R Sandbox Activity - Google Data Analytics

Arafat Ahmed

2024-03-08

## Background for this activity

This is the first R project that I did individually with the help of Google Data Analytics Certificate syllabus. The name of the project is R Sandbox Activity. It is just an exploratory project to see how R functions with an activity from the certification curriculum. #Step 1- In this activity, I will be using a package called tidyverse. The tidyverse package is actually a collection individual packages that can help to perform a wide variety of analysis tasks.

options(repos = c(CRAN = "https://cran.rstudio.com"))

install.packages("tidyverse")

## Installing package into 'C:/Users/Arafat Ahmed/AppData/Local/R/win-library/4.3'  
## (as 'lib' is unspecified)

## package 'tidyverse' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\Arafat Ahmed\AppData\Local\Temp\RtmpQ5dQmM\downloaded\_packages

##Once a package is installed, I can load it by running the library() function with the package name inside the parentheses

library(tidyverse)

## Warning: package 'tidyverse' was built under R version 4.3.3

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.4 ✔ readr 2.1.5  
## ✔ forcats 1.0.0 ✔ stringr 1.5.1  
## ✔ ggplot2 3.5.0 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.3 ✔ tidyr 1.3.1  
## ✔ purrr 1.0.2   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

##Now that I have loaded an R package, I can start exploring some data.

# Step 2: Viewing data Many of the tidyverse packages contain sample datasets that I can use to practice my R skills. The diamonds dataset in the ggplot2 package is a great example for previewing R functions. I can test out how the head() function works by running the chunk below:

head(diamonds)

## # A tibble: 6 × 10  
## carat cut color clarity depth table price x y z  
## <dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43  
## 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31  
## 3 0.23 Good E VS1 56.9 65 327 4.05 4.07 2.31  
## 4 0.29 Premium I VS2 62.4 58 334 4.2 4.23 2.63  
## 5 0.31 Good J SI2 63.3 58 335 4.34 4.35 2.75  
## 6 0.24 Very Good J VVS2 62.8 57 336 3.94 3.96 2.48

##In addition to head() there are a number of other useful functions I can use to summarize or preview the data. For example, the str() and glimpse() functions will both return summaries of each column in this data arranged horizontally.

str(diamonds)

## tibble [53,940 × 10] (S3: tbl\_df/tbl/data.frame)  
## $ carat : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...  
## $ cut : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...  
## $ color : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...  
## $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...  
## $ depth : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...  
## $ table : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...  
## $ price : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...  
## $ x : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...  
## $ y : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...  
## $ z : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...

glimpse(diamonds)

## Rows: 53,940  
## Columns: 10  
## $ carat <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, 0.23, 0.…  
## $ cut <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very Good, Ver…  
## $ color <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, J, J, I,…  
## $ clarity <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI1, VS1, …  
## $ depth <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, 59.4, 64…  
## $ table <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54, 62, 58…  
## $ price <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339, 340, 34…  
## $ x <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, 4.00, 4.…  
## $ y <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, 4.05, 4.…  
## $ z <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, 2.39, 2.…

##Another simple function that I can use regularly is the colnames() function. It returns a list of column names from a dataset.

colnames(diamonds)

## [1] "carat" "cut" "color" "clarity" "depth" "table" "price"   
## [8] "x" "y" "z"

##After running the code chunk, I have noticed a number in brackets. This number helps me to count the number of columns in my dataset. If I have data with lots of columns and colnames() prints the results on multiple lines, each line will have a number in brackets at the start of the line indicating what number column that is! So, for example, “carat” is the first column in the diamonds dataset. On the second line, there is the number seven in brackets; “price” is the seventh column. #Step 3: Cleaning data One of the most frequent tasks I will have to perform as an analyst is to clean and organize your data.For example, I might need to rename the columns, or variables, in your data. There is a function for that: rename(). I can check out how it works in the chunk below:

rename(diamonds, carat\_new = carat)

## # A tibble: 53,940 × 10  
## carat\_new cut color clarity depth table price x y z  
## <dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43  
## 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31  
## 3 0.23 Good E VS1 56.9 65 327 4.05 4.07 2.31  
## 4 0.29 Premium I VS2 62.4 58 334 4.2 4.23 2.63  
## 5 0.31 Good J SI2 63.3 58 335 4.34 4.35 2.75  
## 6 0.24 Very Good J VVS2 62.8 57 336 3.94 3.96 2.48  
## 7 0.24 Very Good I VVS1 62.3 57 336 3.95 3.98 2.47  
## 8 0.26 Very Good H SI1 61.9 55 337 4.07 4.11 2.53  
## 9 0.22 Fair E VS2 65.1 61 337 3.87 3.78 2.49  
## 10 0.23 Very Good H VS1 59.4 61 338 4 4.05 2.39  
## # ℹ 53,930 more rows

##Here, the function is being used to change the name of carat to carat\_new. This is a pretty basic change, but rename() has many options that can help me do more complex changes across all of the variables in my data. For example, I can rename more than one variable in the same rename() code. The code below demonstrates how:

rename(diamonds, carat\_new = carat, cut\_new = cut)

## # A tibble: 53,940 × 10  
## carat\_new cut\_new color clarity depth table price x y z  
## <dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43  
## 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31  
## 3 0.23 Good E VS1 56.9 65 327 4.05 4.07 2.31  
## 4 0.29 Premium I VS2 62.4 58 334 4.2 4.23 2.63  
## 5 0.31 Good J SI2 63.3 58 335 4.34 4.35 2.75  
## 6 0.24 Very Good J VVS2 62.8 57 336 3.94 3.96 2.48  
## 7 0.24 Very Good I VVS1 62.3 57 336 3.95 3.98 2.47  
## 8 0.26 Very Good H SI1 61.9 55 337 4.07 4.11 2.53  
## 9 0.22 Fair E VS2 65.1 61 337 3.87 3.78 2.49  
## 10 0.23 Very Good H VS1 59.4 61 338 4 4.05 2.39  
## # ℹ 53,930 more rows

##Another handy function for summarizing any data is summarize().I can use it to generate a wide range of summary statistics for your data. For example, if I wanted to know what the mean for carat was in this dataset, I could run the code in the chunk below:

summarize(diamonds, mean\_carat = mean(carat))

## # A tibble: 1 × 1  
## mean\_carat  
## <dbl>  
## 1 0.798

These functions are a great way to get more familiar with your data and start making observations about it.

# Step 4: Visualizing data With R, I can create data visualizations that are simple and easy to understand or complicated and beautiful just by changing a bit of code. One of the most commonly used visualization packages is the ggplot2 package, which is loaded automatically when I install and load tidyverse. The diamonds dataset that I have been using so far is a ggplot2 dataset. ##To build a visualization with ggplot2 I layer plot elements together with a + symbol. Here is a preview of how easy and flexible it is to make visuals using code:

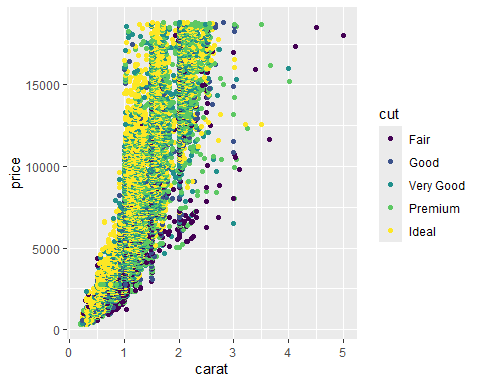
ggplot(data = diamonds, aes(x = carat, y = price)) +  
 geom\_point()



The code above takes the diamonds data, plots the carat column on the X-axis, the price column on the Y-axis, and represents the data as a scatter plot using the geom\_point() command.

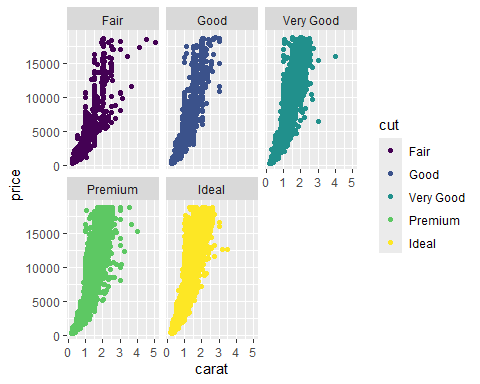
##ggplot2 makes it easy to modify or improve any visuals. For example, if I wanted to change the color of each point so that it represented another variable, such as the cut of the diamond, you can change the code like this:

ggplot(data = diamonds, aes(x = carat, y = price, color = cut)) +  
 geom\_point()



Sometimes when I am trying to represent many different aspects of any data in a visual, it can help to separate out some of the components. For example, I could create a different plot for each type of cut. ggplot2 makes it easy to do this with the facet\_wrap() function:

ggplot(data = diamonds, aes(x = carat, y = price, color = cut)) +  
 geom\_point() +  
 facet\_wrap(~cut)



## Activity Wrap-up I have had a chance to explore more R tools that I can start using on my own. I learned how to install and load R packages; functions for viewing, cleaning, and visualizing data; and using R markdownto export for my work.