Homework 1 - EEB590C

Devin Molnau, Holly Loper, Elizabeth McMurchie

February 18, 2021

EEB590C -Homework 1

Homework 1: Lectures 1-4

This assignment is due prior to class in week 6. You are to self-select and work in groups: 2-3 in a group. For the assignment below submit one R-script. Annotations via comments are highly encouraged. The script should run!

Assignment Instructions:

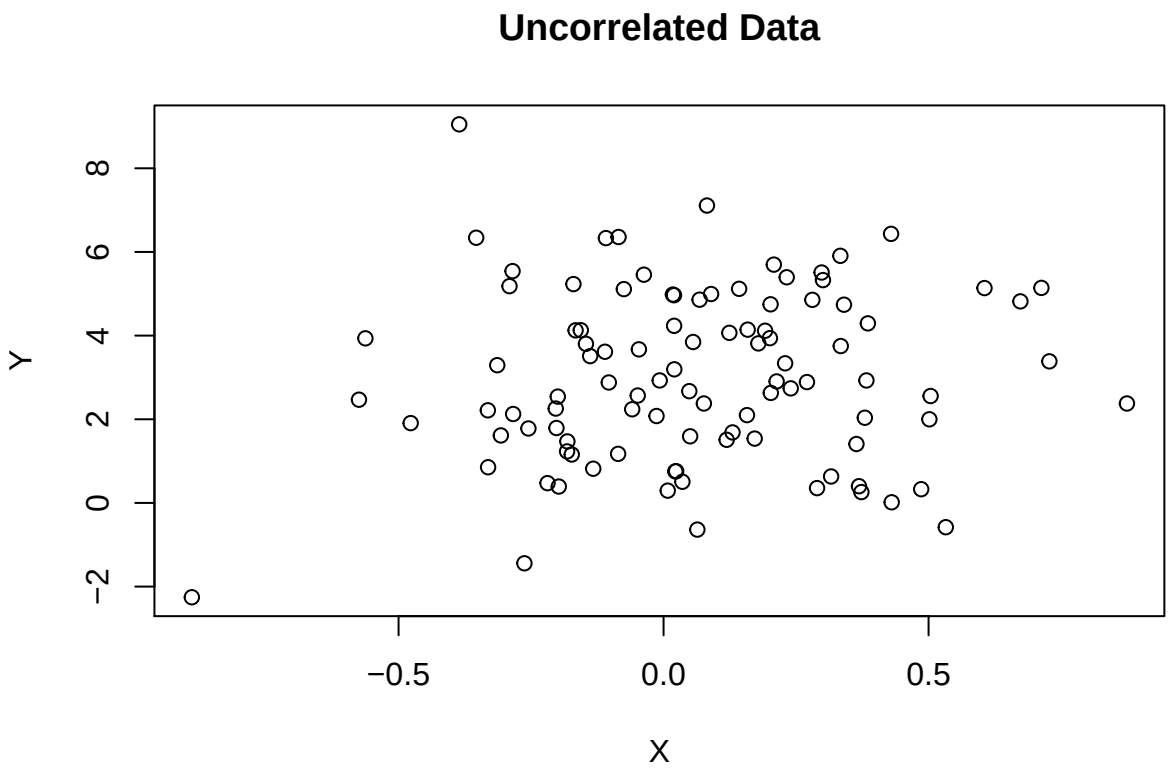
1: Select some form of linear model containing a single dependent variable (continuous) and at least 1 independent variable. Next, simulate two datasets: the first with no relationship between X & Y, and the second with some positive association between X & Y. Perform 100 simulations under each condition. Run the linear models on all datasets to confirm that on average, the patterns for condition 1 (no relationship) and condition 2 (some relationship) are met. (HINT: this requires determining an appropriate summary measure extracted from the linear model).

2: Devise a permutation procedure to evaluate the above linear model. Write code for this permutation procedure. Next, devise a SECOND implementation of the same permutation procedure (ie, code the procedure in a different manner). For a single dataset compare the two implementations for their computational performance. Summarize your findings via comments in the code (e.g., which approach was faster? Which components of the slower approach could be improved, etc.).

Homework Breakdown:

We selected lm as a linear model and we simulated two datasets: the first with no relationship between X amd Y, and the second with positive association between X and Y. Here we have the random uncorrelated data, sampling 100 datapoints with y and x from normal distributions.

```
# data set with no relationship between x & Y
random_corr_gen <- function(elements) {
    x1 <- rnorm(elements, mean = 0, sd = 0.3)
    y1 <- rnorm(elements, mean = 3, sd = 2)
    uncor_dataframe <- data.frame(x1, y1)
}

uncor_data <- random_corr_gen(100)
plot(uncor_data, main = "Uncorrelated Data", xlab = "X", ylab = "Y")
```



Here, we have the positively correlated, normally distributed, Y and X data generated using a correlation of 0.9.

```
# data set with positive association between x & Y
library(mvtnorm)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.6     v dplyr   1.0.4
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
corr.val <- 0.9
pos_cor_gen <- function(elements) {
    cor_data <- data.frame(rmvnorm(n = elements, mean = c(0,
        0), sigma = matrix(c(1, corr.val, corr.val, 1), 2, 2)))
    names(cor_data)[1] <- "x1"
    names(cor_data)[2] <- "y1"
    return(cor_data)
}

pos_cor_data <- pos_cor_gen(100)
```

We then perform 100 simulations under each condition using the replicate function.

First we simulate 100 uncorrelated x and y dataframes and put it a list called uncor_data_sim.

```
# Random Uncorrelated Data List
uncor_data_sim <- replicate(100, random_corr_gen(100), simplify = FALSE)
```

Then we simulate 100 lists of the correlated data, called cor_data_sim.

```
# Random Correlated Data List
cor_data_sim <- replicate(100, pos_cor_gen(100), simplify = FALSE)
```

We then run the linear model lm on the two datasets (uncorrelated and correlated) to confirm that on average, the patterns for condition 1 (no relationship) and condition 2 (some relationship) are met.

Below is the linear model lm function applied to each list of x and y in the uncorrelated dataset. The summary of the linear model is saved to summary_lm_uncor_tests.

```
# model 1 - uncorrelated
lm_uncor_tests <- lapply(uncor_data_sim, lm, formula = y1 ~ x1)
summary_lm_uncor_tests <- lapply(lm_uncor_tests, summary)
anova_lm_uncor_tests <- lapply(lm_uncor_tests, anova)
```

Of the 100 simulations, a grand total of 97 simulations had a pvalue that was greater than 0.05, indicating that there is NOT a linear relationship between X and Y at a significance level of 0.05.

```
sum_pvalue_uncor_nonsig <- 0
sum_pvalue_uncor_sig <- 0
count_pvalue_uncor_nonsig <- 0
count_pvalue_uncor_sig <- 0

for (i in 1:length(anova_lm_uncor_tests)) {
    if (anova_lm_uncor_tests[[i]][["Pr(>F)"]][1] > 0.05) {
        count_pvalue_uncor_nonsig <- count_pvalue_uncor_nonsig +
            1
        sum_pvalue_uncor_nonsig <- sum_pvalue_uncor_nonsig +
            anova_lm_uncor_tests[[i]][["Pr(>F)"]][1]
    } else {
        count_pvalue_uncor_sig <- count_pvalue_uncor_sig + 1
        # print(i)
        sum_pvalue_uncor_sig <- sum_pvalue_uncor_sig + anova_lm_uncor_tests[[i]][["Pr(>F)"]][1]
    }
}



average_pvalue_uncor_tests_nonsig <- sum_pvalue_uncor_nonsig/count_pvalue_uncor_nonsig
print(paste("Average pvalue less than 0.05 for uncorrelated tests:",
    average_pvalue_uncor_tests_nonsig))
```

```
## [1] "Average pvalue less than 0.05 for uncorrelated tests: 0.535079910649765"
```

```
average_pvalue_uncor_tests_sig <- sum_pvalue_uncor_sig/count_pvalue_uncor_sig
print(paste("Average p-value greater than 0.05 for uncorrelated tests:",
    average_pvalue_uncor_tests_sig))
```

```
## [1] "Average p-value greater than 0.05 for uncorrelated tests: 0.0161556417412891"
```

```
average_pvalue_uncor_tests <- (sum_pvalue_uncor_nonsig + sum_pvalue_uncor_sig)/(count_pvalue_uncor_sig +
    count_pvalue_uncor_nonsig)
print(paste("Average p-value for all uncorrelated tests:", average_pvalue_uncor_tests))
```

```
## [1] "Average p-value for all uncorrelated tests: 0.509133697204341"
```

```
# TODO SUM THE NUMBER OF OCCURENCES WHERE THE PVALUE is LESS
# THAN 0.05. THIS SHOULD BE A COUNT AND WILL BE AN INTEGER
```

```
# of 100 simulations, the count of that had a nonsignificant
# p-value
count_pvalue_uncor_nonsig
```

```
## [1] 95
```

```
# of 100 simulations, the count of that had a significant
# p-value
count_pvalue_uncor_sig
```

```
## [1] 5
```

Next is the linear model lm applied to each list of the positively correlated dataset.

```
# model 2 - positively correlated model
lm_cor_tests <- lapply(cor_data_sim, lm, formula = y1 ~ x1)
summary_lm_cor_tests <- lapply(lm_cor_tests, summary)
anova_lm_cor_tests <- lapply(lm_cor_tests, anova)
```

Of the 100 simulations, all 100 simulations have a pvalue that is less than 0.05, indicating that there is a linear relationship between X and Y at a significance level of 0.05.

```
sum_pvalue_cor_nonsig <- 0
sum_pvalue_cor_sig <- 0
count_pvalue_cor_nonsig <- 0
count_pvalue_cor_sig <- 0

for (i in 1:length(anova_lm_cor_tests)) {
    if (anova_lm_cor_tests[[i]][["Pr(>F)"]][1] > 0.05) {
        print(i)
        count_pvalue_cor_nonsig <- count_pvalue_cor_nonsig +
            1
        sum_pvalue_cor_nonsig <- sum_pvalue_cor_nonsig + anova_lm_cor_tests[[i]][["Pr(>F)"]][1]
    } else {
        count_pvalue_cor_sig <- count_pvalue_cor_sig + 1
        sum_pvalue_cor_sig <- sum_pvalue_cor_sig + anova_lm_cor_tests[[i]][["Pr(>F)"]][1]
    }
}

average_pvalue_cor_tests_sig <- sum_pvalue_cor_sig/count_pvalue_cor_sig
print(paste("Average pvalue less than 0.05 for correlated tests:",
    average_pvalue_cor_tests_sig))
```

```
## [1] "Average pvalue less than 0.05 for correlated tests: 1.71263030933319e-30"
```

```
average_pvalue_cor_tests_nonsig <- sum_pvalue_cor_nonsig/count_pvalue_cor_nonsig
print(paste("Average p-value greater than 0.05 for correlated tests:",
    average_pvalue_cor_tests_nonsig))
```

```
## [1] "Average p-value greater than 0.05 for correlated tests: NaN"
```

```
average_pvalue_cor_tests <- (sum_pvalue_cor_nonsig + sum_pvalue_cor_sig)/(count_pvalue_cor_sig +
    count_pvalue_cor_nonsig)
print(paste("Average p-value for all correlated tests:", average_pvalue_cor_tests))
```

```
## [1] "Average p-value for all correlated tests: 1.71263030933319e-30"
```

```
# of 100 simulations for the correlated data, the count of
# simulations with a nonsignificant p-value
count_pvalue_cor_nonsig
```

```
## [1] 0
```

```
# of 100 simulations, the count of simulations with a
# significant p-value
count_pvalue_cor_sig
```

```
## [1] 100
```

Question 2:

The first permutation procedure we used to evaluate the above linear model on the correlated data using the RRPP package.

```
# install.packages('RRPP')
library(RRPP)
ourdata <- rrpp.data.frame(pos_cor_data[, 2], pos_cor_data[,
    1])
model3 <- lm.rrpp(pos_cor_data[, 2] ~ pos_cor_data[, 1], iter = 999,
    print.progress = FALSE, data = ourdata)
anova(model3)
```

```
##
## Analysis of Variance, using Residual Randomization
## Permutation procedure: Randomization of null model residuals
## Number of permutations: 1000
## Estimation method: Ordinary Least Squares
## Sums of Squares and Cross-products: Type I
## Effect sizes (Z) based on F distributions
##
##                   Df      SS     MS   Rsq     F      Z Pr(>F)
## pos_cor_data[, 1]  1  85.205 85.205 0.826 465.2 8.7051  0.001 **
## Residuals         98  17.949  0.183 0.174
## Total             99 103.154
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Call: lm.rrpp(f1 = pos_cor_data[, 2] ~ pos_cor_data[, 1], iter = 999,
##     data = ourdata, print.progress = FALSE)
```

Next, we devised a SECOND implementation of the same permutation procedure. This time we will write a permutation from scratch for 999 iterations using a for loop. We set the observed data's f value to be one of the iterations.

```
pos_cor_lm <- anova(lm(pos_cor_data$y1 ~ pos_cor_data$x1))
f.obs <- pos_cor_lm$`Pr(>F)`[1]
```

We then created a function for 999 permutations so that we could later time the operations.

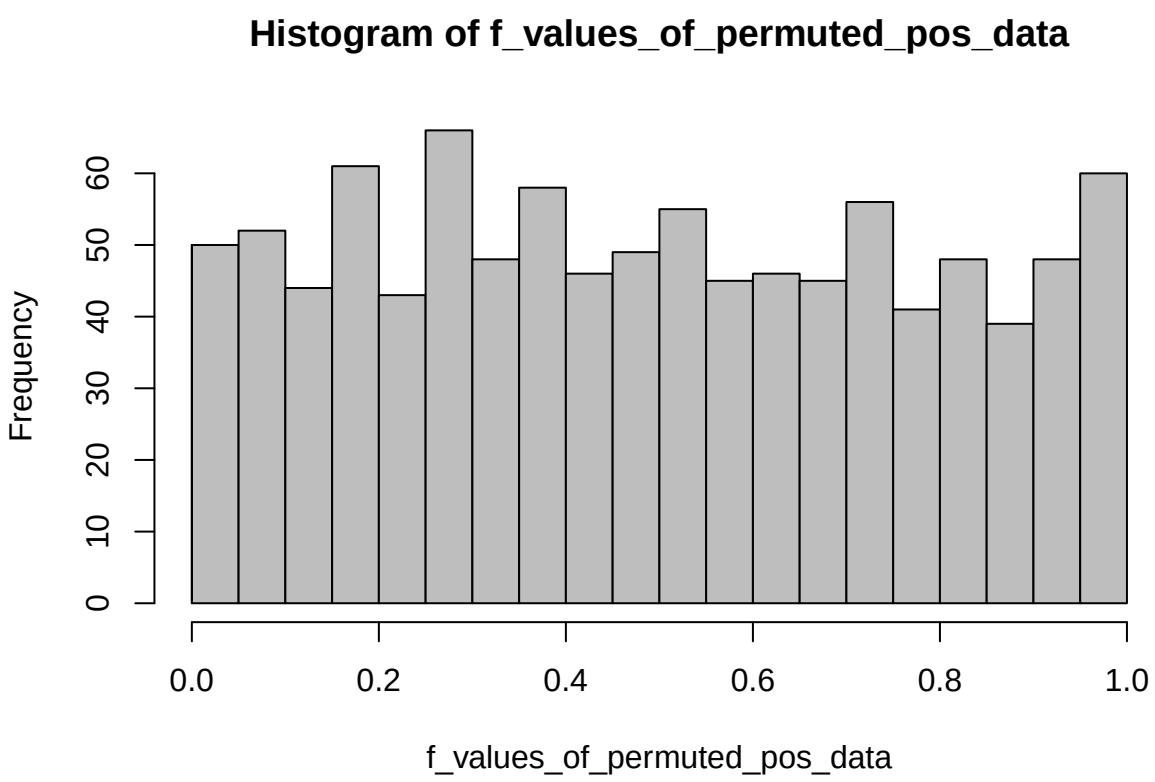
```r
# Randomization function

permute_correlated_data <- function(correlated_data, permute_num,
    f.obs.reported) {

    P.1tailed <- P.2tailed <- 1
    f.rand.vec <- array(NA, (permute_num + 1))
    f.rand.vec[permute_num + 1] <- f.obs.reported

    for (i in 1:permute_num) {
        ### Shuffle Data
        permuted_cor_data <- sample(correlated_data$y1)
        ### Run analysis on random data
        permuted_pos_cor_lm <- anova(lm(permuted_cor_data ~ correlated_data$x1))
        f.rand.vec[i] <- permuted_pos_cor_lm$`Pr(>F)`[1]
    }  #end permute
    return(f.rand.vec)
}
f_values_of_permuted_pos_data <- permute_correlated_data(pos_cor_data,
    999, f.obs)

permute <- 999

## Significance assessment
P.1tailed <- length(which(f_values_of_permuted_pos_data <= f_values_of_permuted_pos_data[permute +
    1]))/(permute + 1)  #because observed is negative
P.2tailed <- length(which(abs(f_values_of_permuted_pos_data) >=
    abs(f_values_of_permuted_pos_data[permute + 1])))/(permute +
    1)
P.1tailed
```

```
## [1] 0.001
```

```r
P.2tailed
```

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

```r
#### Plot
hist(f_values_of_permuted_pos_data, 20, freq = T, col = "gray")
segments(f.obs, 0, f.obs, 50)  ##Plot Observed value
```



Below are the times of the two permutation methods. As you can see, RRPP is the superior method when simply comparing the time as well as ease of use. The time of RRPP permuting 999 iterations is reported in milliseconds (~ 156 milliseconds) while the permutation that was written out in a for loop yields results in terms of seconds (~ 1.866 seconds).

```
library(microbenchmark)
res <- microbenchmark(anova(lm.rrpp(pos_cor_data[, 2] ~ pos_cor_data[,
    1], iter = 999, print.progress = FALSE, data = ourdata)),
    times = 1)  #using rrpp for permutation method
print(res)
```

```
## Unit: milliseconds
##
##  anova(lm.rrpp(pos_cor_data[, 2] ~ pos_cor_data[, 1], iter = 999,      print.progress = FALSE, data =
##       min       lq     mean   median       uq      max neval
##  141.6455 141.6455 141.6455 141.6455 141.6455 141.6455     1
```

```
res2 <- microbenchmark(permute_correlated_data(pos_cor_data,
    999, f.obs), times = 1)
print(res2)  #second permutation method
```

```
## Unit: seconds
##                                           expr      min       lq     mean
##  permute_correlated_data(pos_cor_data, 999, f.obs) 2.318868 2.318868 2.318868
##    median       uq      max neval
##  2.318868 2.318868 2.318868     1
```