Week-3: Code-along

Ian Lee

2023-08-27

# I. Code to edit and execute

**To be submitted on canvas before attending the tutorial**

### Loading packages

# Load package tidyverse  
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.2 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ ggplot2 3.4.3 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ 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

### Assigning values to variables

# Example a.: execute this example  
x <- 'A'  
x

## [1] "A"

# Complete the code for Example b and execute it  
x <- "Apple"  
x

## [1] "Apple"

# Complete the code for Example c and execute it  
x <- FALSE  
x

## [1] FALSE

# Complete the code for Example d and execute it  
x <- 5L  
x

## [1] 5

# Complete the code for Example e and execute it  
x <- 5  
x

## [1] 5

# Complete the code for Example f and execute it  
x <- 1i  
x

## [1] 0+1i

### Checking the type of variables

# Example a.: execute this example  
x <- 'A'  
typeof(x)

## [1] "character"

# Complete the code for Example b and execute it  
x <- "Apple"  
typeof(x)

## [1] "character"

# Complete the code for Example c and execute it  
x <- FALSE  
typeof(x)

## [1] "logical"

# Complete the code for Example d and execute it  
x <- 5L  
typeof(x)

## [1] "integer"

# Complete the code for Example e and execute it  
x <- 5  
typeof(x)

## [1] "double"

# Complete the code for Example f and execute it  
x <- 1i  
typeof(x)

## [1] "complex"

### Need for data types

# import the cat-lovers data from the csv file you downloaded from canvas  
cat\_lovers <- read\_csv("cat-lovers.csv")

# Compute the mean of the number of cats: execute this command  
mean(cat\_lovers$number\_of\_cats)

## Warning in mean.default(cat\_lovers$number\_of\_cats): argument is not numeric or  
## logical: returning NA

## [1] NA

# Get more information about the mean() command using ? operator  
?mean

# Convert the variable number\_of\_cats using as.integer()  
mean(as.integer(cat\_lovers$number\_of\_cats))

## Warning in mean(as.integer(cat\_lovers$number\_of\_cats)): NAs introduced by  
## coercion

## [1] NA

# Display the elements of the column number\_of\_cats   
cat\_lovers$number\_of\_cats

## [1] "0"   
## [2] "0"   
## [3] "1"   
## [4] "3"   
## [5] "3"   
## [6] "2"   
## [7] "1"   
## [8] "1"   
## [9] "0"   
## [10] "0"   
## [11] "0"   
## [12] "0"   
## [13] "1"   
## [14] "3"   
## [15] "3"   
## [16] "2"   
## [17] "1"   
## [18] "1"   
## [19] "0"   
## [20] "0"   
## [21] "1"   
## [22] "1"   
## [23] "0"   
## [24] "0"   
## [25] "4"   
## [26] "0"   
## [27] "0"   
## [28] "0"   
## [29] "0"   
## [30] "0"   
## [31] "0"   
## [32] "0"   
## [33] "0"   
## [34] "0"   
## [35] "0"   
## [36] "0"   
## [37] "0"   
## [38] "0"   
## [39] "0"   
## [40] "0"   
## [41] "0"   
## [42] "0"   
## [43] "1"   
## [44] "3"   
## [45] "3"   
## [46] "2"   
## [47] "1"   
## [48] "1.5 - honestly I think one of my cats is half human"  
## [49] "0"   
## [50] "0"   
## [51] "1"   
## [52] "0"   
## [53] "1"   
## [54] "three"   
## [55] "1"   
## [56] "1"   
## [57] "1"   
## [58] "0"   
## [59] "0"   
## [60] "2"

# Display the elements of the column number\_of\_cats after converting it using as.numeric()  
as.integer(cat\_lovers$number\_of\_cats)

## Warning: NAs introduced by coercion

## [1] 0 0 1 3 3 2 1 1 0 0 0 0 1 3 3 2 1 1 0 0 1 1 0 0 4  
## [26] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 3 3 2 1 NA 0 0  
## [51] 1 0 1 NA 1 1 1 0 0 2

### Create an empty vector

# Empty vector  
x <- vector()  
# Type of the empty vector  
typeof(x)

## [1] "logical"

### Create vectors of type logical

# Method 1  
x<-vector("logical",length=5)  
# Display the contents of x  
print(x)

## [1] FALSE FALSE FALSE FALSE FALSE

# Display the type of x  
print(typeof(x))

## [1] "logical"

# Method 2  
x<-logical(5)  
# Display the contents of x  
print(x)

## [1] FALSE FALSE FALSE FALSE FALSE

# Display the type of x  
print(typeof(x))

## [1] "logical"

# Method 3  
x<-c(TRUE,FALSE,TRUE,FALSE,TRUE)  
# Display the contents of x  
print(x)

## [1] TRUE FALSE TRUE FALSE TRUE

# Display the type of x  
print(typeof(x))

## [1] "logical"

### Create vectors of type character

# Method 1  
x<-vector("character",length=5)  
# Display the contents of x  
print(x)

## [1] "" "" "" "" ""

# Display the type of x  
print(typeof(x))

## [1] "character"

# Method 2  
x<-character(5)  
# Display the contents of x  
print(x)

## [1] "" "" "" "" ""

# Display the type of x  
print(typeof(x))

## [1] "character"

# Method 3  
x<-c('A','b','r','q')  
# Display the contents of x  
print(x)

## [1] "A" "b" "r" "q"

# Display the type of x  
print(typeof(x))

## [1] "character"

### Create vectors of type integer

# Method 1  
x<-vector("integer",length=5)  
# Display the contents of x  
print(x)

## [1] 0 0 0 0 0

# Display the type of x  
print(typeof(x))

## [1] "integer"

# Method 2  
x<-integer(5)  
# Display the contents of x  
print(x)

## [1] 0 0 0 0 0

# Display the type of x  
print(typeof(x))

## [1] "integer"

# Method 3  
x<-c(1,2,3,4,5)  
# Display the contents of x  
print(x)

## [1] 1 2 3 4 5

# Display the type of x  
print(typeof(x))

## [1] "double"

# Method 4  
x<-seq(from=1,to=5,by=0.1)  
# Display the contents of x  
print(x)

## [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8  
## [20] 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7  
## [39] 4.8 4.9 5.0

# Display the type of x  
print(typeof(x))

## [1] "double"

# Method 5  
x<-1:5  
# Display the contents of x  
print(x)

## [1] 1 2 3 4 5

# Display the type of x  
print(typeof(x))

## [1] "integer"

### Create vectors of type double

# Method 1  
x<-vector("double",length=5)  
# Display the contents of x  
print(x)

## [1] 0 0 0 0 0

# Display the type of x  
print(typeof(x))

## [1] "double"

# Method 2  
x<-double(5)  
# Display the contents of x  
print(x)

## [1] 0 0 0 0 0

# Display the type of x  
print(typeof(x))

## [1] "double"

# Method 3  
x<-c(1.787,0.63573,2.3890)  
# Display the contents of x  
print(x)

## [1] 1.78700 0.63573 2.38900

# Display the type of x  
print(typeof(x))

## [1] "double"

### Implicit coercion

#### Example 1

# Create a vector  
x <- c(1.8)  
# Check the type of x  
typeof(x)

## [1] "double"

# Add a character to the vector  
x <- c(x,'a')  
# Check the type of x   
typeof(x)

## [1] "character"

#### Example 2

# Create a vector  
x <- c(TRUE)  
# Check the type of x  
typeof(x)

## [1] "logical"

# Add a number to the vector  
x <- c(x,2)  
# Check the type of x  
typeof(x)

## [1] "double"

#### Example 3

# Create a vector  
x <- c('a')  
# Check the type of x  
typeof(x)

## [1] "character"

# Add a logical value to the vector  
x <- c(x,TRUE)  
# Check the type of x  
typeof(x)

## [1] "character"

#### Example 4

# Create a vector  
x <- c(1L)  
# Check the type of x  
typeof(x)

## [1] "integer"

# Add a number to the vector  
x <- c(x,2)  
# Check the type of x  
typeof(x)

## [1] "double"

### Explicit coercion

#### Example 1

# Create a vector  
x <- c(1L)  
# Check the type of x  
typeof(x)

## [1] "integer"

# Convert the vector to type character  
x <- as.character(x)  
# Check the type of x  
typeof(x)

## [1] "character"

#### Example 2

# Create a vector  
x <- c('A')  
# Check the type of x  
typeof(x)

## [1] "character"

# Convert the vector to type double  
x <- as.numeric(x)

## Warning: NAs introduced by coercion

# Check the type of x  
typeof(x)

## [1] "double"

### Accessing elements of the vector

# Create a vector  
x <- c(1,10,9,8,1,3,5)

# Access one element with index 3  
x[3]

## [1] 9

# Access elements with consecutive indices, 2 to 4: 2,3,4  
x[2:4]

## [1] 10 9 8

# Access elements with non-consecutive indices, 1,3,5  
x[c(1,3,5)]

## [1] 1 9 1

# Access elements using logical vector  
x[c(TRUE,FALSE,FALSE,TRUE,FALSE,FALSE,TRUE)]

## [1] 1 8 5

# Access elements using the conditional operator <  
x[x<10]

## [1] 1 9 8 1 3 5

### Examining vectors

# Display the length of the vector  
print(length(x))

## [1] 7

# Display the type of the vector  
print(typeof(x))

## [1] "double"

# Display the structure of the vector  
print(str(x))

## num [1:7] 1 10 9 8 1 3 5  
## NULL

### Lists

# Initialise a named list  
my\_pie = list(type="key lime", diameter=7, is.vegetarian=TRUE)  
# display the list  
my\_pie

## $type  
## [1] "key lime"  
##   
## $diameter  
## [1] 7  
##   
## $is.vegetarian  
## [1] TRUE

# Print the names of the list  
names(my\_pie)

## [1] "type" "diameter" "is.vegetarian"

# Retrieve the element named type  
my\_pie$type

## [1] "key lime"

# Retrieve a truncated list  
my\_pie["type"]

## $type  
## [1] "key lime"

# Retrieve the element named type  
my\_pie[["type"]]

## [1] "key lime"

#### Exploring data-sets

# Install package  
  
# Load the package  
library(openintro)

## Loading required package: airports

## Loading required package: cherryblossom

## Loading required package: usdata

# Load package  
library(tidyverse)

# Catch a glimpse of the data-set: see how the rows are stacked one below another  
glimpse(loans\_full\_schema)

## Rows: 10,000  
## Columns: 55  
## $ emp\_title <chr> "global config engineer ", "warehouse…  
## $ emp\_length <dbl> 3, 10, 3, 1, 10, NA, 10, 10, 10, 3, 1…  
## $ state <fct> NJ, HI, WI, PA, CA, KY, MI, AZ, NV, I…  
## $ homeownership <fct> MORTGAGE, RENT, RENT, RENT, RENT, OWN…  
## $ annual\_income <dbl> 90000, 40000, 40000, 30000, 35000, 34…  
## $ verified\_income <fct> Verified, Not Verified, Source Verifi…  
## $ debt\_to\_income <dbl> 18.01, 5.04, 21.15, 10.16, 57.96, 6.4…  
## $ annual\_income\_joint <dbl> NA, NA, NA, NA, 57000, NA, 155000, NA…  
## $ verification\_income\_joint <fct> , , , , Verified, , Not Verified, , ,…  
## $ debt\_to\_income\_joint <dbl> NA, NA, NA, NA, 37.66, NA, 13.12, NA,…  
## $ delinq\_2y <int> 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0…  
## $ months\_since\_last\_delinq <int> 38, NA, 28, NA, NA, 3, NA, 19, 18, NA…  
## $ earliest\_credit\_line <dbl> 2001, 1996, 2006, 2007, 2008, 1990, 2…  
## $ inquiries\_last\_12m <int> 6, 1, 4, 0, 7, 6, 1, 1, 3, 0, 4, 4, 8…  
## $ total\_credit\_lines <int> 28, 30, 31, 4, 22, 32, 12, 30, 35, 9,…  
## $ open\_credit\_lines <int> 10, 14, 10, 4, 16, 12, 10, 15, 21, 6,…  
## $ total\_credit\_limit <int> 70795, 28800, 24193, 25400, 69839, 42…  
## $ total\_credit\_utilized <int> 38767, 4321, 16000, 4997, 52722, 3898…  
## $ num\_collections\_last\_12m <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ num\_historical\_failed\_to\_pay <int> 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0…  
## $ months\_since\_90d\_late <int> 38, NA, 28, NA, NA, 60, NA, 71, 18, N…  
## $ current\_accounts\_delinq <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ total\_collection\_amount\_ever <int> 1250, 0, 432, 0, 0, 0, 0, 0, 0, 0, 0,…  
## $ current\_installment\_accounts <int> 2, 0, 1, 1, 1, 0, 2, 2, 6, 1, 2, 1, 2…  
## $ accounts\_opened\_24m <int> 5, 11, 13, 1, 6, 2, 1, 4, 10, 5, 6, 7…  
## $ months\_since\_last\_credit\_inquiry <int> 5, 8, 7, 15, 4, 5, 9, 7, 4, 17, 3, 4,…  
## $ num\_satisfactory\_accounts <int> 10, 14, 10, 4, 16, 12, 10, 15, 21, 6,…  
## $ num\_accounts\_120d\_past\_due <int> 0, 0, 0, 0, 0, 0, 0, NA, 0, 0, 0, 0, …  
## $ num\_accounts\_30d\_past\_due <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ num\_active\_debit\_accounts <int> 2, 3, 3, 2, 10, 1, 3, 5, 11, 3, 2, 2,…  
## $ total\_debit\_limit <int> 11100, 16500, 4300, 19400, 32700, 272…  
## $ num\_total\_cc\_accounts <int> 14, 24, 14, 3, 20, 27, 8, 16, 19, 7, …  
## $ num\_open\_cc\_accounts <int> 8, 14, 8, 3, 15, 12, 7, 12, 14, 5, 8,…  
## $ num\_cc\_carrying\_balance <int> 6, 4, 6, 2, 13, 5, 6, 10, 14, 3, 5, 3…  
## $ num\_mort\_accounts <int> 1, 0, 0, 0, 0, 3, 2, 7, 2, 0, 2, 3, 3…  
## $ account\_never\_delinq\_percent <dbl> 92.9, 100.0, 93.5, 100.0, 100.0, 78.1…  
## $ tax\_liens <int> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ public\_record\_bankrupt <int> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0…  
## $ loan\_purpose <fct> moving, debt\_consolidation, other, de…  
## $ application\_type <fct> individual, individual, individual, i…  
## $ loan\_amount <int> 28000, 5000, 2000, 21600, 23000, 5000…  
## $ term <dbl> 60, 36, 36, 36, 36, 36, 60, 60, 36, 3…  
## $ interest\_rate <dbl> 14.07, 12.61, 17.09, 6.72, 14.07, 6.7…  
## $ installment <dbl> 652.53, 167.54, 71.40, 664.19, 786.87…  
## $ grade <fct> C, C, D, A, C, A, C, B, C, A, C, B, C…  
## $ sub\_grade <fct> C3, C1, D1, A3, C3, A3, C2, B5, C2, A…  
## $ issue\_month <fct> Mar-2018, Feb-2018, Feb-2018, Jan-201…  
## $ loan\_status <fct> Current, Current, Current, Current, C…  
## $ initial\_listing\_status <fct> whole, whole, fractional, whole, whol…  
## $ disbursement\_method <fct> Cash, Cash, Cash, Cash, Cash, Cash, C…  
## $ balance <dbl> 27015.86, 4651.37, 1824.63, 18853.26,…  
## $ paid\_total <dbl> 1999.330, 499.120, 281.800, 3312.890,…  
## $ paid\_principal <dbl> 984.14, 348.63, 175.37, 2746.74, 1569…  
## $ paid\_interest <dbl> 1015.19, 150.49, 106.43, 566.15, 754.…  
## $ paid\_late\_fees <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…

# Selecting numeric variables  
loans <- loans\_full\_schema %>% # <-- pipe operator  
 select(paid\_total, term, interest\_rate,  
 annual\_income,paid\_late\_fees,debt\_to\_income)  
# View the columns stacked one below another  
glimpse(loans)

## Rows: 10,000  
## Columns: 6  
## $ paid\_total <dbl> 1999.330, 499.120, 281.800, 3312.890, 2324.650, 873.130…  
## $ term <dbl> 60, 36, 36, 36, 36, 36, 60, 60, 36, 36, 60, 60, 36, 60,…  
## $ interest\_rate <dbl> 14.07, 12.61, 17.09, 6.72, 14.07, 6.72, 13.59, 11.99, 1…  
## $ annual\_income <dbl> 90000, 40000, 40000, 30000, 35000, 34000, 35000, 110000…  
## $ paid\_late\_fees <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ debt\_to\_income <dbl> 18.01, 5.04, 21.15, 10.16, 57.96, 6.46, 23.66, 16.19, 3…

# Selecting categoric variables  
loans <- loans\_full\_schema %>%   
 select( ) # type the chosen columns as in the lecture slide  
# View the columns stacked one below another  
glimpse(loans)

## Rows: 10,000  
## Columns: 0