Functions and conditionals

Kaitlyn Watson: Group 9

M4 ICA1

Introduction

Today you will get some practice writing functions. Load package `tidyverse`.

```
library(tidyverse)
```

Below is a template for how R functions are structured. The three main components are

- **name**, a short informative verb;
- **arguments**, inputs to the function;
- **body**, code to be executed.

```
my_fcn_name <- function(arg_1, arg_2, arg_3) {
  # body
  # of
  # the
  # function
  # goes
  # here
  return(r_object) # object to return
}
```

Creating `viewr()` function

Your goal is to create a function called `viewr()` that will take a data frame as its data input and return a randomly specified number of rows and specified number of columns.

The following two functions will be useful in your development of `viewr()`.

`dim()` Function `dim()` retrieves the dimension of an object. For example,

```
dim(mtcars)
dim(diamonds)
```

gives the number of rows and columns of the data frames `mtcars` and `diamonds` as a vector of length two.

`sample_n()` Function `sample_n()` is a function in package `dplyr` that randomly selects rows from a data frame. For example

```
mtcars %>%
  sample_n(size = 5)
```

randomly selects five rows from `mtcars`.

Step 1

The first step in building any function is to get your code working before turning it into a function.

1. Use a sequence of `dplyr` functions to select the first 3 columns of `diamonds` and randomly pick 7 rows.

```
# A tibble: 7 x 3
  carat cut       color
  <dbl> <ord>     <ord>
1  0.72 Ideal     G
2  1.2  Premium   H
3  0.41 Ideal     F
4  2.21 Ideal     D
5  0.74 Very Good D
6  0.34 Ideal     I
7  1.01 Good      E
```

2. Modify your above sequence to select all the columns and randomly pick 20 rows.

```
# A tibble: 20 x 10
   carat cut       color clarity depth table price     x     y     z
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  0.36 Ideal     F     VVS2     61.4  56    1059  4.61  4.64  2.84
 2  1.02 Premium   G     VS2      62.6  59    5978  6.39  6.43  4.01
 3  0.5  Premium   D     VS2      62.8  58    1845  5.08  5.05  3.18
 4  0.23 Very Good F     VVS2     59.7  58     530  4.02  4.06  2.41
 5  0.76 Good      E     SI1      63.7  54    2789  5.76  5.85  3.7
 6  0.7  Ideal     F     SI1      62    55    2415  5.72  5.7   3.54
 7  0.31 Very Good G     VS1      63.3  56     802  4.33  4.29  2.73
 8  1.22 Ideal     J     VS2      61    55    6052  6.92  6.96  4.23
 9  0.32 Ideal     D     SI1      61.5  56     589  4.39  4.42  2.71
10  0.36 Good      H     VS2      63.5  54     568  4.55  4.59  2.9
11  1    Very Good H     SI2      59.1  62    3575  6.47  6.5   3.83
12  0.34 Ideal     H     VS1      62.5  57     596  4.43  4.46  2.78
13  1.22 Very Good J     SI2      62.2  60    4058  6.77  6.89  4.25
14  0.51 Premium   F     SI2      61.1  59    1227  5.15  5.09  3.13
15  0.9  Very Good G     SI2      60.1  60    3114  6.22  6.32  3.77
16  0.31 Ideal     H     VS2      62.1  53.8   502  4.33  4.37  2.7
17  1.01 Premium   F     VS2      63    58    6097  6.43  6.36  4.03
18  0.56 Ideal     G     IF       61.8  56    2442  5.28  5.34  3.28
19  1.01 Good      E     SI1      63.5  58    4546  6.28  6.35  4.01
20  0.56 Ideal     H     VS1      60.2  57    1888  5.38  5.35  3.23
```

Step 2

Use your code in Step 1 as a template to create a function named `viewr_1()`. The function should have three arguments:

- `df`: will take a data frame as its input
- `nrow`: will take a positive integer as its input
- `ncol`: will take a positive integer as its input

The function will return part of the data frame provided as an input, but with a randomly chosen number of rows specified by `nrow` and the first `ncol` columns as specified.

After you write your function, test your function with the following calls:

```
viewr_1(df = diamonds, nrow = 4, ncol = 3)
viewr_1(df = diamonds, nrow = 10, ncol = 10)
viewr_1(df = diamonds, nrow = 10, ncol = 11)
```

You should have received an error when you ran the third line. Data set `diamonds` only has 10 variables, so it cannot select 11 columns. You will fix this in step 3.

Step 3

Modify `viewr_1()` to include an if-else statement. If `ncol` does not exceed the number of columns in the data frame, then the function should run as normal. Otherwise, the function should output a message with `cat("The data frame does not have that many columns!")`. Call this new function `viewr_2()`

Test `viewr_2()` with the following calls:

```
viewr_2(df = diamonds, nrow = 4, ncol = 3)
viewr_2(df = diamonds, nrow = 10, ncol = 10)
viewr_2(df = diamonds, nrow = 10, ncol = 11)
viewr_2(df = diamonds, nrow = 10000000, ncol = 3)
```

You should have received an error when you ran the fourth line. Data set `diamonds` does not have 10000000 rows. To fix this, proceed to Step 4.

Step 4

Modify `viewr_1()` to include a stop-if-not statement with function `stopifnot()`. Call this new function `viewr()`. If any of the expressions are not all true, stop is called, producing an error message indicating the first expression which was not true. Separate conditions with a comma. Below is an example.

```
ranger <- function(x){
  # check if x is a numeric vector of length at least 2
  stopifnot(is.numeric(x), length(x)>=2)
  value <- max(x) - min(x)
  return(value)
}

# function test
ranger(x = 1:10)
ranger(x = 3)
ranger(x = letters)
```

Using `stopifnot()` is more convenient than including multiple if-else statements.

Test `viewr()` with the following calls:

```
viewr(df = diamonds, nrow = 4, ncol = 3)
viewr(df = diamonds, nrow = 10, ncol = 11)
viewr(df = diamonds, nrow = 10000000, ncol = 3)
```

Creating `summary()` function

Create a function which will take a vector as an input and calculate the sum and mean of the vector. The function should return the sum, the average and a message stating whether the sum/mean is positive or negative.

```
The sum is positive The sum is :  10The mean is positive The mean is :  2.5
```

Creating `limit_e()`function

Create a function called `limit_e()` that takes one argument, `n`. Argument `n` shoud be a vector of integers greater than zero. Function `limit_e()` will compute and return the evaluated quantity

$$\left(1+\frac{1}{n}\right)^n.$$

From calculus you know that the mathematical constant e is defined as

$$e=\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n.$$

After you write your function, test it with the following function calls. What do you notice happens? Why is the happening?

```
limit_e(100)
limit_e(1000000)
limit_e(c(1, 1000000, 100000000000)
limit_e(c(1, 1000000, 10000000000000000000))
```

There is quite a bit of rounding involved and this is to account for errors in the limit. ## Computer arithmetic

R, as does most software, uses floating point arithmetic, which is not the same as the arithmetic we learn. Computers cannot represent all numbers exactly.

For your mind to be blown, run the following examples.

```
# example 1
0.2 == 0.6 / 3

# example 2
point3 <- c(0.3, 0.4 - 0.1, 0.5 - 0.2, 0.6 - 0.3, 0.7 - 0.4)
point3

point3 == 0.3
```

To work around these issues, use `all.equal()` for checking the equality of two numeric quantities in R.

```r
# example 1, all.equal()
all.equal(0.2, 0.6 / 3)

# example 2, all.equal()
point3 <- c(0.3, 0.4 - 0.1, 0.5 - 0.2, 0.6 - 0.3, 0.7 - 0.4)
point3

all.equal(point3, rep(.3, length(point3)))
```