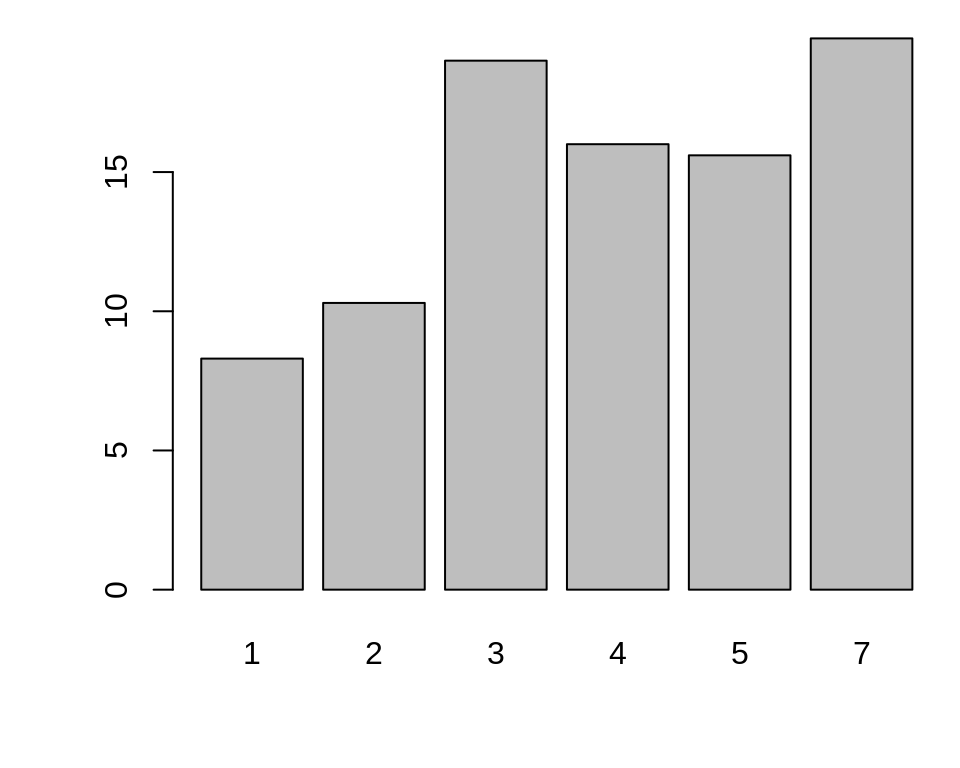
You want to make a bar graph.

2.3.2 Solution

To make a bar graph of values (Figure [2.5](https://r-graphics.org/RECIPE-QUICK-BAR.html#fig:FIG-QUICK-BAR-BASE), left), use barplot() and pass it a vector of values for the height of each bar and (optionally) a vector of labels for each bar. If the vector has names for the elements, the names will automatically be used as labels:

*# First, take a look at the BOD data*BOD*#> Time demand#> 1 1 8.3#> 2 2 10.3#> 3 3 19.0#> 4 4 16.0#> 5 5 15.6#> 6 7 19.8*

**barplot**(BOD$demand, names.arg = BOD$Time)



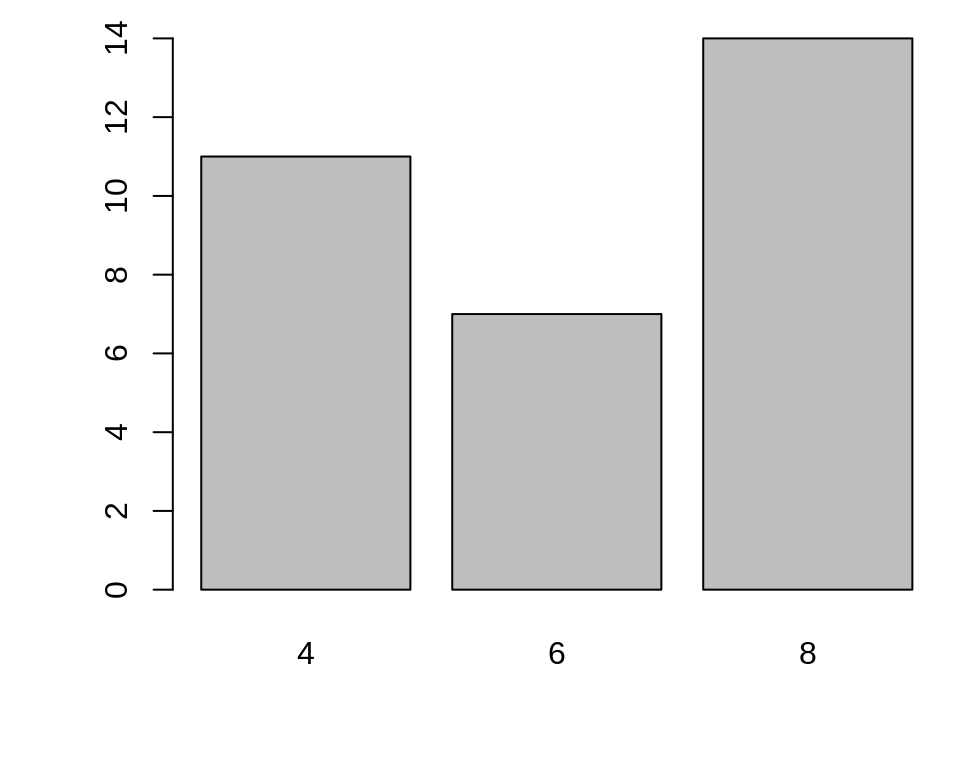


Figure 2.5: Bar graph of values with base graphics (left); Bar graph of counts (right)

Sometimes “bar graph” refers to a graph where the bars represent the *count* of cases in each category. This is similar to a histogram, but with a discrete instead of continuous x-axis. To generate the count of each unique value in a vector, use the table() function:

*# There are 11 cases of the value 4, 7 cases of 6, and 14 cases of 8***table**(mtcars$cyl)

Then pass the table to barplot() to generate the graph of counts:

*# Generate a table of counts***barplot**(**table**(mtcars$cyl))

With ggplot2, you can get a similar result using geom\_col() (Figure [2.6](https://r-graphics.org/RECIPE-QUICK-BAR.html#fig:FIG-QUICK-BAR-GGPLOT)). To plot a bar graph of *values*, use geom\_col(). Notice the difference in the output when the *x* variable is continuous and when it is discrete:

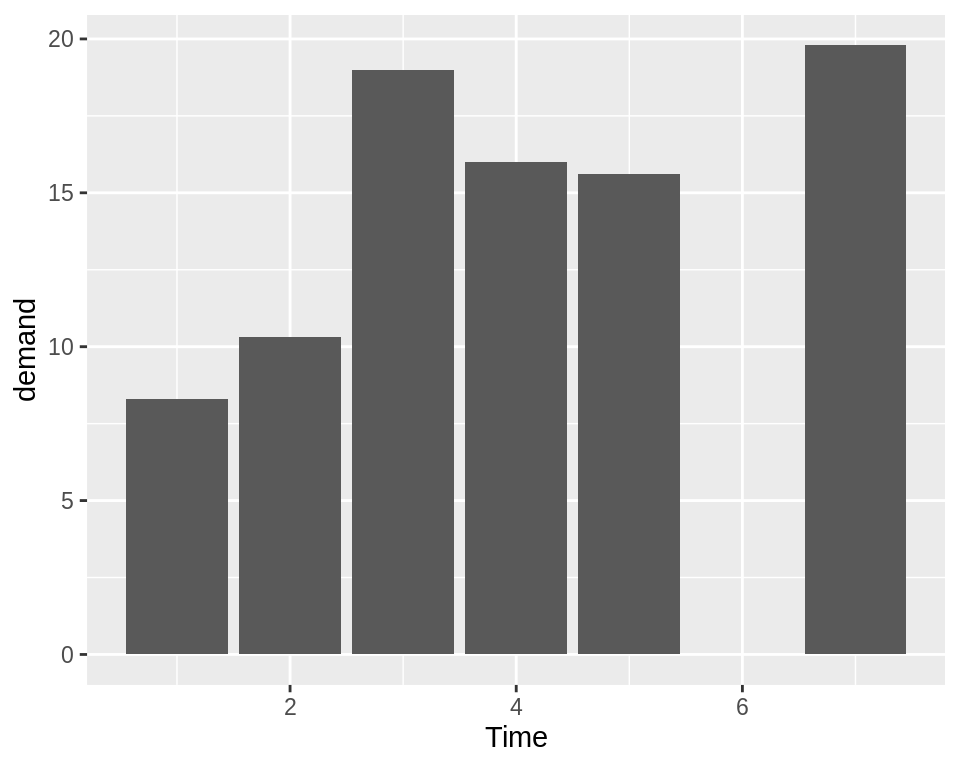
**library**(ggplot2)

*# Bar graph of values. This uses the BOD data frame, with the# "Time" column for x values and the "demand" column for y values.*

**ggplot**(BOD, **aes**(x = Time, y = demand)) + **geom\_col**()

*# Convert the x variable to a factor, so that it is treated as discrete*

**ggplot**(BOD, **aes**(x = **factor**(Time), y = demand)) + **geom\_col**()



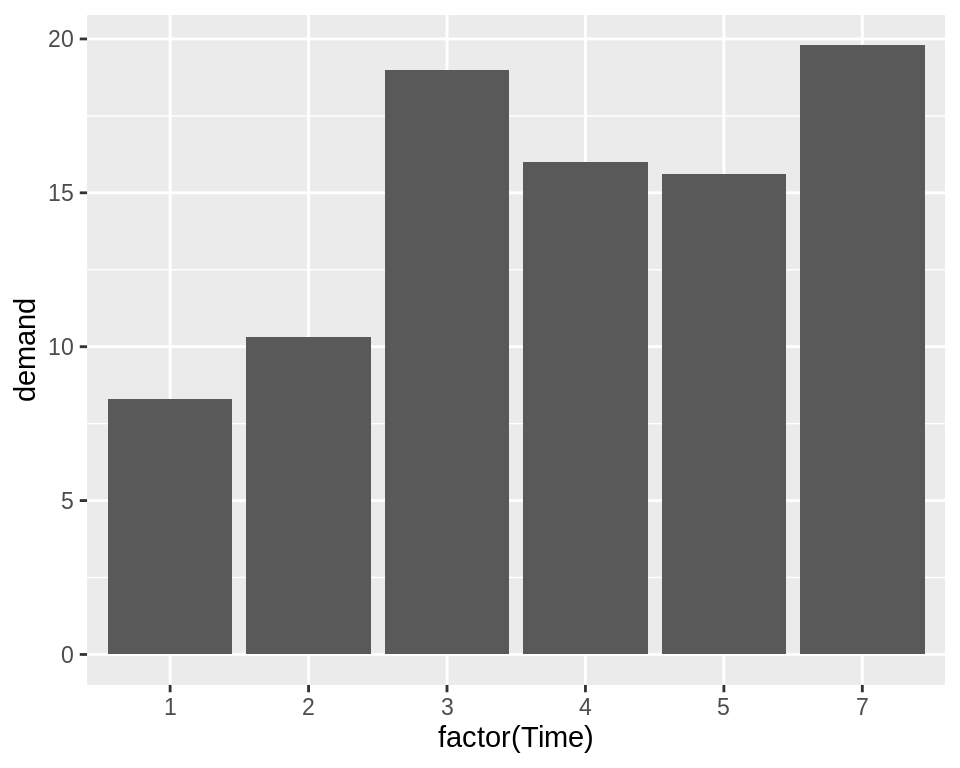


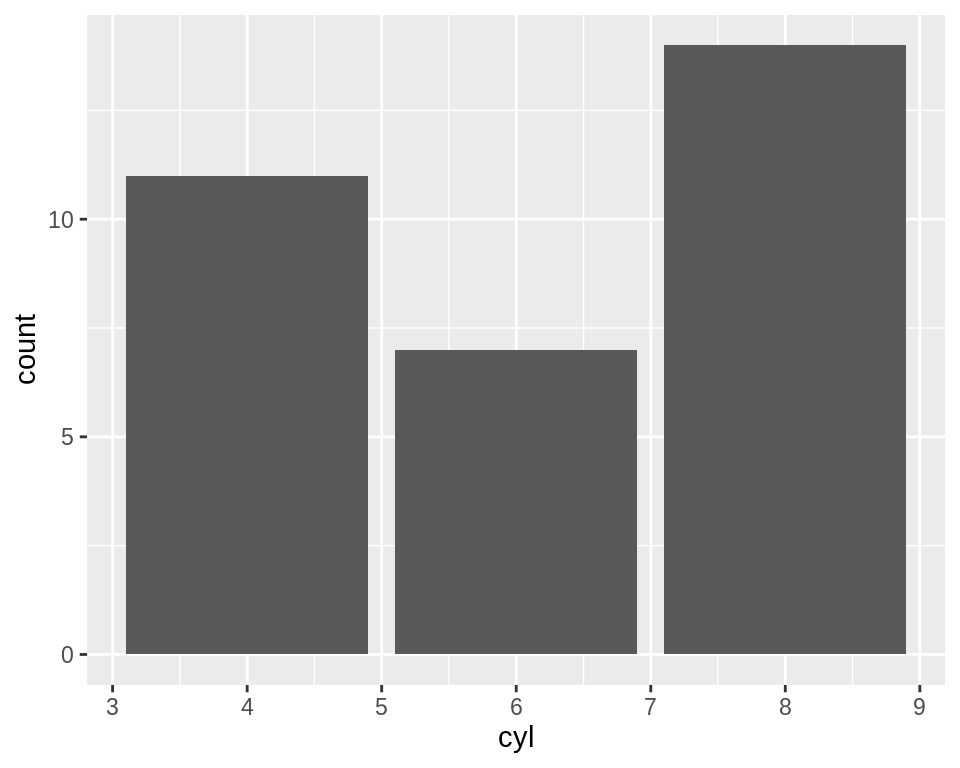
Figure 2.6: Bar graph of values using geom\_col() with a continuous x variable (left); With x variable converted to a factor (notice that there is no entry for 6; right)

ggplot2 can also be used to plot the *count* of the number of data rows in each category (Figure

[2.7](https://r-graphics.org/RECIPE-QUICK-BAR.html#fig:FIG-QUICK-BAR-GGPLOT-COUNT),by using geom\_bar() instead of geom\_col(). Once again, notice the difference between a continuous x-axis and a discrete one. For some kinds of data, it may make more sense to convert the continuous x variable to a discrete one, with the factor() function.

*# Bar graph of counts This uses the mtcars data frame, with the "cyl" column for# x position. The y position is calculated by counting the number of rows for# each value of cyl.***ggplot**(mtcars, **aes**(x = cyl)) + **geom\_bar**()

*# Bar graph of counts***ggplot**(mtcars, **aes**(x = **factor**(cyl))) + **geom\_bar**()



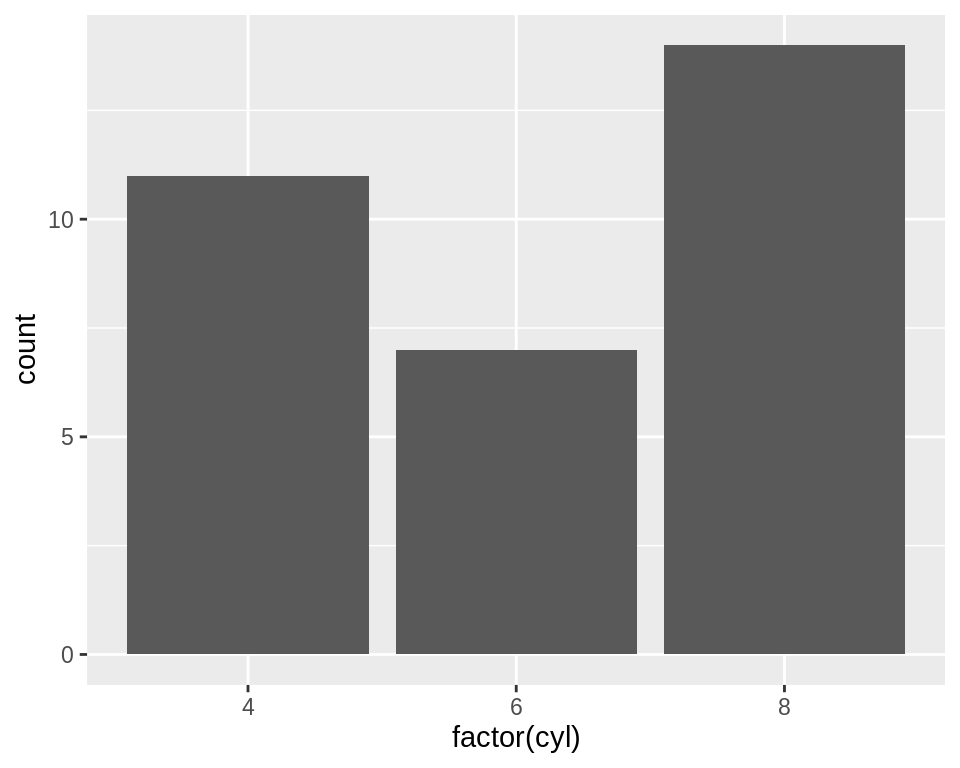


Figure 2.7: Bar graph of counts using geom\_bar() with a continuous x variable (left); With x variable converted to a factor (right)

**Note**

In previous versions of ggplot2, the recommended way to create a bar graph of values was to use geom\_bar(stat = "identity"). As of ggplot2 2.2.0, there is a geom\_col() function which does the same thing.

3.1 Making a Basic Bar Graph

3.1.1 Problem

You have a data frame where one column represents the *x* position of each bar, and another column represents the vertical (y) height of each bar.

3.1.2 Solution

Use ggplot() with geom\_col() and specify what variables you want on the x- and y-axes (Figure [3.1](https://r-graphics.org/RECIPE-BAR-GRAPH-BASIC-BAR.html#fig:FIG-BAR-GRAPH-BASIC-BAR)):

**library**(gcookbook) *# Load gcookbook for the pg\_mean data set***ggplot**(pg\_mean, **aes**(x = group, y = weight)) + **geom\_col**()

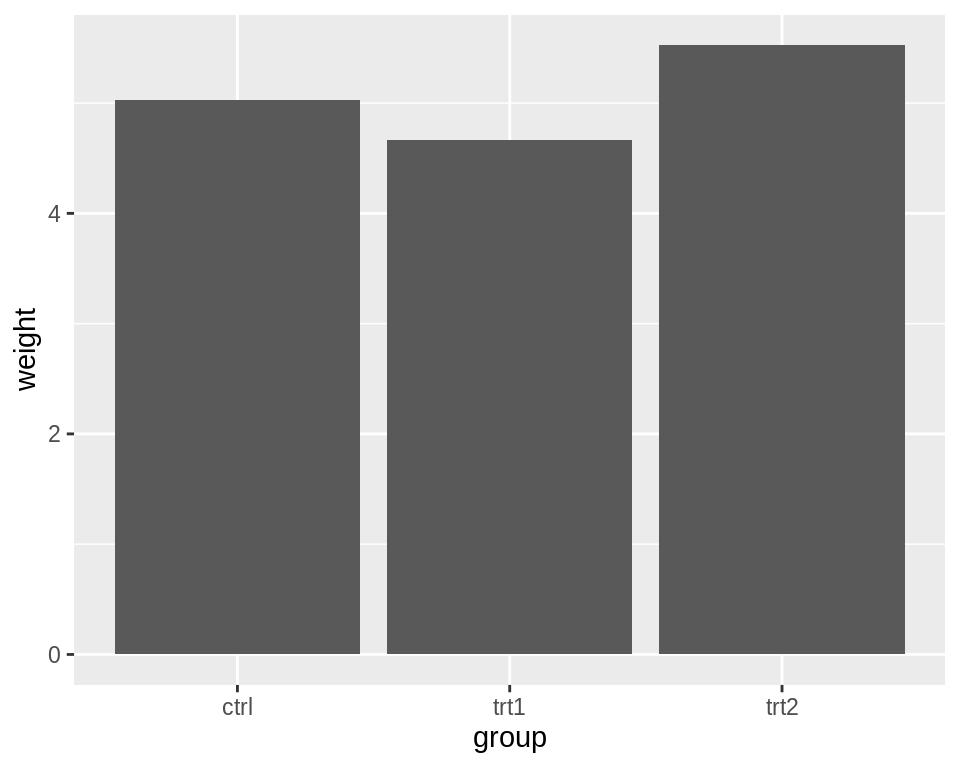


Figure 3.1: Bar graph of values with a discrete x-axis

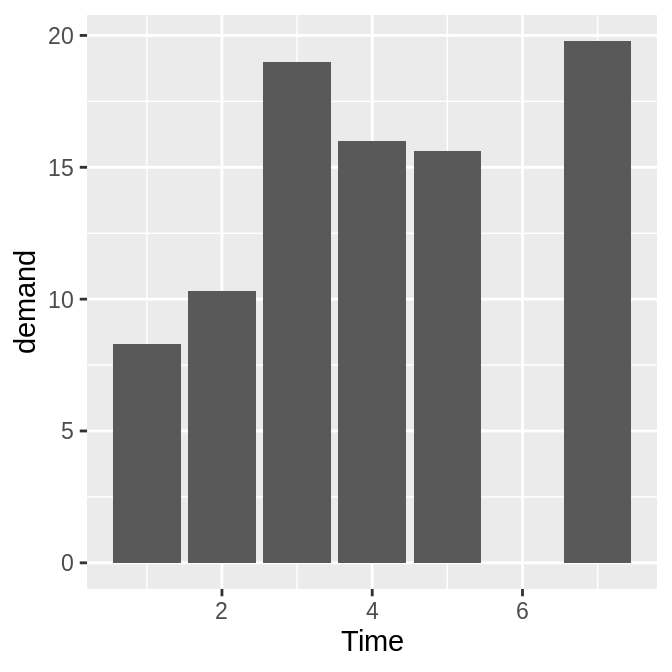
**Note**

In previous versions of ggplot2, the recommended way to create a bar graph of values was to use geom\_bar(stat = "identity"). As of ggplot2 2.2.0, there is a geom\_col() function which does the same thing.

3.1.3 Discussion

When x is a continuous (or numeric) variable, the bars behave a little differently. Instead of having one bar at each actual x value, there is one bar at each possible x value between the minimum and the maximum, as in Figure [3.2](https://r-graphics.org/RECIPE-BAR-GRAPH-BASIC-BAR.html#fig:FIG-BAR-GRAPH-BASIC-BAR-CONT). You can convert the continuous variable to a discrete variable by using factor().

*# There's no entry for Time == 6*BOD*#> Time demand#> 1 1 8.3#> 2 2 10.3#> 3 3 19.0#> 4 4 16.0#> 5 5 15.6#> 6 7 19.8# Time is numeric (continuous)***str**(BOD)*#> 'data.frame': 6 obs. of 2 variables:#> $ Time : num 1 2 3 4 5 7#> $ demand: num 8.3 10.3 19 16 15.6 19.8#> - attr(\*, "reference")= chr "A1.4, p. 270"***ggplot**(BOD, **aes**(x = Time, y = demand)) + **geom\_col**()*# Convert Time to a discrete (categorical) variable with factor()***ggplot**(BOD, **aes**(x = **factor**(Time), y = demand)) + **geom\_col**()



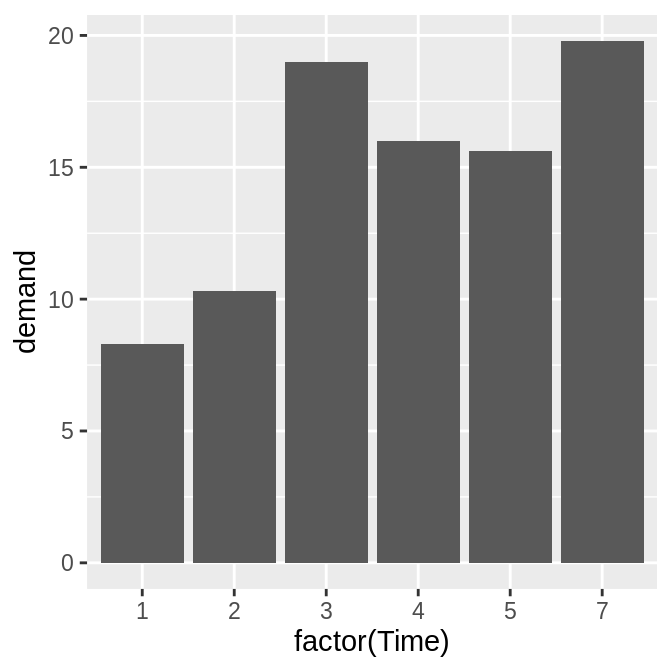


Figure 3.2: Bar graph of values with a continuous x-axis (left); With x variable converted to a factor (notice that the space for 6 is gone; right)

Notice that there was no row in BOD for Time = 6. When the x variable is continuous, ggplot2 will use a numeric axis which will have space for all numeric values within the range – hence the empty space for 6 in the plot. When Time is converted to a factor, ggplot2 uses it as a discrete variable, where the values are treated as arbitrary labels instead of numeric values, and so it won’t allocate space on the x axis for all possible numeric values between the minimum and maximum.

In these examples, the data has a column for x values and another for y values. If you instead want the height of the bars to represent the *count* of cases in each group, see Recipe [3.3](https://r-graphics.org/RECIPE-BAR-GRAPH-COUNTS.html#RECIPE-BAR-GRAPH-COUNTS).

By default, bar graphs use a dark grey for the bars. To use a color fill, use fill. Also, by default, there is no outline around the fill. To add an outline, use colour. For Figure [3.3](https://r-graphics.org/RECIPE-BAR-GRAPH-BASIC-BAR.html#fig:FIG-BAR-GRAPH-BASIC-BAR-SINGLE-FILL), we use a light blue fill and a black outline:

**ggplot**(pg\_mean, **aes**(x = group, y = weight)) + **geom\_col**(fill = "lightblue", colour = "black")

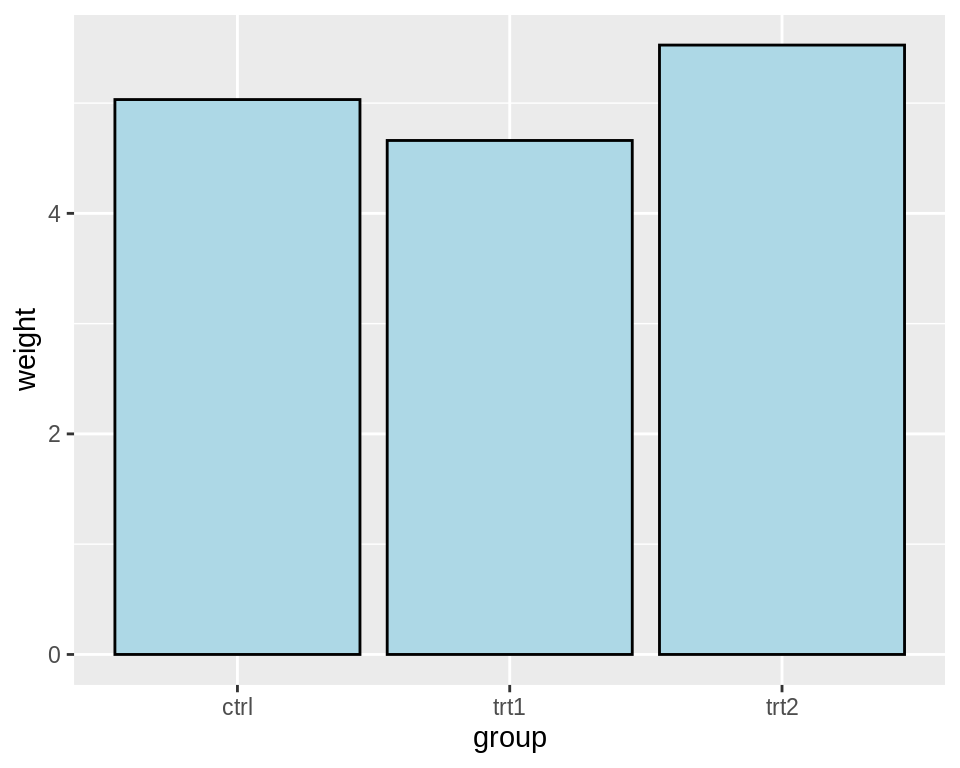


Figure 3.3: A single fill and outline color for all bars

4.1 Making a Basic Line Graph

4.1.1 Problem

You want to make a basic line graph.

4.1.2 Solution

Use ggplot() with geom\_line(), and specify which variables you mapped to x and y (Figure [4.1](https://r-graphics.org/RECIPE-LINE-GRAPH-BASIC-LINE.html#fig:FIG-LINE-GRAPH-BASIC-LINE)):

**ggplot**(BOD, **aes**(x = Time, y = demand)) + **geom\_line**()

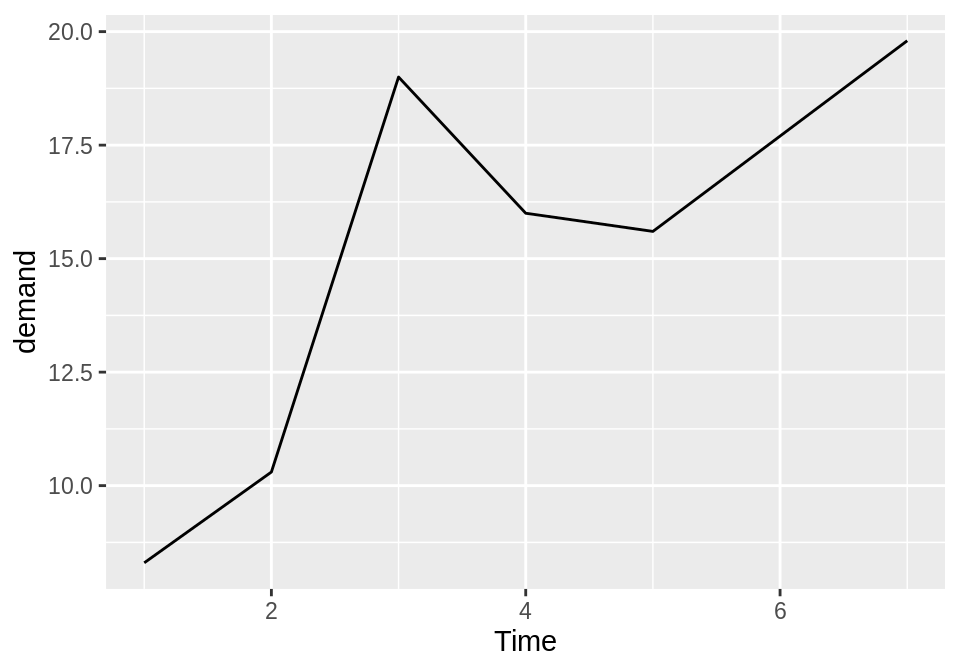


Figure 4.1: Basic line graph

4.1.3 Discussion

In this sample data set, the *x* variable, Time, is in one column and the *y* variable, demand, is in another:

BOD*#> Time demand#> 1 1 8.3#> 2 2 10.3#> 3 3 19.0#> 4 4 16.0#> 5 5 15.6#> 6 7 19.8*

Line graphs can be made with discrete (categorical) or continuous (numeric) variables on the x-axis. In the example here, the variable demand is numeric, but it could be treated as a categorical variable by converting it to a factor with factor() (Figure [4.2](https://r-graphics.org/RECIPE-LINE-GRAPH-BASIC-LINE.html#fig:FIG-LINE-GRAPH-BASIC-LINE-FACTOR)). When the *x* variable is a factor, you must also use aes(group=1) to ensure that ggplot knows that the data points belong together and should be connected with a line (see Recipe [4.3](https://r-graphics.org/RECIPE-LINE-GRAPH-MULTIPLE-LINE.html#RECIPE-LINE-GRAPH-MULTIPLE-LINE) for an explanation of why group is needed with factors):

BOD1 <- BOD *# Make a copy of the data*BOD1$Time <- **factor**(BOD1$Time)**ggplot**(BOD1, **aes**(x = Time, y = demand, group = 1)) + **geom\_line**()

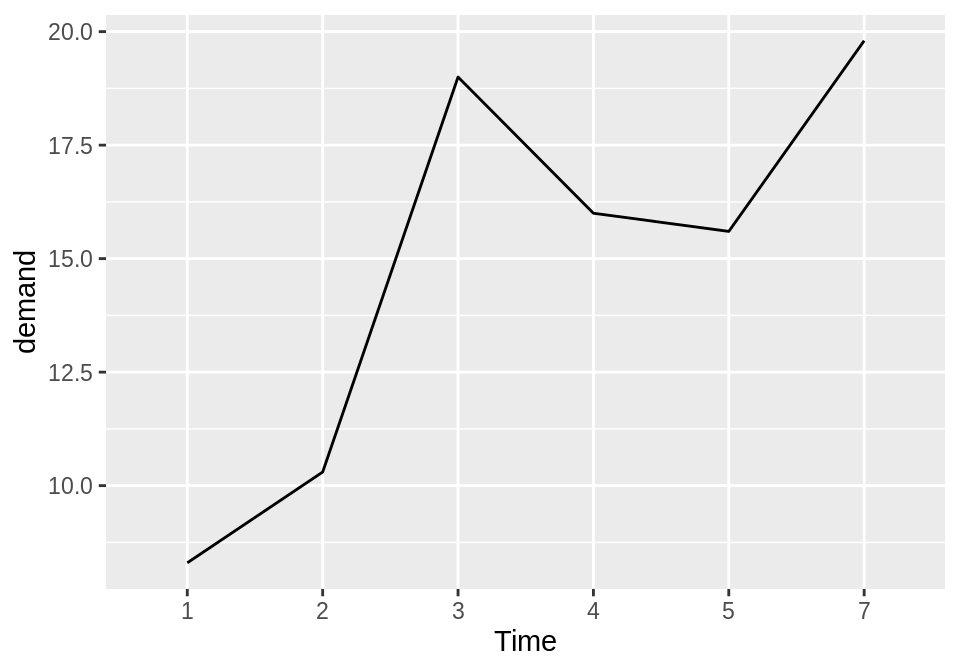


Figure 4.2: Basic line graph with a factor on the x-axis (notice that no space is allocated on the x-axis for 6)

In the BOD data set there is no entry for Time = 6, so there is no level 6 when Time is converted to a factor. Factors hold categorical values, and in that context, 6 is just another value. It happens to not be in the data set, so there’s no space for it on the x-axis.

With ggplot2, the default *y* range of a line graph is just enough to include the *y* values in the data. For some kinds of data, it’s better to have the *y* range start from zero. You can use ylim() to set the range, or you can use expand\_limits() to expand the range to include a value. This will set the range from zero to the maximum value of the demand column in BOD (Figure [4.3](https://r-graphics.org/RECIPE-LINE-GRAPH-BASIC-LINE.html#fig:FIG-LINE-GRAPH-BASIC-LINE-YLIM)):

*# These have the same result***ggplot**(BOD, **aes**(x = Time, y = demand)) + **geom\_line**() + **ylim**(0, **max**(BOD$demand))**ggplot**(BOD, **aes**(x = Time, y = demand)) + **geom\_line**() + **expand\_limits**(y = 0)

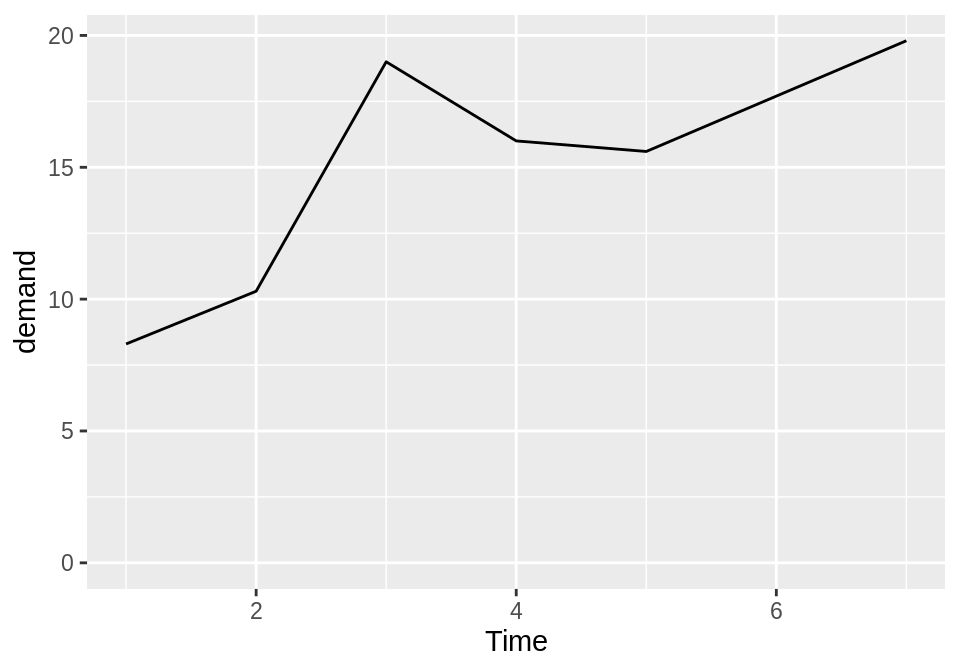


Figure 4.3: Line graph with manually set y range

4.2 Adding Points to a Line Graph

4.2.1 Problem

You want to add points to a line graph.

4.2.2 Solution

Add geom\_point() (Figure [4.4](https://r-graphics.org/RECIPE-LINE-GRAPH-POINTS.html#fig:FIG-LINE-GRAPH-POINT)):

**ggplot**(BOD, **aes**(x = Time, y = demand)) + **geom\_line**() + **geom\_point**()

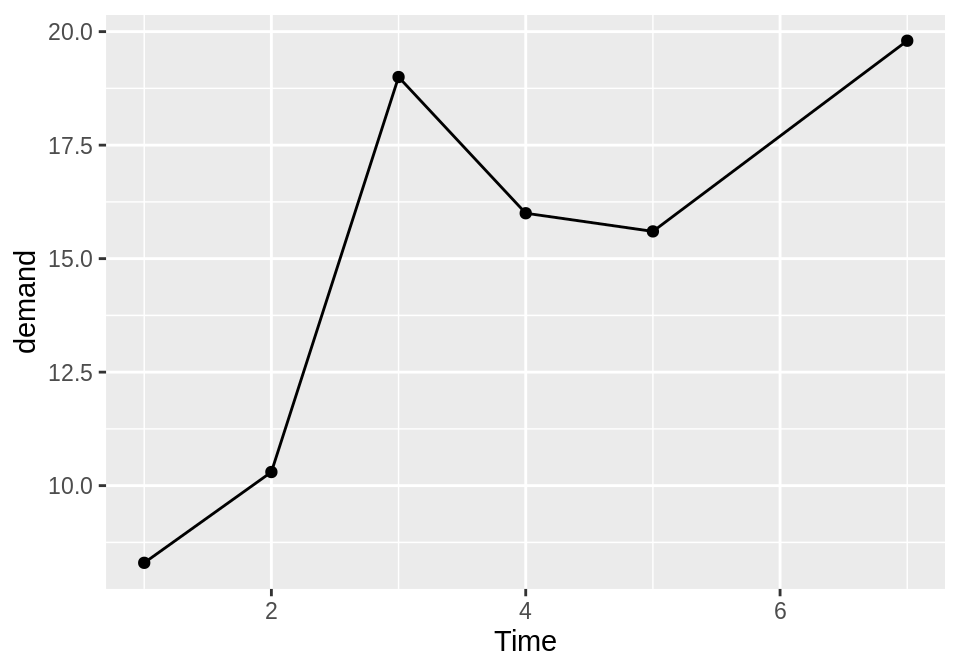


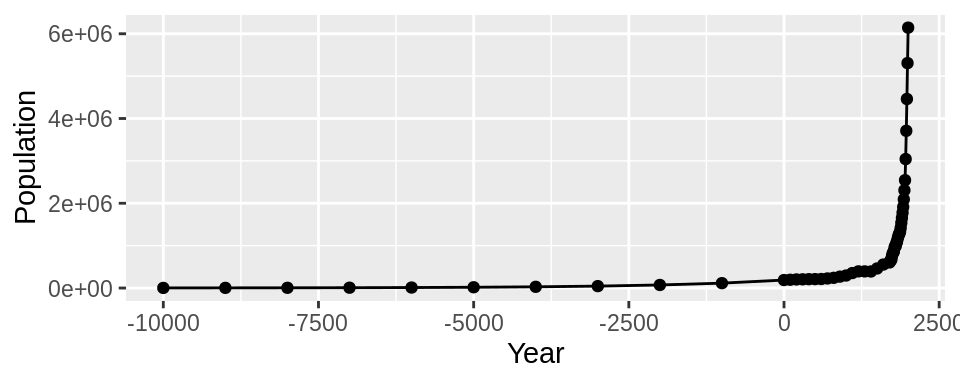
Figure 4.4: Line graph with points

4.2.3 Discussion

Sometimes it is useful to indicate each data point on a line graph. This is helpful when the density of observations is low, or when the observations do not happen at regular intervals. For example, in the BOD data set there is no entry for Time=6, but this is not apparent from just a bare line graph (compare Figure [4.3](https://r-graphics.org/RECIPE-LINE-GRAPH-BASIC-LINE.html#fig:FIG-LINE-GRAPH-BASIC-LINE-YLIM) with Figure [4.4](https://r-graphics.org/RECIPE-LINE-GRAPH-POINTS.html#fig:FIG-LINE-GRAPH-POINT)).

In the worldpop data set, the intervals between each data point are not consistent. In the far past, the estimates were not as frequent as they are in the more recent past. Displaying points on the graph illustrates when each estimate was made (Figure [4.5](https://r-graphics.org/RECIPE-LINE-GRAPH-POINTS.html#fig:FIG-LINE-GRAPH-POINTS-INTERVAL)):

**library**(gcookbook) *# Load gcookbook for the worldpop data set***ggplot**(worldpop, **aes**(x = Year, y = Population)) + **geom\_line**() + **geom\_point**()*# Same with a log y-axis***ggplot**(worldpop, **aes**(x = Year, y = Population)) + **geom\_line**() + **geom\_point**() + **scale\_y\_log10**()



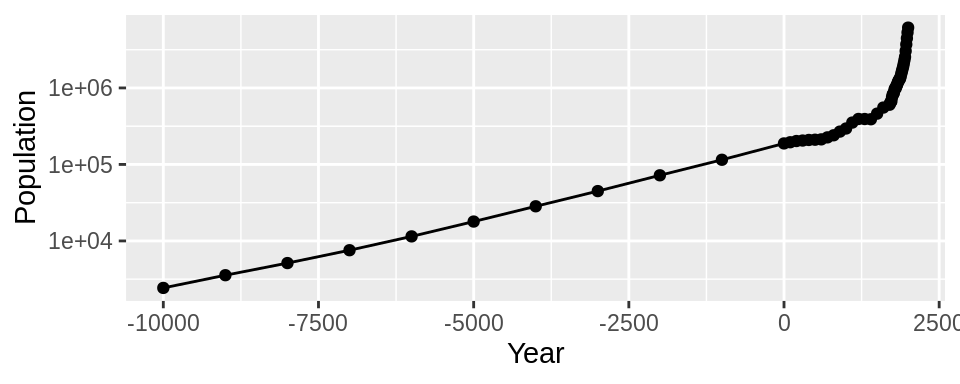


Figure 4.5: Top: points indicate where each data point is; bottom: the same data with a log y-axis

With the log y-axis, you can see that the rate of proportional change has increased in the last thousand years. The estimates for the years before 0 have a roughly constant rate of change of 10 times per 5,000 years. In the most recent 1,000 years, the population has increased at a much faster rate. We can also see that the population estimates are much more frequent in recent times–and probably more accurate!

4.3 Making a Line Graph with Multiple Lines

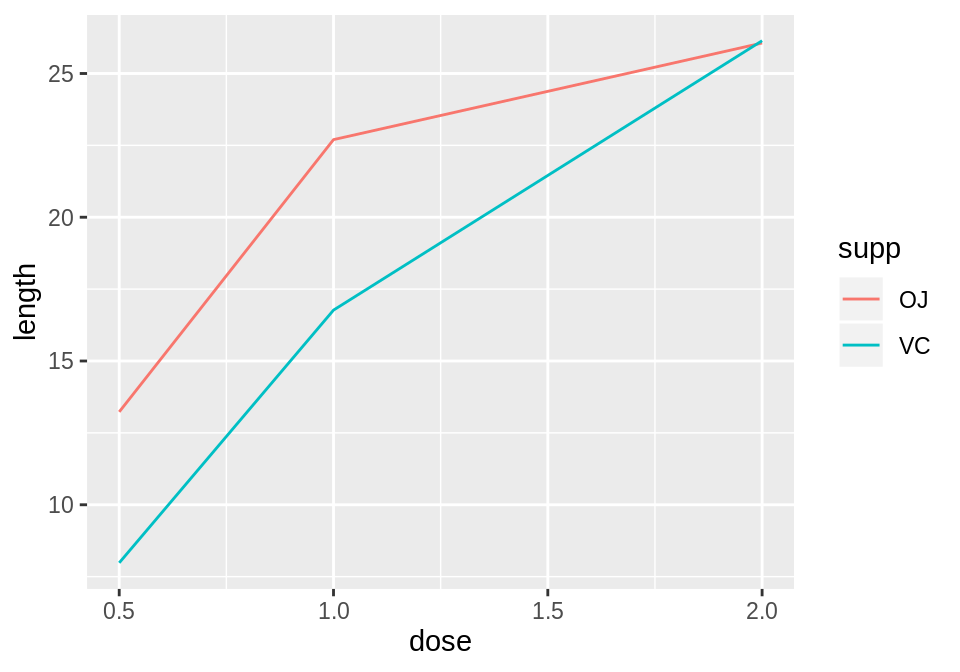
4.3.1 Problem

You want to make a line graph with more than one line.

4.3.2 Solution

In addition to the variables mapped to the x- and y-axes, map another (discrete) variable to colour or linetype, as shown in Figure [4.6](https://r-graphics.org/RECIPE-LINE-GRAPH-MULTIPLE-LINE.html#fig:FIG-LINE-GRAPH-MULTI-LINE-COLOR-TYPE):

**library**(gcookbook) *# Load gcookbook for the tg data set# Map supp to colour***ggplot**(tg, **aes**(x = dose, y = length, colour = supp)) + **geom\_line**()*# Map supp to linetype***ggplot**(tg, **aes**(x = dose, y = length, linetype = supp)) + **geom\_line**()



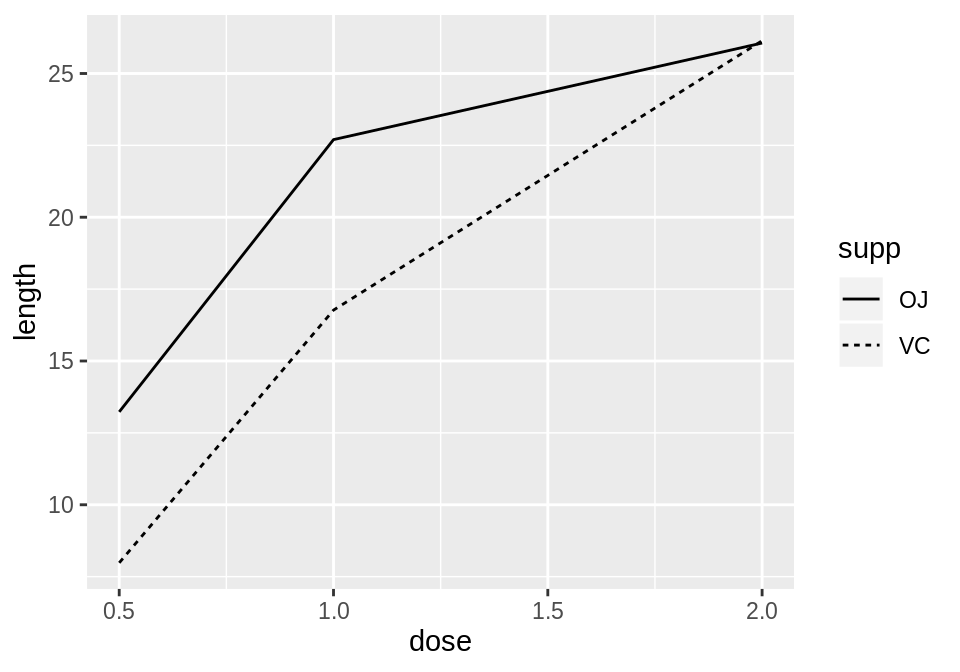


Figure 4.6: A variable mapped to colour (left); A variable mapped to linetype (right)

4.3.3 Discussion

The tg data has three columns, including the factor supp, which we mapped to colour and linetype:

tg*#> supp dose length#> 1 OJ 0.5 13.23#> 2 OJ 1.0 22.70#> 3 OJ 2.0 26.06#> 4 VC 0.5 7.98#> 5 VC 1.0 16.77#> 6 VC 2.0 26.14*

**Note**

If the *x* variable is a factor, you must also tell ggplot to group by that same variable, as described below.

Line graphs can be used with a continuous or categorical variable on the x-axis. Sometimes the variable mapped to the x-axis is *conceived* of as being categorical, even when it’s stored as a number. In the example here, there are three values of dose: 0.5, 1.0, and 2.0. You may want to treat these as categories rather than values on a continuous scale. To do this, convert dose to a factor (Figure [4.7](https://r-graphics.org/RECIPE-LINE-GRAPH-MULTIPLE-LINE.html#fig:FIG-LINE-GRAPH-MULTI-LINE-FACTOR)):

**ggplot**(tg, **aes**(x = **factor**(dose), y = length, colour = supp, group = supp)) + **geom\_line**()

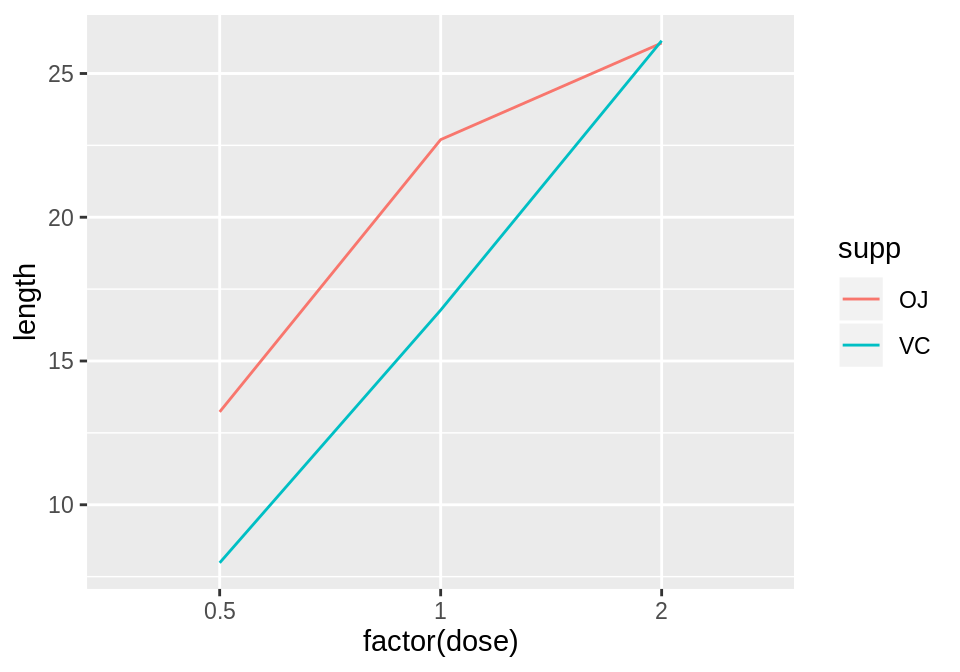


Figure 4.7: Line graph with continuous x variable converted to a factor

Notice the use of group = supp. Without this statement, ggplot won’t know how to group the data together to draw the lines, and it will give an error:

**ggplot**(tg, **aes**(x = **factor**(dose), y = length, colour = supp)) + **geom\_line**()*#> geom\_path: Each group consists of only one observation. Do you need to#> adjust the group aesthetic?*

Another common problem when the incorrect grouping is used is that you will see a jagged sawtooth pattern, as in Figure [4.8](https://r-graphics.org/RECIPE-LINE-GRAPH-MULTIPLE-LINE.html#fig:FIG-LINE-GRAPH-MULTI-LINE-SAWTOOTH):

**ggplot**(tg, **aes**(x = dose, y = length)) + **geom\_line**()

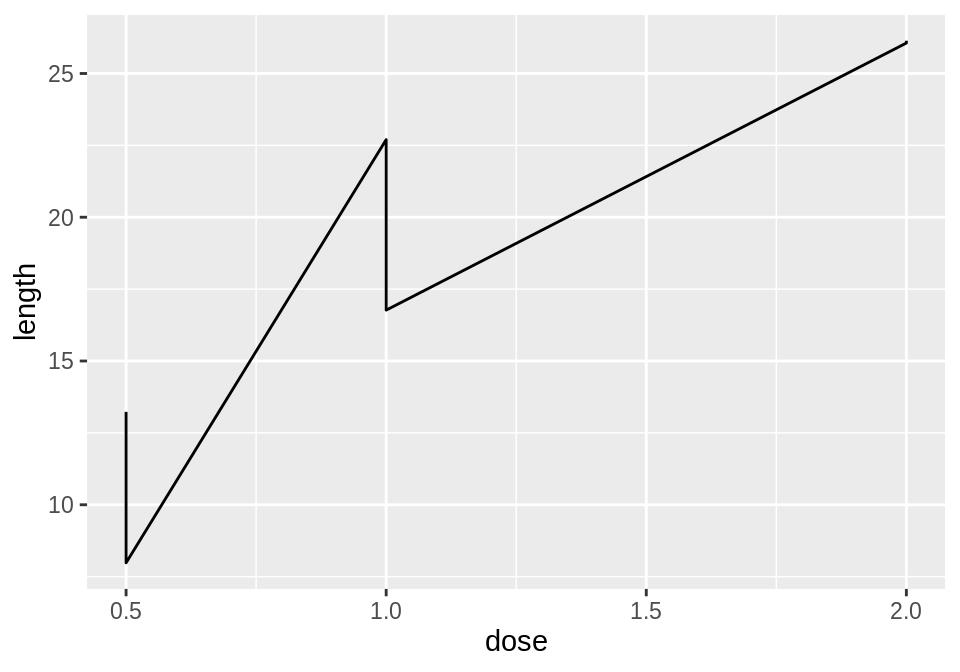


Figure 4.8: A sawtooth pattern indicates improper grouping

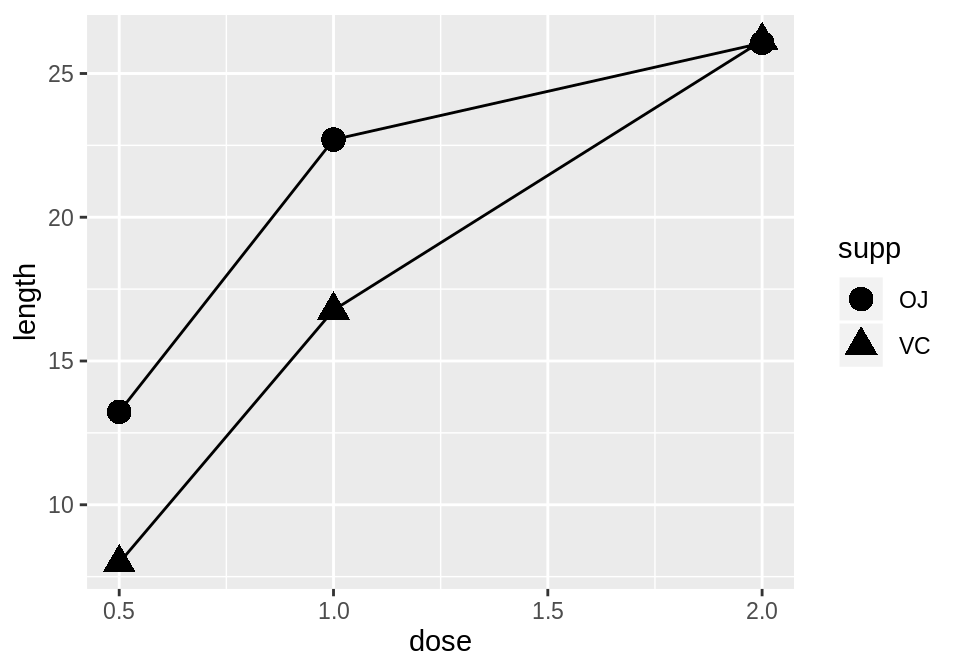
This happens because there are multiple data points at each *y* location, and ggplot thinks they’re all in one group. The data points for each group are connected with a single line, leading to the sawtooth pattern. If any *discrete* variables are mapped to aesthetics like colour or linetype, they are automatically used as grouping variables. But if you want to use other variables for grouping (that aren’t mapped to an aesthetic), they should be used with group.

**Note**

When in doubt, if your line graph looks wrong, try explicitly specifying the grouping variable with group. It’s common for problems to occur with line graphs because ggplot is unsure of how the variables should be grouped.

If your plot has points along with the lines, you can also map variables to properties of the points, such as shape and fill (Figure [4.9](https://r-graphics.org/RECIPE-LINE-GRAPH-MULTIPLE-LINE.html#fig:FIG-LINE-GRAPH-MULTI-LINE-SHAPE-FILL)):

**ggplot**(tg, **aes**(x = dose, y = length, shape = supp)) + **geom\_line**() + **geom\_point**(size = 4) *# Make the points a little larger***ggplot**(tg, **aes**(x = dose, y = length, fill = supp)) + **geom\_line**() + **geom\_point**(size = 4, shape = 21) *# Also use a point with a color fill*



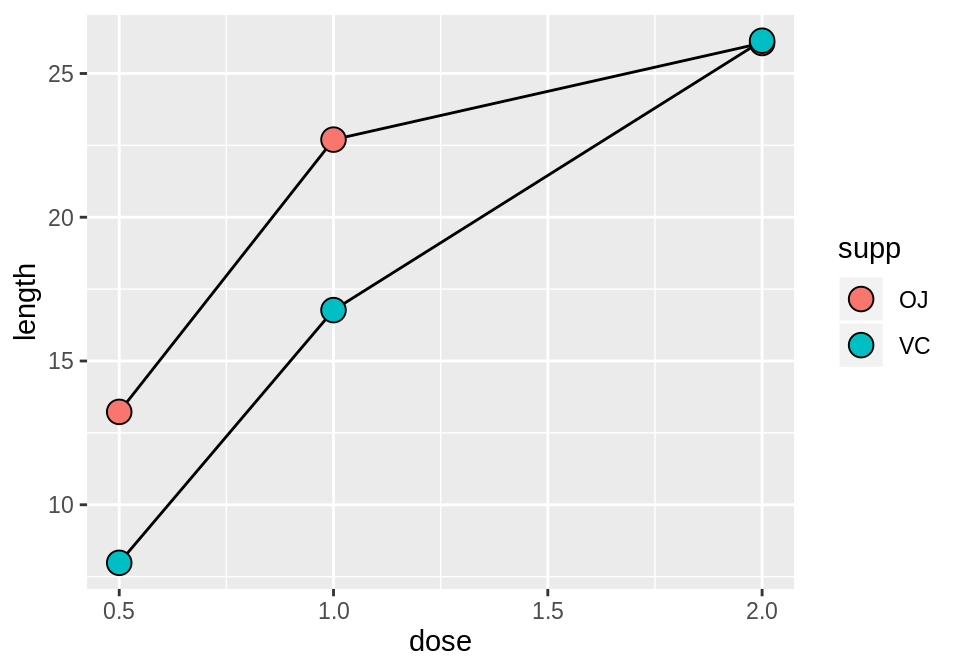


Figure 4.9: Line graph with different shapes (left); With different colors (right)

Sometimes points will overlap. In these cases, you may want to *dodge* them, which means their positions will be adjusted left and right (Figure [4.10](https://r-graphics.org/RECIPE-LINE-GRAPH-MULTIPLE-LINE.html#fig:FIG-LINE-GRAPH-MULTI-LINE-DODGE)). When doing so, you must also dodge the lines, or else only the points will move and they will be misaligned. You must also specify how far they should move when dodged:

**ggplot**(tg, **aes**(x = dose, y = length, shape = supp)) + **geom\_line**(position = **position\_dodge**(0.2)) + *# Dodge lines by 0.2* **geom\_point**(position = **position\_dodge**(0.2), size = 4) *# Dodge points by 0.2*

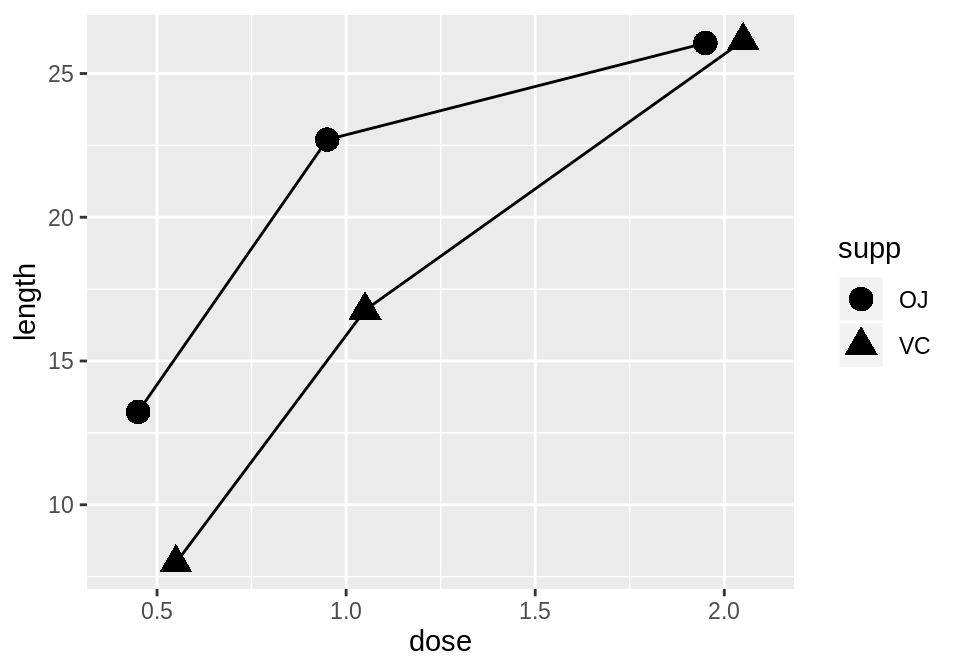


Figure 4.10: Dodging to avoid overlapping points

5.1 Making a Basic Scatter Plot

5.1.1 Problem

You want to make a scatter plot using two continuous variables.

5.1.2 Solution

Use geom\_point(), and map one variable to x and one variable to y.

We will use the heightweight data set. There are a number of columns in this data set, but we’ll only use two in this example (Figure [5.1](https://r-graphics.org/RECIPE-SCATTER-BASIC-SCATTER.html#fig:FIG-SCATTER-BASIC)):

**library**(gcookbook) *# Load gcookbook for the heightweight data set***library**(dplyr)*# Show the head of the two columns we'll use in the plot*heightweight %>% **select**(ageYear, heightIn)*#> ageYear heightIn#> 1 11.92 56.3#> 2 12.92 62.3#> 3 12.75 63.3#> ...<230 more rows>...#> 235 13.67 61.5#> 236 13.92 62.0#> 237 12.58 59.3***ggplot**(heightweight, **aes**(x = ageYear, y = heightIn)) + **geom\_point**()

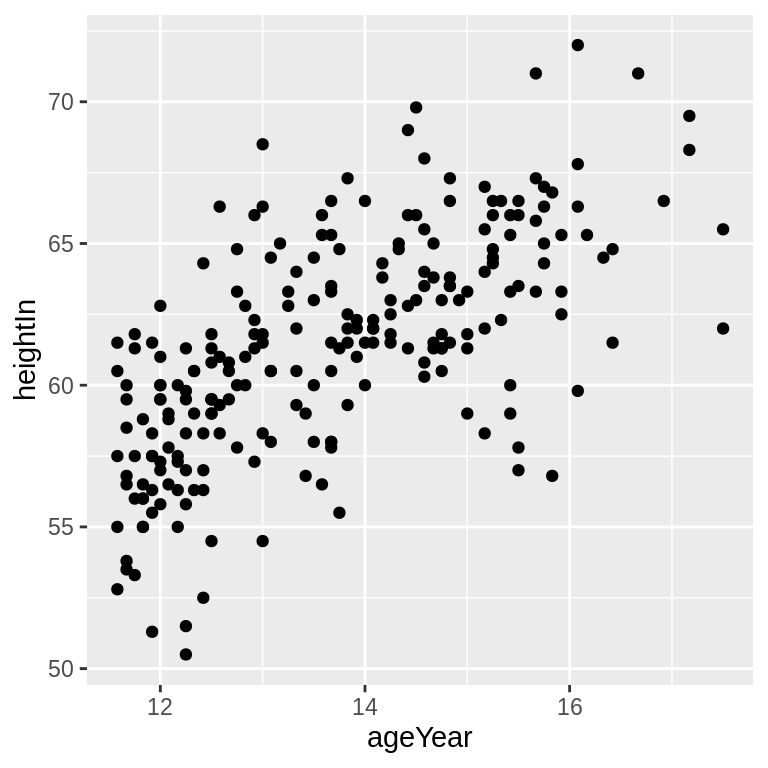


Figure 5.1: A basic scatter plot

5.1.3 Discussion

Instead of points, you can use different shapes for your scatter plot by using the shape aesthetic. A common alternative to the default solid circles (shape #19) is hollow ones (#21), as seen in Figure [5.2](https://r-graphics.org/RECIPE-SCATTER-BASIC-SCATTER.html#fig:FIG-SCATTER-BASIC-SHAPE-SIZE) (left):

**ggplot**(heightweight, **aes**(x = ageYear, y = heightIn)) + **geom\_point**(shape = 21)

The size of the points can be controlled with the size aesthetic. The default value of size is 2 (size = 2). The following code will set size = 1.5 to create smaller points (Figure [5.2](https://r-graphics.org/RECIPE-SCATTER-BASIC-SCATTER.html#fig:FIG-SCATTER-BASIC-SHAPE-SIZE), right):

**ggplot**(heightweight, **aes**(x = ageYear, y = heightIn)) + **geom\_point**(size = 1.5)



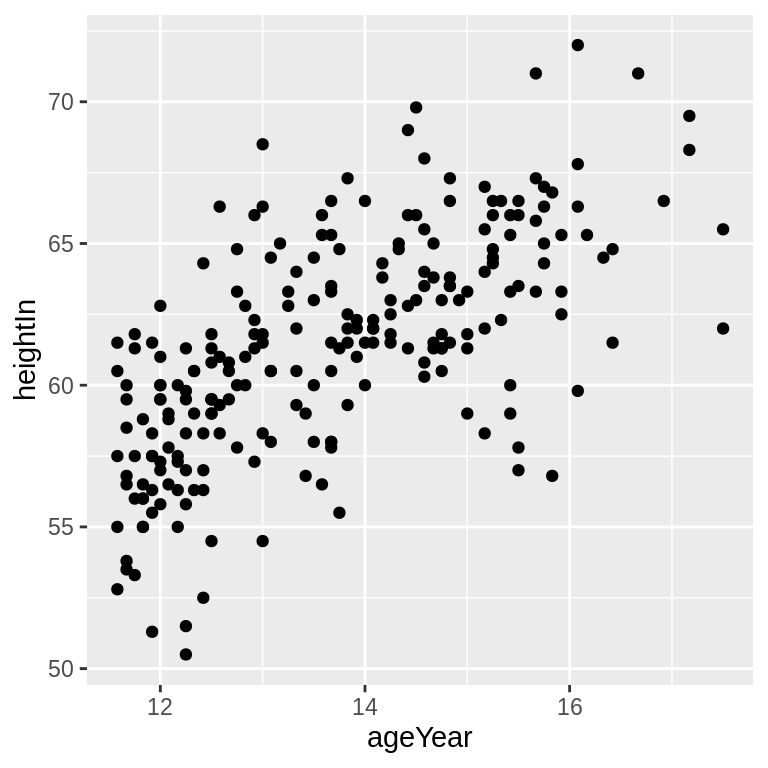


Figure 5.2: Scatter plot with hollow circles (shape 21, left); With smaller points (right)

5.2 Grouping Points Together using Shapes or Colors

5.2.1 Problem

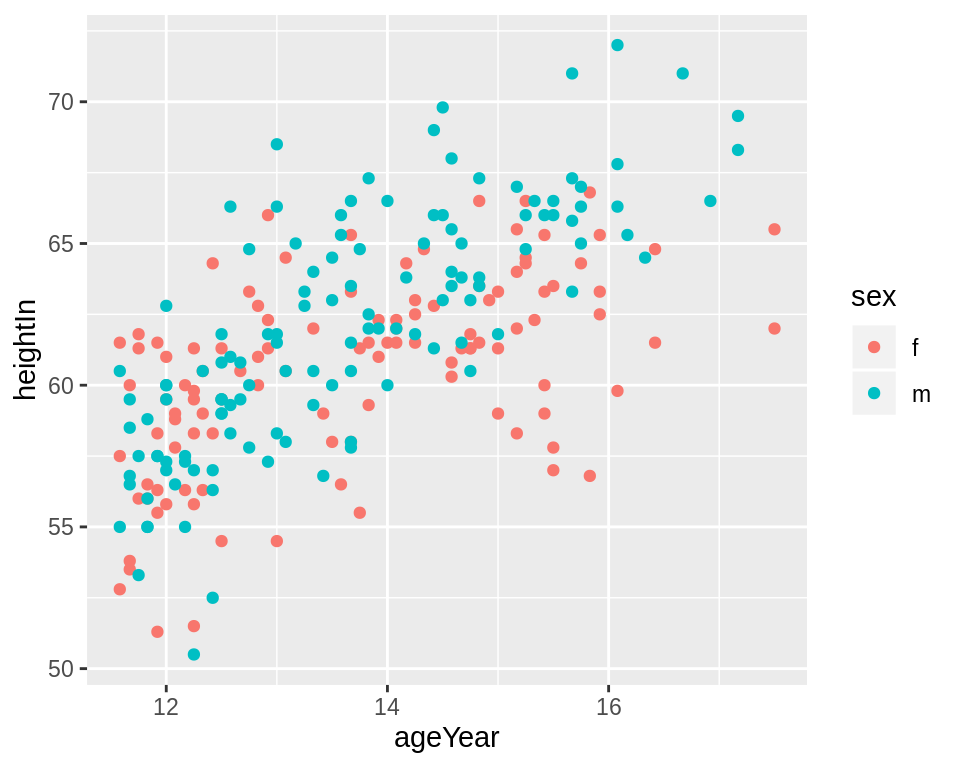
You want to visually group points by some variable (the grouping variable), using different shapes or colors.

5.2.2 Solution

Map the grouping variable to the aesthetic of shape or colour. We’ll use three columns from the heightweight data set for this example:

**library**(gcookbook) *# Load gcookbook for the heightweight data set# Show the head of the three columns we'll use*heightweight %>% **select**(sex, ageYear, heightIn)*#> sex ageYear heightIn#> 1 f 11.92 56.3#> 2 f 12.92 62.3#> 3 f 12.75 63.3#> ...<230 more rows>...#> 235 m 13.67 61.5#> 236 m 13.92 62.0#> 237 m 12.58 59.3*

We can use the aesthetics of colour or shape to visually differentiate the data points belonging to different categories of sex. We do this by mapping sex to one of the aesthetics colour or shape (Figure[5.3](https://r-graphics.org/RECIPE-SCATTER-GROUPED-SCATTER.html#fig:FIG-SCATTER-SHAPE-COLOR)):



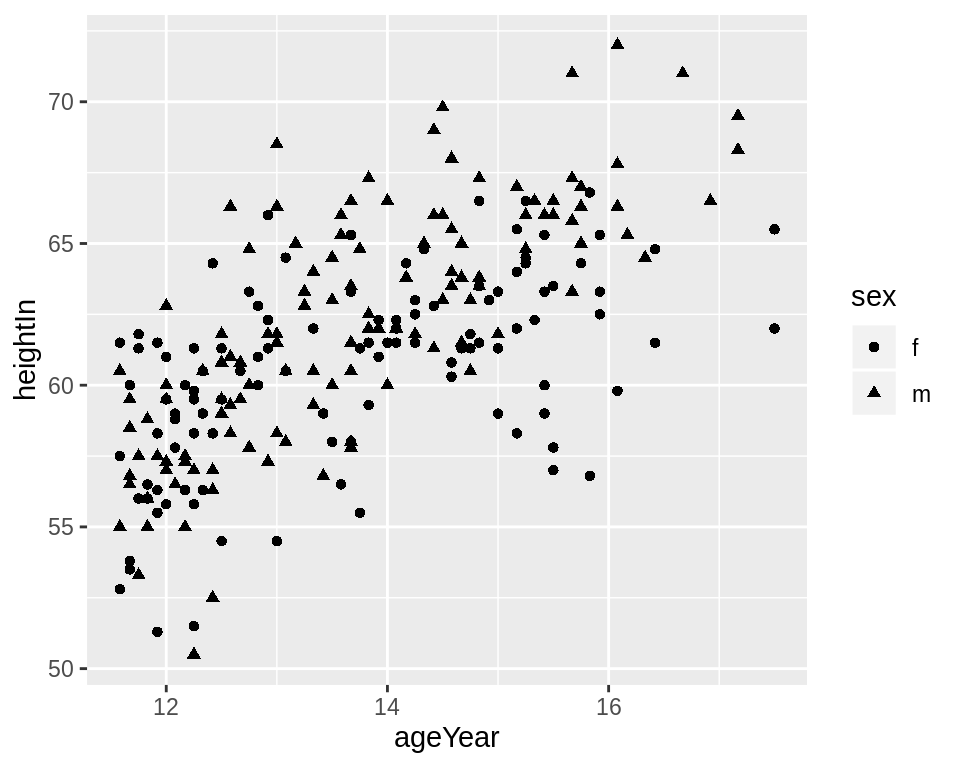


Figure 5.3: Grouping points by a variable mapped to colour (left), or to shape (right)

5.2.3 Discussion

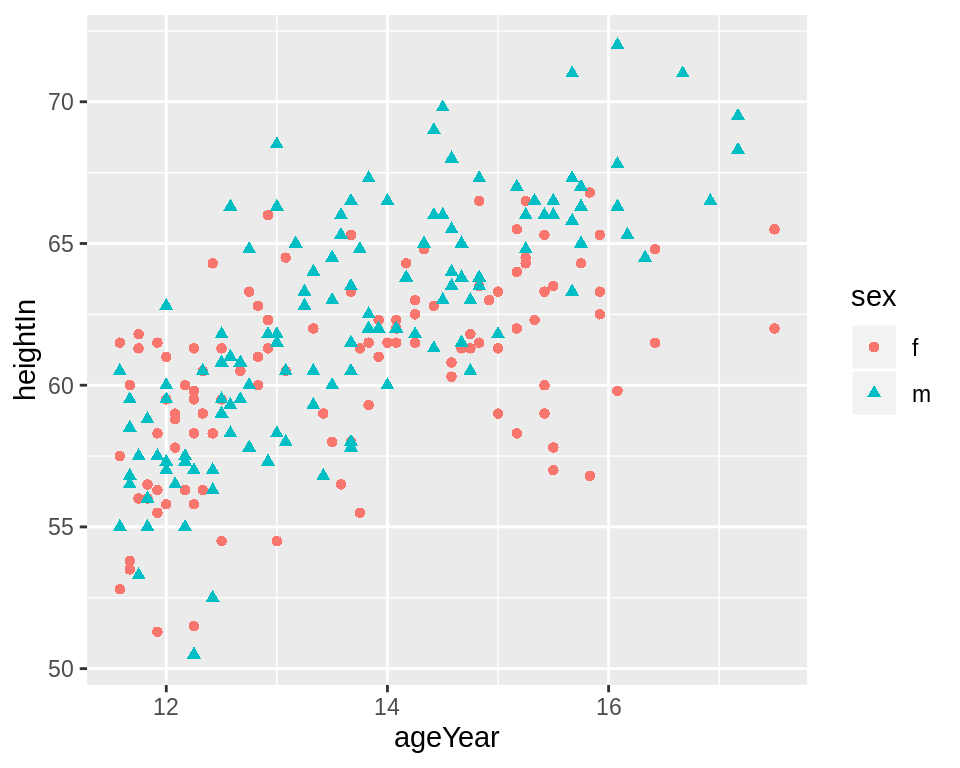
The grouping variable you choose must be categorical – in other words, a factor or character vector. If the grouping variable is a numeric vector, you should convert it to a factor first.

It is possible to map a variable to both shape and colour, or, if you have multiple grouping variables, to map each grouping variable to a different aesthetic. Here, we’ll map the variable sex to both shape and colour aesthetics (Figure [5.4](https://r-graphics.org/RECIPE-SCATTER-GROUPED-SCATTER.html#fig:FIG-SCATTER-SHAPE-COLOR-BOTH), left):

**ggplot**(heightweight, **aes**(x = ageYear, y = heightIn, shape = sex, colour = sex)) + **geom\_point**()

You may want to use different shapes and colors than are given by the default settings. You can select other shapes for the grouping variables using scale\_shape\_manual(), and select other colors using scale\_colour\_brewer() or scale\_colour\_manual(). (Figure [5.4](https://r-graphics.org/RECIPE-SCATTER-GROUPED-SCATTER.html#fig:FIG-SCATTER-SHAPE-COLOR-BOTH), right):

**ggplot**(heightweight, **aes**(x = ageYear, y = heightIn, shape = sex, colour = sex)) + **geom\_point**() + **scale\_shape\_manual**(values = **c**(1,2)) + **scale\_colour\_brewer**(palette = "Set1")



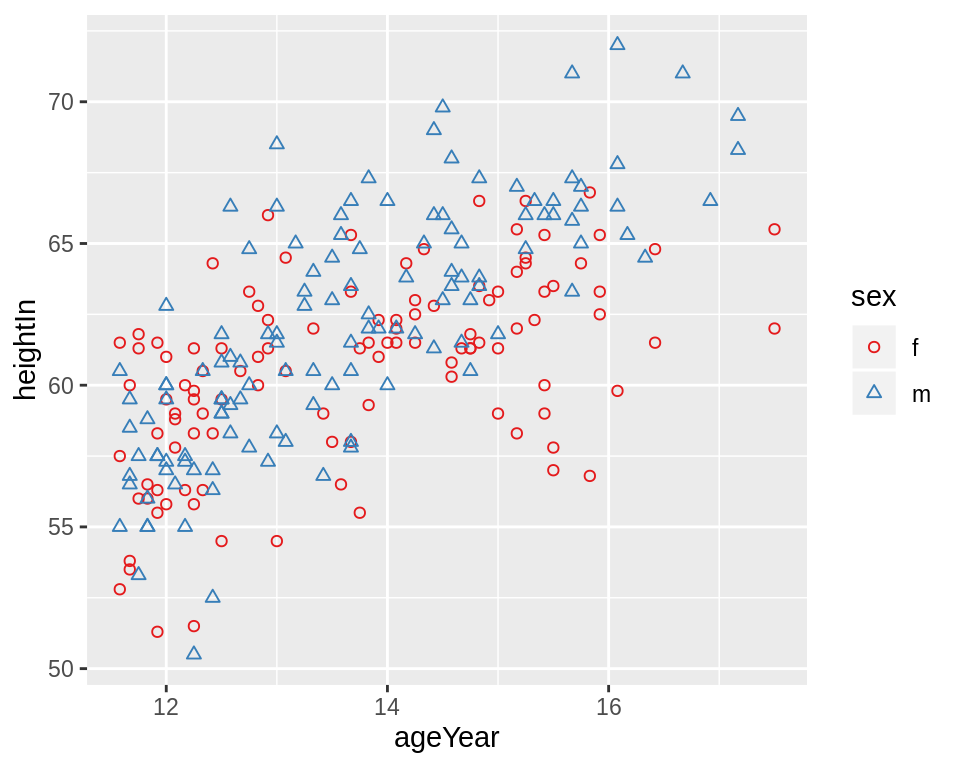


Figure 5.4: Mapping to both shape and colour (left); With manually set shapes and colors (right)

# Sorting

* [Problem](http://www.cookbook-r.com/Manipulating_data/Sorting/#problem)
* [Solution](http://www.cookbook-r.com/Manipulating_data/Sorting/#solution)
  + [Vectors](http://www.cookbook-r.com/Manipulating_data/Sorting/#vectors)
  + [Data frames](http://www.cookbook-r.com/Manipulating_data/Sorting/#data-frames)
    - [Reverse sort](http://www.cookbook-r.com/Manipulating_data/Sorting/#reverse-sort)

## Problem

You want to sort a vector, matrix, or data frame.

## Solution

### Vectors

*# Make up a randomly ordered vector*

v <- sample(101:110)

*# Sort the vector*

sort(v)

*#> [1] 101 102 103 104 105 106 107 108 109 110*

*# Reverse sort*

sort(v, decreasing=TRUE)

*#> [1] 110 109 108 107 106 105 104 103 102 101*

### Data frames

To sort a data frame on one or more columns, you can use the arrange function from plyr package, or use R’s built-in functions. The arrange function is much easier to use, but does require the external package to be installed.

*# Make a data frame*

df <- data.frame (id=1:4,

weight=c(20,27,24,22),

size=c("small", "large", "medium", "large"))

df

library(plyr)

*# Sort by weight column. These have the same result.*

arrange(df, weight) *# Use arrange from plyr package*

df[ order(df$weight), ] *# Use built-in R functions*

*#> id weight size*

*#> 1 1 20 small*

*#> 2 4 22 large*

*#> 3 3 24 medium*

*#> 4 2 27 large*

*# Sort by size, then by weight*

arrange(df, size, weight) *# Use arrange from plyr package*

df[ order(df$size, df$weight), ] *# Use built-in R functions*

*#> id weight size*

*#> 4 4 22 large*

*#> 2 2 27 large*

*#> 3 3 24 medium*

*#> 1 1 20 small*

*# Sort by all columns in the data frame, from left to right*

df[ do.call(order, as.list(df)), ]

*# In this particular example, the order will be unchanged*

Note that the size column is a factor and is sorted by the order of the factor levels. In this case, the levels were automatically assigned alphabetically (when creating the data frame), so **large** is first and **small** is last.

#### Reverse sort

The overall order of the sort can be reversed with the argument decreasing=TRUE.

To reverse the direction of a particular column, the method depends on the data type:

* Numbers: put a - in front of the variable name, e.g. df[ order(-df$weight), ].
* Factors: convert to integer and put a - in front of the variable name, e.g. df[ order(-xtfrm(df$size)), ].
* Characters: there isn’t a simple way to do this. One method is to convert to a factor first and then sort as above.

# Reverse sort by weight column. These all have the same effect:

arrange(df, -weight) # Use arrange from plyr package

df[ order(df$weight, decreasing=TRUE), ] # Use built-in R functions

df[ order(-df$weight), ] # Use built-in R functions

#> id weight size

#> 2 2 27 large

#> 3 3 24 medium

#> 4 4 22 large

#> 1 1 20 small

# Sort by size (increasing), then by weight (decreasing)

arrange(df, size, -weight) # Use arrange from plyr package

df[ order(df$size, -df$weight), ] # Use built-in R functions

#> id weight size

#> 2 2 27 large

#> 4 4 22 large

#> 3 3 24 medium

#> 1 1 20 small

# Sort by size (decreasing), then by weight (increasing)

# The call to xtfrm() is needed for factors

arrange(df, -xtfrm(size), weight) # Use arrange from plyr package

df[ order(-xtfrm(df$size), df$weight), ] # Use built-in R functions

#> id weight size

#> 1 1 20 small

#> 3 3 24 medium

#> 4 4 22 large

#> 2 2 27 large

# Randomizing order

## Problem

You want to randomize the order of a data structure.

## Solution

*# Create a vector*

v <- 11:20

*# Randomize the order of the vector*

v <- sample(v)

*# Create a data frame*

data <- data.frame(label=letters[1:5], number=11:15)

data

*#> label number*

*#> 1 a 11*

*#> 2 b 12*

*#> 3 c 13*

*#> 4 d 14*

*#> 5 e 15*

*# Randomize the order of the data frame*

data <- data[sample(1:nrow(data)), ]

data

*#> label number*

*#> 5 e 15*

*#> 2 b 12*

*#> 4 d 14*

*#> 3 c 13*

*#> 1 a 11*

# *Converting between vector types*

## *Problem*

*You want to convert between numeric vectors, character vectors, and factors.*

## *Solution*

*Suppose you start with this numeric vector n:*

*n <- 10:14*

*n*

*#> [1] 10 11 12 13 14*

*To convert the numeric vector to the other two types (we’ll also save these results in c and f):*

*# Numeric to Character*

*c <- as.character(n)*

*# Numeric to Factor*

*f <- factor(n)*

*# 10 11 12 13 14*

*To convert the character vector to the other two:*

*# Character to Numeric*

*as.numeric(c)*

*#> [1] 10 11 12 13 14*

*# Character to Factor*

*factor(c)*

*#> [1] 10 11 12 13 14*

*#> Levels: 10 11 12 13 14*

*Converting a factor to a character vector is straightforward:*

*# Factor to Character*

*as.character(f)*

*#> [1] "10" "11" "12" "13" "14"*

*However, converting a factor to a numeric vector is a little trickier. If you just convert it with as.numeric, it will give you the numeric coding of the factor, which probably isn’t what you want.*

*as.numeric(f)*

*#> [1] 1 2 3 4 5*

*# Another way to get the numeric coding, if that's what you want:*

*unclass(f)*

*#> [1] 1 2 3 4 5*

*#> attr(,"levels")*

*#> [1] "10" "11" "12" "13" "14"*

*The way to get the text values converted to numbers is to first convert it to a character, then a numeric vector.*

*# Factor to Numeric*

*as.numeric(as.character(f))*

*#> [1] 10 11 12 13 14*

# Finding and removing duplicate records

## Problem

You want to find and/or remove duplicate entries from a vector or data frame.

## Solution

With vectors:

# Generate a vector

set.seed(158)

x <- round(rnorm(20, 10, 5))

x

#> [1] 14 11 8 4 12 5 10 10 3 3 11 6 0 16 8 10 8 5 6 6

# For each element: is this one a duplicate (first instance of a particular value

# not counted)

duplicated(x)

#> [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE TRUE TRUE FALSE FALSE FALSE

#> [15] TRUE TRUE TRUE TRUE TRUE TRUE

# The values of the duplicated entries

# Note that '6' appears in the original vector three times, and so it has two

# entries here.

x[duplicated(x)]

#> [1] 10 3 11 8 10 8 5 6 6

# Duplicated entries, without repeats

unique(x[duplicated(x)])

#> [1] 10 3 11 8 5 6

# The original vector with all duplicates removed. These do the same:

unique(x)

#> [1] 14 11 8 4 12 5 10 3 6 0 16

x[!duplicated(x)]

#> [1] 14 11 8 4 12 5 10 3 6 0 16

With data frames:

*# A sample data frame:*

df <- read.table(header=TRUE, text='

label value

A 4

B 3

C 6

B 3

B 1

A 2

A 4

A 4

')

*# Is each row a repeat?*

duplicated(df)

*#> [1] FALSE FALSE FALSE TRUE FALSE FALSE TRUE TRUE*

*# Show the repeat entries*

df[duplicated(df),]

*#> label value*

*#> 4 B 3*

*#> 7 A 4*

*#> 8 A 4*

*# Show unique repeat entries (row names may differ, but values are the same)*

unique(df[duplicated(df),])

*#> label value*

*#> 4 B 3*

*#> 7 A 4*

*# Original data with repeats removed. These do the same:*

unique(df)

*#> label value*

*#> 1 A 4*

*#> 2 B 3*

*#> 3 C 6*

*#> 5 B 1*

*#> 6 A 2*

df[!duplicated(df),]

*#> label value*

*#> 1 A 4*

*#> 2 B 3*

*#> 3 C 6*

*#> 5 B 1*

*#> 6 A 2*

# Mapping vector values

## Problem

You want to change all instances of value x to value y in a vector.

## Solution

# Create some example data

str <- c("alpha", "beta", "gamma")

num <- c(1, 2, 3)

The easiest way is to use revalue() or mapvalues() from the plyr package:

library(plyr)

revalue(str, c("beta"="two", "gamma"="three"))

#> [1] "alpha" "two" "three"

mapvalues(str, from = c("beta", "gamma"), to = c("two", "three"))

#> [1] "alpha" "two" "three"

# For numeric vectors, revalue() won't work, since it uses a named vector, and

# the names are always strings, not numbers. mapvalues() will work, though:

mapvalues(num, from = c(2, 3), to = c(5, 6))

#> [1] 1 5 6

If you don’t want to rely on plyr, you can do the following with R’s built-in functions. Note that these methods will modify the vectors directly; that is, you don’t have to save the result back into the variable.

# Rename by name: change "beta" to "two"

str[str=="beta"] <- "two"

str

#> [1] "alpha" "two" "gamma"

num[num==2] <- 5

num

#> [1] 1 5 3

It’s also possible to use R’s string search-and-replace functions to remap values in character vectors. Note that the ^ and $ surrounding alpha are there to ensure that the entire string matches. Without them, if there were a value named alphabet, it would also match, and the replacement would be onebet.

str <- c("alpha", "beta", "gamma")

sub("^alpha$", "one", str)

#> [1] "one" "beta" "gamma"

# Across all columns, replace all instances of "a" with "X"

gsub("a", "X", str)

#> [1] "XlphX" "betX" "gXmmX"

# gsub() replaces all instances of the pattern in each element

# sub() replaces only the first instance in each element

# Adding and removing columns from a data frame

## Problem

You want to add or remove columns from a data frame.

## Solution

There are many different ways of adding and removing columns from a data frame.

data <- read.table(header=TRUE, text='

id weight

1 20

2 27

3 24

')

*# Ways to add a column*

data$size <- c("small", "large", "medium")

data[["size"]] <- c("small", "large", "medium")

data[,"size"] <- c("small", "large", "medium")

data$size <- 0 *# Use the same value (0) for all rows*

*# Ways to remove the column*

data$size <- NULL

data[["size"]] <- NULL

data[,"size"] <- NULL

data[[3]] <- NULL

data[,3] <- NULL

data <- subset(data, select=-size)

# Reordering the columns in a data frame

## Problem

You want to do reorder the columns in a data frame.

## Solution

*# A sample data frame*

data <- read.table(header=TRUE, text='

id weight size

1 20 small

2 27 large

3 24 medium

')

*# Reorder by column number*

data[c(1,3,2)]

*#> id size weight*

*#> 1 1 small 20*

*#> 2 2 large 27*

*#> 3 3 medium 24*

*# To actually change `data`, you need to save it back into `data`:*

*# data <- data[c(1,3,2)]*

*# Reorder by column name*

data[c("size", "id", "weight")]

*#> size id weight*

*#> 1 small 1 20*

*#> 2 large 2 27*

*#> 3 medium 3 24*

The above examples index into the data frame by treating it as a list (a data frame is essentially a list of vectors). You can also use matrix-style indexing, as in data[row, col], where row is left blank.

data[, c(1,3,2)]

#> id size weight

#> 1 1 small 20

#> 2 2 large 27

#> 3 3 medium 24

The drawback to matrix indexing is that it gives different results when you specify just one column. In these cases, the returned object is a *vector*, not a data frame. Because the returned data type isn’t always consistent with matrix indexing, it’s generally safer to use list-style indexing, or the drop=FALSE option:

*# List-style indexing of one column*

data[2]

*#> weight*

*#> 1 20*

*#> 2 27*

*#> 3 24*

*# Matrix-style indexing of one column - drops dimension to become a vector*

data[,2]

*#> [1] 20 27 24*

*# Matrix-style indexing with drop=FALSE - preserves dimension to remain data frame*

data[, 2, drop=FALSE]

*#> weight*

*#> 1 20*

*#> 2 27*

*#> 3 24*

# Merging data frames

## Problem

You want to merge two data frames on a given column from each (like a **join** in SQL).

## Solution

*# Make a data frame mapping story numbers to titles*

stories <- read.table(header=TRUE, text='

storyid title

1 lions

2 tigers

3 bears

')

*# Make another data frame with the data and story numbers (no titles)*

data <- read.table(header=TRUE, text='

subject storyid rating

1 1 6.7

1 2 4.5

1 3 3.7

2 2 3.3

2 3 4.1

2 1 5.2

')

*# Merge the two data frames*

merge(stories, data, "storyid")

*#> storyid title subject rating*

*#> 1 1 lions 1 6.7*

*#> 2 1 lions 2 5.2*

*#> 3 2 tigers 1 4.5*

*#> 4 2 tigers 2 3.3*

*#> 5 3 bears 1 3.7*

*#> 6 3 bears 2 4.1*

If the two data frames have different names for the columns you want to match on, the names can be specified:

*# In this case, the column is named 'id' instead of storyid*

stories2 <- read.table(header=TRUE, text='

id title

1 lions

2 tigers

3 bears

')

*# Merge on stories2$id and data$storyid.*

merge(x=stories2, y=data, by.x="id", by.y="storyid")

*#> id title subject rating*

*#> 1 1 lions 1 6.7*

*#> 2 1 lions 2 5.2*

*#> 3 2 tigers 1 4.5*

*#> 4 2 tigers 2 3.3*

*#> 5 3 bears 1 3.7*

*#> 6 3 bears 2 4.1*

*# Note that the column name is inherited from the first data frame (x=stories2).*

It is possible to merge on multiple columns:

*# Make up more data*

animals <- read.table(header=T, text='

size type name

small cat lynx

big cat tiger

small dog chihuahua

big dog "great dane"

')

observations <- read.table(header=T, text='

number size type

1 big cat

2 small dog

3 small dog

4 big dog

')

merge(observations, animals, c("size","type"))

*#> size type number name*

*#> 1 big cat 1 tiger*

*#> 2 big dog 4 great dane*

*#> 3 small dog 2 chihuahua*

*#> 4 small dog 3 chihuahua*

# Summarizing data

* [Problem](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#problem)
* [Solution](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#solution)
  + [Using ddply](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#using-ddply)
    - [Handling missing data](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#handling-missing-data)
    - [A function for mean, count, standard deviation, standard error of the mean, and confidence interval](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#a-function-for-mean-count-standard-deviation-standard-error-of-the-mean-and-confidence-interval)
    - [Filling empty combinations with zeros](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#filling-empty-combinations-with-zeros)
  + [Using summaryBy](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#using-summaryby)
    - [Handling missing data](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#handling-missing-data-1)
    - [A function for mean, count, standard deviation, standard error of the mean, and confidence interval](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#a-function-for-mean-count-standard-deviation-standard-error-of-the-mean-and-confidence-interval-1)
    - [Filling empty combinations with zeros](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#filling-empty-combinations-with-zeros-1)
  + [Using aggregate](http://www.cookbook-r.com/Manipulating_data/Summarizing_data/#using-aggregate)

## Problem

You want to do summarize your data (with mean, standard deviation, etc.), broken down by group.

## Solution

There are three ways described here to group data based on some specified variables, and apply a summary function (like mean, standard deviation, etc.) to each group.

* The ddply() function. It is the easiest to use, though it requires the plyr package. This is probably what you want to use.
* The summarizeBy() function. It is easier to use, though it requires the doBy package.
* The aggregate() function. It is more difficult to use but is included in the base install of R.

Suppose you have this data and want to find the N, mean of **change**, standard deviation, and standard error of the mean for each group, where the groups are specified by each combination of sex and condition: ***F-placebo***, ***F-aspirin***, ***M-placebo***, and ***M-aspirin***.

data <- read.table(header=TRUE, text='

subject sex condition before after change

1 F placebo 10.1 6.9 -3.2

2 F placebo 6.3 4.2 -2.1

3 M aspirin 12.4 6.3 -6.1

4 F placebo 8.1 6.1 -2.0

5 M aspirin 15.2 9.9 -5.3

6 F aspirin 10.9 7.0 -3.9

7 F aspirin 11.6 8.5 -3.1

8 M aspirin 9.5 3.0 -6.5

9 F placebo 11.5 9.0 -2.5

10 M placebo 11.9 11.0 -0.9

11 F aspirin 11.4 8.0 -3.4

12 M aspirin 10.0 4.4 -5.6

13 M aspirin 12.5 5.4 -7.1

14 M placebo 10.6 10.6 0.0

15 M aspirin 9.1 4.3 -4.8

16 F placebo 12.1 10.2 -1.9

17 F placebo 11.0 8.8 -2.2

18 F placebo 11.9 10.2 -1.7

19 M aspirin 9.1 3.6 -5.5

20 M placebo 13.5 12.4 -1.1

21 M aspirin 12.0 7.5 -4.5

22 F placebo 9.1 7.6 -1.5

23 M placebo 9.9 8.0 -1.9

24 F placebo 7.6 5.2 -2.4

25 F placebo 11.8 9.7 -2.1

26 F placebo 11.8 10.7 -1.1

27 F aspirin 10.1 7.9 -2.2

28 M aspirin 11.6 8.3 -3.3

29 F aspirin 11.3 6.8 -4.5

30 F placebo 10.3 8.3 -2.0

')

### Using ddply

library(plyr)

*# Run the functions length, mean, and sd on the value of "change" for each group,*

*# broken down by sex + condition*

cdata <- ddply(data, c("sex", "condition"), summarise,

N = length(change),

mean = mean(change),

sd = sd(change),

se = sd / sqrt(N)

)

cdata

*#> sex condition N mean sd se*

*#> 1 F aspirin 5 -3.420000 0.8642916 0.3865230*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867*

*#> 3 M aspirin 9 -5.411111 1.1307569 0.3769190*

*#> 4 M placebo 4 -0.975000 0.7804913 0.3902456*

#### Handling missing data

If there are NA’s in the data, you need to pass the flag na.rm=TRUE to each of the functions. length() doesn’t take na.rm as an option, so one way to work around it is to use sum(!is.na(...)) to count how many non-NA’s there are.

*# Put some NA's in the data*

dataNA <- data

dataNA$change[11:14] <- NA

cdata <- ddply(dataNA, c("sex", "condition"), summarise,

N = sum(!is.na(change)),

mean = mean(change, na.rm=TRUE),

sd = sd(change, na.rm=TRUE),

se = sd / sqrt(N)

)

cdata

*#> sex condition N mean sd se*

*#> 1 F aspirin 4 -3.425000 0.9979145 0.4989572*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867*

*#> 3 M aspirin 7 -5.142857 1.0674848 0.4034713*

*#> 4 M placebo 3 -1.300000 0.5291503 0.3055050*

#### A function for mean, count, standard deviation, standard error of the mean, and confidence interval

Instead of manually specifying all the values you want and then calculating the standard error, as shown above, this function will handle all of those details. It will do all the things described here:

* Find the mean, standard deviation, and count (N)
* Find the standard error of the mean (***again, this may not be what you want if you are collapsing over a within-subject variable.*** See [../../Graphs/Plotting means and error bars (ggplot2)](http://www.cookbook-r.com/Graphs/Plotting_means_and_error_bars_(ggplot2)) for information on how to make error bars for graphs with within-subjects variables.)
* Find a 95% confidence interval (or other value, if desired)
* Rename the columns so that the resulting data frame is easier to work with

To use, put this function in your code and call it as demonstrated below.

*## Summarizes data.*

*## Gives count, mean, standard deviation, standard error of the mean, and confidence interval (default 95%).*

*## data: a data frame.*

*## measurevar: the name of a column that contains the variable to be summariezed*

*## groupvars: a vector containing names of columns that contain grouping variables*

*## na.rm: a boolean that indicates whether to ignore NA's*

*## conf.interval: the percent range of the confidence interval (default is 95%)*

summarySE <- function(data=NULL, measurevar, groupvars=NULL, na.rm=FALSE,

conf.interval=.95, .drop=TRUE) {

library(plyr)

*# New version of length which can handle NA's: if na.rm==T, don't count them*

length2 <- function (x, na.rm=FALSE) {

if (na.rm) sum(!is.na(x))

else length(x)

}

*# This does the summary. For each group's data frame, return a vector with*

*# N, mean, and sd*

datac <- ddply(data, groupvars, .drop=.drop,

.fun = function(xx, col) {

c(N = length2(xx[[col]], na.rm=na.rm),

mean = mean (xx[[col]], na.rm=na.rm),

sd = sd (xx[[col]], na.rm=na.rm)

)

},

measurevar

)

*# Rename the "mean" column*

datac <- rename(datac, c("mean" = measurevar))

datac$se <- datac$sd / sqrt(datac$N) *# Calculate standard error of the mean*

*# Confidence interval multiplier for standard error*

*# Calculate t-statistic for confidence interval:*

*# e.g., if conf.interval is .95, use .975 (above/below), and use df=N-1*

ciMult <- qt(conf.interval/2 + .5, datac$N-1)

datac$ci <- datac$se \* ciMult

return(datac)

}

Example usage (with 95% confidence interval). Instead of doing all the steps manually, as done previously, the summarySE function does it all in one step:

summarySE(data, measurevar="change", groupvars=c("sex", "condition"))

*#> sex condition N change sd se ci*

*#> 1 F aspirin 5 -3.420000 0.8642916 0.3865230 1.0731598*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867 0.3334201*

*#> 3 M aspirin 9 -5.411111 1.1307569 0.3769190 0.8691767*

*#> 4 M placebo 4 -0.975000 0.7804913 0.3902456 1.2419358*

*# With a data set with NA's, use na.rm=TRUE*

summarySE(dataNA, measurevar="change", groupvars=c("sex", "condition"), na.rm=TRUE)

*#> sex condition N change sd se ci*

*#> 1 F aspirin 4 -3.425000 0.9979145 0.4989572 1.5879046*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867 0.3334201*

*#> 3 M aspirin 7 -5.142857 1.0674848 0.4034713 0.9872588*

*#> 4 M placebo 3 -1.300000 0.5291503 0.3055050 1.3144821*

#### Filling empty combinations with zeros

Sometimes there will be empty combinations of factors in the summary data frame – that is, combinations of factors that are possible, but don’t actually occur in the original data frame. It is often useful to automatically fill in those combinations in the summary data frame with NA’s. To do this, set .drop=FALSE in the call to ddply or summarySE.

Example usage:

*# First remove some all Male+Placebo entries from the data*

dataSub <- subset(data, !(sex=="M" & condition=="placebo"))

*# If we summarize the data, there will be a missing row for Male+Placebo,*

*# since there were no cases with this combination.*

summarySE(dataSub, measurevar="change", groupvars=c("sex", "condition"))

*#> sex condition N change sd se ci*

*#> 1 F aspirin 5 -3.420000 0.8642916 0.3865230 1.0731598*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867 0.3334201*

*#> 3 M aspirin 9 -5.411111 1.1307569 0.3769190 0.8691767*

*# Set .drop=FALSE to NOT drop those combinations*

summarySE(dataSub, measurevar="change", groupvars=c("sex", "condition"), .drop=FALSE)

*#> Warning in qt(conf.interval/2 + 0.5, datac$N - 1): NaNs produced*

*#> sex condition N change sd se ci*

*#> 1 F aspirin 5 -3.420000 0.8642916 0.3865230 1.0731598*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867 0.3334201*

*#> 3 M aspirin 9 -5.411111 1.1307569 0.3769190 0.8691767*

*#> 4 M placebo 0 NaN NA NA NA*

### Using summaryBy

To collapse the data using the summarizeBy() function:

library(doBy)

*# Run the functions length, mean, and sd on the value of "change" for each group,*

*# broken down by sex + condition*

cdata <- summaryBy(change ~ sex + condition, data=data, FUN=c(length,mean,sd))

cdata

*#> sex condition change.length change.mean change.sd*

*#> 1 F aspirin 5 -3.420000 0.8642916*

*#> 2 F placebo 12 -2.058333 0.5247655*

*#> 3 M aspirin 9 -5.411111 1.1307569*

*#> 4 M placebo 4 -0.975000 0.7804913*

*# Rename column change.length to just N*

names(cdata)[names(cdata)=="change.length"] <- "N"

*# Calculate standard error of the mean*

cdata$change.se <- cdata$change.sd / sqrt(cdata$N)

cdata

*#> sex condition N change.mean change.sd change.se*

*#> 1 F aspirin 5 -3.420000 0.8642916 0.3865230*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867*

*#> 3 M aspirin 9 -5.411111 1.1307569 0.3769190*

*#> 4 M placebo 4 -0.975000 0.7804913 0.3902456*

***Note that if you have any within-subjects variables, these standard error values may not be useful for comparing groups.*** See [../../Graphs/Plotting means and error bars (ggplot2)](http://www.cookbook-r.com/Graphs/Plotting_means_and_error_bars_(ggplot2)) for information on how to make error bars for graphs with within-subjects variables.

#### Handling missing data

If there are NA’s in the data, you need to pass the flag na.rm=TRUE to the functions. Normally you could pass it to summaryBy() and it would get passed to each of the functions called, but length() does not recognize it and so it won’t work. One way around it is to define a new length function that handles the NA’s.

*# New version of length which can handle NA's: if na.rm==T, don't count them*

length2 <- function (x, na.rm=FALSE) {

if (na.rm) sum(!is.na(x))

else length(x)

}

*# Put some NA's in the data*

dataNA <- data

dataNA$change[11:14] <- NA

cdataNA <- summaryBy(change ~ sex + condition, data=dataNA,

FUN=c(length2, mean, sd), na.rm=TRUE)

cdataNA

*#> sex condition change.length2 change.mean change.sd*

*#> 1 F aspirin 4 -3.425000 0.9979145*

*#> 2 F placebo 12 -2.058333 0.5247655*

*#> 3 M aspirin 7 -5.142857 1.0674848*

*#> 4 M placebo 3 -1.300000 0.5291503*

*# Now, do the same as before*

#### A function for mean, count, standard deviation, standard error of the mean, and confidence interval

Instead of manually specifying all the values you want and then calculating the standard error, as shown above, this function will handle all of those details. It will do all the things described here:

* Find the mean, standard deviation, and count (N)
* Find the standard error of the mean (***again, this may not be what you want if you are collapsing over a within-subject variable.*** See [../../Graphs/Plotting means and error bars (ggplot2)](http://www.cookbook-r.com/Graphs/Plotting_means_and_error_bars_(ggplot2)) for information on how to make error bars for graphs with within-subjects variables.)
* Find a 95% confidence interval (or other value, if desired)
* Rename the columns so that the resulting data frame is easier to work with

To use, put this function in your code and call it as demonstrated below.

*## Summarizes data.*

*## Gives count, mean, standard deviation, standard error of the mean, and confidence*

*## interval (default 95%).*

*## data: a data frame.*

*## measurevar: the name of a column that contains the variable to be summariezed*

*## groupvars: a vector containing names of columns that contain grouping variables*

*## na.rm: a boolean that indicates whether to ignore NA's*

*## conf.interval: the percent range of the confidence interval (default is 95%)*

summarySE <- function(data=NULL, measurevar, groupvars=NULL, na.rm=FALSE, conf.interval=.95) {

library(doBy)

*# New version of length which can handle NA's: if na.rm==T, don't count them*

length2 <- function (x, na.rm=FALSE) {

if (na.rm) sum(!is.na(x))

else length(x)

}

*# Collapse the data*

formula <- as.formula(paste(measurevar, paste(groupvars, collapse=" + "), sep=" ~ "))

datac <- summaryBy(formula, data=data, FUN=c(length2,mean,sd), na.rm=na.rm)

*# Rename columns*

names(datac)[ names(datac) == paste(measurevar, ".mean", sep="") ] <- measurevar

names(datac)[ names(datac) == paste(measurevar, ".sd", sep="") ] <- "sd"

names(datac)[ names(datac) == paste(measurevar, ".length2", sep="") ] <- "N"

datac$se <- datac$sd / sqrt(datac$N) *# Calculate standard error of the mean*

*# Confidence interval multiplier for standard error*

*# Calculate t-statistic for confidence interval:*

*# e.g., if conf.interval is .95, use .975 (above/below), and use df=N-1*

ciMult <- qt(conf.interval/2 + .5, datac$N-1)

datac$ci <- datac$se \* ciMult

return(datac)

}

Example usage (with 95% confidence interval). Instead of doing all the steps manually, as done previously, the summarySE function does it all in one step:

summarySE(data, measurevar="change", groupvars=c("sex","condition"))

*#> sex condition N change sd se ci*

*#> 1 F aspirin 5 -3.420000 0.8642916 0.3865230 1.0731598*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867 0.3334201*

*#> 3 M aspirin 9 -5.411111 1.1307569 0.3769190 0.8691767*

*#> 4 M placebo 4 -0.975000 0.7804913 0.3902456 1.2419358*

*# With a data set with NA's, use na.rm=TRUE*

summarySE(dataNA, measurevar="change", groupvars=c("sex","condition"), na.rm=TRUE)

*#> sex condition N change sd se ci*

*#> 1 F aspirin 4 -3.425000 0.9979145 0.4989572 1.5879046*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867 0.3334201*

*#> 3 M aspirin 7 -5.142857 1.0674848 0.4034713 0.9872588*

*#> 4 M placebo 3 -1.300000 0.5291503 0.3055050 1.3144821*

#### Filling empty combinations with zeros

Sometimes there will be empty combinations of factors in the summary data frame – that is, combinations of factors that are possible, but don’t actually occur in the original data frame. It is often useful to automatically fill in those combinations in the summary data frame with zeros.

This function will fill in those missing combinations with zeros:

fillMissingCombs <- function(df, factors, measures) {

*# Make a list of the combinations of factor levels*

levelList <- list()

for (f in factors) { levelList[[f]] <- levels(df[,f]) }

fullFactors <- expand.grid(levelList)

dfFull <- merge(fullFactors, df, all.x=TRUE)

*# Wherever there is an NA in the measure vars, replace with 0*

for (m in measures) {

dfFull[is.na(dfFull[,m]), m] <- 0

}

return(dfFull)

}

Example usage:

*# First remove some all Male+Placebo entries from the data*

dataSub <- subset(data, !(sex=="M" & condition=="placebo"))

*# If we summarize the data, there will be a missing row for Male+Placebo,*

*# since there were no cases with this combination.*

cdataSub <- summarySE(dataSub, measurevar="change", groupvars=c("sex", "condition"))

cdataSub

*#> sex condition N change sd se ci*

*#> 1 F aspirin 5 -3.420000 0.8642916 0.3865230 1.0731598*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867 0.3334201*

*#> 3 M aspirin 9 -5.411111 1.1307569 0.3769190 0.8691767*

*# This will fill in the missing combinations with zeros*

fillMissingCombs(cdataSub, factors=c("sex","condition"), measures=c("N","change","sd","se","ci"))

*#> sex condition N change sd se ci*

*#> 1 F aspirin 5 -3.420000 0.8642916 0.3865230 1.0731598*

*#> 2 F placebo 12 -2.058333 0.5247655 0.1514867 0.3334201*

*#> 3 M aspirin 9 -5.411111 1.1307569 0.3769190 0.8691767*

*#> 4 M placebo 0 0.000000 0.0000000 0.0000000 0.0000000*

### Using aggregate

The aggregate function is more difficult to use, but it is included in the base R installation and does not require the installation of another package.

*# Get a count of number of subjects in each category (sex\*condition)*

cdata <- aggregate(data["subject"], by=data[c("sex","condition")], FUN=length)

cdata

*#> sex condition subject*

*#> 1 F aspirin 5*

*#> 2 M aspirin 9*

*#> 3 F placebo 12*

*#> 4 M placebo 4*

*# Rename "subject" column to "N"*

names(cdata)[names(cdata)=="subject"] <- "N"

cdata

*#> sex condition N*

*#> 1 F aspirin 5*

*#> 2 M aspirin 9*

*#> 3 F placebo 12*

*#> 4 M placebo 4*

*# Sort by sex first*

cdata <- cdata[order(cdata$sex),]

cdata

*#> sex condition N*

*#> 1 F aspirin 5*

*#> 3 F placebo 12*

*#> 2 M aspirin 9*

*#> 4 M placebo 4*

*# We also keep the \_\_before\_\_ and \_\_after\_\_ columns:*

*# Get the average effect size by sex and condition*

cdata.means <- aggregate(data[c("before","after","change")],

by = data[c("sex","condition")], FUN=mean)

cdata.means

*#> sex condition before after change*

*#> 1 F aspirin 11.06000 7.640000 -3.420000*

*#> 2 M aspirin 11.26667 5.855556 -5.411111*

*#> 3 F placebo 10.13333 8.075000 -2.058333*

*#> 4 M placebo 11.47500 10.500000 -0.975000*

*# Merge the data frames*

cdata <- merge(cdata, cdata.means)

cdata

*#> sex condition N before after change*

*#> 1 F aspirin 5 11.06000 7.640000 -3.420000*

*#> 2 F placebo 12 10.13333 8.075000 -2.058333*

*#> 3 M aspirin 9 11.26667 5.855556 -5.411111*

*#> 4 M placebo 4 11.47500 10.500000 -0.975000*

*# Get the sample (n-1) standard deviation for "change"*

cdata.sd <- aggregate(data["change"],

by = data[c("sex","condition")], FUN=sd)

*# Rename the column to change.sd*

names(cdata.sd)[names(cdata.sd)=="change"] <- "change.sd"

cdata.sd

*#> sex condition change.sd*

*#> 1 F aspirin 0.8642916*

*#> 2 M aspirin 1.1307569*

*#> 3 F placebo 0.5247655*

*#> 4 M placebo 0.7804913*

*# Merge*

cdata <- merge(cdata, cdata.sd)

cdata

*#> sex condition N before after change change.sd*

*#> 1 F aspirin 5 11.06000 7.640000 -3.420000 0.8642916*

*#> 2 F placebo 12 10.13333 8.075000 -2.058333 0.5247655*

*#> 3 M aspirin 9 11.26667 5.855556 -5.411111 1.1307569*

*#> 4 M placebo 4 11.47500 10.500000 -0.975000 0.7804913*

*# Calculate standard error of the mean*

cdata$change.se <- cdata$change.sd / sqrt(cdata$N)

cdata

*#> sex condition N before after change change.sd change.se*

*#> 1 F aspirin 5 11.06000 7.640000 -3.420000 0.8642916 0.3865230*

*#> 2 F placebo 12 10.13333 8.075000 -2.058333 0.5247655 0.1514867*

*#> 3 M aspirin 9 11.26667 5.855556 -5.411111 1.1307569 0.3769190*

*#> 4 M placebo 4 11.47500 10.500000 -0.975000 0.7804913 0.3902456*

If you have NA’s in your data and wish to skip them, use na.rm=TRUE:

cdata.means <- aggregate(data[c("before","after","change")],

by = data[c("sex","condition")],

FUN=mean, na.rm=TRUE)

cdata.means

#> sex condition before after change

#> 1 F aspirin 11.06000 7.640000 -3.420000

#> 2 M aspirin 11.26667 5.855556 -5.411111

#> 3 F placebo 10.13333 8.075000 -2.058333

#> 4 M placebo 11.47500 10.500000 -0.975000

# Filling in NAs with last non-NA value

## Problem

You want to replace NA’s in a vector or factor with the last non-NA value.

## Solution

This code shows how to fill gaps in a vector. If you need to do this repeatedly, see the function below. The function also can fill in leading NA’s with the first good value and handle factors properly.

# Sample data

x <- c(NA,NA, "A","A", "B","B","B", NA,NA, "C", NA,NA,NA, "A","A","B", NA,NA)

goodIdx <- !is.na(x)

goodIdx

#> [1] FALSE FALSE TRUE TRUE TRUE TRUE TRUE FALSE FALSE TRUE FALSE FALSE FALSE TRUE

#> [15] TRUE TRUE FALSE FALSE

# These are the non-NA values from x only

# Add a leading NA for later use when we index into this vector

goodVals <- c(NA, x[goodIdx])

goodVals

#> [1] NA "A" "A" "B" "B" "B" "C" "A" "A" "B"

# Fill the indices of the output vector with the indices pulled from

# these offsets of goodVals. Add 1 to avoid indexing to zero.

fillIdx <- cumsum(goodIdx)+1

fillIdx

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

# The original vector with gaps filled

goodVals[fillIdx]

#> [1] NA NA "A" "A" "B" "B" "B" "B" "B" "C" "C" "C" "C" "A" "A" "B" "B" "B"

### A function for filling gaps

This function does the same as the code above. It can also fill leading NA’s with the first good value, and handle factors properly.

fillNAgaps <- function(x, firstBack=FALSE) {

## NA's in a vector or factor are replaced with last non-NA values

## If firstBack is TRUE, it will fill in leading NA's with the first

## non-NA value. If FALSE, it will not change leading NA's.

# If it's a factor, store the level labels and convert to integer

lvls <- NULL

if (is.factor(x)) {

lvls <- levels(x)

x <- as.integer(x)

}

goodIdx <- !is.na(x)

# These are the non-NA values from x only

# Add a leading NA or take the first good value, depending on firstBack

if (firstBack) goodVals <- c(x[goodIdx][1], x[goodIdx])

else goodVals <- c(NA, x[goodIdx])

# Fill the indices of the output vector with the indices pulled from

# these offsets of goodVals. Add 1 to avoid indexing to zero.

fillIdx <- cumsum(goodIdx)+1

x <- goodVals[fillIdx]

# If it was originally a factor, convert it back

if (!is.null(lvls)) {

x <- factor(x, levels=seq\_along(lvls), labels=lvls)

}

x

}

# Sample data

x <- c(NA,NA, "A","A", "B","B","B", NA,NA, "C", NA,NA,NA, "A","A","B", NA,NA)

x

#> [1] NA NA "A" "A" "B" "B" "B" NA NA "C" NA NA NA "A" "A" "B" NA NA

fillNAgaps(x)

#> [1] NA NA "A" "A" "B" "B" "B" "B" "B" "C" "C" "C" "C" "A" "A" "B" "B" "B"

# Fill the leading NA's with the first good value

fillNAgaps(x, firstBack=TRUE)

#> [1] "A" "A" "A" "A" "B" "B" "B" "B" "B" "C" "C" "C" "C" "A" "A" "B" "B" "B"

# It also works on factors

y <- factor(x)

y

#> [1] <NA> <NA> A A B B B <NA> <NA> C <NA> <NA> <NA> A A B <NA>

#> [18] <NA>

#> Levels: A B C

fillNAgaps(y)

#> [1] <NA> <NA> A A B B B B B C C C C A A B B

#> [18] B

#> Levels: A B C

# Regression and correlation

* [Problem](http://www.cookbook-r.com/Statistical_analysis/Regression_and_correlation/#problem)
* [Solution](http://www.cookbook-r.com/Statistical_analysis/Regression_and_correlation/#solution)
  + [Correlation](http://www.cookbook-r.com/Statistical_analysis/Regression_and_correlation/#correlation)
  + [Correlation matrices (for multiple variables)](http://www.cookbook-r.com/Statistical_analysis/Regression_and_correlation/#correlation-matrices-for-multiple-variables)
  + [Linear regression](http://www.cookbook-r.com/Statistical_analysis/Regression_and_correlation/#linear-regression)
  + [Linear regression with multiple predictors](http://www.cookbook-r.com/Statistical_analysis/Regression_and_correlation/#linear-regression-with-multiple-predictors)
    - [Interactions](http://www.cookbook-r.com/Statistical_analysis/Regression_and_correlation/#interactions)

## Problem

You want to perform linear regressions and/or correlations.

## Solution

Some sample data to work with:

*# Make some data*

*# X increases (noisily)*

*# Z increases slowly*

*# Y is constructed so it is inversely related to xvar and positively related to xvar\*zvar*

set.seed(955)

xvar <- 1:20 + rnorm(20,sd=3)

zvar <- 1:20/4 + rnorm(20,sd=2)

yvar <- -2\*xvar + xvar\*zvar/5 + 3 + rnorm(20,sd=4)

*# Make a data frame with the variables*

dat <- data.frame(x=xvar, y=yvar, z=zvar)

*# Show first few rows*

head(dat)

*#> x y z*

*#> 1 -4.252354 4.5857688 1.89877152*

*#> 2 1.702318 -4.9027824 -0.82937359*

*#> 3 4.323054 -4.3076433 -1.31283495*

*#> 4 1.780628 0.2050367 -0.28479448*

*#> 5 11.537348 -29.7670502 -1.27303976*

*#> 6 6.672130 -10.1458220 -0.09459239*

### Correlation

*# Correlation coefficient*

cor(dat$x, dat$y)

*#> [1] -0.7695378*

### Correlation matrices (for multiple variables)

It is also possible to run correlations between many pairs of variables, using a matrix or data frame.

*# A correlation matrix of the variables*

cor(dat)

*#> x y z*

*#> x 1.0000000 -0.769537849 0.491698938*

*#> y -0.7695378 1.000000000 0.004172295*

*#> z 0.4916989 0.004172295 1.000000000*

*# Print with only two decimal places*

round(cor(dat), 2)

*#> x y z*

*#> x 1.00 -0.77 0.49*

*#> y -0.77 1.00 0.00*

*#> z 0.49 0.00 1.00*

To visualize a correlation matrix, see [../../Graphs/Correlation matrix](http://www.cookbook-r.com/Graphs/Correlation_matrix).

### Linear regression

Linear regressions, where dat$x is the predictor, and dat$y is the outcome. This can be done using two columns from a data frame, or with numeric vectors directly.

*# These two commands will have the same outcome:*

fit <- lm(y ~ x, data=dat) *# Using the columns x and y from the data frame*

fit <- lm(dat$y ~ dat$x) *# Using the vectors dat$x and dat$y*

fit

*#>*

*#> Call:*

*#> lm(formula = dat$y ~ dat$x)*

*#>*

*#> Coefficients:*

*#> (Intercept) dat$x*

*#> -0.2278 -1.1829*

*# This means that the predicted y = -0.2278 - 1.1829\*x*

*# Get more detailed information:*

summary(fit)

*#>*

*#> Call:*

*#> lm(formula = dat$y ~ dat$x)*

*#>*

*#> Residuals:*

*#> Min 1Q Median 3Q Max*

*#> -15.8922 -2.5114 0.2866 4.4646 9.3285*

*#>*

*#> Coefficients:*

*#> Estimate Std. Error t value Pr(>|t|)*

*#> (Intercept) -0.2278 2.6323 -0.087 0.932*

*#> dat$x -1.1829 0.2314 -5.113 7.28e-05 \*\*\**

*#> ---*

*#> Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1*

*#>*

*#> Residual standard error: 6.506 on 18 degrees of freedom*

*#> Multiple R-squared: 0.5922, Adjusted R-squared: 0.5695*

*#> F-statistic: 26.14 on 1 and 18 DF, p-value: 7.282e-05*

To visualize the data with regression lines, see [../../Graphs/Scatterplots (ggplot2)](http://www.cookbook-r.com/Graphs/Scatterplots_(ggplot2)) and [../../Graphs/Scatterplot](http://www.cookbook-r.com/Graphs/Scatterplot).

### Linear regression with multiple predictors

Linear regression with y as the outcome, and x and z as predictors.

Note that the formula specified below does **not** test for interactions between x and z.

*# These have the same result*

fit2 <- lm(y ~ x + z, data=dat) *# Using the columns x, y, and z from the data frame*

fit2 <- lm(dat$y ~ dat$x + dat$z) *# Using the vectors x, y, z*

fit2

*#>*

*#> Call:*

*#> lm(formula = dat$y ~ dat$x + dat$z)*

*#>*

*#> Coefficients:*

*#> (Intercept) dat$x dat$z*

*#> -1.382 -1.564 1.858*

summary(fit2)

*#>*

*#> Call:*

*#> lm(formula = dat$y ~ dat$x + dat$z)*

*#>*

*#> Residuals:*

*#> Min 1Q Median 3Q Max*

*#> -7.974 -3.187 -1.205 3.847 7.524*

*#>*

*#> Coefficients:*

*#> Estimate Std. Error t value Pr(>|t|)*

*#> (Intercept) -1.3816 1.9878 -0.695 0.49644*

*#> dat$x -1.5642 0.1984 -7.883 4.46e-07 \*\*\**

*#> dat$z 1.8578 0.4753 3.908 0.00113 \*\**

*#> ---*

*#> Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1*

*#>*

*#> Residual standard error: 4.859 on 17 degrees of freedom*

*#> Multiple R-squared: 0.7852, Adjusted R-squared: 0.7599*

*#> F-statistic: 31.07 on 2 and 17 DF, p-value: 2.1e-06*

#### Interactions

The topic of how to properly do multiple regression and test for interactions can be quite complex and is not covered here. Here we just fit a model with x, z, and the interaction between the two.

To model interactions between x and z, a x:z term must be added. Alternatively, the formula x\*z expands to x+z+x:z.

*# These are equivalent; the x\*z expands to x + z + x:z*

fit3 <- lm(y ~ x \* z, data=dat)

fit3 <- lm(y ~ x + z + x:z, data=dat)

fit3

*#>*

*#> Call:*

*#> lm(formula = y ~ x + z + x:z, data = dat)*

*#>*

*#> Coefficients:*

*#> (Intercept) x z x:z*

*#> 2.2820 -2.1311 -0.1068 0.2081*

summary(fit3)

*#>*

*#> Call:*

*#> lm(formula = y ~ x + z + x:z, data = dat)*

*#>*

*#> Residuals:*

*#> Min 1Q Median 3Q Max*

*#> -5.3045 -3.5998 0.3926 2.1376 8.3957*

*#>*

*#> Coefficients:*

*#> Estimate Std. Error t value Pr(>|t|)*

*#> (Intercept) 2.28204 2.20064 1.037 0.3152*

*#> x -2.13110 0.27406 -7.776 8e-07 \*\*\**

*#> z -0.10682 0.84820 -0.126 0.9013*

*#> x:z 0.20814 0.07874 2.643 0.0177 \**

*#> ---*

*#> Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1*

*#>*

*#> Residual standard error: 4.178 on 16 degrees of freedom*

*#> Multiple R-squared: 0.8505, Adjusted R-squared: 0.8225*

*#> F-statistic: 30.34 on 3 and 16 DF, p-value: 7.759e-07*

# Bar and line graphs (ggplot2)

* [Problem](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#problem)
* [Solution](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#solution)
  + [Basic graphs with discrete x-axis](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#basic-graphs-with-discrete-x-axis)
    - [Bar graphs of values](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#bar-graphs-of-values)
    - [Bar graphs of counts](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#bar-graphs-of-counts)
    - [Line graphs](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#line-graphs)
  + [Graphs with more variables](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#graphs-with-more-variables)
    - [Bar graphs](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#bar-graphs)
    - [Line graphs](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#line-graphs-1)
    - [Finished examples](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#finished-examples)
  + [With a numeric x-axis](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#with-a-numeric-x-axis)
    - [With x-axis treated as continuous](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#with-x-axis-treated-as-continuous)
    - [With x-axis treated as categorical](http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#with-x-axis-treated-as-categorical)

## Problem

You want to do make basic bar or line graphs.

## Solution

To make graphs with ggplot2, the data must be in a data frame, and in “long” (as opposed to wide) format. If your data needs to be restructured, see [this page](http://www.cookbook-r.com/Manipulating_data/Converting_data_between_wide_and_long_format) for more information.

### Basic graphs with discrete x-axis

With bar graphs, there are two different things that the heights of bars commonly represent:

* The *count* of cases for each group – typically, each x value represents one group. This is done with stat\_bin, which calculates the number of cases in each group (if x is discrete, then each x value is a group; if x is continuous, then all the data is automatically in one group, unless you specifiy grouping with group=xx).
* The *value* of a column in the data set. This is done with stat\_identity, which leaves the y values unchanged.

|  |  |  |
| --- | --- | --- |
| **x axis is** | **Height of bar represents** | **Common name** |
| ***Continuous*** | ***Count*** | Histogram |
| ***Discrete*** | ***Count*** | Bar graph |
| ***Continuous*** | ***Value*** | Bar graph |
| ***Discrete*** | ***Value*** | Bar graph |

In ggplot2, the default is to use stat\_bin, so that the bar height represents the count of cases.

##### Bar graphs of values

Here is some sample data (derived from the tips dataset in the reshape2 package):

dat <- data.frame(

time = factor(c("Lunch","Dinner"), levels=c("Lunch","Dinner")),

total\_bill = c(14.89, 17.23)

)

dat

*#> time total\_bill*

*#> 1 Lunch 14.89*

*#> 2 Dinner 17.23*

*# Load the ggplot2 package*

library(ggplot2)

In these examples, the height of the bar will represent the *value* in a column of the data frame. This is done by using stat="identity" instead of the default, stat="bin".

These are the variable mappings used here:

* time: x-axis and sometimes color fill
* total\_bill: y-axis

*# Very basic bar graph*

ggplot(data=dat, aes(x=time, y=total\_bill)) +

geom\_bar(stat="identity")

*# Map the time of day to different fill colors*

ggplot(data=dat, aes(x=time, y=total\_bill, fill=time)) +

geom\_bar(stat="identity")

*## This would have the same result as above*

*# ggplot(data=dat, aes(x=time, y=total\_bill)) +*

*# geom\_bar(aes(fill=time), stat="identity")*

*# Add a black outline*

ggplot(data=dat, aes(x=time, y=total\_bill, fill=time)) +

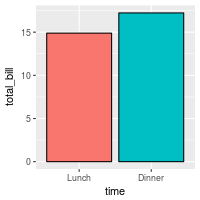
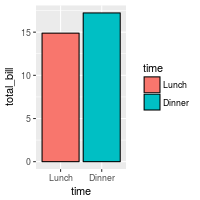
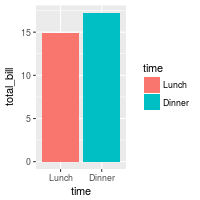
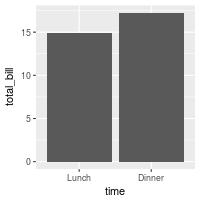
geom\_bar(colour="black", stat="identity")

*# No legend, since the information is redundant*

ggplot(data=dat, aes(x=time, y=total\_bill, fill=time)) +

geom\_bar(colour="black", stat="identity") +

guides(fill=FALSE)



The desired bar graph might look something like this:

# Add title, narrower bars, fill color, and change axis labels

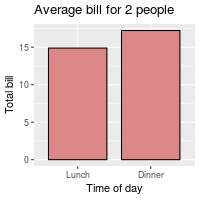
ggplot(data=dat, aes(x=time, y=total\_bill, fill=time)) +

geom\_bar(colour="black", fill="#DD8888", width=.8, stat="identity") +

guides(fill=FALSE) +

xlab("Time of day") + ylab("Total bill") +

ggtitle("Average bill for 2 people")



See [../Colors (ggplot2)](http://www.cookbook-r.com/Graphs/Colors_(ggplot2)) for more information on colors.

##### Bar graphs of counts

In these examples, the height of the bar will represent the *count* of cases. This is done by using stat="bin" (which is the default).

We’ll start with the tips data from the reshape2 package:

library(reshape2)

*# Look at fist several rows*

head(tips)

*#> total\_bill tip sex smoker day time size*

*#> 1 16.99 1.01 Female No Sun Dinner 2*

*#> 2 10.34 1.66 Male No Sun Dinner 3*

*#> 3 21.01 3.50 Male No Sun Dinner 3*

*#> 4 23.68 3.31 Male No Sun Dinner 2*

*#> 5 24.59 3.61 Female No Sun Dinner 4*

*#> 6 25.29 4.71 Male No Sun Dinner 4*

To get a bar graph of counts, don’t map a variable to y, and use stat="bin" (which is the default) instead of stat="identity":

*# Bar graph of counts*

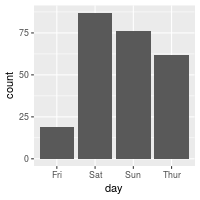
ggplot(data=tips, aes(x=day)) +

geom\_bar(stat="count")

*## Equivalent to this, since stat="bin" is the default:*

*# ggplot(data=tips, aes(x=day)) +*

*# geom\_bar()*



#### Line graphs

For line graphs, the data points must be grouped so that it knows which points to connect. In this case, it is simple – all points should be connected, so group=1. When more variables are used and multiple lines are drawn, the grouping for lines is usually done by variable (this is seen in later examples).

These are the variable mappings used here:

* time: x-axis
* total\_bill: y-axis

# Basic line graph

ggplot(data=dat, aes(x=time, y=total\_bill, group=1)) +

geom\_line()

## This would have the same result as above

# ggplot(data=dat, aes(x=time, y=total\_bill)) +

# geom\_line(aes(group=1))

# Add points

ggplot(data=dat, aes(x=time, y=total\_bill, group=1)) +

geom\_line() +

geom\_point()

# Change color of both line and points

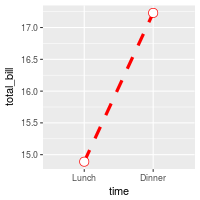
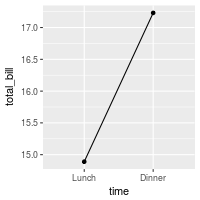
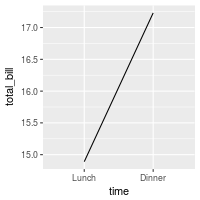
# Change line type and point type, and use thicker line and larger points

# Change points to circles with white fill

ggplot(data=dat, aes(x=time, y=total\_bill, group=1)) +

geom\_line(colour="red", linetype="dashed", size=1.5) +

geom\_point(colour="red", size=4, shape=21, fill="white")



The desired line graph might look something like this:

# Change the y-range to go from 0 to the maximum value in the total\_bill column,

# and change axis labels

ggplot(data=dat, aes(x=time, y=total\_bill, group=1)) +

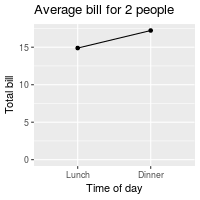
geom\_line() +

geom\_point() +

expand\_limits(y=0) +

xlab("Time of day") + ylab("Total bill") +

ggtitle("Average bill for 2 people")



See [../Colors (ggplot2)](http://www.cookbook-r.com/Graphs/Colors_(ggplot2)) for more information on colors, and [../Shapes and line types](http://www.cookbook-r.com/Graphs/Shapes_and_line_types) for information on shapes and line types.

### Graphs with more variables

This data will be used for the examples below:

dat1 <- data.frame(

sex = factor(c("Female","Female","Male","Male")),

time = factor(c("Lunch","Dinner","Lunch","Dinner"), levels=c("Lunch","Dinner")),

total\_bill = c(13.53, 16.81, 16.24, 17.42)

)

dat1

*#> sex time total\_bill*

*#> 1 Female Lunch 13.53*

*#> 2 Female Dinner 16.81*

*#> 3 Male Lunch 16.24*

*#> 4 Male Dinner 17.42*

This is derived from the tips dataset in the reshape2 package.

#### Bar graphs

These are the variable mappings used here:

* time: x-axis
* sex: color fill
* total\_bill: y-axis.

*# Stacked bar graph -- this is probably not what you want*

ggplot(data=dat1, aes(x=time, y=total\_bill, fill=sex)) +

geom\_bar(stat="identity")

*# Bar graph, time on x-axis, color fill grouped by sex -- use position\_dodge()*

ggplot(data=dat1, aes(x=time, y=total\_bill, fill=sex)) +

geom\_bar(stat="identity", position=position\_dodge())

ggplot(data=dat1, aes(x=time, y=total\_bill, fill=sex)) +

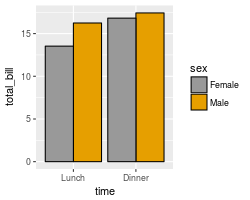
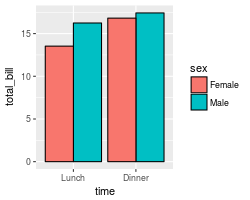
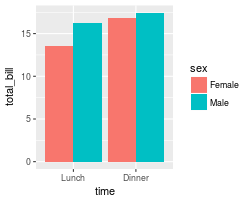
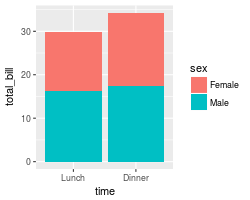
geom\_bar(stat="identity", position=position\_dodge(), colour="black")

*# Change colors*

ggplot(data=dat1, aes(x=time, y=total\_bill, fill=sex)) +

geom\_bar(stat="identity", position=position\_dodge(), colour="black") +

scale\_fill\_manual(values=c("#999999", "#E69F00"))

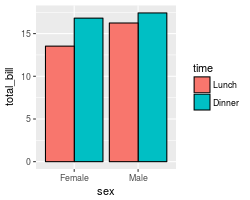


It’s easy to change which variable is mapped the x-axis and which is mapped to the fill.

*# Bar graph, time on x-axis, color fill grouped by sex -- use position\_dodge()*

ggplot(data=dat1, aes(x=sex, y=total\_bill, fill=time)) +

geom\_bar(stat="identity", position=position\_dodge(), colour="black")



See [../Colors (ggplot2)](http://www.cookbook-r.com/Graphs/Colors_(ggplot2)) for more information on colors.

#### Line graphs

These are the variable mappings used here:

* time: x-axis
* sex: line color
* total\_bill: y-axis.

To draw multiple lines, the points must be grouped by a variable; otherwise all points will be connected by a single line. In this case, we want them to be grouped by sex.

# Basic line graph with points

ggplot(data=dat1, aes(x=time, y=total\_bill, group=sex)) +

geom\_line() +

geom\_point()

# Map sex to color

ggplot(data=dat1, aes(x=time, y=total\_bill, group=sex, colour=sex)) +

geom\_line() +

geom\_point()

# Map sex to different point shape, and use larger points

ggplot(data=dat1, aes(x=time, y=total\_bill, group=sex, shape=sex)) +

geom\_line() +

geom\_point()

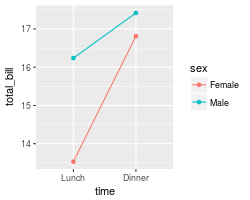
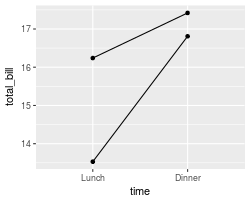
# Use thicker lines and larger points, and hollow white-filled points

ggplot(data=dat1, aes(x=time, y=total\_bill, group=sex, shape=sex)) +

geom\_line(size=1.5) +

geom\_point(size=3, fill="white") +

scale\_shape\_manual(values=c(22,21))

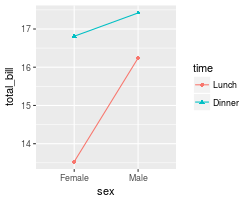


It’s easy to change which variable is mapped the x-axis and which is mapped to the color or shape.

ggplot(data=dat1, aes(x=sex, y=total\_bill, group=time, shape=time, color=time)) +

geom\_line() +

geom\_point()



See [../Colors (ggplot2)](http://www.cookbook-r.com/Graphs/Colors_(ggplot2)) for more information on colors, and [../Shapes and line types](http://www.cookbook-r.com/Graphs/Shapes_and_line_types) for information on shapes and line types.

#### Finished examples

The finished graphs might look like these:

# A bar graph

ggplot(data=dat1, aes(x=time, y=total\_bill, fill=sex)) +

geom\_bar(colour="black", stat="identity",

position=position\_dodge(),

size=.3) + # Thinner lines

scale\_fill\_hue(name="Sex of payer") + # Set legend title

xlab("Time of day") + ylab("Total bill") + # Set axis labels

ggtitle("Average bill for 2 people") + # Set title

theme\_bw()

# A line graph

ggplot(data=dat1, aes(x=time, y=total\_bill, group=sex, shape=sex, colour=sex)) +

geom\_line(aes(linetype=sex), size=1) + # Set linetype by sex

geom\_point(size=3, fill="white") + # Use larger points, fill with white

expand\_limits(y=0) + # Set y range to include 0

scale\_colour\_hue(name="Sex of payer", # Set legend title

l=30) + # Use darker colors (lightness=30)

scale\_shape\_manual(name="Sex of payer",

values=c(22,21)) + # Use points with a fill color

scale\_linetype\_discrete(name="Sex of payer") +

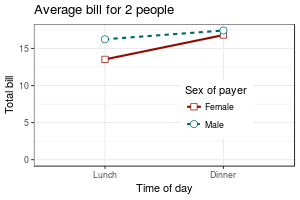
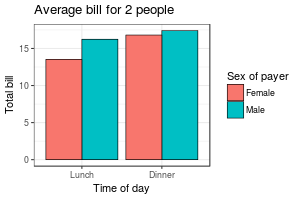
xlab("Time of day") + ylab("Total bill") + # Set axis labels

ggtitle("Average bill for 2 people") + # Set title

theme\_bw() +

theme(legend.position=c(.7, .4)) # Position legend inside

# This must go after theme\_bw



In the line graph, the reason that the legend title, “Sex of payer”, must be specified three times is so that there is only one legend. The issue is explained [here](http://www.cookbook-r.com/Graphs/Legends_(ggplot2)#With_lines_and_points).

### With a numeric x-axis

When the variable on the x-axis is numeric, it is sometimes useful to treat it as continuous, and sometimes useful to treat it as categorical. In this data set, the dose is a numeric variable with values 0.5, 1.0, and 2.0. It might be useful to treat these values as equal categories when making a graph.

datn <- read.table(header=TRUE, text='

supp dose length

OJ 0.5 13.23

OJ 1.0 22.70

OJ 2.0 26.06

VC 0.5 7.98

VC 1.0 16.77

VC 2.0 26.14

')

This is derived from the ToothGrowth dataset included with R.

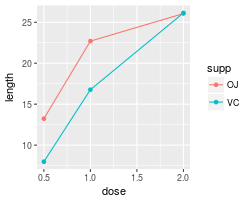
#### With x-axis treated as continuous

A simple graph might put dose on the x-axis as a numeric value. It is possible to make a line graph this way, but not a bar graph.

ggplot(data=datn, aes(x=dose, y=length, group=supp, colour=supp)) +

geom\_line() +

geom\_point()



#### With x-axis treated as categorical

If you wish to treat it as a categorical variable instead of a numeric one, it must be converted to a factor. This can be done by modifying the data frame, or by changing the specification of the graph.

*# Copy the data frame and convert dose to a factor*

datn2 <- datn

datn2$dose <- factor(datn2$dose)

ggplot(data=datn2, aes(x=dose, y=length, group=supp, colour=supp)) +

geom\_line() +

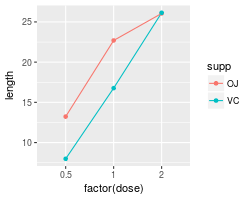
geom\_point()

*# Use the original data frame, but put factor() directly in the plot specification*

ggplot(data=datn, aes(x=factor(dose), y=length, group=supp, colour=supp)) +

geom\_line() +

geom\_point()



It is also possible to make a bar graph when the variable is treated as categorical rather than numeric.

# Use datn2 from above

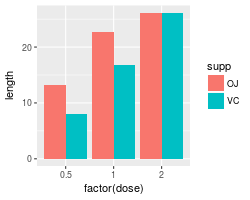
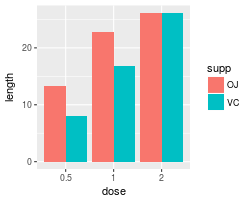
ggplot(data=datn2, aes(x=dose, y=length, fill=supp)) +

geom\_bar(stat="identity", position=position\_dodge())

# Use the original data frame, but put factor() directly in the plot specification

ggplot(data=datn, aes(x=factor(dose), y=length, fill=supp)) +

geom\_bar(stat="identity", position=position\_dodge())



**Quiz 1**

1. What function is used to determine the length of an object?

Length or length()

1. What is the function to remove an object?

Rm

1. R is free as in beer?

False

1. R is capable of connecting http services”?

true

1. Which of the following examples were used to load excel data into “R”

XLConnect

1. When calculating in “R” a mean what string would most correctly be used to remove “na” values?

na.rm= true

1. R is case sensitive?

true

1. R has a yearly licensing charge?

false

1. When installing R you should install which version if you are running a modern CPU.

64 bit version

**Quiz 2**

1. Attempting to calculate the min of a column in R will complete successfully if the column contains an empty row in R.

false

1. The \_\_\_ method is used to calculate the mode in R?

mode

1. The \_\_\_ method is used to calculate the variance in R?

var

1. When performing an analysis on the following data what would be the most appropriate for correcting the outlier in R?

0,1,2,3,10,100,15,30,31,29,25,-1,null,1000

Set the 1000 parameter to the mean

1. Attempting to calculate the variance of a column contains an empty row in R?

False

1. Attempting to calculate the max of a column in R will complete successfully if the column contains an empty row in R.

false

**Quiz 3**

1. Why would we connect R to a database

Connecting R to databases allows direct access to data for analytics

1. Which method would be used to connect MySQL and R?

An odbc driver

1. Which method would be used to connect JavaDB and R?

A jdbc driver

1. Which method would create a line graph in R?

plot

1. True or False it is possible to set NA value to the mode of the column in R?

True

1. When working with R the following logical statement can be used to determine if a field is missing data?

is.na

**Quiz 4**

1. In R when creating a plot which short type is used to specify a line graph?

I

1. When presenting data to users which of the following is important?

Charts should take advantage of color shades not different colors

1. Proper visualizations should make sure to emphasize important variables

True

1. In R which function is used to create a scatter plot?

stripchart

1. When creating a metric which of the following is important to remember?

Unstructured investigation may uncover important insights

1. Which would create a simple bar plot?

barplot(table(iris$Sepal.Length))

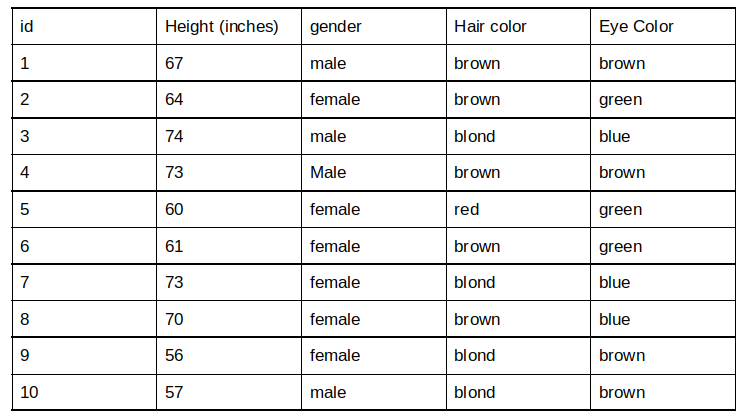
1. When creating a metric which of the following is important to remember?

A metrics should not cost more than the potential business benefit.

ISM 4403 Homework Week 2

### **Tasks:**

1. Create a new Excel spreadsheet from the following table.



**Paste your code here**

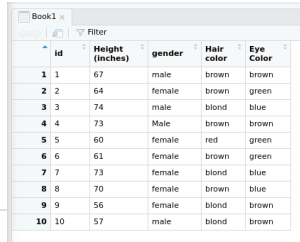
> library(readxl)

> Book1 <- read\_excel("Desktop/Book1.xlsx")

> View(Book1)

**End of Paste**

**Paste your results here**



**End of Paste**

1. Using the World Bank Development Indicators import and chart the United States, Canada and Mexico’s GDP per capita as demonstrated on <https://github.com/vincentarelbundock/WDI>/.

**Paste your resulting code here**

**> install.packages('WDI')**

The downloaded source packages are in

‘/tmp/RtmpDsrS4k/downloaded\_packages’

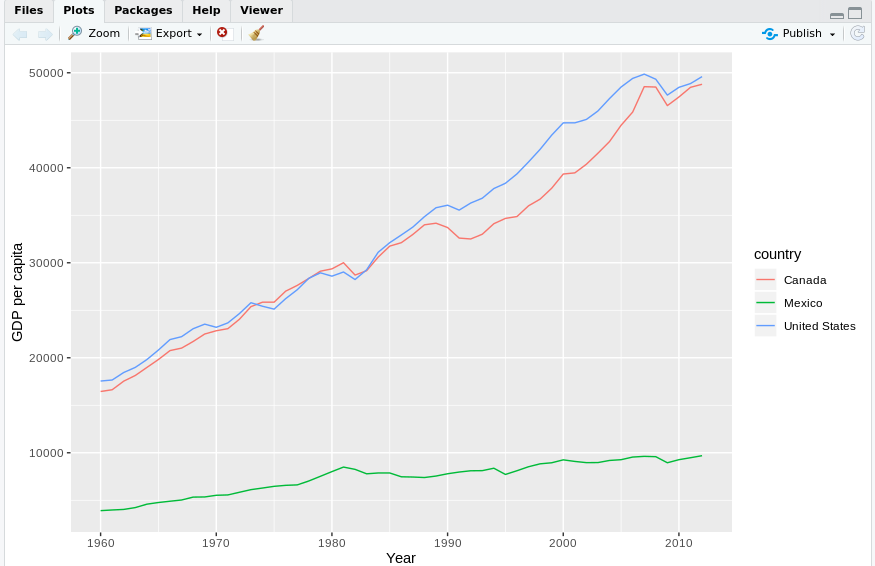
> dat=WDI::WDI(country = c('MX','CA','US'),indicator = 'NY.GDP.PCAP.KD',start = 1960,end = 2012)

> library(ggplot2)

> ggplot(dat, aes(year, NY.GDP.PCAP.KD, color=country)) + geom\_line() + xlab('Year') + ylab('GDP per capita')

**END OF Paste**

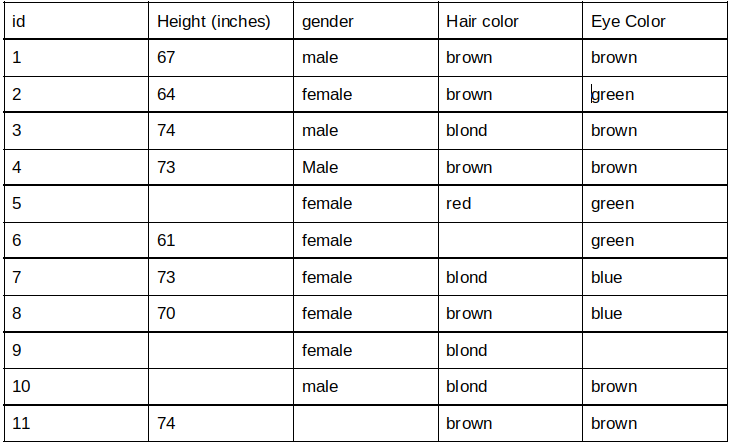
**Paste your resulting graph here**



ISM 4403 Homework Week 3

### **Tasks:**

Create a new Excel spreadsheet from the following table.



Import the the sheet into R as a dataset using the read package.

To recode the height to be the mean for males and females respectively. In summary, you must enter the mean value for females into the height for any females with missing data. You must also enter the mean value for males into the height for any males with missing data. You MUST use R’s recode functionality to do this.

For any missing data in the Hair color and eye color fields insert the mode of that field.

Hint the following template can be used to accomplish this.

x[is.na(x)] <- mean(x, na.rm = TRUE)

x = na.omit(x)

Once you have filled in any missing data calculate the following items.

The mean height.

The sum of all heights.

The mode for gender, hair color, and eye color.

**Paste your code here**

> library(readxl)

> Book2 <- read\_excel("Desktop/Book2.csv")

> View(Book2)

> Book2

# A tibble: 11 x 5

id `Height (inches)` gender `Hair color` `Eye Color`

*<dbl>* *<dbl>* *<chr>* *<chr>* *<chr>*

1 1 67 male brown brown

2 2 64 female brown green

3 3 74 male blond brown

4 4 73 Male brown brown

5 5 NA female red green

6 6 61 female NA green

7 7 73 female blond blue

8 8 70 female brown blue

9 9 NA female blond NA

10 10 NA male blond brown

11 11 74 NA brown brown

> Book2$`Height (inches)`[which(is.na(Book2$`Height (inches)`))] <-mean(Book2$`Height (inches)`,na.rm = TRUE)

>Book2$`Hair color`[which(is.na(Book2$`Hair color`))] <- 'brown'

>Book2$`Eye Color`[which(is.na(Book2$`Eye Color`))]<- 'brown'

>na.omit(Book2)

id `Height (inches)` gender `Hair color` `Eye Color`

*<dbl>* *<dbl>* *<chr>* *<chr>* *<chr>*

1 1 67 male brown brown

2 2 64 female brown green

3 3 74 male blond brown

4 4 73 Male brown brown

5 5 69.5 female red green

6 6 61 female brown green

7 7 73 female blond blue

8 8 70 female brown blue

9 9 69.5 female blond brown

10 10 69.5 male blond brown

>

> mean(Book2$`Height (inches)`)

[1] 69.5

> sum(Book2$`Height (inches)`)

[1] 764.5

> mode(Book2$gender)

[1] "character"

> mode(Book2$`Hair color`)

[1] "character"

> mode(Book2$`Eye Color`)

[1] "character"

**End of Paste**

**Paste your results here**

id `Height (inches)` gender `Hair color` `Eye Color`

*<dbl>* *<dbl>* *<chr>* *<chr>* *<chr>*

1 1 67 male brown brown

2 2 64 female brown green

3 3 74 male blond brown

4 4 73 Male brown brown

5 5 69.5 female red green

6 6 61 female brown green

7 7 73 female blond blue

8 8 70 female brown blue

9 9 69.5 female blond brown

10 10 69.5 male blond brown

> mean(Book2$`Height (inches)`)

[1] 69.5

> sum(Book2$`Height (inches)`)

[1] 764.5

> mode(Book2$gender)

[1] "character"

> mode(Book2$`Hair color`)

[1] "character"

> mode(Book2$`Eye Color`)

[1] "character"

**End of Paste**

Using the Quantmod interface get data for Boeing, Apple, Nvida, AMD, and Intel. (See the following for more information<https://github.com/joshuaulrich/quantmod>) Yahoo finance should be used as your source.

Print the mean stock price for each stock

Chart it in a Bar chart

**Paste your resulting code here**

>library(quantmod)

> getSymbols("AAPL", src = "yahoo")

[1] "AAPL"

> getSymbols("BA", srs= "yahoo")

[1] "BA"

> getSymbols("NVDA", srs= "yahoo")

[1] "NVDA"

> getSymbols("AMD", srs= "yahoo")

[1] "AMD"

> getSymbols("INTC")

[1] "INTC"

> summary(AAPL)

Index AAPL.Open AAPL.High

Min. :2007-01-03 Min. : 11.34 Min. : 11.71

1st Qu.:2010-04-13 1st Qu.: 34.35 1st Qu.: 34.69

Median :2013-07-22 Median : 79.38 Median : 80.22

Mean :2013-07-19 Mean : 91.11 Mean : 91.95

3rd Qu.:2016-10-26 3rd Qu.:127.00 3rd Qu.:127.85

Max. :2020-02-06 Max. :324.45 Max. :327.85

AAPL.Low AAPL.Close AAPL.Volume

Min. : 11.17 Min. : 11.17 Min. : 11362000

1st Qu.: 33.86 1st Qu.: 34.33 1st Qu.: 35749000

Median : 78.81 Median : 79.28 Median : 79055900

Mean : 90.25 Mean : 91.13 Mean :108695473

3rd Qu.:125.87 3rd Qu.:126.82 3rd Qu.:149827300

Max. :321.38 Max. :325.21 Max. :843242400

AAPL.Adjusted

Min. : 9.72

1st Qu.: 29.87

Median : 70.44

Mean : 85.75

3rd Qu.:117.22

Max. :325.21

> summary(BA)

Index BA.Open BA.High

Min. :2007-01-03 Min. : 29.22 Min. : 30.02

1st Qu.:2010-04-13 1st Qu.: 71.73 1st Qu.: 72.47

Median :2013-07-22 Median :105.13 Median :106.00

Mean :2013-07-19 Mean :143.48 Mean :144.91

3rd Qu.:2016-10-26 3rd Qu.:151.96 3rd Qu.:153.17

Max. :2020-02-06 Max. :446.01 Max. :446.01

BA.Low BA.Close BA.Volume

Min. : 29.05 Min. : 29.36 Min. : 788900

1st Qu.: 71.08 1st Qu.: 71.80 1st Qu.: 3313700

Median :104.23 Median :104.99 Median : 4455100

Mean :142.00 Mean :143.51 Mean : 5155975

3rd Qu.:151.07 3rd Qu.:152.32 3rd Qu.: 6035200

Max. :440.19 Max. :440.62 Max. :44391800

BA.Adjusted

Min. : 22.21

1st Qu.: 57.62

Median : 88.58

Mean :129.68

3rd Qu.:135.54

Max. :432.86

> summary(NVDA)

Index NVDA.Open NVDA.High

Min. :2007-01-03 Min. : 6.00 Min. : 6.38

1st Qu.:2010-04-13 1st Qu.: 14.27 1st Qu.: 14.49

Median :2013-07-22 Median : 19.99 Median : 20.24

Mean :2013-07-19 Mean : 60.33 Mean : 61.22

3rd Qu.:2016-10-26 3rd Qu.: 69.08 3rd Qu.: 70.52

Max. :2020-02-06 Max. :289.32 Max. :292.76

NVDA.Low NVDA.Close NVDA.Volume

Min. : 5.75 Min. : 5.90 Min. : 1141100

1st Qu.: 14.08 1st Qu.: 14.24 1st Qu.: 8241700

Median : 19.64 Median : 19.97 Median :12130400

Mean : 59.35 Mean : 60.31 Mean :14186436

3rd Qu.: 68.02 3rd Qu.: 68.52 3rd Qu.:17734900

Max. :285.58 Max. :289.36 Max. :92323200

NVDA.Adjusted

Min. : 5.433

1st Qu.: 13.234

Median : 18.888

Mean : 59.265

3rd Qu.: 67.698

Max. :287.946

> summary(AMD)

Index AMD.Open AMD.High

Min. :2007-01-03 Min. : 1.620 Min. : 1.690

1st Qu.:2010-04-13 1st Qu.: 3.720 1st Qu.: 3.780

Median :2013-07-22 Median : 6.690 Median : 6.840

Mean :2013-07-19 Mean : 9.361 Mean : 9.551

3rd Qu.:2016-10-26 3rd Qu.:12.130 3rd Qu.:12.360

Max. :2020-02-06 Max. :52.280 Max. :52.810

AMD.Low AMD.Close AMD.Volume

Min. : 1.61 Min. : 1.620 Min. : 0

1st Qu.: 3.65 1st Qu.: 3.700 1st Qu.: 15411600

Median : 6.51 Median : 6.700 Median : 24274700

Mean : 9.16 Mean : 9.355 Mean : 34902428

3rd Qu.:11.92 3rd Qu.:12.120 3rd Qu.: 42737100

Max. :51.20 Max. :51.710 Max. :325058400

AMD.Adjusted

Min. : 1.620

1st Qu.: 3.700

Median : 6.700

Mean : 9.355

3rd Qu.:12.120

Max. :51.710

> summary(INTC)

Index INTC.Open INTC.High

Min. :2007-01-03 Min. :12.17 Min. :12.63

1st Qu.:2010-04-13 1st Qu.:21.42 1st Qu.:21.69

Median :2013-07-22 Median :25.89 Median :26.19

Mean :2013-07-19 Mean :29.87 Mean :30.19

3rd Qu.:2016-10-26 3rd Qu.:35.43 3rd Qu.:35.64

Max. :2020-02-06 Max. :67.63 Max. :69.29

INTC.Low INTC.Close INTC.Volume

Min. :12.05 Min. :12.08 Min. : 5893800

1st Qu.:21.19 1st Qu.:21.43 1st Qu.: 23642100

Median :25.68 Median :25.93 Median : 36687300

Mean :29.57 Mean :29.89 Mean : 43770078

3rd Qu.:35.11 3rd Qu.:35.40 3rd Qu.: 57196100

Max. :66.77 Max. :68.47 Max. :309347600

INTC.Adjusted

Min. : 8.62

1st Qu.:15.97

Median :20.73

Mean :25.72

3rd Qu.:32.39

Max. :68.13

> Stock\_mean <- c(85.75,129.68,59.265,9.355,25.72)

> Stock\_name <- c("APPL","BA","NVDA","AMD","INTC")

> barplot(Stock\_mean, names.arg = Stock\_name, xlab = "Stock", ylab = "USD", main = "Mean Stock Price", col = "green")

> barChart(AAPL)

> barChart(BA)

> barChart(NVDA)

> barChart(AMD)

> barChart(INTC)

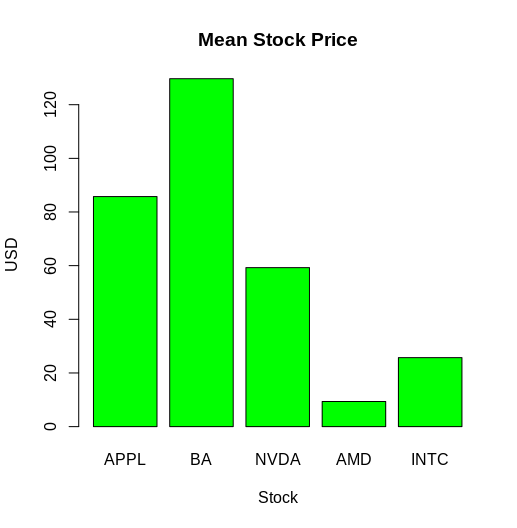
**END OF Paste**

**Paste your resulting graph here**

> Stock\_mean <- c(85.75,129.68,59.265,9.355,25.72)

> Stock\_name <- c("APPL","BA","NVDA","AMD","INTC")

**> barplot(Stock\_mean, names.arg = Stock\_name, xlab = "Stock", ylab = "USD", main = "Mean Stock Price", col = "green")**



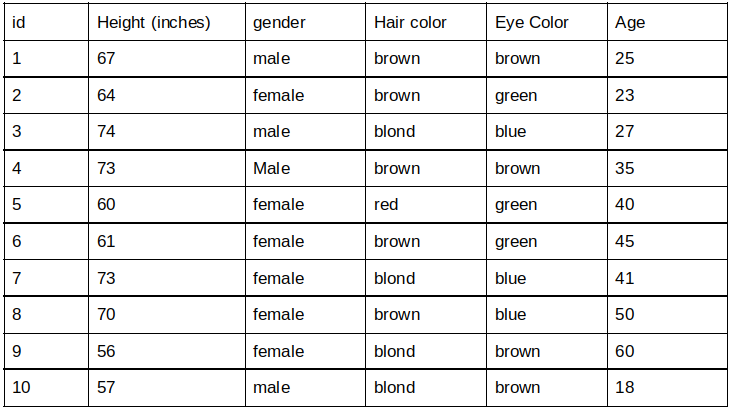
**> barChart(AAPL)**



ISM 4403 Homework Week 4

### **Tasks:**

Create a new Excel spreadsheet from the following table.



Import the the sheet into R.

Calculate the following.

The mean height.

The sum of all heights.

The mode for gender, hair color, and eye color.

Plot a pie graph for each of the following parameters, gender (count), hair color (count), eye color (count).

Plot a line graph of height and age. In your opinion is there a relationship between height and age? Please explain why or why not.

**Paste your code here**

> library(readxl)

> Book3 <- read\_excel("Desktop/Book3.xlsx")

> View(Book3)

> mean(Book3$`Height (inches)`)

[1] 65.5

> sum(Book3$`Height (inches)`)

[1] 655

> mode(Book3$gender)

[1] "character"

> mode(Book3$`Hair color`)

[1] "character"

> mode(Book3$`Eye Color`)

[1] "character"

> lbls <- c("male","female")

> slices <- c(4,6)

> pie(slices, lables = lbls, main='Pie Chart of Gender')

> pie(slices,labels = lbls, col=rainbow(length(lbls)), main="Pie Chart of Gender")

> lbls <- c("brown","blond","red")

> slices <- c(5,4,1)

> pie(slices,labels = lbls, col=rainbow(length(lbls)), main="Pie Chart of Hair Color")

> lbls <- c("brown","green","blue",)

> lbls <- c("brown","green","blue")

> slices <- c(4,3,3)

> pie(slices,labels = lbls, col=rainbow(length(lbls)), main="Pie Chart of Eye Color")

> slices <- c(4,6,5,4,1,4,3,3)

> lbls <-c("male","female","brown","blond","red","brown","green","blue")

> pct <- round(slices/sum(slices)\*100)

> lbls <- paste(lbls,pct)

> lbls <- paste(lbls,"%",sep="")

> pie(slices,labels = lbls, col=rainbow(length(lbls)), main="Pie Chart")

> h <- c(67,64,74,73,60,61,73,70,56,57)

> plot(h ,type = "o", col = "blue" ,ylab = "Inches" , main= "Height")

> a <- c(25,23,27,35,40,45,41,50,60,18)

> plot(a ,type = "o", col = "red" ,ylab = "Years" , main= "Age")

**End of Paste**

**Paste your results here**

> mean(Book3$`Height (inches)`)

[1] 65.5

> sum(Book3$`Height (inches)`)

[1] 655

> mode(Book3$gender)

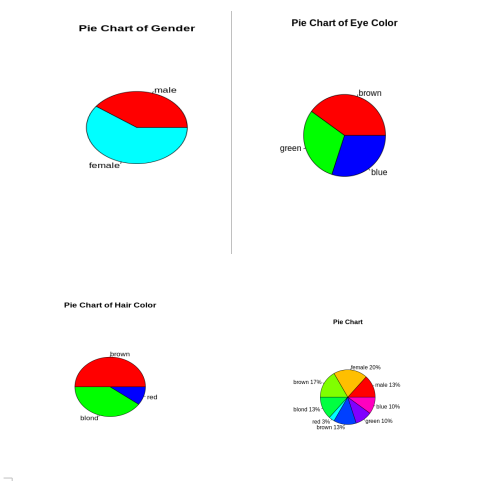
[1] "character"

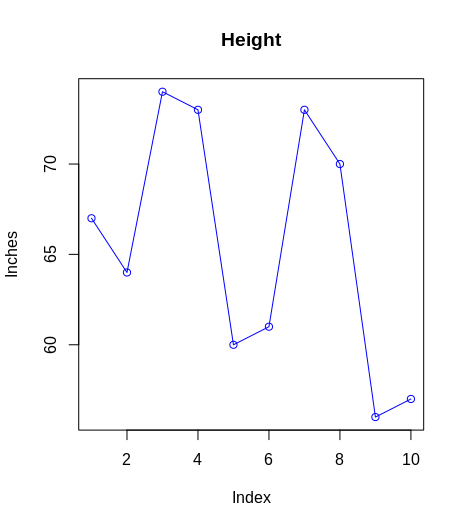
> mode(Book3$`Hair color`)

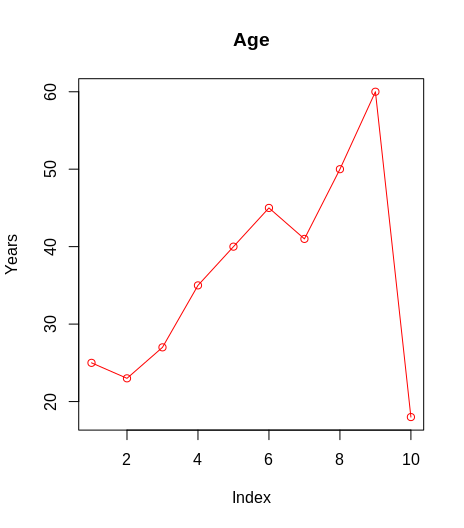
[1] "character"

> mode(Book3$`Eye Color`)

[1] "character"







ISM 4403 Homework Week 5

### **Tasks:**

Create a new Excel spreadsheet from the following table.

Import the the sheet into R.

Resolve any issues with missing data as done in homework 4.

Calculate the following.

The mean height.

The sum of all heights.

The mode for gender, hair color, and eye color.

Plot a pie graph for each of the following parameters, gender (count), hair color (count), eye color (count).

Plot a line graph of height and age. In your opinion is there a relationship between height and age? Please explain why or why not.

**There does not seem to be a relationship between height and age. Although the data is misleading data because we do not have any samples under the age of 18**

**Paste your code here**

> library(readxl)

> Book4 <- read\_excel("Documents/Book4.xlsx")

> View(Book4)

> Book4$`Height (inches)`[which(is.na(Book4$`Height (inches)`))] <-mean(Book4$`Height (inches)`,na.rm = TRUE)

> mean(Book4$`Height (inches)`)

[1] 67.82143

> sum(Book4$`Height (inches)`)

[1] 2034.643

> Book4$gender[which(is.na(Book4$gender))] <- 'female'

> Book4$`Hair color`[which(is.na(Book4$`Hair color`))] <- 'brown'

> Book4$`Eye Color`[which(is.na(Book4$`Eye Color`))] <- 'brown'

> mode(Book4$gender)

[1] "character"

> mode(Book4$`Hair color`)

[1] "character"

> mode(Book4$`Eye Color`)

[1] "character"

> Book4$Age[which(is.na(Book4$Age))] <-mean(Book4$Age,na.rm = TRUE)

#pie chart gender

lbls <- c("male","female")

> slices <- c(13,17)

> pie(slices,labels = lbls, col = rainbow(length(lbls)),main="Pie Chart of Gender")

#pie chart hair

> lbls <- c("brown","blond","red")

> slices <- c(15,12,3)

> pie(slices, labels = lbls,col = rainbow(length(lbls)),main = "Pie Chart of Hair Color")

#pie chart eye

> lbls <- c ("brown","green","blue")

> slices <- c(13,9,8)

> pie(slices, labels = lbls,col = rainbow(length(lbls)),main = "Pie Chart of Eye Color"

#line graph height

> h <-c(67,64,74,73,60,61,73,70,56,57,64,69,70,71,60,80,75,78,69,67.8,66,74,72,68,64,63,67.8,70,62)

> plot(h, type = "o", col = "red", ylab = "Inches", main = "Height")

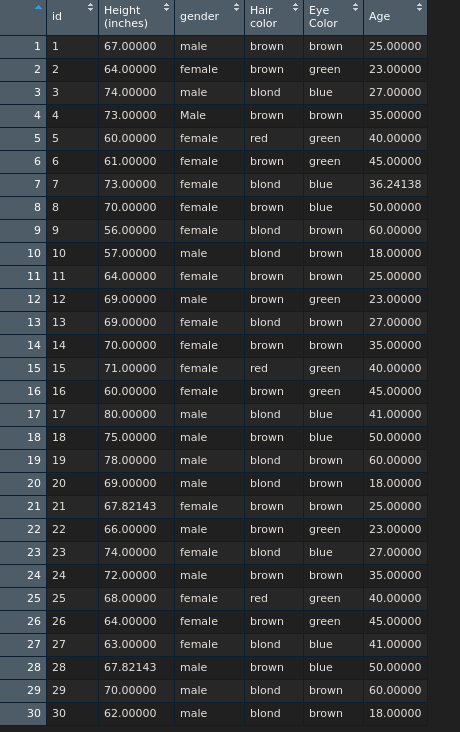
#line graph age

> a <- c(25,23,27,35,40,45,35.2,50,60,18,25,23,27,35,40,45,41,50,60,18,25,23,27,35,40,45,41,50,41,50,60,18)

> plot(a, type = "o", col = "blue", ylab = "Years Old", main = "Age")

**End of Paste**

**Paste your results here**



> mean(Book4$`Height (inches)`)

[1] 67.82143

> sum(Book4$`Height (inches)`)

[1] 2034.643

> mode(Book4$gender)

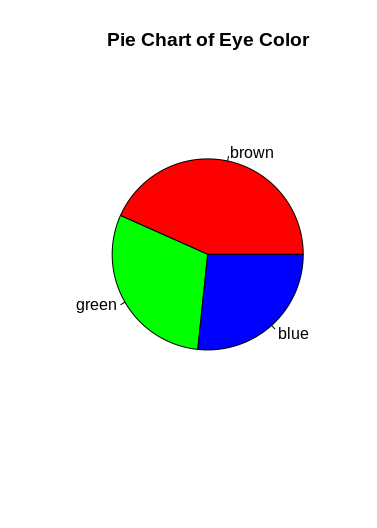
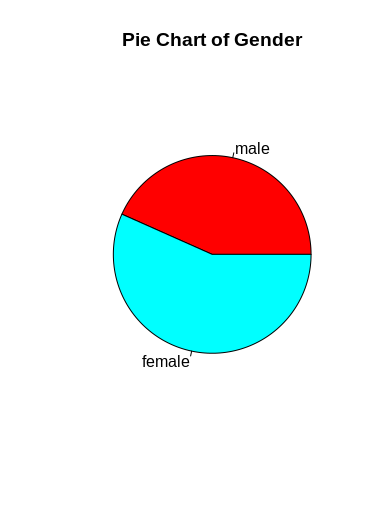
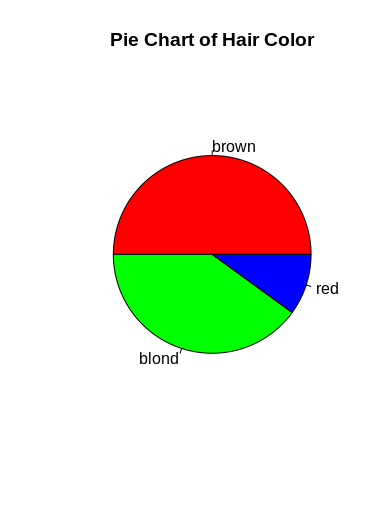
[1] "character"

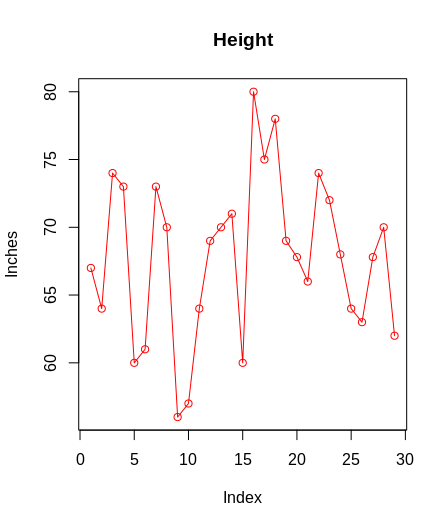
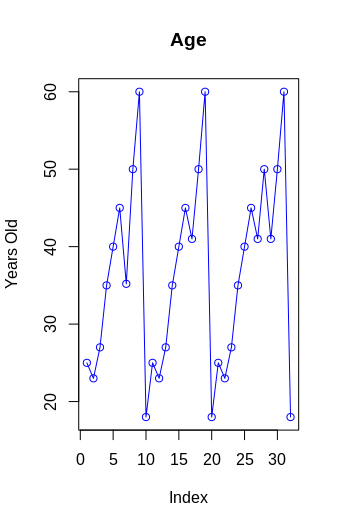
> mode(Book4$`Hair color`)

[1] "character"

> mode(Book4$`Eye Color`)

[1] "character"





**There does not seem to be a relationship between height and age. Although the data is misleeding data because we do not have any samples under the age of 18**

**End of Paste**

Using the example in 6.6 to create a regression model for the above data. Use this model to describe the relationship between the data in the above table.

Is there a relationship between height and age? Please explain?

**There does not seem to be a relationship between height and age. Although the data is misleeding data because we do not have any samples under the age of 18**

**Paste your answer here**

>library(caTools)

>set.seed(123)

>split = sample.split(Book4$’Height (inches)’), SplitRatio = 2/3)

>training\_set = subset(Book4, split == TRUE)

>test\_set = subset(Book4, split == FALSE)

#Linear Regression to Training set

>regressor = lm(formula = ’Height (inches)’ ~ Age,

data = training\_set)

#Predicting Test results

>y\_pred = predict(regressor, newdata = test\_set)

#Training set results

>library(ggplot2)

>ggplot() +

geom\_point(aes(x = training\_set$’Height (inches)’, y = training\_set$Age),

colour = 'red') +

gemo\_line(aes(x = training\_set$’Height (inches)’, y = predict(regressor, newdata = training\_set)),

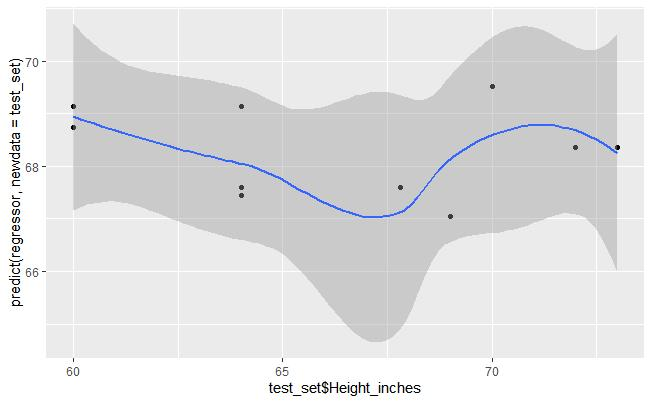
color = 'blue') +

> ggtitle(’Height (inches)’(Training set)') +

> xlab('Height') +

> ylab('Age')

> ggplot()



END PASTE



Given the above table (chart 5.2) plot a line graph of customer height to jean length.

If we accept that chart 5.1 represents 1% of the population who will buy jeans and we have a monopoly. Chart a line graph that indicates how many individuals will buy each length of jean. Be sure to check the quality of the data and resolve any issues you find. The fix must be implemented in R. We will not accept solutions where you manually clean the data.

**Paste your code here**

> H <- c(67,72,60,60,65,66,80,76,80,72)

> L <- c(30,36,24,26,28,28,40,38,40,36)

> plot(H,type = "o",col = "red", xlab = "Customer Height", ylab = "Jean Length",

+ main = "Height to Length Chart")

> lines(L, type = "o", col = 'blue')

>

> plot(H,type = "o",col = "red", xlab = "Customer Height", ylab = "Jean Length", main = "Height to Length")

> lines(L, type = "o", col = 'blue')

> lines(L, type = "o", col = "blue")

> plot(L,type = "o",col = "red", xlab = "Customer Height", ylab = "Jean Length", main = "Height to Length")

> lines(H, type = "o", col = "blue")

> library(ggplot2)

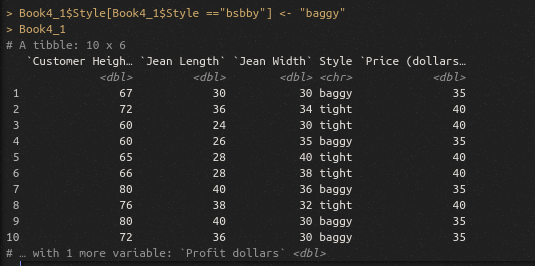
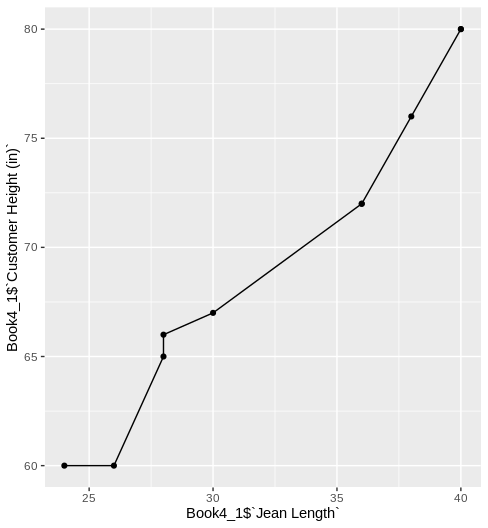
> ggplot(Book4\_1, aes(x = Book4\_1$`Jean Length`, y = Book4\_1$`Customer Height (in)`)) + geom\_line() + geom\_point()

> Book4\_1$Style[Book4\_1$Style =="bsbby"] <- "baggy

# do not know how to “Chart a line graph that indicates how many individuals will buy each length of jean” I feel like there is no way of knowing that with the information provided to us

**End of Paste**

**Paste your results here**



ISM 4403 Homework Week 6

### **Tasks**

Using the information provided in the video perform a similar analysis upon the works by Edgar Rice Burrows that are available on Project Gutenberg. (<https://www.gutenberg.org/ebooks/author/48>)

Please provide the following data making sure to remove stop words and clean up punctuation etc…

What is the most common word used in these works?

Attempt to use inverse document frequency to determine the people and places in these books. If this results in alternative data please describe.

Attempt to determine the most common names using the Tidy framework. What are they?

PASTE YOUR CODE HERE

**#install.packages("tidyverse")**

**#install.packages("gutenbergr")**

**#install.packages("tidytext")**

**>library(gutenbergr)**

**>library(tidyverse)**

**>library(tidytext)library(dplyr)**

**#looking in the books**

**>edgar\_df <- gutenberg\_download(c(62, 78, 64, 1154, 551, 106, 106, 90, 605, 92, 123, 85, 1401, 58874, 3475, 59752, 3475, 59752, 363, 331, 552, 149, 369, 96, 554, 2020, 29405, 58904, 20802), meta\_fields = "title")**

**# turning text df into tidy data**

**>edgar\_tidy <- edgar\_df %>%**

**>unnest\_tokens(word, text)**

**>edgar\_tidy**

**# remove stop words**

**>edgar\_clean <- edgar\_tidy %>%**

**>anti\_join(get\_stopwords())**

**# sort**

**>edgar\_clean %>%**

**>count(word, sort = TRUE)**

**>edgar\_count <- edgar\_clean %>%**

**>count(word, sort = TRUE)**

**>edgar\_king <- edgar\_tidy %>%**

**>anti\_join(get\_stopwords())**

**>edgar\_king**

**#Calculate the frequency of each unique word and sort by frequency**

**#to determine important words and items.**

**>edgar\_count <- edgar\_count %>%**

**>bind\_tf\_idf(word, n)**

**>edgar\_idf**

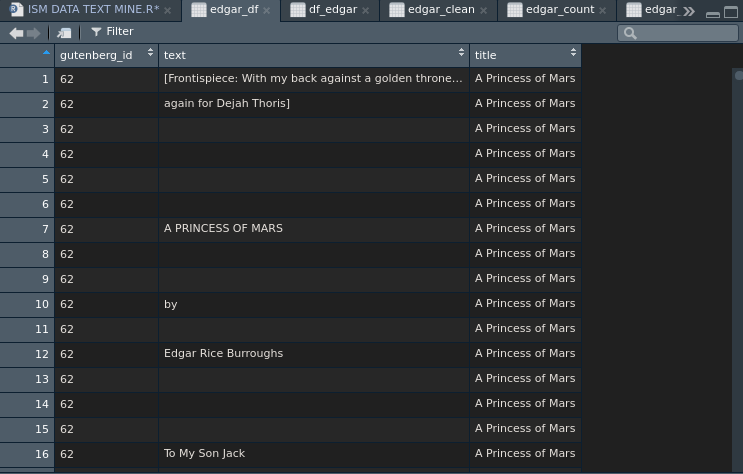
**>edgar\_count %>%**

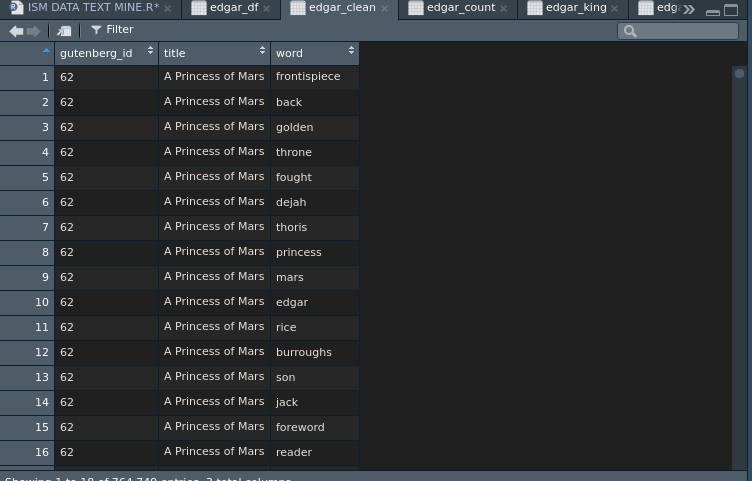
**>select(-word) %>%**

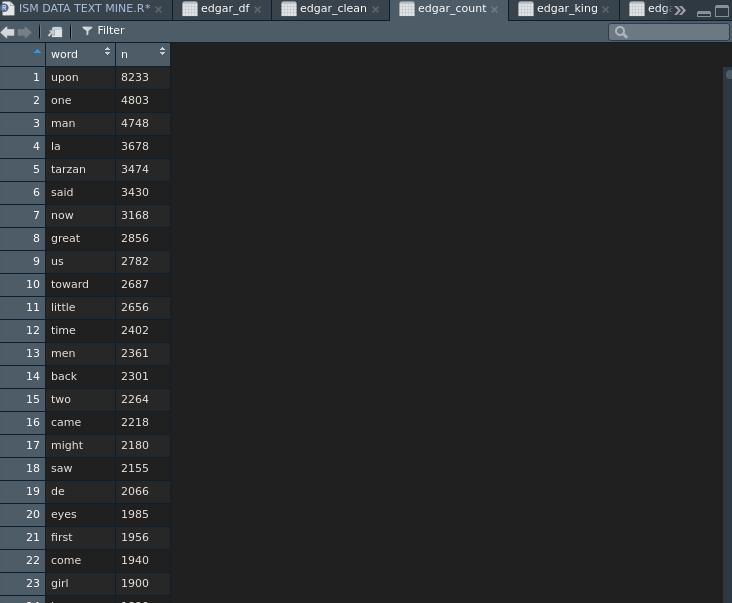
**>arrange(desc(word))**

END PASTE

PASTE YOUR RESULTS HERE







END PASTE

20% Installed Tidy framework and imported the library, or loaded a similar library

20% Load the words of each book into a data structure, normalized text to lowercase, remove punctuation, and spaces.

20% Calculated the frequency of each word for each book.

20% Remove common words from the collections of each book to determine unique words

20% Calculate the frequency of each unique word and sort by frequency to determine important words and items.