R101 Primer

Matthew Byrne

02/10/2020

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

# Some basic samples of R Data

myString <- 'Hello my name is Matt and this is R'  
print(myString)

## [1] "Hello my name is Matt and this is R"

7 + 7

## [1] 14

x = 1  
y = 2  
z = x + y

## Data Types in R

In contrast to other programming languages like C or Java, variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-Object becomes the data type of the variable. They are many types of R-Objects. Here is the most common ones!

## R Objects

-vectors -Lists -Matrices -Arrays -Factors -Data Frames

### Vector - value assigned to a variable

# Logical True or False  
v = TRUE  
print(class(v))

## [1] "logical"

# Numeric 12.3, 5, 999  
v = 23.5  
print(class(v))

## [1] "numeric"

# Integer 2L, 34L, 0L   
v = 2L  
print(class(v))

## [1] "integer"

# Character vectors are strings 'a' , '"good", "TRUE", '23.4'   
v = "this is a string"  
print(class(v))

## [1] "character"

# Raw Value "Hello" is stored as 48 65 6c 6c 6f   
v = charToRaw("hello")  
print(class(v))

## [1] "raw"

v

## [1] 68 65 6c 6c 6f

##################### Multiple Values ###############  
  
# Create a vector with multiple values.  
apple = c('red','green',"yellow")  
print(apple)

## [1] "red" "green" "yellow"

# Get the class of the multiple vector which is "character".  
print(class(apple))

## [1] "character"

# Create a list (contains a vector, float, max function and boolean)  
list1 = list(c(2,5,3),21.3,max, TRUE)  
# Print the list. (Double brackets is the field of the list)  
print(list1)

## [[1]]  
## [1] 2 5 3  
##   
## [[2]]  
## [1] 21.3  
##   
## [[3]]  
## function (..., na.rm = FALSE) .Primitive("max")  
##   
## [[4]]  
## [1] TRUE

## A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.  
# Create a matrix.  
M = matrix(c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)  
print(M)

## [,1] [,2] [,3]  
## [1,] "a" "a" "b"   
## [2,] "c" "b" "a"

# Create an array(repeats over and over if values are too short)  
a = array(c('green','yellow','purple'),dim = c(3,3,2))  
print(a)

## , , 1  
##   
## [,1] [,2] [,3]   
## [1,] "green" "green" "green"   
## [2,] "yellow" "yellow" "yellow"  
## [3,] "purple" "purple" "purple"  
##   
## , , 2  
##   
## [,1] [,2] [,3]   
## [1,] "green" "green" "green"   
## [2,] "yellow" "yellow" "yellow"  
## [3,] "purple" "purple" "purple"

# Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.  
  
# Factors are created using the `factor()` function. The `nlevels` functions gives the count of levels.  
# Create a vector.  
apple\_colors = c('green','green','yellow','red','red','red','green')  
# Create a factor object.  
factor\_apple = factor(apple\_colors)  
# Print the factor equal 3  
print(factor\_apple)

## [1] green green yellow red red red green   
## Levels: green red yellow

print(nlevels(factor\_apple))

## [1] 3

# Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.  
  
# Data Frames are created using the `data.frame()` function.  
  
# Create the data frame.  
BMI = data.frame(  
 gender = factor(c("Male", "Male","Female")),   
 height = c(152, 171.5, 165),   
 weight = c(81,93, 78),  
 Age = c(42,38,26)  
)  
print(BMI)

## gender height weight Age  
## 1 Male 152.0 81 42  
## 2 Male 171.5 93 38  
## 3 Female 165.0 78 26

# Get your Data into R

# imports a multitude of packages designed to make your life much easier, a ecosystem and works together using a tidydata format

install.packages("tidyverse")

## Warning: package 'tidyverse' is in use and will not be installed

library(tidyverse)

# Clean you data and reshape it

# brings in a sample data set of "miles per gallon" in car manufacturers to tinker with  
  
# dplyr and the pipe (piping the input into another action)  
mpg %>%   
  
# return and filter models that are equal to a4  
dplyr::filter(model == 'a4') %>%   
  
# the previous output from the filter is now the input for arrange  
# use the arrange function, sort by this, this and this  
dplyr::arrange(displ, cyl) %>%   
  
# create a new variable at the end "ageOfCar", using existing data while still retaining the old column  
dplyr::mutate(ageOfCar = year < 2000, fullWheelDrive = drv == 'f') %>% # add new preserve old  
  
  
# if you wanna calculate, transform and completely replace the older data  
# transmute give you back the field you asked for!  
# dplyr::transmute(full\_wheel\_drive = drv == 'f')  
  
  
# select explicitly or pipe all the output into select and say everything minus more columns below  
dplyr::select(-drv, -trans, -year, -manufacturer, -model)

## # A tibble: 7 x 8  
## displ cyl cty hwy fl class ageOfCar fullWheelDrive  
## <dbl> <int> <int> <int> <chr> <chr> <lgl> <lgl>   
## 1 1.8 4 18 29 p compact TRUE TRUE   
## 2 1.8 4 21 29 p compact TRUE TRUE   
## 3 2 4 20 31 p compact FALSE TRUE   
## 4 2 4 21 30 p compact FALSE TRUE   
## 5 2.8 6 16 26 p compact TRUE TRUE   
## 6 2.8 6 18 26 p compact TRUE TRUE   
## 7 3.1 6 18 27 p compact FALSE TRUE

mpg %>% # take the full data set to what it can do  
dplyr::count(model, sort = TRUE) #return back 234 records if no condition

## # A tibble: 38 x 2  
## model n  
## <chr> <int>  
## 1 caravan 2wd 11  
## 2 ram 1500 pickup 4wd 10  
## 3 civic 9  
## 4 dakota pickup 4wd 9  
## 5 jetta 9  
## 6 mustang 9  
## 7 a4 quattro 8  
## 8 grand cherokee 4wd 8  
## 9 impreza awd 8  
## 10 a4 7  
## # ... with 28 more rows

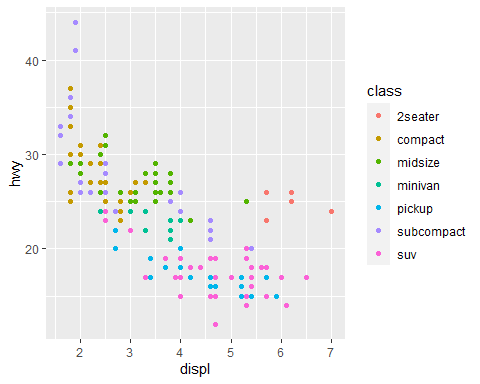
# Make Some Graphics and plot with the GGPLOT package one of the biggest in R

# graphical package that lets use complex highled detailed images used everywhere

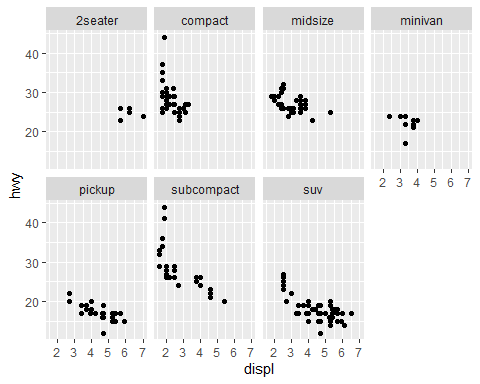
# ggplot(data = <DATA>) +   
# <GEOM\_FUNCTION>(mapping = aes(<MAPPINGS>))  
  
# geom\_point is a scatter plot, mapping the aesthetic which represents "aes" to x and y axis of "miles per gallon" car data  
  
# can you the previous pipe but can you a plus symbol to combine  
  
# can use multiple geoms on your chart also  
  
ggplot(data = mpg) +   
 geom\_point(mapping = aes(x = displ, y = hwy))



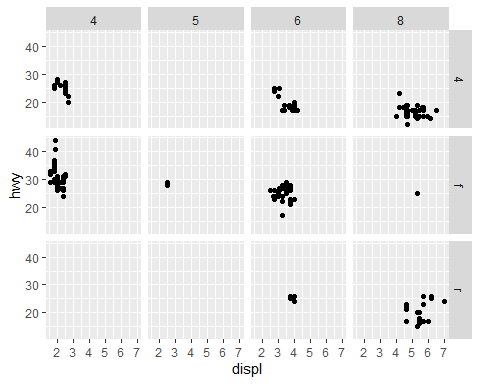
ggplot(data = mpg) +   
 geom\_point(mapping = aes(x = displ, y = hwy, color = class))



ggplot(data = mpg) +   
 geom\_point(mapping = aes(x = displ, y = hwy)) +   
 facet\_wrap(~ class, nrow = 2)



ggplot(data = mpg) +   
 geom\_point(mapping = aes(x = displ, y = hwy)) +   
 facet\_grid(drv ~ cyl)

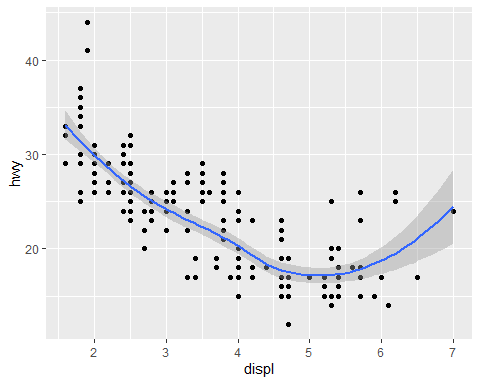


ggplot(data = mpg) +   
 geom\_point(mapping = aes(x = displ, y = hwy))



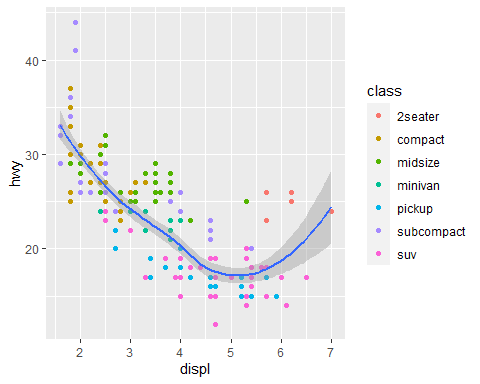
ggplot(data = mpg) +   
 geom\_point(mapping = aes(x = displ, y = hwy)) +  
 geom\_smooth(mapping = aes(x = displ, y = hwy))

## `geom\_smooth()` using method = 'loess' and formula 'y ~ x'

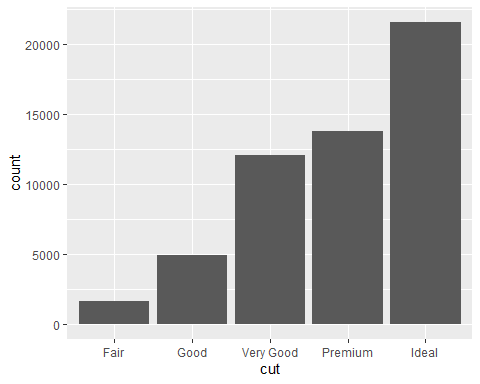


ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +   
 geom\_point(mapping = aes(color = class)) +   
 geom\_smooth()

## `geom\_smooth()` using method = 'loess' and formula 'y ~ x'

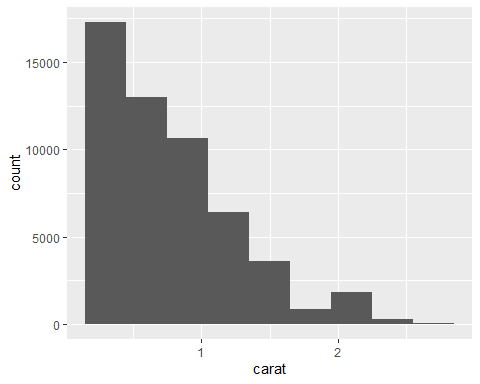


ggplot(data = diamonds) +   
 geom\_bar(mapping = aes(x = cut))

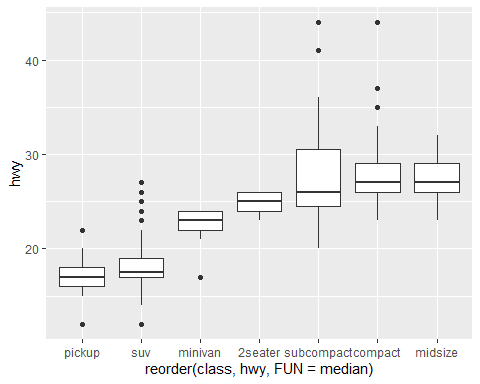


# Examples of EDA

smaller <- diamonds %>%   
 filter(carat < 3)  
   
ggplot(data = smaller, mapping = aes(x = carat)) +  
 geom\_histogram(binwidth = 0.3)



ggplot(data = mpg) +  
 geom\_boxplot(mapping = aes(x = reorder(class, hwy, FUN = median), y = hwy))



diamonds %>%   
 count(color, cut, sort = T) %>%   
 ggplot(mapping = aes(x = color, y = cut)) +  
 geom\_tile(mapping = aes(fill = n))

