Coding Challenge6

Muhtarin Khayer Brohee

2025-04-04

[Git link](https://github.com/mzb0226/PLPA-6820/tree/main/Coding%20challenge%206)

### Writing Functions and For Loops in R

#### Importance of Writing Your Own Functions and Using Iterations

* Improves **reproducibility** by preventing copy-paste errors
* Makes code easier to **read, debug, and reuse**
* Automates **repetitive tasks**, which is crucial when handling large datasets

#### How to Write a Function

In R, a function is a **reusable block of code** that performs a specific task. To write a function, use the syntax:

my\_function <- function(input1, input2) {  
 # Code to perform some operation  
 result <- input1 + input2  
 return(result)  
}

##### Explanation

* The function starts with a name, followed by <- function(…) to define the inputs.
* Code is written inside {} curly brackets.
* The return() statement defines what the function will output.
* I usually write functions at the top of my script or .Rmd file so I can reuse them later.

### How to Write a For Loop

A for loop is used when I want to **repeat an action multiple times**. The basic syntax is:

for (i in 1:5) {  
 print(i)  
}

##### Explanation

* The loop runs the code inside {} for each value of i in the sequence 1:5.
* I write the loop where I need repeated actions, like applying a function to every row in a dataset.
* Loops are useful for automating calculations, simulations, or building data structures.

### Load data

cities <- read.csv("Cities.csv")

### Load libraries

library(tidyverse)

library(tidyverse)

## ── 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 3.5.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.4 ✔ tidyr 1.3.1  
## ✔ purrr 1.0.4   
## ── 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

### Haversine distance function

haversine\_dist <- function(lat1, lon1, lat2, lon2) {  
 rad.lat1 <- lat1 \* pi/180  
 rad.lon1 <- lon1 \* pi/180  
 rad.lat2 <- lat2 \* pi/180  
 rad.lon2 <- lon2 \* pi/180  
   
 delta\_lat <- rad.lat2 - rad.lat1  
 delta\_lon <- rad.lon2 - rad.lon1  
   
 a <- sin(delta\_lat / 2)^2 + cos(rad.lat1) \* cos(rad.lat2) \* sin(delta\_lon / 2)^2  
 c <- 2 \* asin(sqrt(a))  
   
 earth\_radius <- 6378137  
 distance\_km <- (earth\_radius \* c)/1000  
   
 return(distance\_km)  
}

### Compute distance between Auburn and New York

auburn <- cities %>% filter(city == "Auburn")  
nyc <- cities %>% filter(city == "New York")  
  
distance\_to\_nyc <- haversine\_dist(  
 lat1 = auburn$lat, lon1 = auburn$long,  
 lat2 = nyc$lat, lon2 = nyc$long  
)  
  
print(distance\_to\_nyc)

## [1] 1367.854

### Loop to calculate distance from all cities to Auburn

#Initialize empty dataframe  
distance\_df <- data.frame(City1 = character(), City2 = character(), Distance\_km = numeric())  
  
# Auburn’s coordinates  
auburn\_lat <- auburn$lat  
auburn\_lon <- auburn$long  
  
# Loop through each city except Auburn  
for (i in 1:nrow(cities)) {  
 if (cities$city[i] != "Auburn") {  
 city\_name <- cities$city[i]  
 city\_lat <- cities$lat[i]  
 city\_lon <- cities$long[i]  
   
 dist\_km <- haversine\_dist(auburn\_lat, auburn\_lon, city\_lat, city\_lon)  
   
 new\_row <- data.frame(City1 = city\_name, City2 = "Auburn", Distance\_km = dist\_km)  
 distance\_df <- rbind(distance\_df, new\_row)  
 }  
}  
  
# View first few distances  
head(distance\_df)

## City1 City2 Distance\_km  
## 1 New York Auburn 1367.8540  
## 2 Los Angeles Auburn 3051.8382  
## 3 Chicago Auburn 1045.5213  
## 4 Miami Auburn 916.4138  
## 5 Houston Auburn 993.0298  
## 6 Dallas Auburn 1056.0217