# R Workshop ACMS2019

*Judith E. Canner, Alana Unfried, Ryan Botts*

*2019-06-01*

# Contents

# Preface

Welcome to the 2018 ACMS R workshop!

## Welcome

R is a freely available language and environment for statistical computing and graphics that has become popular in academia and in many industries. In addition, RStudio and RMarkdown have become standard tools in reproducible analysis and reporting. This mini-course will introduce participants to teaching applied statistics courses using integrated computing to facilitate learning and develop best practices in statistical learning. The presenters will share different approaches to teaching statistics using R and some favorite examples for using R to teach statistics to undergraduates at all levels, including in introductory courses for mathematics and science majors.

Topics will include simple approaches to provide novices with a powerful, but manageable, set of tools, workflow in the RStudio environment, data visualization, basic statistical inference using R, and using R Markdown to create documents that include both text and R output. This mini-course is designed to be accessible to those with little or no experience using R or teaching with R and will provide participants with skills, examples, and resources that they can use in their own teaching.

## Purpose

The aim of this workshop is to introduce the users to beginning and intermediate uses of R, along with best practices of using R in the classroom as well as in real-world data analysis projects. We assume no prior background with R. As you learn R, we also aim to provide many practical examples of the tools we commonly use in our own work. During the workshop there are many places where optional questions are given. If you have been successful, with the code provided we encourage you to try these extra bonuses to test your abilities. At the end of the workshop we will provide a real-world project to give you the opportunity to apply your skills.

## Who we are

**Judith E Canner** Associate Professor of Statistics, California State University, Monterey Bay

Dr. Canner has been using R to teach undergraduate introductory statistics for biologists and mathematicians/statisticians since 2010. She has presented numerous times on R pedagogy in statistics courses and regularly teaches a statistical computing seminar using R, R Studio, and R Markdown.

**Alana Unfried** Assistant Professor of Statistics, California State University, Monterey Bay

Dr. Unfried uses R to teach sampling and statistical modeling in upper-division undergraduate statistics courses. She uses R extensively for her own research and for research projects involving students. Much of her research involves improving undergraduate statistics education.

**Ryan Botts** Associate Professor of Mathematics, Point Loma Nazarene University

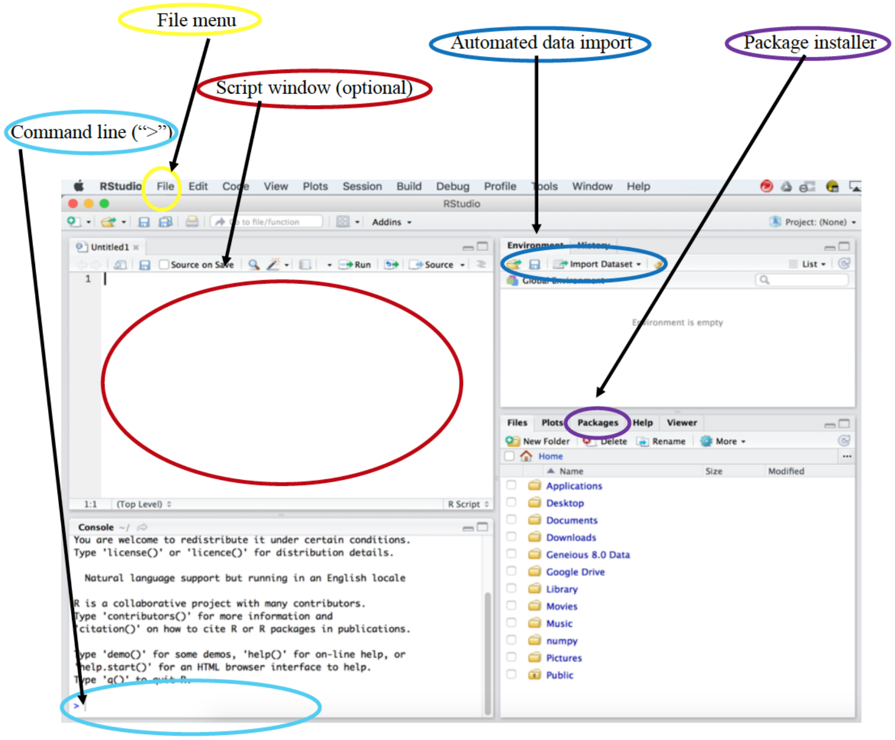
Dr. Botts has been teaching R extensively to undergraduates from across many disciplines since 2014, and regularly advises students in undergraduate research using R and RShiny. Additionally, he has published on teaching pedagogy in non-major mathematics courses, including introductory statistics.

# Chapter 1

# RStudio Environment

Here we will gain some basic familiarity with the **RStudio** environment. Usually the first confusion for students in using R is in actually installing it. We typically encourage students to develop and interface with **R** through **RStudio**. So let's take a look at what **RStudio** has to offer.

Below is an image of the typical **RStudio** environment.



```
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

When R is opened, a workspace is created and code run from a script, at the command line or from one of the built in functions and by default is run from this workspace.

## 1.1 Importing data

We will begin by importing a dataset posted by Merijn Coumans from a Tesla User's Group about the battery life, charge history, location, etc. of Tesla cars. A slightly cleaned version of the data can be found cleanTeslaBattery.csv. Download this dataset and save it somewhere you can find it.

On the upper right select the *Import Dataset* button, for this case select *From Text (base)*. When you find the dataset, select open and you will see a new menu open. A few things to note are the "Name" box specifying the variable name assigned to the dataset, and also note the "Header" option and select appropriately depending on whether there is a header row or not. Import the data. Note what appears in the terminal.

```
Tesla <- read.csv("Data/cleanTeslaBattery.csv")
View(Tesla)
```

```
Teaching tip: when first introducing this to students, it is helpful to identify how they can cha
```

For entry level students, this is an easy way to import data, but it also provides us with an example of the command that we could have manually typed to load the data, and which we can put into our script or notebook document. We will save this command for use in just a bit.

Without introducing object types or anything advanced, a quick summary of the data can give some useful information and let students know if the data read in as expected. We will try that command shortly.

```
Teaching tip: Students with Mac's using Numbers frequently open the files they download using Num
```

## 1.2 Command line

Let's go ahead and compute a summary of the data we just imported to double-check how R is treating each variable, e.g. categorical, quantitative, or character.

```r
summary(Tesla)
```

```
                            Location                ManufactureDate
 Asia Pacific & Europe (excl UK):1058   2015-04-03T00:00:00Z:  46  
 Canada                         :  47   2014-06-04T00:00:00Z:  32  
 UK                             :  12   2015-06-10T00:00:00Z:  32  
 USA                            : 222   2013-09-10T00:00:00Z:  28  
                                        2013-11-20T00:00:00Z:  27  
                                        (Other)             :1169  
                                        NA's                :   5  
                 ReadDate      AgeInDays                Model    
 2015-01-15T00:00:00Z:   8   Min.   :    0.0   Model S 85  :475  
 2015-10-21T00:00:00Z:   8   1st Qu.:  252.0   Model S P85 :178  
 2014-09-24T00:00:00Z:   7   Median :  494.0   Model S 85D :112  
 2015-12-09T00:00:00Z:   7   Mean   :  734.1   Model S 90D :107  
 2014-09-18T00:00:00Z:   6   3rd Qu.:  832.8   Model S 70D :103  
 2014-09-26T00:00:00Z:   6   Max.   :43219.0   Model S P85D: 91  
 (Other)             :1297   NA's   :1         (Other)     :273  
   MileageKM        MileagePerDay      MaxRangeKM     ReplacementBatt
 Min.   :     6   Min.   :  0.70   Min.   :174.0   No :1265      
 1st Qu.: 17838   1st Qu.: 57.40   1st Qu.:358.0   Yes:  74      
 Median : 39929   Median : 84.10   Median :377.0                 
 Mean   : 51325   Mean   : 88.63   Mean   :361.5                 
 3rd Qu.: 69278   3rd Qu.:113.42   3rd Qu.:389.0                 
 Max.   :361500   Max.   :282.90   Max.   :509.0                 
                  NA's   :5        NA's   :1                     
 MileageSinceNewKM BatteryAgeDays   WattHoursPerKM  OriginalRangeKM
 Min.   :     6    Min.   :   1.0   Min.   :155.0   Min.   :180.0  
 1st Qu.: 17687    1st Qu.: 246.0   1st Qu.:208.0   1st Qu.:365.0
 Median : 39120    Median : 487.0   Median :221.0   Median :400.0
 Mean   : 50616    Mean   : 571.2   Mean   :234.8   Mean   :378.8
 3rd Qu.: 69182    3rd Qu.: 823.0   3rd Qu.:241.0   3rd Qu.:400.0
 Max.   :284500    Max.   :2466.0   Max.   :450.0   Max.   :515.0
 NA's   :1         NA's   :6        NA's   :22
          SuperchargeFreq             MaxChargeFreq
 twice a month     :357    a few times a year  :503
 monthly           :261    monthly             :346
 a few times a year:240    twice a month       :164
 weekly            :189    once or twice a year: 84
 twice a week      :107    weekly              : 55
 (Other)           : 75    (Other)             : 71
 NA's              :110    NA's                :116
             RunToEmptyFreq DailyChargeLevel     Cycles       ModelXS
 never               :438    Min.   :10.00    Min.   :  0.02   3 :   4
 a few times a year  :347    1st Qu.:80.00    1st Qu.: 55.05   85:  24
 once or twice a year:331    Median :80.00    Median :123.95   S :1277
 monthly             : 72    Mean   :80.83    Mean   :155.24   X :  34
 twice a month       : 27    3rd Qu.:90.00    3rd Qu.:213.89
 (Other)             :  9    Max.   :90.00    Max.   :906.37
 NA's                :115    NA's   :279      NA's   :23
    ModNum
 85     :475
 P85    :178
 90D    :136
 85D    :112
 70D    :103
 P85D   : 91
 (Other):244
```

Based on the summary, what can you say about how R is summarizing the *age* column? This is good when working with introductory students.

```
**Questions**
1. The *head* and *tail* functions allow you to inspect the first few rows or last few
2. Try the *names* function on this data?  What does it produce?
```

To demonstrate the utility of R, let's compute the mean of the Mileage column.

```
mean(Tesla$MileageKM)
```

```
[1] 51325.09
```

```
Teaching tip:  this is a useful place to introduce students to case sensitivity and wha
```

**Programmer's note:** R treats columns in a dataframe much like dictionaries in Python, as well as using matri notation like Python, Java or Matlab. Aside from the "$" operator, one can reference the row or column using numbers or names (if they have them) The code below computes the standard deviation of the same column.

```
sd(Tesla[,"MileageKM"])
```

```
[1] 46712.35
```

**Practice** Try the histogram function `hist` and the boxplot function `boxplot` on one of the quantitative variables. Teaching tip: When teaching R to introductory students, make it accessible as a go to tool for completing homework the first day. Teaching Tip: When importing data, frequently students will select the *readr* option. In many cases this won't matter, but readr reads the data in as a tibble instead of a data frame. In many cases students won't notice, but because of data structure of the tibble object, there are cases where downstream analysis will not behave the same, so for simplicity the base option is often easier.

For intro student make it the go to tool, they often need a calculator for their homework, show how it can be done easily. Here we show how to input your own data and do some basic computations on it.

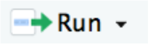
```
dat = c(1,9,23,35)
median(dat)
```

```
[1] 16
```

Note that the command `c()` specifies a list that is assigned to the variable `dat` that can then be used for calculation later.

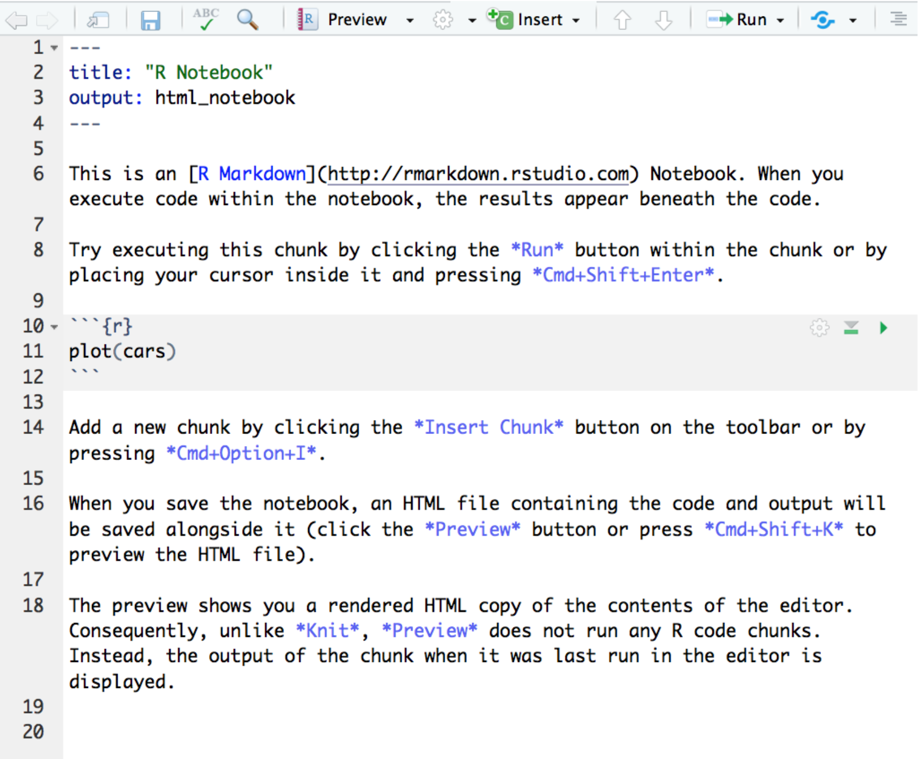
## 1.3 Scripts and Notebooks

This brings us to reproducability. One of the great strengths of R is in creating reproducible analyses. So let's save the code we just entered so that we can run it later. First we will create a new R Script.

Under file, open a new R Script. Copy the code you just entered at the terminal and past it into the script. Note that you will have to delete the ">" character from each new line. Some things to know about running from scripts: The run button is the little green arrow 

at the top of the text editor, if you select "run selected line(s)," the line the cursor is currently on will be run, or the current selected code will be executed. Try re-running all of the code you have just entered in your script from the terminal.

Notebooks are another nice way to create reproducible analyses. Under the file menu open a new notebook and you should see something like this:



~~~
---
title: "R Notebook"
output: html_notebook
---

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```{r}
plot(cars)
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.


~~~

You will find that the R community makes getting started in many of these files quite straightforward, with many detail on how to use them in the new files. Information on notebooks will be given later.

Note that the *cars* variable here is a pre-installed dataset that comes with R that is frequently used in examples.

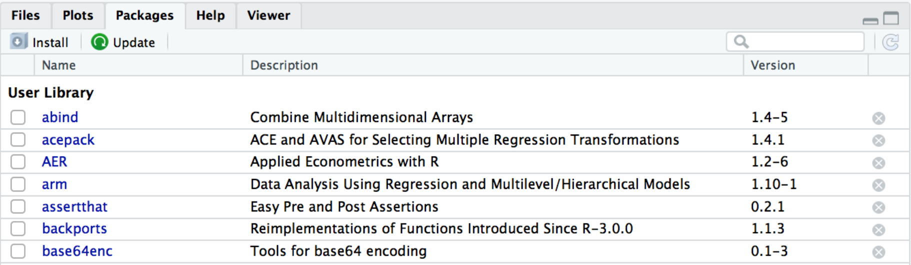
```
Teaching Tip: Remind students to test what they have saved.  I find that often they ma
```

---

**Question?** When would students want to use the command line over a notebook or a script?

## 1.4 Loading Packages

RStudio provides easy features for package management. The lower right-hand window has a package manager. The packages listed are currently installed, while others may be installed using the *Install* option. Note that when you are ready to use a package you must also load it using the `library()` function. Try loading the *ggplot2* package, which we will be using later in this workshop.



| | Name | Description | Version |
|---|---|---|---|
| **User Library** | | | |
| □ | abind | Combine Multidimensional Arrays | 1.4–5 |
| □ | acepack | ACE and AVAS for Selecting Multiple Regression Transformations | 1.4.1 |
| □ | AER | Applied Econometrics with R | 1.2–6 |
| □ | arm | Data Analysis Using Regression and Multilevel/Hierarchical Models | 1.10–1 |
| □ | assertthat | Easy Pre and Post Assertions | 0.2.1 |
| □ | backports | Reimplementations of Functions Introduced Since R–3.0.0 | 1.1.3 |
| □ | base64enc | Tools for base64 encoding | 0.1–3 |

`Teaching Tip: Students often forget that the workspace and the variables in it use memory, so enc`

# Chapter 2

# Managing Projects

## 2.1 Introduction to the R User Community

First of all, one of the best things about using R and R Studio is the vibrant and supportive community that wants to help you learn! The following materials are heavily adapted from Chester Ismay and Patrick Kennedy's Getting used to R, RStudio, and R Markdown. In addition, we will include materials from Garrett Grolemund's Master the Tidyverse.

On social media, the #rstats and #tidytuesday tags are great ways to see what others are doing in R and to learn new things.

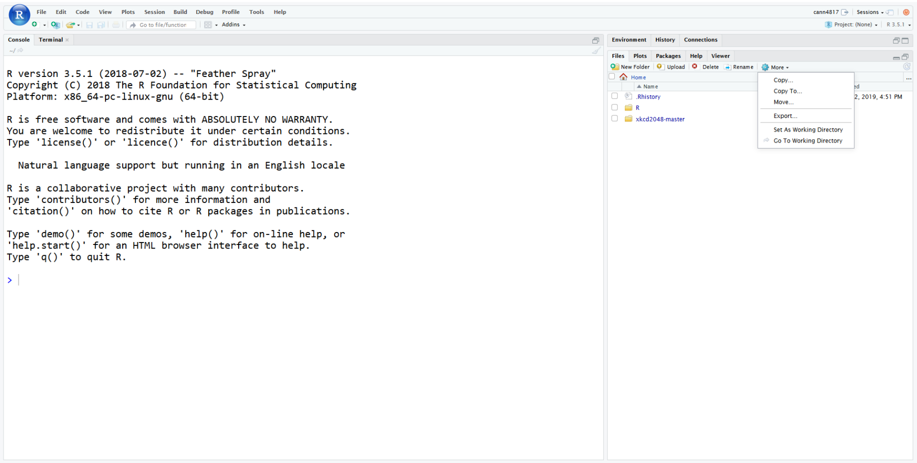
## 2.2 Introduction to Projects and Working Directories

Workflow is an important aspect of any data analysis project. It saves you time, allows for better collaboration, and reproducibility of analysis. Workflow is definitely one of R's strengths, as the R/RStudio environment was designed to support a productive and seamless workflow for analysis projects.

### 2.2.1 Working Directories

In R Studio, go to your lower left hand pane and be sure you are on the *Files* tab.

Click on and select **Go to Working Directory**. Whatever folder is displayed is your working directory.



```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Figure 2.1:

The ***Working Directory*** is where R looks for files, images, folders, etc. that you want to read into R. If you have a data set you want to load into R you could use the Import Dataset, but for reproducibility and ease, it is easier to write a script to read in your data.If `data.csv` is in your working directory, this is easy and straightforward:

```
newdata<-read.csv("data.csv")
```

If it is not, then you would have to tell R the full path name to the location on your computer where the data is stored:

```
newdata<-read.csv("C:/Users/admin/Foldername1/Foldername2/data.csv")
```

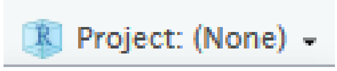
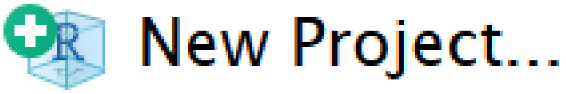
You can manually set the working directory from the More settings, but again, the advantage of R is to move away from manual tasks to automated tasks. To easily set a working directory for an analysis project, the best workflow option is to create a "PROJECT" in R.

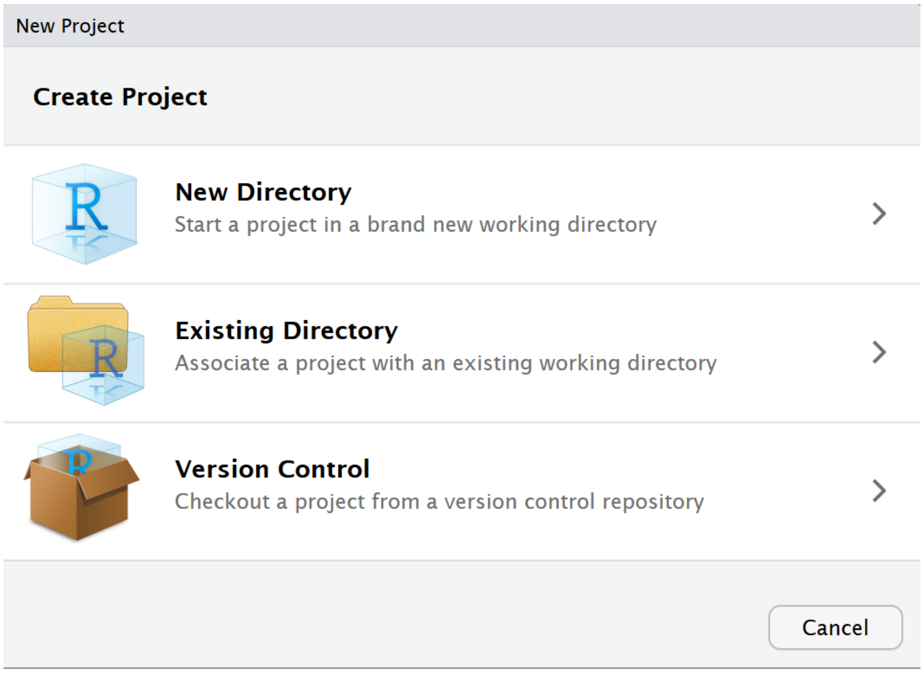
### 2.2.2 Projects in R

"When starting a new R project, it is good practice to create a new RStudio project to go along with it. RStudio project files have the extension `.Rproj` and store metadata and information about the R environment you are working in. More information about RStudio projects is available from RStudio, Inc." (Ismay and Kennedy).

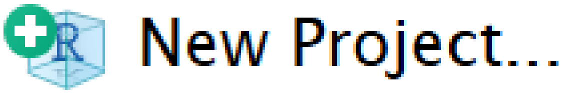
In the creation of an R Project, a working directory is automatically set when the project is opened. In addition, all created content (images, new data files, output files) is stored within the Project folder.

Let's create our first R Project and put a data file into the folder.

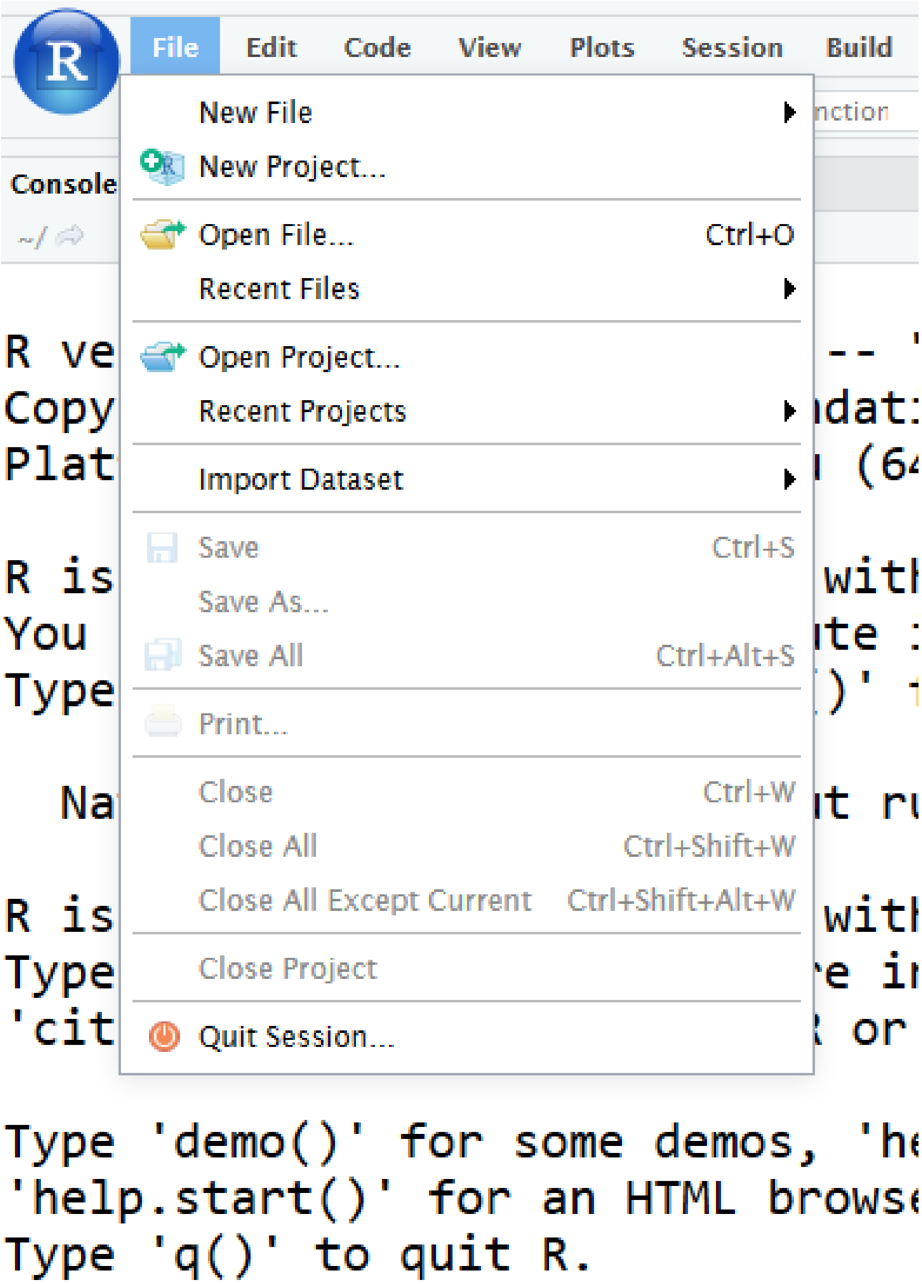
In the top right corner of R Studio is the . Click and then select . Or you can go to File in the top left corner and select New Project.

Select "New Directory" 

New Project
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

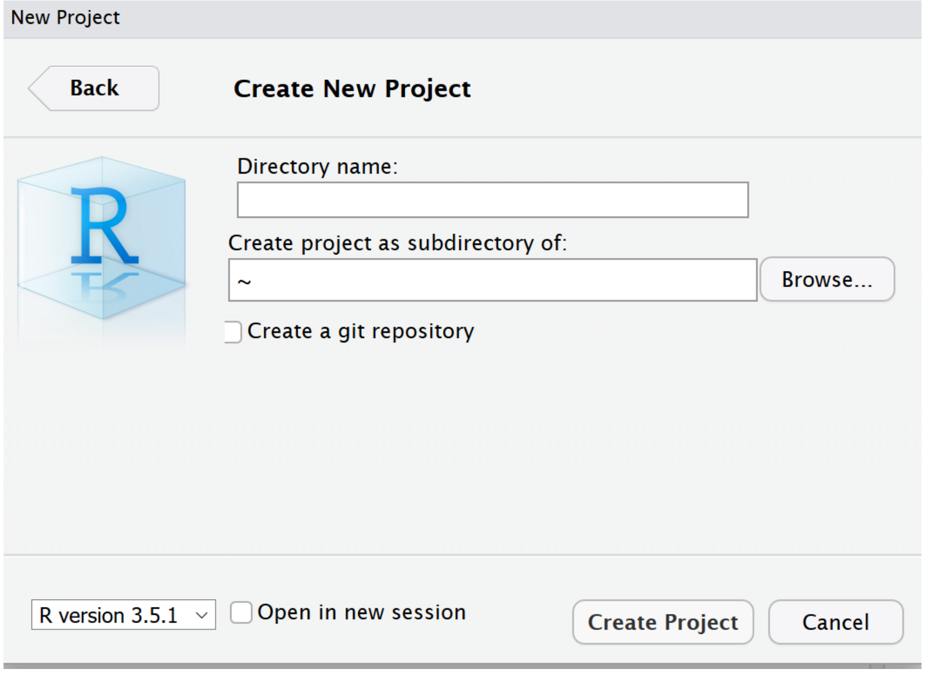
Select  and then type in the name "First_Project" into the Directory Name. You may also set the location of the file folder that will be created for the project using the *Browse...* button. Then click *Create Project.*

Congrats! You just created your first R Project. Notice that in your files you have



```
R ve                                  -- '
Copy                                  dat:
Plat                                  (6

R is                                  with
You                                   te :
Type                                  )' 1

  Na                                  t ru

R is                                  with
Type                                  e in
'cit                                  or

Type 'demo()' for some demos, 'he
'help.start()' for an HTML browse
Type 'q()' to quit R.
```

Figure 2.2:



Directory name:

Create project as subdirectory of: ~

□ Create a git repository

R version 3.5.1

□ Open in new session

Figure 2.3:

a special file type called

.

The process created a new folder and a `.Rproj` file. The `.Rproj` file, when opened, will set the working directory to the folder it is located.

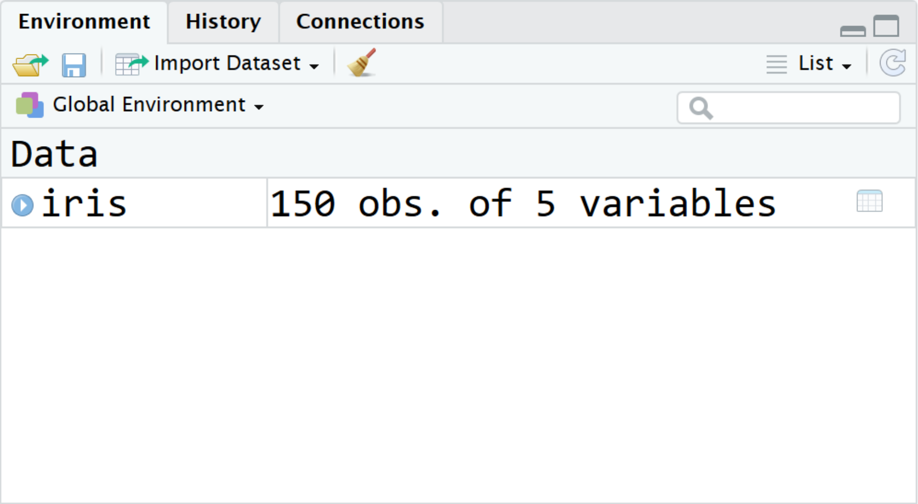
When a new project is created RStudio:

1. Creates a project file (with an `.Rproj` extension) within the project directory. This file contains various project options and can also be used as a shortcut for opening the project directly from the file system.
2. Creates a hidden directory (named .Rproj.user) where project-specific temporary files (e.g. auto-saved source documents, window-state, etc.) are stored.
3. Loads the project into RStudio and display its name in the Projects toolbar (which is located on the far right side of the main toolbar).

Now download the data set iris.csv and save it into the R project file location. Notice that it appears in the working directory. We can then load the data into R using the following command

```
iris<-read.csv("Data/iris.csv")
```

Notice that it appears in the environment as a data frame:



Data

iris 150 obs. of 5 variables

We can now start the analysis of the data!

```
summary(iris)
```

```
        X              Sepal.Length     Sepal.Width      Petal.Length
```

```
 Min.   :  1.00   Min.   :4.300   Min.   :2.000   Min.   :1.000  
 1st Qu.: 38.25   1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600  
 Median : 75.50   Median :5.800   Median :3.000   Median :4.350  
 Mean   : 75.50   Mean   :5.843   Mean   :3.057   Mean   :3.758  
 3rd Qu.:112.75   3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100  
 Max.   :150.00   Max.   :7.900   Max.   :4.400   Max.   :6.900  
  Petal.Width          Species  
 Min.   :0.100   setosa    :50  
 1st Qu.:0.300   versicolor:50  
 Median :1.300   virginica :50  
 Mean   :1.199                  
 3rd Qu.:1.800                  
 Max.   :2.500                  
```

```
*Teaching Tip: Students often forget to check their working directory and/or forget to open the .
```

**Now it is your turn!** Under the Files tab of the bottom left panel in RStudio, select New Folder. Call the New Folder “Data”. Now move the file TeslaBattery-Survey.csv into the “Data” folder. Does the following code work to import the data? How would you modify the following code to import the data correctly into R?

```
tesla<-read.csv("cleanTeslaBattery.csv")
```

*Solution*

```
tesla<-read.csv("Data/cleanTeslaBattery.csv")
```

## 2.3 More Resources on Projects

To learn more about version control and other features of R Projects, see the following RStudio Support Documenation on Using Projects and this introduction to Workflow: projects from R for Data Science.

# Chapter 3

# R Markdown and Reproducible Documentation

## 3.1 Communication with R

One of the benefits of using R is that is comes with its own documentation system, called R Markdown. Markdown has actually been around for a while. It is a markup language with plain text formatting syntax that allows it to be converted to many output formats, such as HTML. R Markdown adapts the Markdown language and incorporates the ability to include R code and output (and other languages) seamlessly.

## 3.2 R Markdown

The beauty of R Markdown is that it allows you to create reproducible reports of analysis, without copy and pasting code/output, and to export those reports into HTML, Word .docx, and PDF. Let's watch a short video to get a better overview.

R Markdown Introduction

Now let's create our first R Markdown document! Download the following document and save it into your Project folder: Introduction to R.Rmd

## 3.3 Learning More about R Markdown

R Markdown can also be used to make presentations, tutorials, and data dashboards. To learn more, check out R Markdown: The Definitive Guide.

You can also check out the R Markdown Cheatsheet and R Mardkown Reference Guide.

# Chapter 4

# Visualization using base graphics

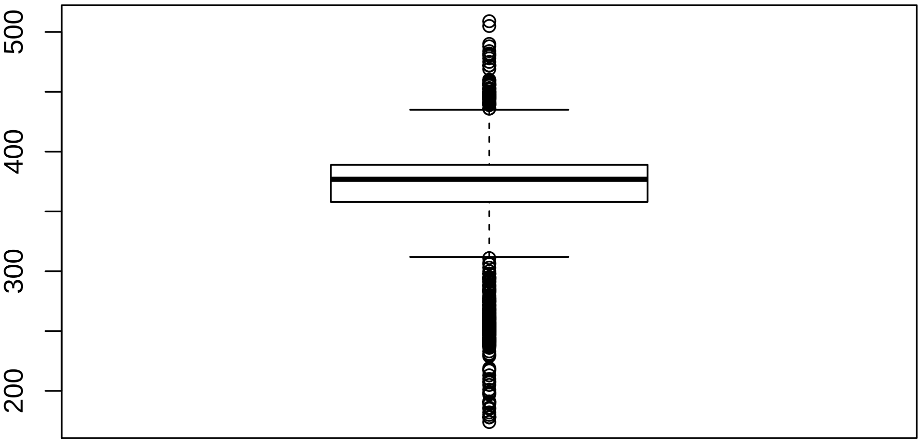
The base graphics often provide a very simple way for students to get plots quickly and explore data. We will continue using the `cleanTeslaBattery` data. Load it now and view it.

## 4.1 Histograms and boxplots

One wonderful thing about R is how intuitive the functions are, if you want a mean, the command is `mean`, if you want a boxplot, the command is ‘boxplot’.

Run the command ‘boxplot(cleanTeslaBattery$MaxRangeKM)’. You should get the following:

```
boxplot(cleanTeslaBattery$MaxRangeKM)
```



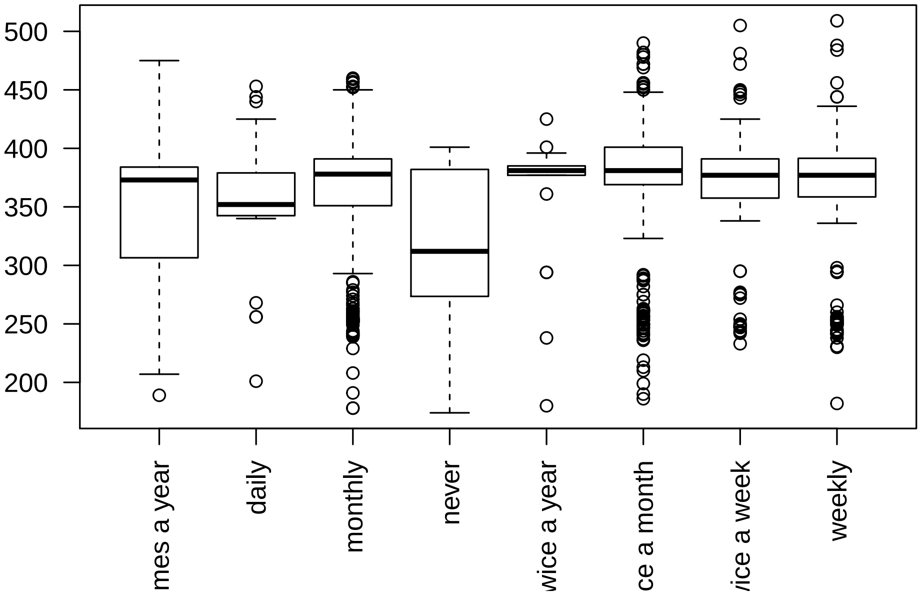
The x- and y-axis labels, main title, as well as color can be set using optional variables, xlab, ylab, main and color, respectively. Modify your command to `boxplot(cleanTeslaBattery$MaxRangeKM, xlab = "All", ylab = "KM", main = "Max Range", col = "green")`. R has many other colors built in and also accepts hexadecimals. Search the web to find some other possible colors to modify your graph.

Replacing `boxplot()` with `hist` will give the expected effect. Try it on one of the other quantitative variables.

- Note that histograms have an optional argument to control how many bins called breaks. Try adding the option 'breaks = 50'
- Set the number of breaks to 5, notice anything strange?
- Set your histogram to another color

One other feature that is nice is that comparisons between quantitative variables can easily be done as well. Suppose we want to compare the maximum range based on how frequently they supercharge their batteries. Enter the command

```
boxplot(MaxRangeKM ~ SuperchargeFreq, data = cleanTeslaBattery, las = 2)
```



- Note the relational notation `y ~ x`

- In many functions we can specify the dataset to avoid extra typing

- The `las` argument specifies turning the axis labels 90 degrees

- The optional axis labels are as before

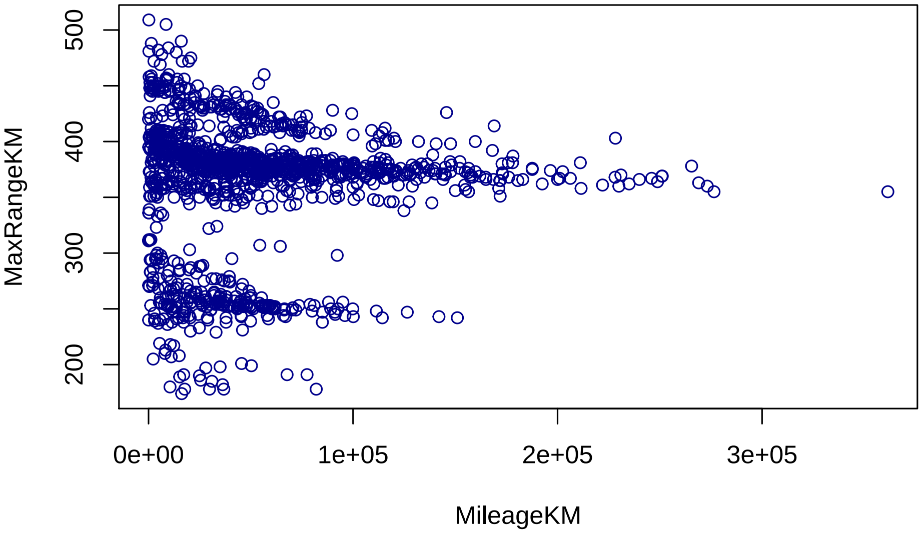
- Colors can still be specified, but now we need a list of colors, one for each box. Add the optional argument `col = heat.colors(8)`. Note you must specify one for each category. There are many color palettes. You can view some with the command `?colors`.

  Teaching tip: One of the most common problems students run into here is not putting the variables in the correct order in the 'y ∼ x'

## 4.2 Scatterplots

Scatterplots follow a very similar syntax as doing side-by-side boxplots, but using the `plot` function and replacing the qualitative explanatory variable with a quantitative one. Let's explore the effect of Mileage (`MileageKM`) and the Maximum Range (`MaxRangeKM`).

```
plot(MaxRangeKM ~ MileageKM, data = cleanTeslaBattery, col = "darkblue")
```



Note that the type of point can be controlled using the optional `pch = 2` command and changing the number for different point types. Color and labels are done as before. Try adjusting your colors and labelling your plot.

Another nice alternative for large datasets is the `smoothScatter`, however it doesn't support the specification of data and will require explicitly referring to the x and y using the data frame name and '$' symbol. Try it if you are interested.

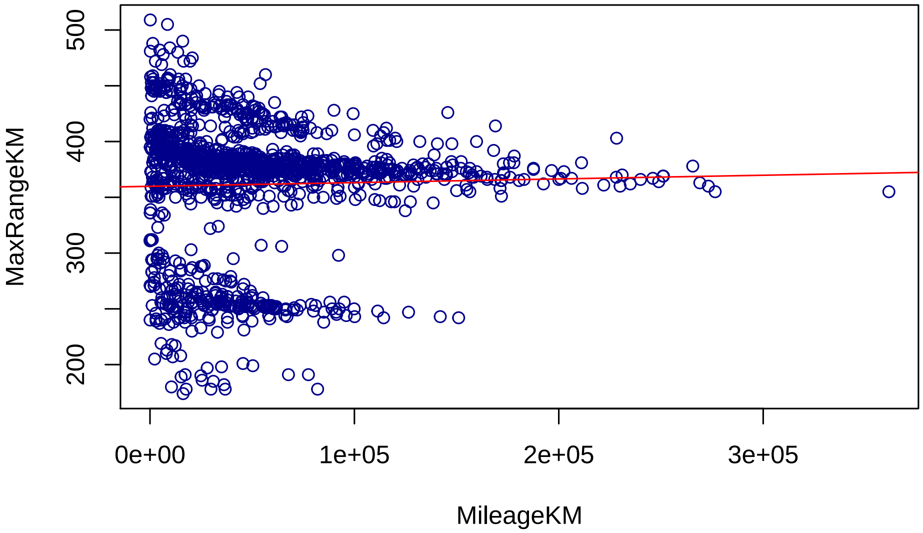
Clearly this graph shows multiple trends due to the different battery sizes, so we would want to separate these out separately for regression or use a more advanced regression model. For simplicity we are going to add a regression line without accounting for the car model.

Regression lines are implemented using the followng process:

1. Fit and save the model to the data using `lm()` (x and y are specified as in the plot). This model is used for all other steps.
2. Add the line to the scatterplot that has already been created using `abline()` applied to the model you have saved. Colors for the line can be specified as before.
3. Summarize the model using `summary()`.

Try it.

```
plot(MaxRangeKM ~ MileageKM, data = cleanTeslaBattery, col = "darkblue")
line = lm(MaxRangeKM ~ MileageKM, data = cleanTeslaBattery)
abline(line, col = "red")
```



```
summary(line)
```

```
Call:
lm(formula = MaxRangeKM ~ MileageKM, data = cleanTeslaBattery)

Residuals:
     Min       1Q   Median       3Q      Max
-186.380   -4.404   14.744   28.588  149.153

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.598e+02  2.389e+00 150.597   <2e-16 ***
MileageKM   3.313e-05  3.442e-05   0.962    0.336
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 58.81 on 1336 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.0006928, Adjusted R-squared:  -5.522e-05
F-statistic: 0.9262 on 1 and 1336 DF,  p-value: 0.336
```

```
Teaching tip:  students frequently forget to create the plot first.
```

The correlation can be computed using the `cor` function. Look up how to use it using `?cor` and try it.

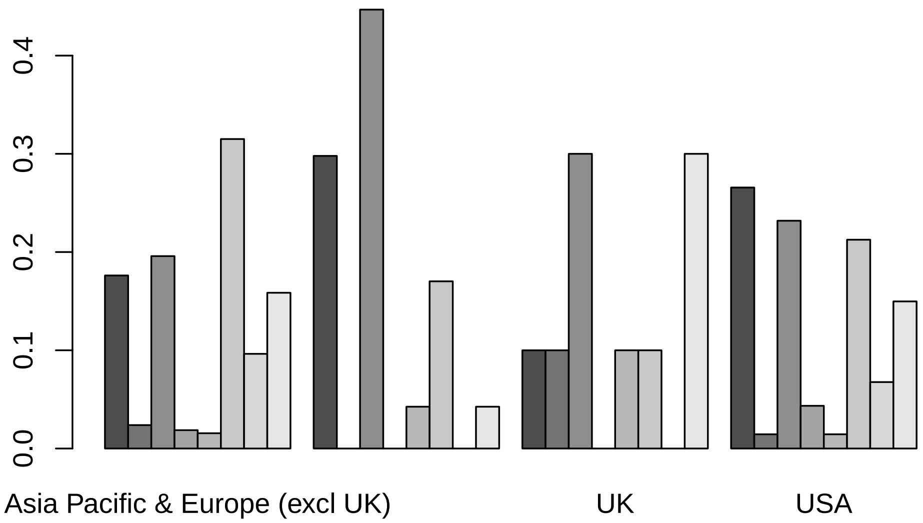
## 4.3 Bar graphs

Categorical data isn't much harder, but does require a preprocessing step for base graphics. Let's create a boxplot of the Supercharging Frequency. We will go straight to grouped bar plots. Let's summarize the relationship between supercharging frequency and the location. This is done in three steps:

1. tabulate (using `table`, stored as `tble`)
2. compute conditional percentages (`prop.table`, stored as `ptable`)
3. plot (`barplot`)

Try it:

```
tble = table(cleanTeslaBattery$SuperchargeFreq, cleanTeslaBattery$Location)
ptable = prop.table(tble,2)
barplot(ptable, beside = TRUE)
```



Note the organization of the table and the relationship between the rows and columns.

We can also add a legend and a location with the options `legend.text = TRUE` and `args.legend = list(x = "topright")`.

Try adding them to your plot.

Note, there are many options for the placement of the legend other than "topright" that you may want to use if the legend overlaps your bars. These include "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", and "center."

Set the main title and colors (the number of colors should be the same as the number of rows in the table). A quick way to get the number of colors is to count the number of levels of a factor. You can replace the count with `length(levels(vblname))`, and substitute your 'vblname'.

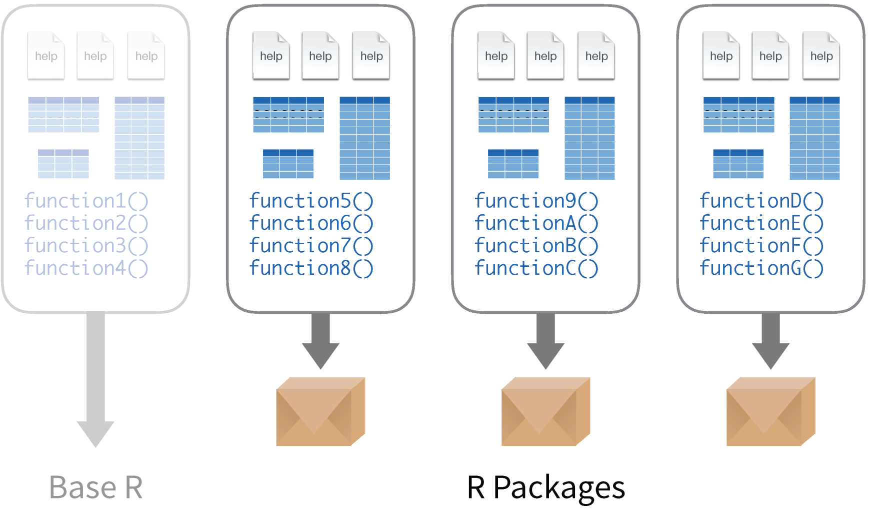
A bar plot for a single variable can be accomplished by only passing in one variable. Try it on one of the other categorical variables.

Chapter 5

# Data Visualization with ggplot2

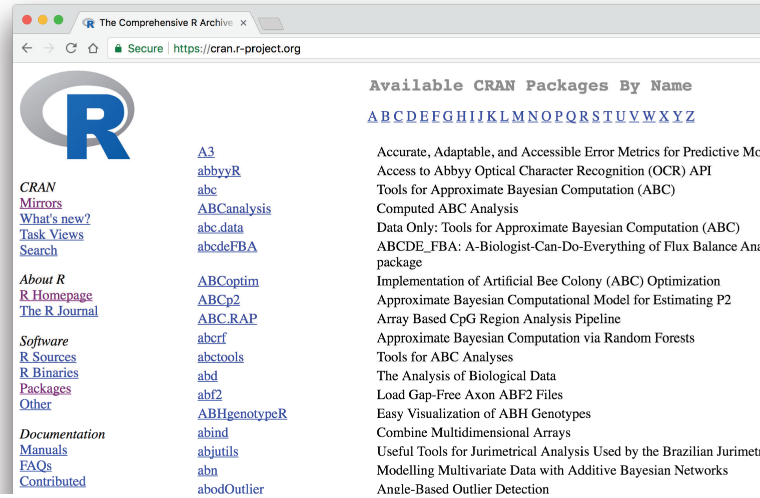
## 5.1 What is a package?

Packages are collections of R functions, data, and compiled code in a well-defined format. The directory where packages are stored is called the library. R comes with a standard set of packages, called the base package. Others are available for download and installation.



Packages that are curated, maintained, and reviewed are stored on the CRAN

(Comprehensive R Archive Network).



The Comprehensive R Archive
Secure https://cran.r-project.org

**Available CRAN Packages By Name**

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

CRAN
Mirrors
What's new?
Task Views
Search

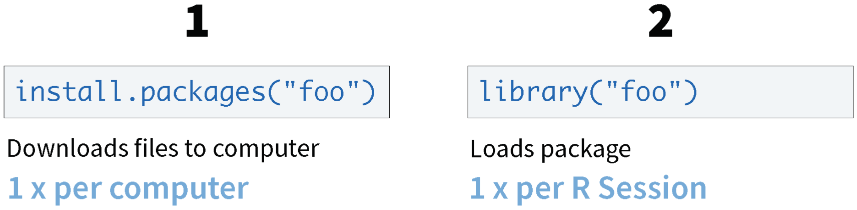
About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

| | |
|---|---|
| A3 | Accurate, Adaptable, and Accessible Error Metrics for Predictive Mo |
| abbyyR | Access to Abbyy Optical Character Recognition (OCR) API |
| abc | Tools for Approximate Bayesian Computation (ABC) |
| ABCanalysis | Computed ABC Analysis |
| abc.data | Data Only: Tools for Approximate Bayesian Computation (ABC) |
| abcdeFBA | ABCDE_FBA: A-Biologist-Can-Do-Everything of Flux Balance Ana package |
| ABCoptim | Implementation of Artificial Bee Colony (ABC) Optimization |
| ABCp2 | Approximate Bayesian Computational Model for Estimating P2 |
| ABC.RAP | Array Based CpG Region Analysis Pipeline |
| abcrf | Approximate Bayesian Computation via Random Forests |
| abctools | Tools for ABC Analyses |
| abd | The Analysis of Biological Data |
| abf2 | Load Gap-Free Axon ABF2 Files |
| ABHgenotypeR | Easy Visualization of ABH Genotypes |
| abind | Combine Multidimensional Arrays |
| abjutils | Useful Tools for Jurimetrical Analysis Used by the Brazilian Jurimetr |
| abn | Modelling Multivariate Data with Additive Bayesian Networks |
| abodOutlier | Angle-Based Outlier Detection |

Once installed, packages have to be loaded into the session to use the functions stored within the package.



```
install.packages("foo")
```

```
library("foo")
```

If you ever need R to do something that is not obvious, there is probably a package for that. Many packages also provide vignettes or publish background and examples in the Journal of Statistical Software.

One of the more popular packages of packages is the `tidyverse`, which includes packages such as `dplyr`, `ggplot2`, and `readr` which make using R for data cleaning, graphics, and data science. We will explore a few of these packages today, starting with `ggplot2` and the grammar of graphics.

## 5.2 The Grammar of Graphics

A data visualization is a set of visual **geometries** whose **aesthetics** are mapped from data.

### 5.2.1 Geometry

- A geometry is a visual entity in space.
- Some common geometries encountered in data visualizations:
  - Point
  - Line
  - Bar

### 5.2.2 Aesthetics

- An aesthetic is a visual attribute of a geometry
- Common aesthetics:
  - Position on horizontal (X)
  - Position on vertical (Y)
  - Shape
  - Size
  - Color
    - Hue
    - Saturation (“intensity”)
    - Value (“brightness”)
  - Text
- Not all aesthetics are available for every geometry

### 5.2.3 Data

- To visualize, must have data in row-by-column format where:
  - Rows represent cases: at most one geometry per case (assuming no aggregation)
  - Columns represent variables: to be mapped to aesthetic attributes

- Differences in geometry aesthetics map to differences in data variables
- Available mappings depend on whether data variable is continuous (height) or discrete (race)
- The following caveats apply:
  - An aesthetic attribute can be mapped back to at most one variable
  - A variable can be mapped to more than one aesthetic
  - Not all mappings make sense

## 5.3 Package ggplot2

The package `ggplot2` stands for grammar of graphic plots. It works to layer details onto a graphic map data to specific aesthetics using specific geometries. If you have not already done so, install the `ggplot2` package.

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

Open up the the R Notebook for Data Visualization with ggplot2 and follow along.

Now load the package into R using either `library()` or `require()`

```
library(ggplot2)
```

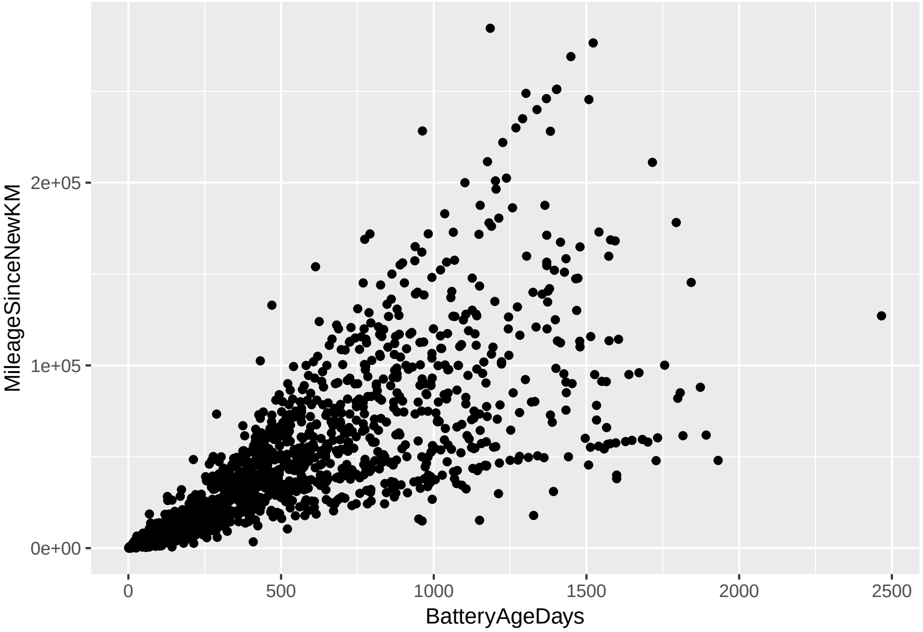
Let's make a graphic for the Tesla data:

```
tesla<-read.csv("Data/cleanTeslaBattery.csv")
```

**Your Turn 1** Talk in a group - what relationship do you expect to see in the `tesla` data between the age of the battery (`Battery AgeDays`) and the amount of mileage put on the car since the battery was installed (`MileageSinceNewKM`)? (No peeking!)

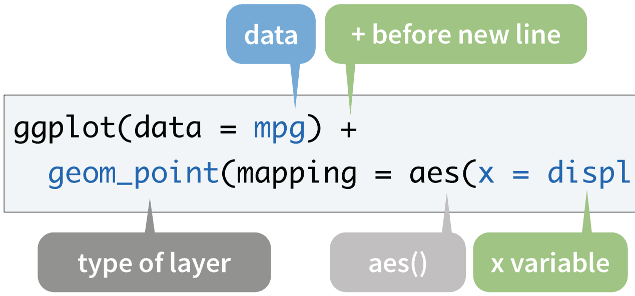
Run the following code in your notebook to make a graph, paying strict attention to spelling, capitalization, and parentheses.

```
ggplot(data=tesla) +
  geom_point(mapping=aes(x=BatteryAgeDays, y=MileageSinceNewKM))
```



Notice a few key things.

1. `ggplot(data)` initializes the plot
2. adding a `+` after a line of ggplot code adds a new layer
3. start with the geometry
4. then specify the aesthetics



```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ
```

CC by RStudio

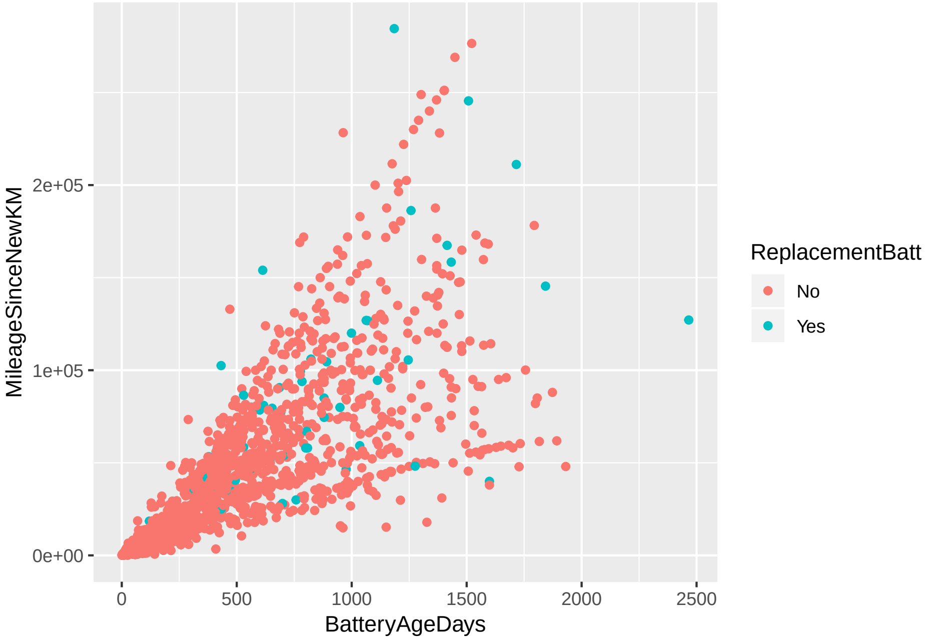
Here is a similar graphic code with annotation.

Therefore, you can use this code template to make thousands of graphs with **ggplot2**.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

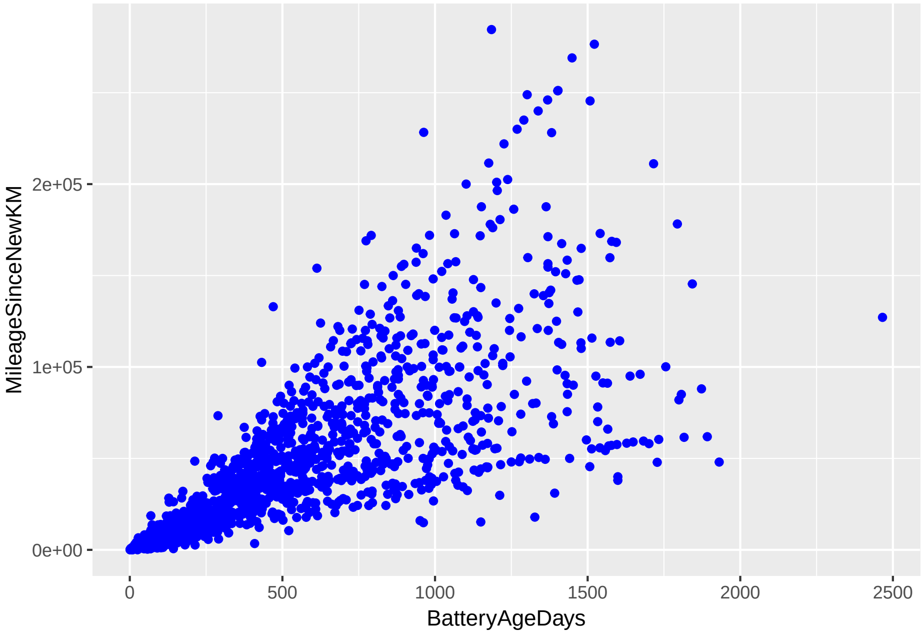
Let's add on some more layers! What if we want to make the variable Replacement Battery to a color aesthetic?

```
ggplot(data=tesla) +
  geom_point(mapping=aes(x=BatteryAgeDays, y=MileageSinceNewKM, color=ReplacementBatt)
```



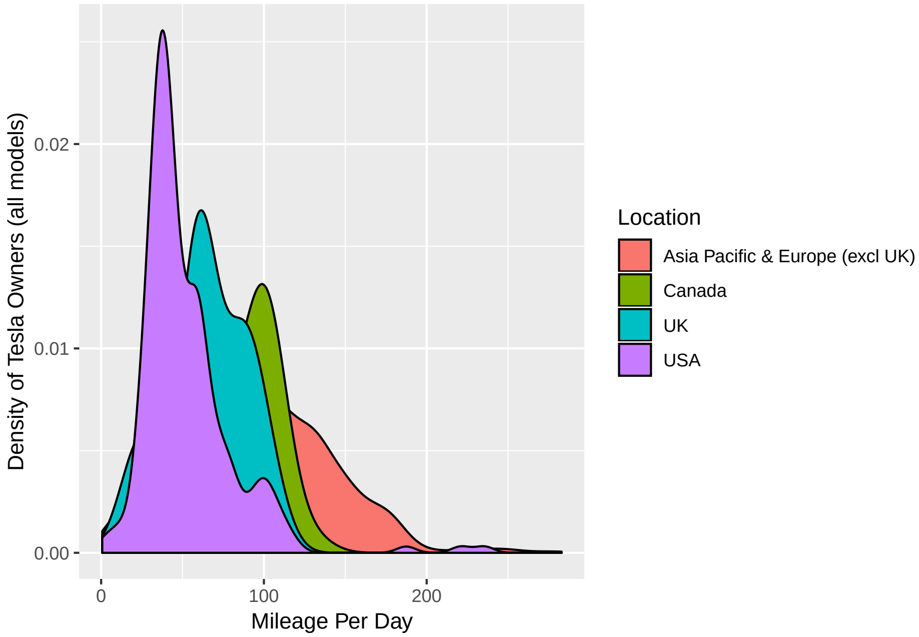
If we didn't care about mapping the new variable to an aesthetic, but wanted to change the color, that would come outside of the `aes()` function.

```r
ggplot(data=tesla) +
  geom_point(mapping=aes(x=BatteryAgeDays, y=MileageSinceNewKM), color="blue")
```



There are a lot of resources available, including a the “ggplot2 Cheatsheet”

**Your Turn 2** Make the following density plot, using the Cheatsheet to help you. What does the plot tell you about the Tesla driver habits in different regions?



### 5.3.1 Layering Graphics

To get a better idea about how `ggplot2` layers aesthetics and and specific geometries onto a graphic, check out the ggplot flipbook which demonstrates how each layer modifies the graphic.

## 5.4 Additional Resources

There are a lot of resources for `ggplot2`, here are three good places to start:

1. The data visualisation and graphics for communication chapters in R for data science. R for data science is designed to give you a comprehensive introduction to the `tidyverse`, and these two chapters will you get up to speed with the essentials of `ggplot2` as quickly as possible.

2. If you’d like to take an online course, try Data Visualization in R With ggplot2 by Kara Woo.

3. If you want to dive into making common graphics as quickly as possible, I recommend The R Graphics Cookbook by Winston Chang. It provides a set of recipes to solve common graphics problems.

If you've mastered the basics and want to learn more, read ggplot2: Elegant Graphics for Data Analysis. It describes the theoretical underpinnings of `ggplot2` and shows you how all the pieces fit together. This book helps you understand the theory that underpins `ggplot2`, and will help you create new types of graphics specifically tailored to your needs. The book is not available for free, but you can find the complete source for the book at https://github.com/hadley/ggplot2-book.

# Chapter 6

# Statistical Inference

In this section we will cover some of the basic tools for statistical inference covered in an introductory statistics class.

## 6.1 ANOVA

We will demonstrate the use of ANOVA, but $t$-tests are quite similar.

`Teaching Tip:  When teaching non-majors incorporating data from the research of faculty teaching`

For this section we are going to work with a dataset from a cancer drug development project that is part of a biology faculty member's ongoing research at our campus, the data came from one of the student projects. Below is an image of one of the tumors.

The measurements in the file are the percent of tumor growth under various drug treatments. TumorGrowth.csv. Load the data now and inspect the dataset.

Construct a graphical summary of your data using one of the tools from earlier.

We will also construct a numerical summaries of each group using the `doBy` package. Install and load it now.

For this I define a helper function:

```
msd = function (x) {c(m = mean(x), stdev = sd(x))}
```

We apply this function using `summaryBy`.

To run an ANOVA use the command `aov()` by specifying the relationship and the `data`, as you did for summarizing the data.

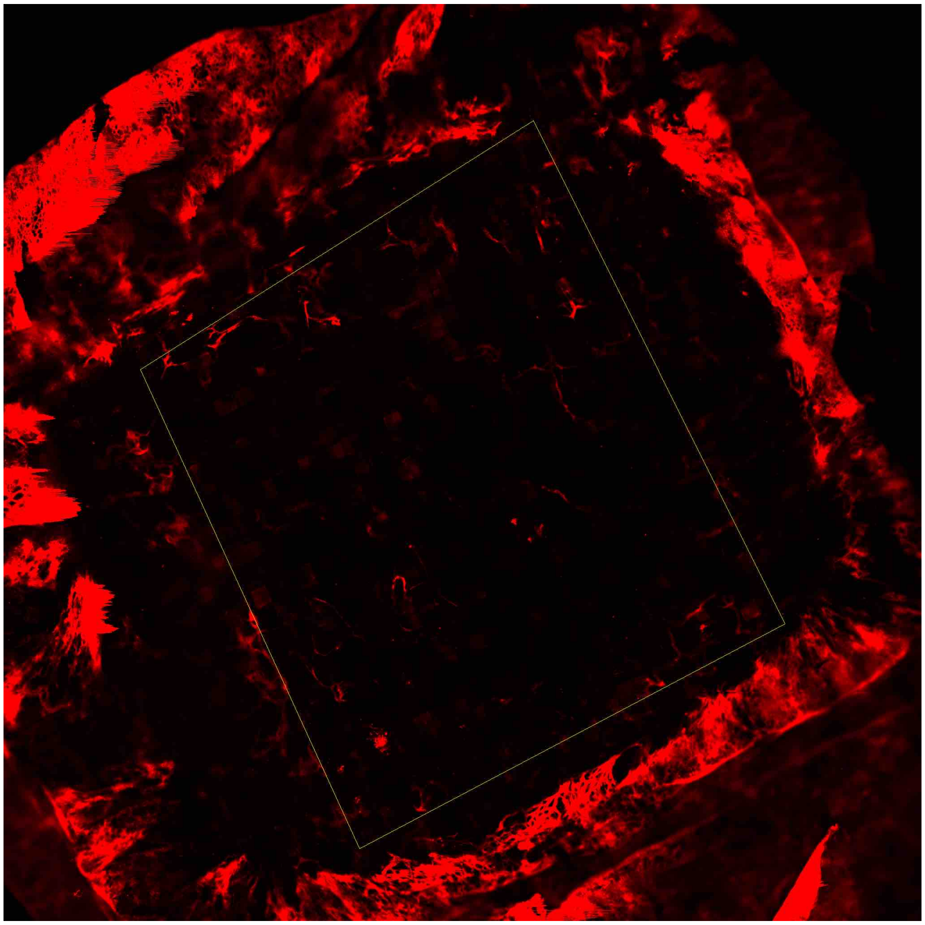


Figure 6.1: CancerTumor

```
mod = aov(Growth ~ Treatment, data = Tumor)
```

Note that there is something missing that most intro stats students are asked for, what is it?

This can be fixed by wrapping the `aov` command with a `summary()`. Try it.

Post-hoc tests can be run by saving the ANOVA model to a variable and then passing it to `TukeyHSD()`.

Try it here:

```
TukeyHSD(mod)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Growth ~ Treatment, data = Tumor)

$Treatment
                        diff          lwr          upr     p adj
Control-Avastin  0.002222222 -0.029180472  0.033624916 0.9976809
Tor+Ava-Avastin -0.037222222 -0.068624916 -0.005819528 0.0137317
Torisel-Avastin -0.004444444 -0.035847138  0.026958250 0.9821827
Tor+Ava-Control -0.039444444 -0.070847138 -0.008041750 0.0080067
Torisel-Control -0.006666667 -0.038069361  0.024736027 0.9437365
Torisel-Tor+Ava  0.032777778  0.001375084  0.064180472 0.0374652
```

There are other functions for the other types of post-hoc tests.

## 6.2 Testing for Proportions

For simplicity we are going to continue with the `cleanTeslaBattery`. Suppose we want to look at the relationship between `Country` and `Model`.

Create a two-way table and a bar plot for these variables.

```
cleanTeslaBattery = read.csv("Data/cleanTeslaBattery.csv")
tbl = table(cleanTeslaBattery$Location,cleanTeslaBattery$Model)
tbl
```

```

                                 Model 3 LR Model S 100D Model S 60
  Asia Pacific & Europe (excl UK)         0            3         22
  Canada                                  0            0          3
  UK                                      0            0          1
  USA                                     4            2         20
```

```
                                Model S 60D Model S 70 Model S 70D
Asia Pacific & Europe (excl UK)           0          8          99
Canada                                    1          0           0
UK                                        1          0           0
USA                                       4          0           4

                                Model S 75 Model S 75D Model S 85
Asia Pacific & Europe (excl UK)         28          20        376
Canada                                   1           0         20
UK                                       0           1          7
USA                                     10           7         72

                                Model S 85D Model S 90 Model S 90D
Asia Pacific & Europe (excl UK)          88          1          90
Canada                                    9          2           2
UK                                        2          0           0
USA                                      13          0          15

                                Model S 90D 2015 Model S P100D
Asia Pacific & Europe (excl UK)                3             8
Canada                                         0             0
UK                                             0             0
USA                                           10             0

                                Model S P85 Model S P85+ Model S P85D
Asia Pacific & Europe (excl UK)         141           45           75
Canada                                    5            1            1
UK                                        0            0            0
USA                                      32            2           15

                                Model S P90D Model X 100D Model X 60D
Asia Pacific & Europe (excl UK)            4            7           0
Canada                                     0            1           1
UK                                         0            0           0
USA                                        3            4           0

                                Model X 75D Model X 90D Model X P90D
Asia Pacific & Europe (excl UK)           4          11            1
Canada                                    0           0            0
UK                                        0           0            0
USA                                       0           5            0

                                Unspecified 85 kWh
Asia Pacific & Europe (excl UK)                 24
Canada                                           0
UK                                               0
```

```
  USA                                              0
```

Chi-squared is as simple as running `chisq.test()` on the two-way table.

```
chisq.test(tbl)
```

```

    Pearson's Chi-squared test

data:  tbl
X-squared = 254.16, df = 69, p-value < 2.2e-16
```

Tests for proportions require the `prop.test` command and require just inputting the desired counts and/or population proportions. We won't do an application here, but if 23 of 120 have a trait and we would like to see if the proprtion is significantly greater than 0.15, we might use the command

```
prop.test(23,120, p=0.15, alternative = "greater")
```

```

    1-sample proportions test with continuity correction

data:  23 out of 120, null probability 0.15
X-squared = 1.3235, df = 1, p-value = 0.125
alternative hypothesis: true p is greater than 0.15
95 percent confidence interval:
 0.136025 1.000000
sample estimates:
        p
0.1916667
```

As with the t-tests, notice that the confidence interval is an added bonus, but requires the two-sided option.

A two-sample test for proportions may be run similarly with a slight modification:

```
prop.test(c(31,40),c(60,85), alternative = "greater")
```

```

    2-sample test for equality of proportions with continuity
    correction

data:  c(31, 40) out of c(60, 85)
X-squared = 0.14289, df = 1, p-value = 0.3527
alternative hypothesis: greater
95 percent confidence interval:
 -0.1066671  1.0000000
sample estimates:
   prop 1    prop 2
0.5166667 0.4705882
```

where the first list specifies the counts with the trait in each of the two groups and the second list consists of the sample sizes for each of the two groups.

## 6.3 Linear Regression

Let's go back to the Tesla Battery Data from earlier. We did linear regression when we added the regression line earlier, but now we will explore some of the other available tools with regression.

```
cleanTeslaBattery = read.csv("Data/cleanTeslaBattery.csv")
```

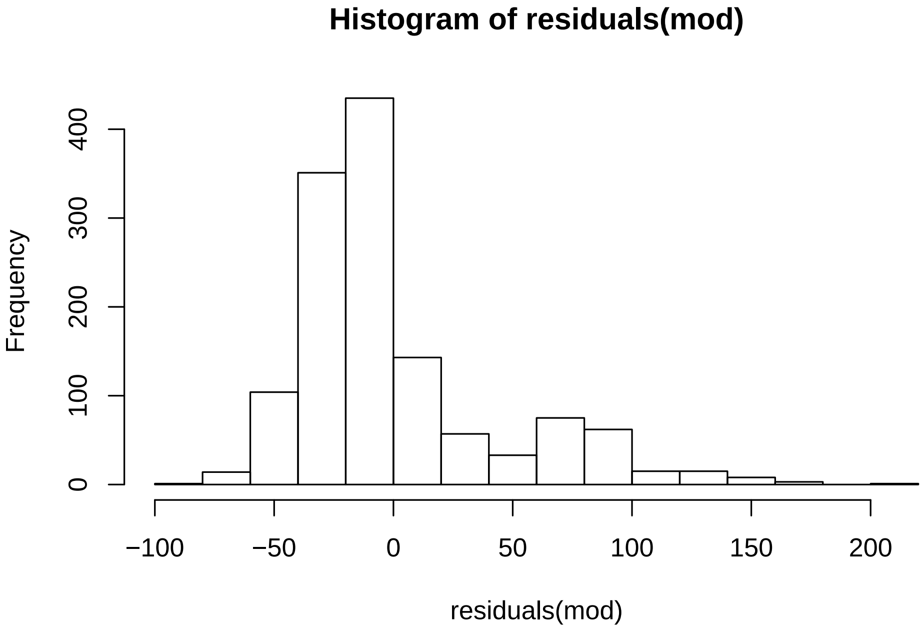
Implement the linear model from before to look at the relationship between the Mileage (`MileageKM`) and Watt Hours Per KM (`WattHoursPerKM`) and assign the model to the variable `mod`

```
mod = lm(WattHoursPerKM ~ MileageKM, data = cleanTeslaBattery)
summary(mod)
```

```

Call:
lm(formula = WattHoursPerKM ~ MileageKM, data = cleanTeslaBattery)

Residuals:
    Min      1Q  Median      3Q     Max 
-80.060 -26.278 -11.197   8.352 206.763 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.439e+02  1.743e+00 139.901  < 2e-16 ***
MileageKM   -1.766e-04  2.500e-05  -7.063 2.63e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 42.5 on 1315 degrees of freedom
  (22 observations deleted due to missingness)
Multiple R-squared:  0.03655,	Adjusted R-squared:  0.03582 
F-statistic: 49.89 on 1 and 1315 DF,  p-value: 2.628e-12
```

We can also easily inspect the residuals by calling the `residual()` function on the model. Try it:

```
hist(residuals(mod))
```



Additionally transforms of any of the variables can be performed in the following way:

```
lm(WattHoursPerKM ~ log(MileageKM), data = cleanTeslaBattery)
```

```

Call:
lm(formula = WattHoursPerKM ~ log(MileageKM), data = cleanTeslaBattery)

Coefficients:
   (Intercept)  log(MileageKM)  
        286.59           -5.02  
```

```
lm(sqrt(WattHoursPerKM) ~ MileageKM, data = cleanTeslaBattery)
```

```

Call:
lm(formula = sqrt(WattHoursPerKM) ~ MileageKM, data = cleanTeslaBattery)

Coefficients:
(Intercept)    MileageKM  
  1.555e+01   -5.510e-06  
```

Finally, we left out the correlation coefficient. The call for the correlation coefficient is a little different as it doesn't use relations `cor(x,y, use = "pairwise.complete.obs", method = "pearson")`. Try the command here for the Mileage and Watt Hours Per KM.

Note that the `use` option has been specified to omit pairs of correlations where ther is an NA, and the `method` controls the type of correlation.

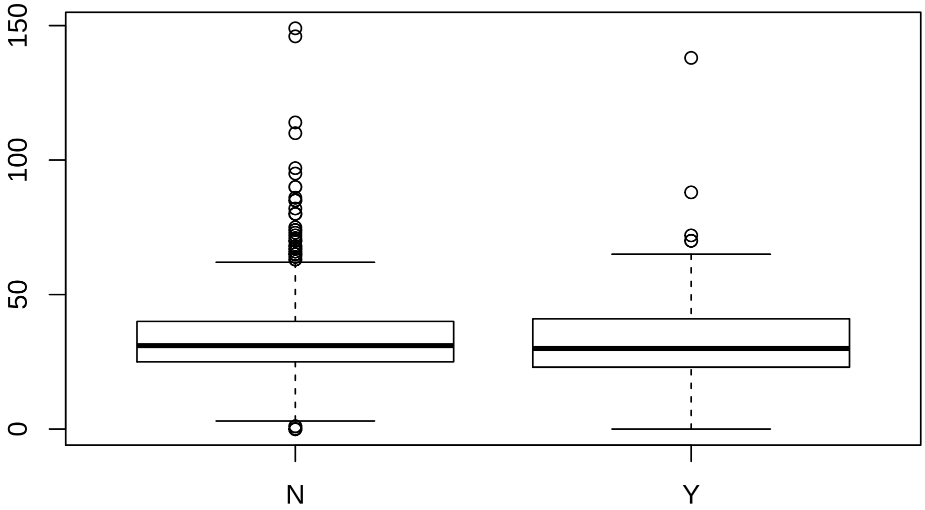
## 6.4 t-tests

Here is a very quick overview of t-tests. We are going to take a look at the difference in weight gain be smoking and non-smoking mothers in Kings County. Load the dataset

```
Kings <- read.csv("Data/KingCounty2001.csv")
```

Take a look at the relationship between `smoker` and `wgain`.

```
boxplot(wgain ~ smoker, data = Kings )
```



```
summary(Kings$smoker)
```

```
   N    Y 
2325  175 
```

two-sample *t*-test syntax is nearly identical to ANOVA syntax, except we need to replace the `aov` with `t.test` and specify the type of alternative, either “greater”, “less” or “two.tailed”, using the `alternative` option.

```
t.test(wgain ~ smoker, data = Kings, alternative = "greater")
```

```

    Welch Two Sample t-test

data:  wgain by smoker
t = -0.65319, df = 189.21, p-value = 0.7428
```

```
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -3.083485       Inf
sample estimates:
mean in group N mean in group Y
       32.21806        33.09143
```

Note that confidence intervals are included, but the alternative should be "two.sided."

One-sided *t*-tests require specifying only a single quantitative variable and also specifying the value from the null hypothesis. An example for testing that the mean `wgain` is different from 9.4 is

```
t.test(Kings$wgain, mu = 9.4, alt = "two.sided")
```

```

    One Sample t-test

data:  Kings$wgain
t = 85.31, df = 2499, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 9.4
95 percent confidence interval:
 31.75331 32.80509
sample estimates:
mean of x
  32.2792
```

Other options like the `conf.level` are described under the help for the function.

Try computing a 90% confidence interval for the mean `gest`.

## 6.5 Paired t-test

Paired t-tests require data in the wide format, that is two columns side by side, perhaps like this

```
pairdat = data.frame(pre=c(23,20,30,29),post=c(25,21,27,31))
pairdat
```

```
  pre post
1  23   25
2  20   21
3  30   27
4  29   31
```

Note this is for example only, not for validity.

```
t.test(pairdat$pre, pairdat$post, paired=TRUE)
```

```

	Paired t-test

data:  pairdat$pre and pairdat$post
t = -0.42008, df = 3, p-value = 0.7027
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.287869  3.287869
sample estimates:
mean of the differences 
                   -0.5 
```

# Chapter 7

# Classroom hints and pedagogy

## 7.1 Introductory General Statistics

- Make it a go to tool the first day. First Day Handout
- Highlight common errors and how to fix them
- Have students us it regularly.
- Integrate it regularly in class so students are familiar with output

- Classroom demos, e.g. the Central Limit Theorem or the effects of outliers
- Follow – Apply Example Lab
- Choose your own adventure to accomodate other departmental needs. Excel Version of Lab
- Avoid the black box, show in class how R produces output that matches with the computations/methods they have seen in class.

---

Group work in R R Task Card

Shiny Web Apps

# Chapter 8

# R Packages for Teaching and Learning

## 8.1 Base R vs. Packages for Learning

There are a lot of options in base R, but often they can be clunky and a bit of a challenge for students. Download the Packages for Learning Notebook

Let's look at an example (you can find the data called `KingCounty2011.csv` in linked here: In 2001, a sample of 2500 births from King County, Washington contained information on both the mother and the infant at birth.

```
king<-read.csv("Data/KingCounty2001.csv")
```

Suppose we want to compare the birth weight (`bwt`) of a baby between mothers who smoke (Y) versus do not smoke (N) (`smoker`).

In base R, here is what the code would look like:

```
mean(king$bwt[king$smoker=="Y"]) #smoke
```

```
[1] 3185.737
```

```
mean(king$bwt[king$smoker=="N"]) #do not smoke
```

```
[1] 3431.201
```

How would we calculate the standard deviation as well?

```
sd(king$bwt[king$smoker=="Y"]) #smoke
```

```
[1] 583.4798
```

```
sd(king$bwt[king$smoker=="N"]) #do not smoke
```

```
[1] 553.8304
```

It quickly becomes cumbersome to do this for multiple summary statistics. So what if there is a better way?

## 8.2 Package mosaic

The `mosaic` package was developed by Randall Pruim, Danny Kaplan, and Nicholas Horton. The goal of Project mosiac is to support the learning of R in colleges and universities. Let's try our previous problem using functions in `mosaic`.

First, install the package if you haven't done so already.

```
install.packages("mosaic")
```

Now, before we load the package, let's take a peak at the `mean()` function in base R. What error results?

```
mean(king$bwt~king$smoker)
```

```
     argument is not numeric or logical: returning NA[1] NA
```

What does that error message mean?

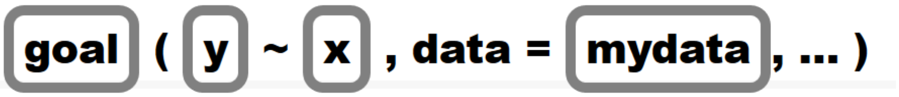
Now load the `mosaic` package and note the message provided when loaded:

```
library(mosaic)
```

Notice that it states that

> The following objects are masked from package:base:
> max, mean, min, prod, range, sample, sum

Package Mosaic overwrites several of the base package functions with new functions of the same name to make them adaptable to the formula notation.



We can now run our `mean()` function (overwritten by `mosaic`)

```
mean(bwt~smoker, data=king)
```

```
       N        Y 
3431.201 3185.737 
```

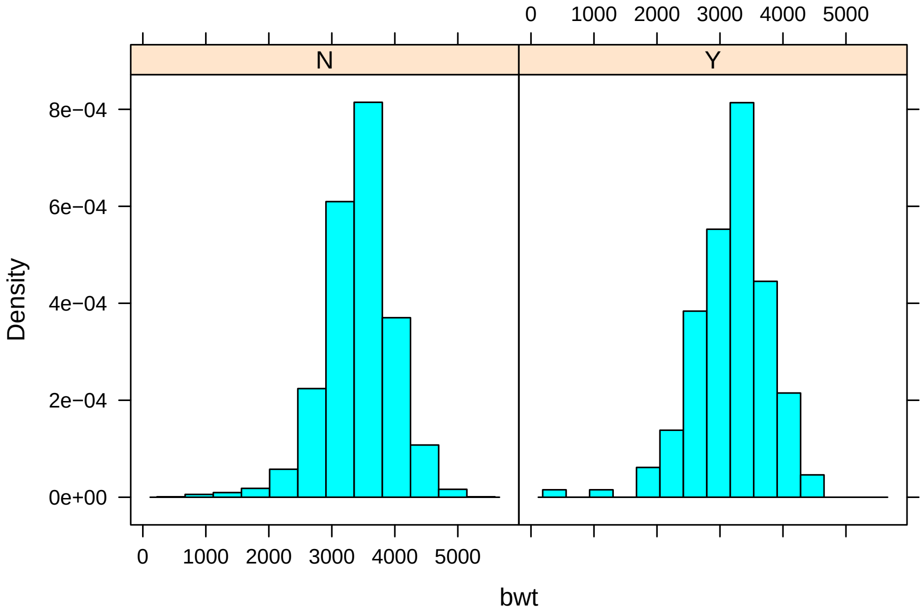
Many of the summary statistic functions of base R will now work in function form. `mosaic` also adds new functions, such as `favstats`

```
favstats(bwt~smoker, data=king)
```

```
  smoker min     Q1 median   Q3  max     mean       sd    n missing
1      N 255 3118.0   3459 3770 5175 3431.201 553.8304 2325       0
2      Y 414 2856.5   3275 3544 4508 3185.737 583.4798  175       0
```

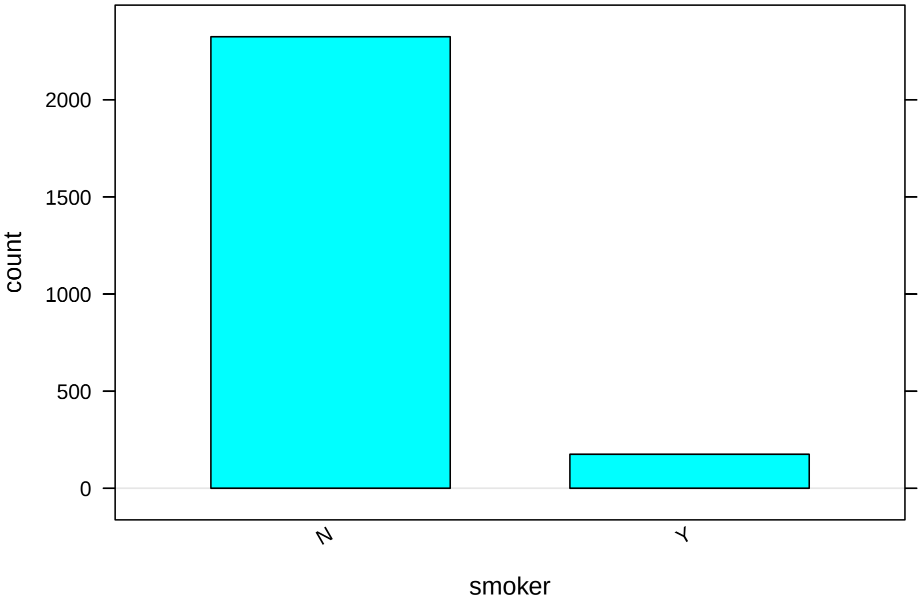
In addition, it allows for some simple graphics that allow faceting:

```
histogram(~bwt|smoker, data=king)
```



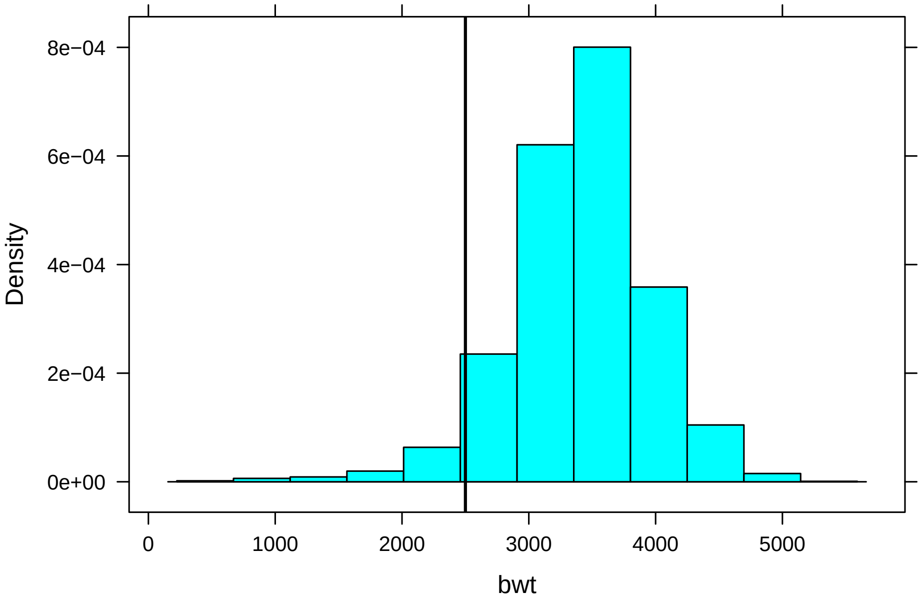
and easy bar graphs without the need to create a count table:

```
bargraph(~smoker, data=king)
```



and overlay summary values:

```r
histogram(~bwt, v=2500, data = king)
```



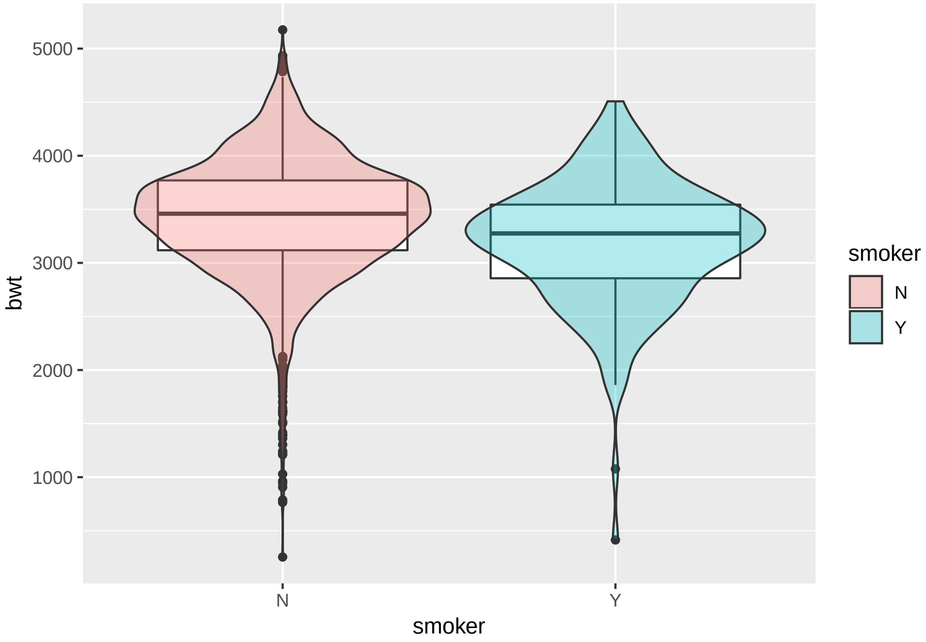
## 8.3 Package ggformula

Just like mosaic, ggformula was written to get students doing powerful visualization quickly, without having to learn the ins and outs of ggplot2 or even base R. While mosaic has some graphing functionality, ggformula serves as an overlay for ggplot2, allowing the user to create quality graphics and to support multivariate reasoning via formulas. To learn more, check out the vignette from the ggformula package.
Install the package and load it into R.

```
#install.packages("ggformula")
library(ggformula)
```

You can easily "pipe" using the %>% symbol to overlay two graphs and you can use many of the ggplot2 arguments within the ggformula functions. For example:

```
gf_boxplot(bwt~smoker, data=king) %>%
  gf_violin(bwt~smoker, data=king, fill=~smoker, alpha=0.3)
```



Try out a few other plots using the King County birth weight data and ggformula.

To learn more about the pipe %>% in R, read the Pipes chapter in R for Data Science.

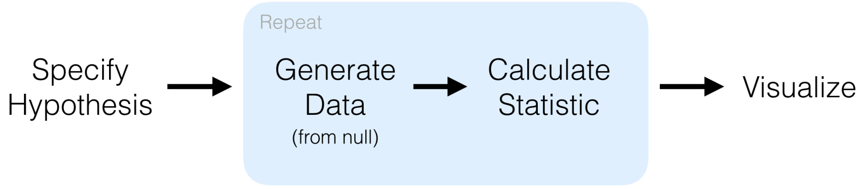


Figure 8.1: `infer` Grammar for Inference

## 8.4 Package infer

The objective of the `infer` package is to perform statistical inference using an expressive statistical grammar that mimics the `tidyverse` design framework. The goal is to have students run a hypothesis test by using functions that follow these steps:

Install the package if you haven't done so already.

```r
install.packages("infer")
```

and load the `infer` package:

```r
library(infer)
```

Let's look at an example, to compare the mean birth weights for infants born to smokers and non-smokers.

First, let's calculate the test statistic, what would we want to compare?

```r
king %>%
  specify(bwt ~ smoker) %>%
  calculate(stat = "diff in means", order = c("Y", "N")) -> d_hat
```

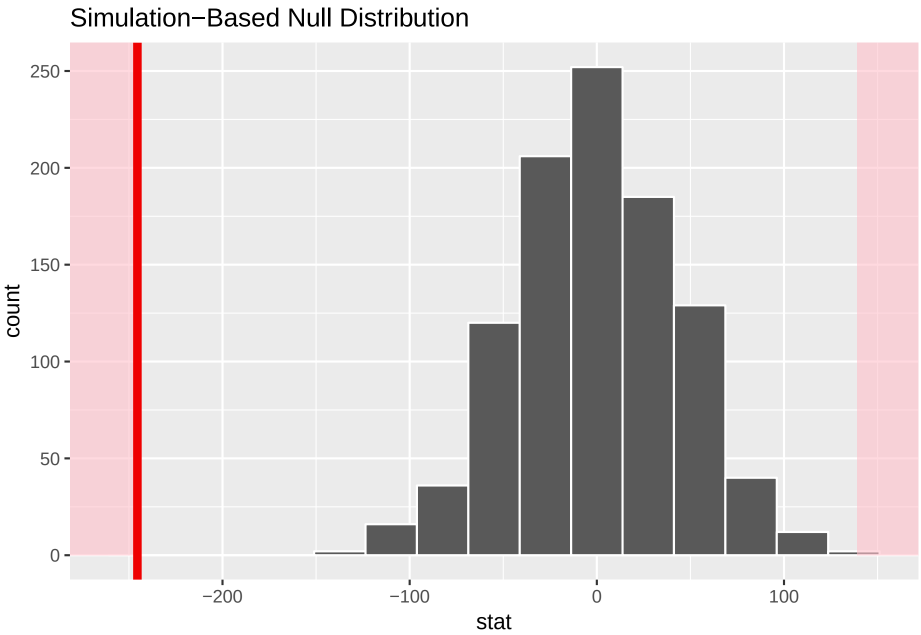
Next we generate data under the null hypothesis and calculate the test statistic under the null hypothesis to determine the "null distribution" of the null hypothesis. What is our null hypothesis?

$H_0 : \mu_{smoker} = \mu_{nonsmoker}$
$H_0 : \mu_{smoker} \neq \mu_{nonsmoker}$

```r
king %>%
  specify(bwt ~ smoker) %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute") %>%
  calculate(stat = "diff in means", order = c("Y", "N")) -> null_dist
```

Finally, we can visualize the null distribution and p-value. What conclusion should we draw from our inference?

```
visualise(null_dist) +
  shade_p_value(obs_stat = d_hat, direction = "two_sided")
```



You can learn more about the infer package for different tests by reading the `infer` vignettes.

# Chapter 9

# Packages for Learning R - Swirl

`swirl` is a package designed to let you learn R within R. Check out the Swirl webpage to learn more. To get started, you just have to load the library (after you install it of course).

```r
install.packages("swirl")
library(swirl)
```

# Chapter 10

# Untidy Data

## 10.1 Tidy Data

> “Happy families are all alike; every unhappy family is unhappy in its own way.” —- Leo Tolstoy
>
> “Tidy datasets are all alike, but every messy dataset is messy in its own way.” —- Hadley Wickham

R works best with “tidy data”. There are three interrelated rules which make a dataset tidy:

1. Each variable must have its own column.

2. Each observation must have its own row.

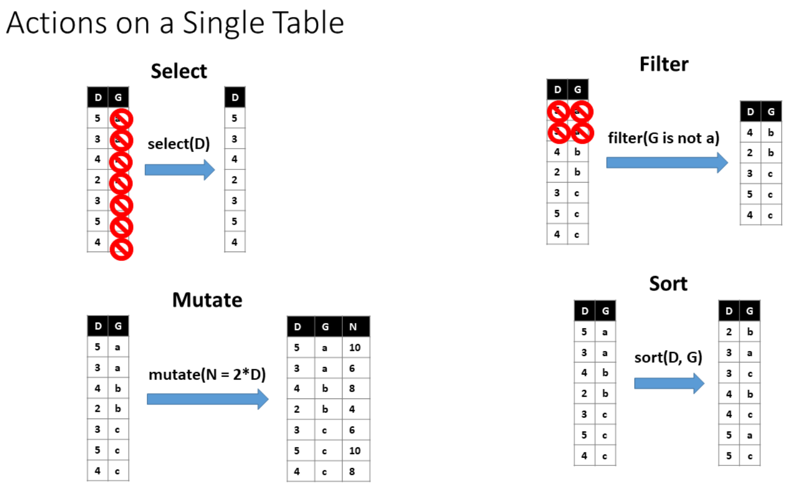
3. Each value must have its own cell.

R for Data Science

Most data does not start out “tidy”. Fortunately, the `tidyverse` was designed to help with the process of turning messy data into tidy data. Packages such as `dplyr`, `tidyr`, `lubridate`, `forcats`, and `stringr` allow the user to clean up data and eventually turn it into nice and tidy data.

It would be impossible to teach you how to clean data in all circumstances in the 15 minutes we have, but I wanted to overview the grammar of data cleaning for you. The functions in `tidyr` and `dplyr` mimic the grammar of data cleaning to make it easy to learn.

- Select - choose specific columns



Actions on a Single Table

Select

| D | G |
|---|---|
| 5 | |
| 3 | |
| 4 | |
| 2 | |
| 3 | |
| 5 | |
| 4 | |

select(D)

| D |
|---|
| 5 |
| 3 |
| 4 |
| 2 |
| 3 |
| 5 |
| 4 |

Filter

| D | G |
|---|---|
| | |
| | |
| 4 | b |
| 2 | b |
| 3 | c |
| 5 | c |
| 4 | c |

filter(G is not a)

| D | G |
|---|---|
| 4 | b |
| 2 | b |
| 3 | c |
| 5 | c |
| 4 | c |

Mutate

| D | G |
|---|---|
| 5 | a |
| 3 | a |
| 4 | b |
| 2 | b |
| 3 | c |
| 5 | c |
| 4 | c |

mutate(N = 2*D)

| D | G | N |
|---|---|---|
| 5 | a | 10 |
| 3 | a | 6 |
| 4 | b | 8 |
| 2 | b | 4 |
| 3 | c | 6 |
| 5 | c | 10 |
| 4 | c | 8 |

Sort

| D | G |
|---|---|
| 5 | a |
| 3 | a |
| 4 | b |
| 2 | b |
| 3 | c |
| 5 | c |
| 4 | c |

sort(D, G)

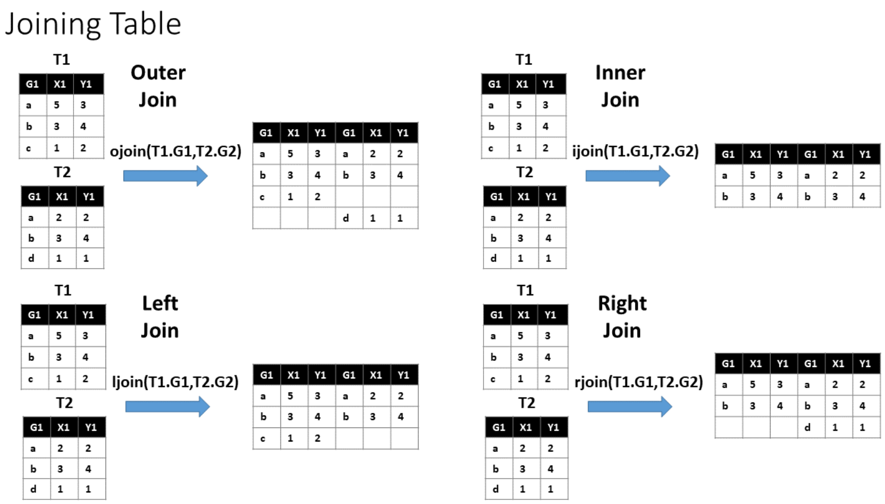
| D | G |
|---|---|
| 2 | b |
| 3 | a |
| 3 | c |
| 4 | b |
| 4 | c |
| 5 | a |
| 5 | c |

Figure 10.1: from Winona State

- Filter - choose specific rows
- Sort/Arrange - order the rows in the columns
- Mutate - add a new column
- Join - join two data tables together
- Gather - gather column names that are values instead of variables into a single column
- Spread - spread out observation values to new column variables with they are spread across multiple rows

The best way to learn to clean data is to clean data! I highly recommend working through R for Data Science and the exercises included (here are the solutions).

In addition to navigating the shape of the data, there are often fun surprises within the data even with technically "tidy". You might have character strings that are messy and contain information that needs to be extracted (use `stringr`), or dates (use `lubridate`), or that need to be specified as ordered factors (use `forcats`). `base` R will often make decisions for you when reading in data as well. To avoid that, use `readr` to specify variable/data types when reading in the data to save a headache later!



Joining Table

Outer Join

T1

| G1 | X1 | Y1 |
|---|---|---|
| a | 5 | 3 |
| b | 3 | 4 |
| c | 1 | 2 |

T2

| G1 | X1 | Y1 |
|---|---|---|
| a | 2 | 2 |
| b | 3 | 4 |
| d | 1 | 1 |

ojoin(T1.G1,T2.G2)

| G1 | X1 | Y1 | G1 | X1 | Y1 |
|---|---|---|---|---|---|
| a | 5 | 3 | a | 2 | 2 |
| b | 3 | 4 | b | 3 | 4 |
| c | 1 | 2 | | | |
| | | | d | 1 | 1 |

Inner Join

T1

| G1 | X1 | Y1 |
|---|---|---|
| a | 5 | 3 |
| b | 3 | 4 |
| c | 1 | 2 |

T2

| G1 | X1 | Y1 |
|---|---|---|
| a | 2 | 2 |
| b | 3 | 4 |
| d | 1 | 1 |

ijoin(T1.G1,T2.G2)

| G1 | X1 | Y1 | G1 | X1 | Y1 |
|---|---|---|---|---|---|
| a | 5 | 3 | a | 2 | 2 |
| b | 3 | 4 | b | 3 | 4 |

Left Join

T1

| G1 | X1 | Y1 |
|---|---|---|
| a | 5 | 3 |
| b | 3 | 4 |
| c | 1 | 2 |

T2

| G1 | X1 | Y1 |
|---|---|---|
| a | 2 | 2 |
| b | 3 | 4 |
| d | 1 | 1 |

ljoin(T1.G1,T2.G2)

| G1 | X1 | Y1 | G1 | X1 | Y1 |
|---|---|---|---|---|---|
| a | 5 | 3 | a | 2 | 2 |
| b | 3 | 4 | b | 3 | 4 |
| c | 1 | 2 | | | |

Right Join

T1

| G1 | X1 | Y1 |
|---|---|---|
| a | 5 | 3 |
| b | 3 | 4 |
| c | 1 | 2 |

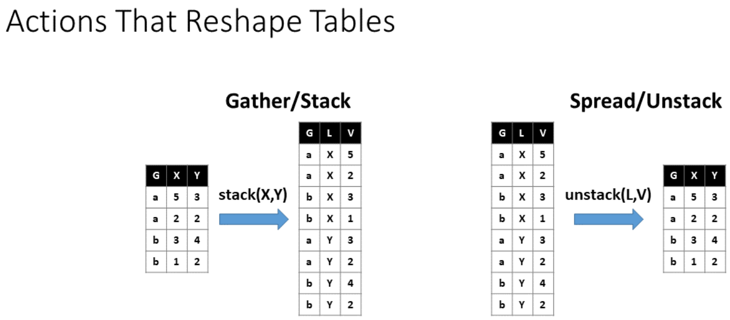
T2

| G1 | X1 | Y1 |
|---|---|---|
| a | 2 | 2 |
| b | 3 | 4 |
| d | 1 | 1 |

rjoin(T1.G1,T2.G2)

| G1 | X1 | Y1 | G1 | X1 | Y1 |
|---|---|---|---|---|---|
| a | 5 | 3 | a | 2 | 2 |
| b | 3 | 4 | b | 3 | 4 |
| | | | d | 1 | 1 |

Figure 10.2: from Winona State



Actions That Reshape Tables

Gather/Stack

| G | X | Y |
|---|---|---|
| a | 5 | 3 |
| a | 2 | 2 |
| b | 3 | 4 |
| b | 1 | 2 |

stack(X,Y)

| G | L | V |
|---|---|---|
| a | X | 5 |
| a | X | 2 |
| b | X | 3 |
| b | X | 1 |
| a | Y | 3 |
| a | Y | 2 |
| b | Y | 4 |
| b | Y | 2 |

Spread/Unstack

| G | L | V |
|---|---|---|
| a | X | 5 |
| a | X | 2 |
| b | X | 3 |
| b | X | 1 |
| a | Y | 3 |
| a | Y | 2 |
| b | Y | 4 |
| b | Y | 2 |

unstack(L,V)

| G | X | Y |
|---|---|---|
| a | 5 | 3 |
| a | 2 | 2 |
| b | 3 | 4 |
| b | 1 | 2 |

Figure 10.3: from Winona State

### 10.1.1 Example - Cleaning the Tesla Data

The Tesla data was actually “tidy” in format, but had a lot of dates (in multiple formats) and strings that needed some massaging. Here is what I did (more could be done!) to get the data ready for your use.

```
library(readr)
library(dplyr)
library(stringr)
library(lubridate)
tesla<-read_csv("Data/TeslaBatterySurvey2018.csv")
```

```
tesla %>%
  rename(BatteryAgeDays=`Battery AgeDays`) %>%
  mutate(DailyChargeLevel=as.numeric(str_sub(DailyChargeLevel,1,2))) %>%
  mutate(Location=str_replace_all(tesla$Location,"uK", "UK")) %>%
  mutate(ReadDate=parse_date_time(ReadDate, c("%d-%b-%y", "%m/%d/%Y", "%d. %b %Y"),exac
  mutate(ManufactureDate=parse_date_time(ManufactureDate,
                                        c("%d-%b-%y", "%m/%d/%Y", "%d. %b %Y"),exact=
  mutate(ModelXS=str_split(Model," ", simplify = TRUE)[,2],
         ModNum=str_split(Model," ",simplify = TRUE)[,3]) %>%
  write_csv("cleanTeslaBattery.csv")
```

```
newtesla<-read.csv("cleanTeslaBattery.csv")
str(newtesla)
```

```
'data.frame':   1339 obs. of  20 variables:
 $ Location         : Factor w/ 4 levels "Asia Pacific & Europe (excl UK)",..: 3 3 3 3
 $ ManufactureDate  : Factor w/ 388 levels "2012-09-12T00:00:00Z",..: 233 121 256 215
 $ ReadDate         : Factor w/ 769 levels "2014-03-27T00:00:00Z",..: 169 219 444 457
 $ AgeInDays        : int  45 520 387 556 653 187 475 839 36 42965 ...
 $ Model            : Factor w/ 24 levels "Model 3 LR","Model S 100D",..: 9 3 10 9 9 9
 $ MileageKM        : int  2800 10500 34653 45959 37895 16800 49129 49900 2271 3000 ..
 $ MileagePerDay    : num  60.9 20.2 89.3 82.5 57.9 ...
 $ MaxRangeKM       : int  246 180 256 231 238 242 266 239 205 239 ...
 $ ReplacementBatt  : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ MileageSinceNewKM: int  2800 10500 34653 45959 37895 16800 49129 49900 2271 3000 ..
 $ BatteryAgeDays   : int  46 521 388 557 654 188 476 840 37 NA ...
 $ WattHoursPerKM   : num  323 353 345 379 358 318 320 323 NA 305 ...
 $ OriginalRangeKM  : int  249 180 267 249 249 249 267 249 207 242 ...
 $ SuperchargeFreq  : Factor w/ 8 levels "a few times a year",..: 6 5 2 8 1 3 8 3 NA N
 $ MaxChargeFreq    : Factor w/ 8 levels "a few times a year",..: 3 1 3 5 3 1 4 1 NA N
 $ RunToEmptyFreq   : Factor w/ 7 levels "a few times a year",..: 4 4 1 3 3 3 3 3 NA N
 $ DailyChargeLevel : int  80 70 90 70 80 90 80 80 NA NA ...
 $ Cycles           : num  11.7 63.4 156.6 231.8 179 ...
 $ ModelXS          : Factor w/ 4 levels "3","85","S","X": 3 3 3 3 3 3 3 3 3 3 ...
 $ ModNum           : Factor w/ 18 levels "100D","60","60D",..: 8 2 9 8 8 8 9 8 3 7 ..
```

## 10.2 Miscellany on data types in R

R has several built-in data types that you may find useful:

*Vector*, like arrays in other languages, a collection of entries, each with the same type of data.

```
1:4
```

```
[1] 1 2 3 4
```

*Lists* with mixed data types

```
list("28",14)
```

```
[[1]]
[1] "28"

[[2]]
[1] 14
```

*Matrices* for storing data of all the same type:

```
 A = matrix(
   c(1, 3, 5, 9, 2, 1), # the data elements
   nrow=2,              # number of rows
   ncol=3,              # number of columns
   byrow = TRUE)        # specify to fill in across the rows or down the columns
```

Note that we have been using data frames that can be built from the ground up.

```
weight = c(55,33,21,19)
height = c(67,41,35,24)
pet = c("cat","dog","cat","lizard")
dat = data.frame(height,weight, pet)
summary(dat)
```

```
     height          weight         pet   
 Min.   :24.00   Min.   :19.0   cat   :2  
 1st Qu.:32.25   1st Qu.:20.5   dog   :1  
 Median :38.00   Median :27.0   lizard:1  
 Mean   :41.75   Mean   :32.0             
 3rd Qu.:47.50   3rd Qu.:38.5             
 Max.   :67.00   Max.   :55.0             
```

Note that entries in data frames can be accessed much as the way they are in other languages. Try each of the following. What do each of them do?

```
dat[1,2]
dat[,2]
dat[2:3,]
```

```
dat[c(1,4),]
dat[,"weight"]
dat[,c("height","pet")]
dat[-2,]
```

One of the other incredibly handy features is that rows can also be selected based on various criteria. We can select the rows where the pet is a cat using the command or the weight of the rows where the pet is a cat using the following two commands:

```
dat[dat$pet=="cat",]
```

```
  height weight pet
1     67     55 cat
3     35     21 cat
```

```
dat[dat$pet=="cat","weight"]
```

```
[1] 55 21
```

R also casts vectors to apply arithmetic operations to lists and also allows dynamic variable assignment in data frames. Below we will rescale the weight by multiplying by 16 and save it in the column new height

```
dat$newheight = dat$height *16
dat
```

```
  height weight    pet newheight
1     67     55    cat      1072
2     41     33    dog       656
3     35     21    cat       560
4     24     19 lizard       384
```

This can be handy for normalizing and transforming data. Note there is also a *scale* function for rescaling data.

# Chapter 11

# Project

Let's analyze some data. We are going to consider the data from the Youth Risk Behavior Surveillance System collected by the CDC. YRBS2015.csv

# Chapter 12

# Other Helpful Resources

One of the wonderful things about R is all of the publicly available support. There are nearly limitless places to go. Here are a few links that we find helpful.

- RStudio Cheatsheets
- ggplot Cheatsheet
- R Graphics Gallery