Lab 09 Solutions

2026-03-26

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Load data
police <- read.csv("https://remiller1450.github.io/data/Police2019.csv")
college <- read.csv("https://collinn.github.io/data/college2019.csv")
```

Question 1

```
top_table <- function(x) {
  v <- table(x)
  sort(v, decreasing = TRUE)[1:5]
}
top_table(police$state)
```

```
## x
##  CA  TX  FL  AZ  CO
## 825 496 369 259 204
```

```
top_table(college$State)
```

```
## x
## PA NY CA TX OH
## 85 67 63 60 48
```

Question 2

```
top_table <- function(x, n) {
  v <- table(x)
  sort(v, decreasing = TRUE)[1:n]
}
top_table(police$state, n = 10)
```

```
## x
##  CA  TX  FL  AZ  CO  GA  OK  NC  OH  WA
## 825 496 369 259 204 189 170 163 159 156
```

```
top_table(college$State, n = 10)
```

```
## x
## PA NY CA TX OH IL NC MA MI IN
## 85 67 63 60 48 45 40 36 34 33
```

Question 3

```
long_square <- function(n) {
  if (length(n) > sqrt(sum(n))) {
    print("long!")
  } else {
    print("not long!")
  }
}

x1 <- c(1, 2, 3, 4, 5)
x2 <- c(5, 8, 10, 12)
x3 <- c(2, 5, 9, 10, 1, 1, 1)

long_square(n = x1)
```

```
## [1] "long!"
```

```
long_square(n = x2)
```

```
## [1] "not long!"
```

```
long_square(n = x3)
```

```
## [1] "long!"
```

Question 4

```
top_table <- function(x, n, top = TRUE) {
  v <- table(x)

  if (top == TRUE) {
    sort(v, decreasing = TRUE)[1:n]
  } else {
    sort(v, decreasing = FALSE)[1:n]
  }
}
top_table(police$state, n = 5, top = TRUE)
```

```
## x
##  CA  TX  FL  AZ  CO
## 825 496 369 259 204
```

```
top_table(college$State, n = 10, top = FALSE)
```

```
## x
## AK WY NV DE NM AZ DC ID NH RI
##  1  1  2  3  4  5  5  5  5  5
```

Question 5

Make the name grabber function that works like this:

```
name_grabber <- function(l, name) {
  if (name  %in% names(l)) {
    return(l[[name]])
  } else {
      return(NULL)
    }
}

l <- list("apple" = 1:5,
          "bumblebee" = letters[1:5],
          "capybara" = iris[1:5, ])

name_grabber(l, "capybara")
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

```
name_grabber(l, "doggo")
```

```
## NULL
```

Question 6

1. Using the `college` dataset, create a 2 way table with the variables `Type` and `Region`
2. Use the `chisq.test()` function to test the hypothesis that these variables are independent
3. Using list constructs, extract the expected values under independent and save it to an object. Recreate the χ^2 test statistic using the tables of observed and expected values (note: in the same way we can add two vectors of the same length, we can do the same with matrices. This idea extends to many arithmetical operations)

```
obs <- with(college, table(Region, Type))
ex <- chisq.test(obs)$expected

## These should be the same
((obs - ex)^2 / ex) %>% sum()
```

```
## [1] 30.215
```

```
chisq.test(obs)$statistic
```

```
## X-squared
##    30.215
```

Question 7

Given a a linear model and a Type I error rate alpha, write a function that returns how many predictors are significant at a given alpha level. Verify it works on given models

```r
countsig <- function(fit, alpha = 0.05) {
  ss <- summary(fit)$coefficients[, 4]
  sum(ss < alpha)
}

## Use these models
fit1 <- lm(circumference ~ age, data = Orange)
fit2 <- lm(Murder ~ Assault + UrbanPop, data = USArrests)
fit3 <- lm(mpg ~ disp + hp + wt + qsec, data = mtcars)


## You function should take these arguments and return the same values
countsig(fit1, alpha = 0.05)
```

```
## [1] 1
```

```r
countsig(fit2, alpha = 0.1)
```

```
## [1] 3
```

```r
countsig(fit3, alpha = 0.2)
```

```
## [1] 2
```

Question 8

Rewrite last lab thing with lapply

```r
exp_files <- list.files(path = "../../labs/fundir/hwk/", pattern = "csv", full.names = TRUE)

## Create storage vectors
subject_name <- character(length = length(exp_files))
sex <- character(length = length(exp_files))
group <- character(length = length(exp_files))
mean_value <- numeric(length = length(exp_files))

tt <- lapply(exp_files, function(x) {
  dat <- read.csv(x)
  fname <- sub("\\.csv$", "", basename(x))
  df <- data.frame(
    sex = sub(".*([MF])exp[ABC]$", "\\1", fname),
    group = sub(".*exp([ABC])$", "\\1", fname),
    sub = sub("([MF])exp[ABC]$", "", fname),
    val = mean(dat$x)
  )
})

df <- Reduce(rbind, tt)
head(df)
```

```
##   sex group     sub      val
## 1   F     A  Amanda -4.19982
## 2   F     B     Amy -1.14893
## 3   M     C  Andrew  8.16950
```

```
## 4   F     B  Angela  0.52793
## 5   F     C  Ashley 18.72791
## 6   F     A Barbara  2.37037
```

Question 9

Look at the documentation for `lapply()`. How is `sapply()` different? The code below is not run, but you can use it to investigate. Can you perform the same task with `vapply()`? See examples in the documentation if you get stuck

```
## Create list of data
df_list <- lapply(1:5, function(x) {
  data.frame(x = rnorm(10),
             y = rnorm(10))
})

lapply(df_list, function(df) {
  cor(df$x, df$y)
})
```

```
## [[1]]
## [1] 0.35858
##
## [[2]]
## [1] 0.12126
##
## [[3]]
## [1] 0.14535
##
## [[4]]
## [1] -0.36297
##
## [[5]]
## [1] -0.58125
```

```
lapply(df_list, function(df) {
  cor(df$x, df$y)
})
```

```
## [[1]]
## [1] 0.35858
##
## [[2]]
## [1] 0.12126
##
## [[3]]
## [1] 0.14535
##
## [[4]]
## [1] -0.36297
##
## [[5]]
## [1] -0.58125
```

- lapply returns a list
- sapply will try to simplify it to an array if possible
- vapply works by specifying the return type

```
## Using vapply
vapply(df_list, function(df) {
  with(df, cor(x,y))
}, numeric(1))
```

```
## [1]  0.35858  0.12126  0.14535 -0.36297 -0.58125
```

Question 10

The `apply()` function is slightly different than the the rest of the *apply family in that rather than taking a vector or list of objects, it operates on a matrix (a matrix is similar to a data.frame with the exception that all of the entries must be of the same type e.g., numeric or character). Use `apply()` to verify the row and column sums given below.

```
set.seed(123)
m <- matrix(rnorm(25), nrow = 5)

## Verify these values with apply
rowSums(m)
```

```
## [1]  3.09776  0.87043 -2.29820 -0.53320 -1.97005
```

```
colSums(m)
```

```
## [1]  0.96785 -0.22159  1.53951  0.54671 -3.66573
```

```
## Row sums
apply(m, 1, sum)
```

```
## [1]  3.09776  0.87043 -2.29820 -0.53320 -1.97005
```

```
## Col sums
apply(m, 2, sum)
```

```
## [1]  0.96785 -0.22159  1.53951  0.54671 -3.66573
```