Assignment II

Aldo Giovanni e Giacomo,Anuja Saira Abraham 5204982,Alessia Marzotti 5108443,Miriam Mercuri 5207

2023-06-06

EXERCISE 1 - RISIKO! is back, with friends

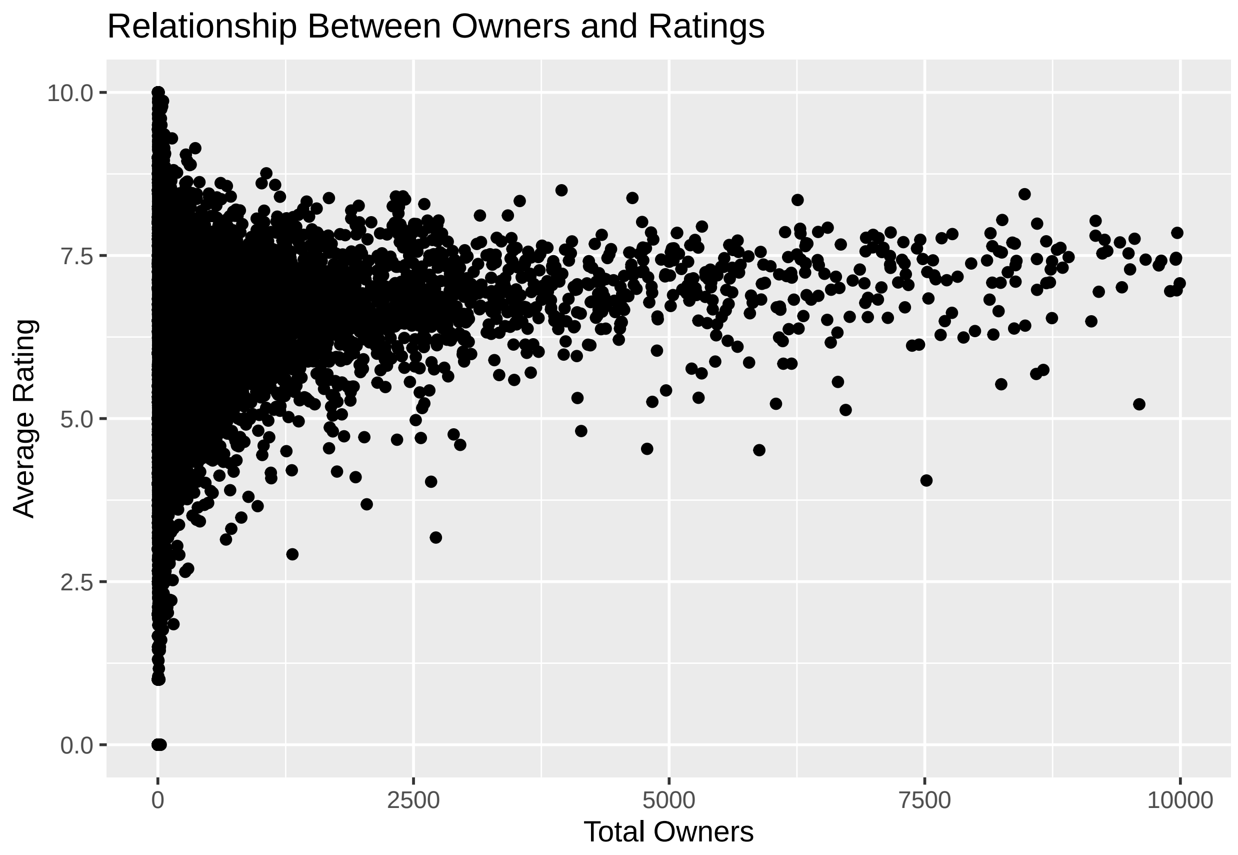
```
library(ggplot2)
library(ggcorrplot)
library(boot)
library(tidyverse)
library(reshape2)
library(combinat)
```

Point a

```
games <- readRDS("games_preprocessed.RDS")
```

The dataset *game* contains data about around 25000 board games.The variables that describe each game are eight. They are the year in which it has been published (*yearpublished*), the minimum and the maximum number of players that can play the game (*minplayers*, *maxplayers*) and the minimun age to play (*minage*). Then there is the *average_rating*, *total_owners* and *average_weight*.All are quantitative variables except for the first (*id*) and second (*name*) ones that we are not going to consider for our exercise.

```
ggplot(games, aes(x = total_owners, y = average_rating)) +
  geom_point() +
  xlab("Total Owners") +
  ylab("Average Rating") +
  ggtitle("Relationship Between Owners and Ratings")
```

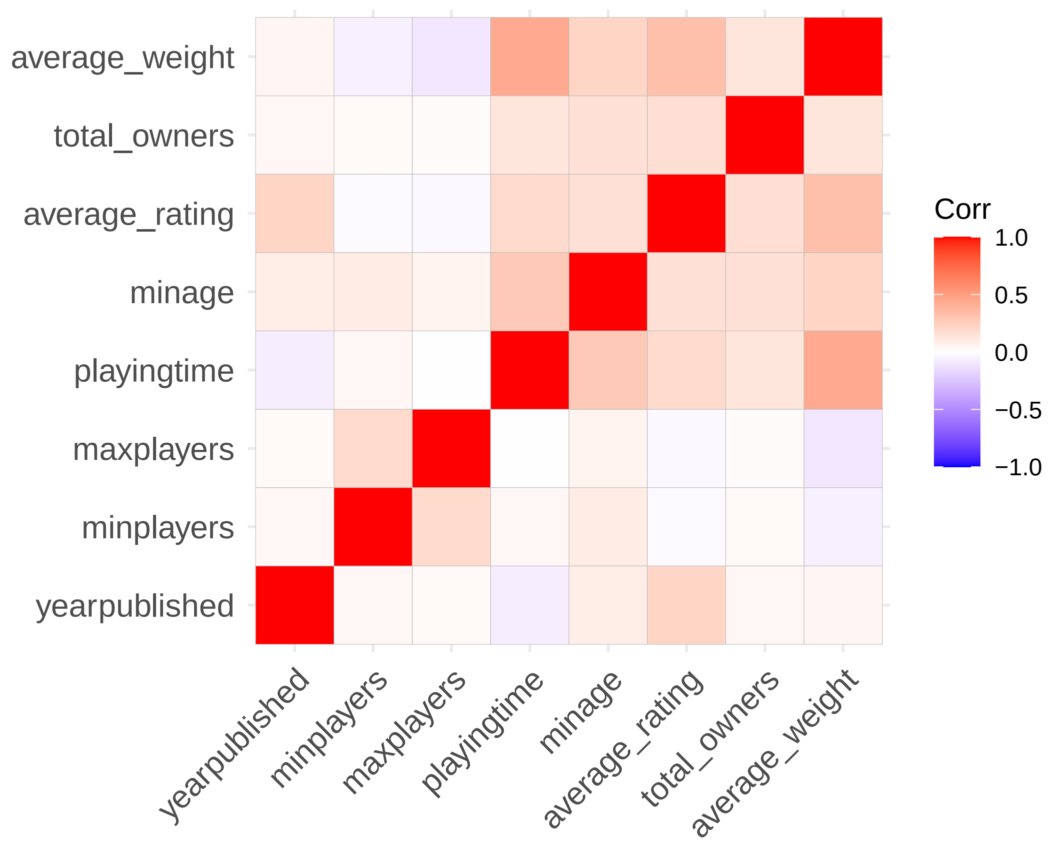


Considering, for example, the two variable *average_rating* and *total_owners*, the plot above make us see which is the relationship between them.

Point b

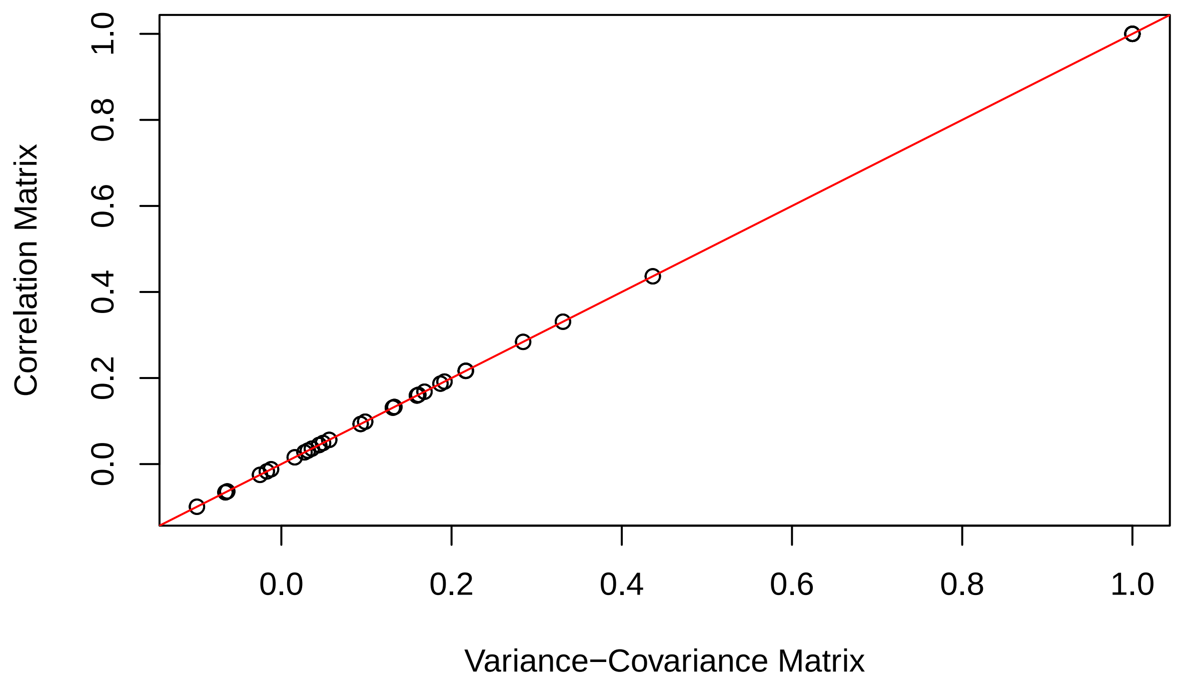
```
games <- games[,-c(1,2)]
games <- scale(games)
```

```
cov_matrix <- cov(games)
ggcorrplot(cov_matrix)
```



From the plot and the variance-covariance matrix values we can see that the variables don't have a strong relationship in general.

```r
cor_matrix <- cor(games)
plot(cov_matrix, cor_matrix,
     xlab = "Variance-Covariance Matrix",
     ylab = "Correlation Matrix")
abline(0,1,col = "red")
```



The correlation matrix is the same as the variance-covariance matrix because our dataset has been scaled, resulting in equivalent covariance and correlation values.

Point c

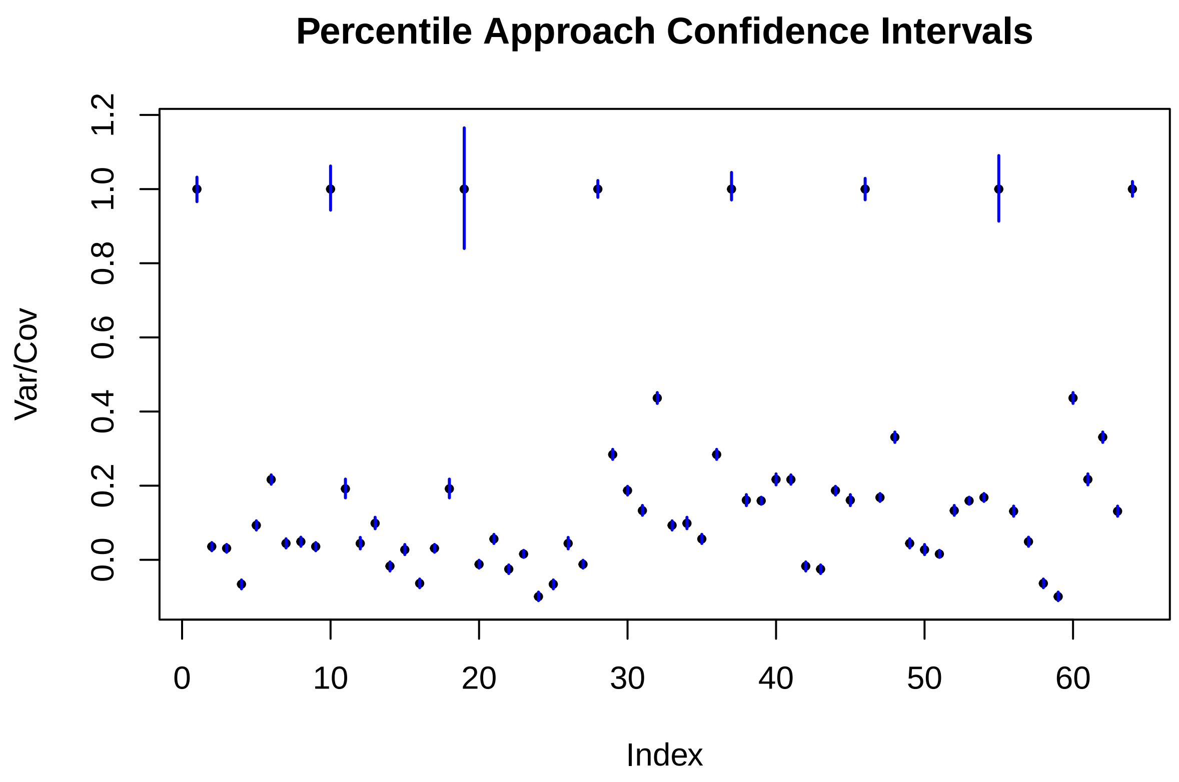
Now we perform nonparametric bootstrap to obtain standard error estimates for each of the entries of the variance-covariance matrix.

```
set.seed(12345)

B <- 1000
boot <- function(B, games){

  results <- matrix(NA, nrow = B, ncol = ncol(games)^2)
  bootstrap <- replicate(B, {

    bootSample <- games[sample(nrow(games), replace = T),]
    bootCov <- cov(bootSample)
    i <- seq_len(B)
    results[i,] <- as.vector(bootCov)
  })
  return(bootstrap)
}

boot <- boot(1000,games)

standard_errors <- apply(boot, 1, sd)
standard_errors
```

```
##  [1] 0.016767571 0.005726342 0.005392461 0.006026930 0.006611704 0.006527865
##  [7] 0.006366352 0.006391746 0.005726342 0.030878533 0.012807945 0.008199441
## [13] 0.007928323 0.006332489 0.007044940 0.006013313 0.005392461 0.012807945
## [19] 0.082934774 0.005253855 0.006605301 0.005991273 0.004863436 0.005935743
## [25] 0.006026930 0.008199441 0.005253855 0.012317292 0.006937599 0.006049556
## [31] 0.006927497 0.007735376 0.006611704 0.007928323 0.006605301 0.006937599
## [37] 0.018955155 0.007341887 0.004957443 0.007548761 0.006527865 0.006332489
## [43] 0.005991273 0.006049556 0.007341887 0.015166272 0.005447973 0.007162792
## [49] 0.006366352 0.007044940 0.004863436 0.006927497 0.004957443 0.005447973
## [55] 0.044025974 0.006719733 0.006391746 0.006013313 0.005935743 0.007735376
## [61] 0.007548761 0.007162792 0.006719733 0.010072221
```

Now we obtain confidence intervals of the bootstrap distribution using the percentile approach and the central limit theorem (CLT).

```r
means <- apply(boot, 1, mean)
qnt_CI <- t(apply(boot, 1, function(z) quantile(z, probs = c(0.025,0.975))))
clt_CI <- means + c(-1,1)*1.96*standard_errors
cov_hat <- as.vector(cov_matrix)
x <- seq(length(cov_hat))
y <- cov_hat
y_upper <- unlist(qnt_CI[,2])
y_lower <- unlist(qnt_CI[,1])
plot(y, pch = 19, cex = 0.5,ylim = range(c(y_lower, y_upper)), ylab = "Var/Cov", xlab = "Index",
     main = "Percentile Approach Confidence Intervals")
segments(x, y_lower, x, y_upper, lwd = 1.5,col = "blue")
```



Above there is the plot of the confidence intervals of each entry of the variance-covariance matrix (Index) using the percentile approach.

Point d

Show the behavior of the values $\{\theta_j\}_{j=1}^{p}$ and estimate j^*.

```
eigen_values <- eigen(cov_matrix)$values
theta <- cumsum(eigen_values) / sum(eigen_values)
theta
```

```
## [1] 0.2426729 0.4008816 0.5410761 0.6550491 0.7568552 0.8551548 0.9359288
## [8] 1.0000000
```

```
jstar <- min(which(theta > 0.76))
jstar
```

```
## [1] 6
```

Point e

Perform bootstrap to estimate the bias and standard error of the estimators for the parameters in the sequence $\{\theta_j\}_{j=1}^{p}$.

```
boot_theta <- matrix(0, nrow = B, ncol = length(theta))
boot_jstar <- numeric(B)
for (i in 1:B) {
  boot_sample <- games[sample(nrow(games), replace= TRUE),]
  cov_boot <- cov(boot_sample)
  eigen_boot <- eigen(cov_boot)$values
  boot_theta[i,] <- cumsum(eigen_boot) / sum(eigen_boot)
  boot_jstar[i] <- min(which(boot_theta[i,] > 0.76))
}
```

```
bias <- colMeans(boot_theta)-theta
se <- apply(boot_theta, 2, sd)
bias
```

```
## [1] 2.660301e-04 8.058590e-04 7.381508e-04 9.166828e-04 1.767983e-03
## [6] 2.008943e-04 9.475436e-05 0.000000e+00
```

```
se
```

```
## [1] 0.003174574 0.002647125 0.002873812 0.003279194 0.002552820 0.002106016
## [7] 0.001101662 0.000000000
```

Also report the bootstrap estimates and standard errors for the estimator of j^*.

```
bias_jstar <- mean(boot_jstar) - jstar
se_jstar <- sd(boot_jstar)
bias_jstar
```

```
## [1] -0.278
```

```
se_jstar
```

```
## [1] 0.4482376
```

Finally, give an estimate for $P(j^* \hat{=} 5)$.

```
p_jstar <- mean(boot_jstar==5)
p_jstar
```

```
## [1] 0.278
```

Point f

Run a linear regression with average_rating as target variable, regressed over all the other variables.

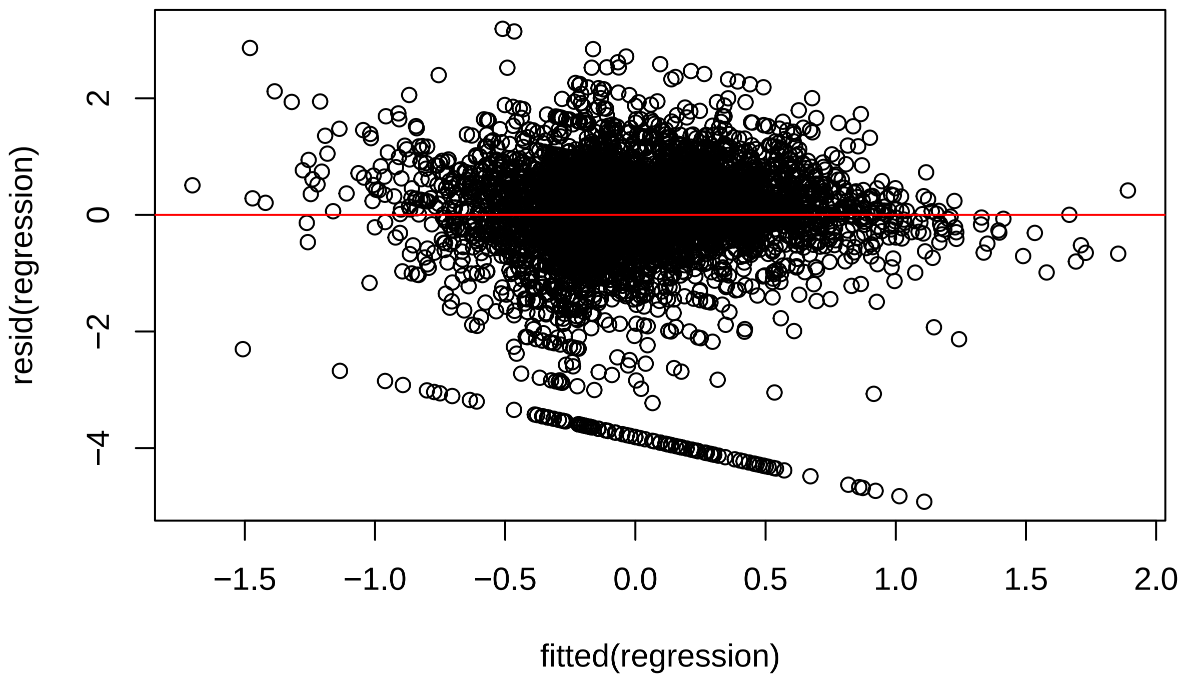
```
set.seed(abs(636-555-3226))
ind <- sample(1:nrow(games),5000,FALSE)
sub_data <- games[ind,]

sub_data <- as.data.frame(sub_data)
regression <- lm(sub_data$average_rating~., data = sub_data)
summary(regression)
```

```
##
## Call:
## lm(formula = sub_data$average_rating ~ ., data = sub_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.9202 -0.3275  0.0989  0.4873  3.1911
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.008392   0.012916   0.650  0.51587
## yearpublished   0.199808   0.013454  14.851  < 2e-16 ***
## minplayers     -0.010653   0.013532  -0.787  0.43119
## maxplayers     -0.006019   0.013517  -0.445  0.65614
## playingtime     0.077747   0.014689   5.293 1.26e-07 ***
## minage          0.041984   0.013271   3.164  0.00157 **
## total_owners    0.111174   0.013423   8.283  < 2e-16 ***
## average_weight  0.228837   0.014440  15.848  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9129 on 4992 degrees of freedom
## Multiple R-squared:  0.151,  Adjusted R-squared:  0.1498
## F-statistic: 126.9 on 7 and 4992 DF,  p-value: < 2.2e-16
```

The intercept is equal to 0.008392. It corresponds to the expected average rate of a board game when the other variables are equal to zero. And so it is not interpretable in this problem because we are talking about the average rating of a board game that does not exist. We can use the R^2 to assess the goodness of fit of our linear regression model, that is equal to 0.151. It means that only 15,1% of the the variation in *average_rating* is explained by the regressors.

Point g

```
plot(fitted(regression), resid(regression))
abline(0,0,col = "red")
```



Below we used the paired bootstrap method to provide bias, standard error and confidence intervals for regression coefficients, adjusted R^2 index and our $\hat{\theta}$. In particular, we chose this method and, for example, not the bootstrap of the errors because in the plot above it is possible to see that the residuals seem to not have constant variance.

```
set.seed(abs(636-555-3226))
perform_paired_bootstrap <- function(data, B) {
  num_coeffs <- length(coef(regression))
  bootstrap_estimates <- matrix(0, nrow = B, ncol = num_coeffs)

  for (i in 1:B) {
    bootstrap_indices <- sample(nrow(data), replace = TRUE)
    bootstrap_sample <- data[bootstrap_indices, ]
    bootstrap_regression <- lm(average_rating ~ ., data = bootstrap_sample)
    bootstrap_estimates[i, ] <- coef(bootstrap_regression)
  }

  coefficient_bias <- apply(bootstrap_estimates, 2, mean) - coef(regression)
  coefficient_std_error <- apply(bootstrap_estimates, 2, sd)
  coefficient_confidence_intervals <- t(apply(bootstrap_estimates, 2, function(x) quantile(x, c(0.025, 0.975))))

  adjusted_r_squared <- numeric(B)
  for (i in 1:B) {
    bootstrap_indices <- sample(nrow(data), replace = TRUE)
    bootstrap_sample <- data[bootstrap_indices, ]
    bootstrap_regression <- lm(average_rating ~ ., data = bootstrap_sample)
    adjusted_r_squared[i] <- 1 - (1 - summary(bootstrap_regression)$r.squared) * ((nrow(bootstrap_sample) - 1) / (n
  }
  adjusted_r_squared_bias <- mean(adjusted_r_squared) - summary(regression)$adj.r.squared
  adjusted_r_squared_std_error <- sd(adjusted_r_squared)
  adjusted_r_squared_confidence_interval <- quantile(adjusted_r_squared, c(0.025, 0.975))

  theta_estimates <- numeric(B)
  coeff <- coef(regression)

  for (i in 1:B) {
    boot_sample <- data[sample(nrow(data), replace = TRUE), ]
    regression_boot <- lm(average_rating ~ ., data = boot_sample)
    coeff_boot <- coef(regression_boot)
    theta_boot <- max(((coeff_boot[5] - coeff_boot[6])/(coeff_boot[3]+coeff_boot[2])), 0)
    theta_estimates[i] <- theta_boot
  }

  bias <- mean(theta_estimates) - ((coeff[5] - coeff[6])/(coeff[3]+coeff[2]))
  se <- sd(theta_estimates)
  ci <- quantile(theta_estimates, probs = c(0.025, 0.975))

  results <- list(
    coefficient_bias = coefficient_bias,
    coefficient_std_error = coefficient_std_error,
    coefficient_confidence_intervals = coefficient_confidence_intervals,
    adjusted_r_squared_bias = adjusted_r_squared_bias,
    adjusted_r_squared_std_error = adjusted_r_squared_std_error,
    adjusted_r_squared_confidence_interval = adjusted_r_squared_confidence_interval,
    theta_bias = bias,
    theta_std_error = se,
    theta_confidence_interval = ci
  )

  return(results)
}

results <- perform_paired_bootstrap(sub_data, B)
results
```

```
## $coefficient_bias
##    (Intercept)  yearpublished     minplayers     maxplayers    playingtime
##   2.590343e-04  -5.313236e-04   5.391762e-05  -3.615397e-04  -2.715688e-04
##         minage   total_owners average_weight
##  -9.566880e-05   4.931990e-04  -6.421485e-05
##
## $coefficient_std_error
## [1] 0.012917019 0.014882237 0.014706065 0.009575085 0.013402492 0.013637539
## [7] 0.007677498 0.017932378
##
## $coefficient_confidence_intervals
##              2.5%      97.5%
## [1,] -0.01545676 0.03319964
## [2,]  0.17137553 0.22927323
## [3,] -0.03924184 0.01808103
## [4,] -0.02516037 0.01208000
## [5,]  0.05100863 0.10383718
## [6,]  0.01570397 0.06778159
## [7,]  0.09695728 0.12696910
## [8,]  0.19381726 0.26420035
##
## $adjusted_r_squared_bias
## [1] 0.0005191378
```

```
##
## $adjusted_r_squared_std_error
## [1] 0.01122473
##
## $adjusted_r_squared_confidence_interval
##      2.5%     97.5%
## 0.1292476 0.1717044
##
## $theta_bias
## playingtime
## 0.008709073
##
## $theta_std_error
## [1] 0.1101246
##
## $theta_confidence_interval
##      2.5%     97.5%
## 0.0000000 0.4221232
```

Point h

```
set.seed(abs(636-555-3226))
perform_jackknife <- function(data) {
  num_coeffs <- length(coef(regression))
  num_samples <- nrow(data)
  jackknife_estimates <- matrix(0, nrow = num_samples, ncol = num_coeffs)

  for (i in 1:num_samples) {
    jackknife_sample <- data[-i, ]
    jackknife_regression <- lm(average_rating ~ ., data = jackknife_sample)
    jackknife_estimates[i, ] <- coef(jackknife_regression)
  }

  coefficient_bias <- num_samples * (colMeans(jackknife_estimates) - coef(regression)) / (num_samples - 1)
  coefficient_std_error <- sqrt(num_samples * (colMeans(jackknife_estimates^2) - (colMeans(jackknife_estimates))^2)
  coefficient_confidence_intervals <- t(apply(jackknife_estimates, 2, function(x) quantile(x, c(0.025, 0.975))))

  adjusted_r_squared <- numeric(num_samples)
  for (i in 1:num_samples) {
    jackknife_sample <- data[-i, ]
    jackknife_regression <- lm(average_rating ~ ., data = jackknife_sample)
    adjusted_r_squared[i] <- 1 - (1 - summary(jackknife_regression)$r.squared) * ((nrow(jackknife_sample) - 1) / (n
  }
  adjusted_r_squared_bias <- num_samples * (mean(adjusted_r_squared) - summary(regression)$adj.r.squared) / (num_sa
  adjusted_r_squared_std_error <- sqrt(num_samples * (mean(adjusted_r_squared^2) - (mean(adjusted_r_squared))^2) /
  adjusted_r_squared_confidence_interval <- quantile(adjusted_r_squared, c(0.025, 0.975))

  theta_estimates <- numeric(num_samples)
  coeff <- coef(regression)

  for (i in 1:num_samples) {
    jackknife_sample <- data[-i, ]
    jackknife_regression <- lm(average_rating ~ ., data = jackknife_sample)
    coeff_boot <- coef(jackknife_regression)
    theta_estimates[i] <- max(((coeff_boot[5] - coeff_boot[6])/(coeff_boot[3] + coeff_boot[2])), 0)
  }

  theta_bias <- num_samples * (mean(theta_estimates) - (coeff[5] - coeff[6])/(coeff[3] + coeff[2])) / (num_samples
  theta_std_error <- sqrt(num_samples * (mean(theta_estimates^2) - (mean(theta_estimates))^2) / (num_samples - 1))
```

```r
  theta_confidence_interval <- quantile(theta_estimates, probs = c(0.025, 0.975))

  results <- list(
    coefficient_bias = coefficient_bias,
    coefficient_std_error = coefficient_std_error,
    coefficient_confidence_intervals = coefficient_confidence_intervals,
    adjusted_r_squared_bias = adjusted_r_squared_bias,
    adjusted_r_squared_std_error = adjusted_r_squared_std_error,
    adjusted_r_squared_confidence_interval = adjusted_r_squared_confidence_interval,
    theta_bias = theta_bias,
    theta_std_error = theta_std_error,
    theta_confidence_interval = theta_confidence_interval
  )

  return(results)
}
results_jackknife <- perform_jackknife(sub_data)
results_jackknife
```

```
## $coefficient_bias
##    (Intercept)  yearpublished     minplayers     maxplayers    playingtime
##  -1.045900e-08   3.160400e-08   7.927530e-09  -6.638226e-08   7.077104e-10
##         minage   total_owners average_weight
##  -2.011886e-08   7.649813e-08  -7.538168e-09
##
## $coefficient_std_error
## [1] 0.0001819372 0.0002094197 0.0001985008 0.0001395934 0.0001921217
## [6] 0.0001934172 0.0001108170 0.0002526914
##
## $coefficient_confidence_intervals
##              2.5%        97.5%
## [1,]  0.008092459  0.008923529
## [2,]  0.199486341  0.200284337
## [3,] -0.010942538 -0.010345240
## [4,] -0.006186247 -0.005870274
## [5,]  0.077324374  0.078128852
## [6,]  0.041632962  0.042396845
## [7,]  0.110976154  0.111320283
## [8,]  0.228385290  0.229235412
##
## $adjusted_r_squared_bias
## [1] -0.0001703989
##
## $adjusted_r_squared_std_error
## [1] 0.0001616389
##
## $adjusted_r_squared_confidence_interval
##      2.5%     97.5%
## 0.1494391 0.1499510
##
## $theta_bias
##  playingtime
## 1.791877e-07
##
## $theta_std_error
## [1] 0.00155067
##
## $theta_confidence_interval
##      2.5%     97.5%
## 0.1859348 0.1921601
```

The differences between the results of the paired bootstrap and the jackknife methods can be attributed to their different resampling techniques. The paired bootstrap randomly samples with replacement, while the jackknife systematically leaves out observations.They also have different statistical properties.The paired bootstrap provides an estimate of the distribution of the coefficients, while the jackknife estimates the variance of the coefficients. This leads to variations in the results.

EXERCISE 2- WE NEED SOME MUSIC!

Point a

Considering the data saved in spot.RDS, a description of the variables can be found on Kaggle:

- Acousticness: a confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.

- Danceability: danceability describes how suitable a track is for dancing. A value of 0.0 is least danceable and 1.0 is most danceable.

- Duration_ms: the track length in milliseconds.

- Energy:energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity.

- Explicit: whether or not the track has explicit lyrics (true = yes it does; false = no it does not OR unknown)

- id: the Spotify ID for the track.

- Instrumentalness: predicts whether a track contains no vocals. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content.

- Liveness: detects the presence of an audience in the recording. A value above 0.8 provides strong likelihood that the track is live.

- Loudness: the overall loudness of a track in decibels (dB).

- Name:name of the track.

- Popularity: the popularity of a track is a value between 0 and 100, with 100 being the most popular. Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past.

- Release_date of the song

- Speechiness: speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value.

- Tempo: the overall estimated tempo of a track in beats per minute (BPM).

- Valence: a measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track.

- First_auth: first author of the song.

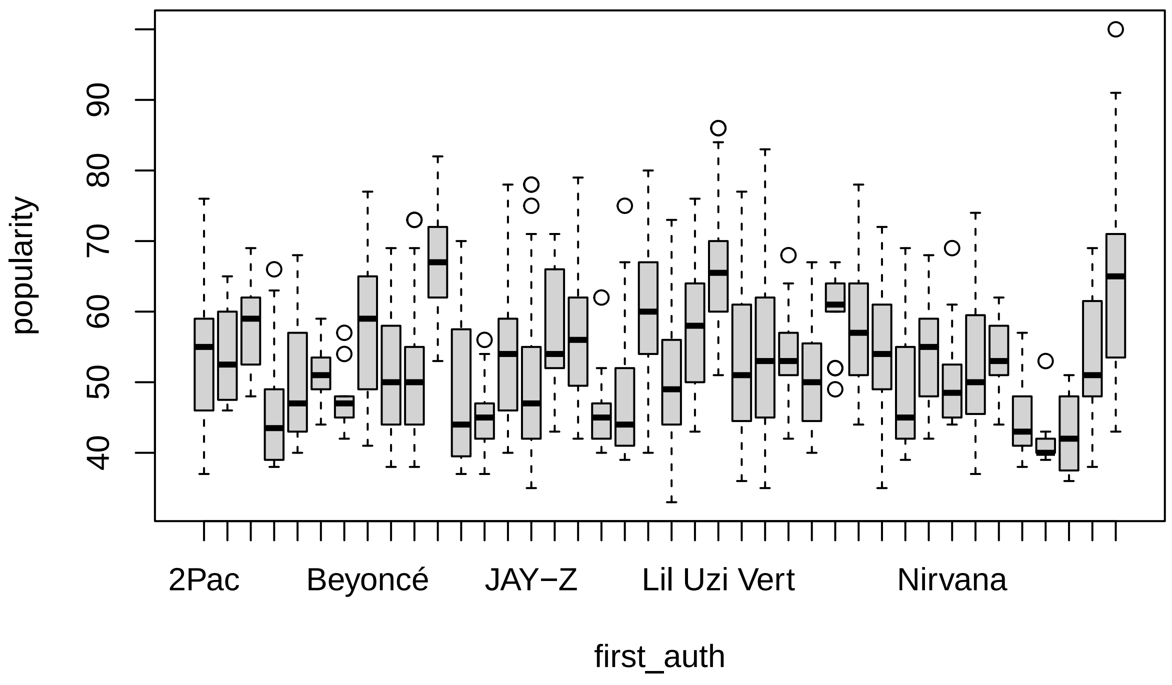
- n: number of song for each artist.

- pop: variable containing the original column popularity rescaled between 0 and 1.

```r
songs <- readRDS("spot (1).RDS")
head(songs)
```

```
## # A tibble: 6 x 18
##   acousticness danceability duration_ms energy explicit id      instrumentalness
##          <dbl>        <dbl>       <int>  <dbl>    <int> <chr>              <dbl>
## 1       0.116         0.908      203520  0.573        1 3iNzFu~         0.000547
## 2       0.299         0.703      247600  0.639        1 3eBBbA~         0
## 3       0.283          0.874      230227  0.64         1 1SgENh~       0.000002
## 4       0.0672         0.748      230493  0.614        1 36UjTT~       0.0000781
## 5       0.0634         0.536      267813  0.836        1 01qKrm~       0
## 6       0.114          0.824      286893  0.847        1 06jYyA~       0.00000284
## # i 11 more variables: liveness <dbl>, loudness <dbl>, name <chr>,
## #   popularity <int>, release_date <chr>, speechiness <dbl>, tempo <dbl>,
## #   valence <dbl>, first_auth <chr>, n <int>, pop <dbl>
```

```
boxplot(popularity~first_auth,data=songs)
```



Point b

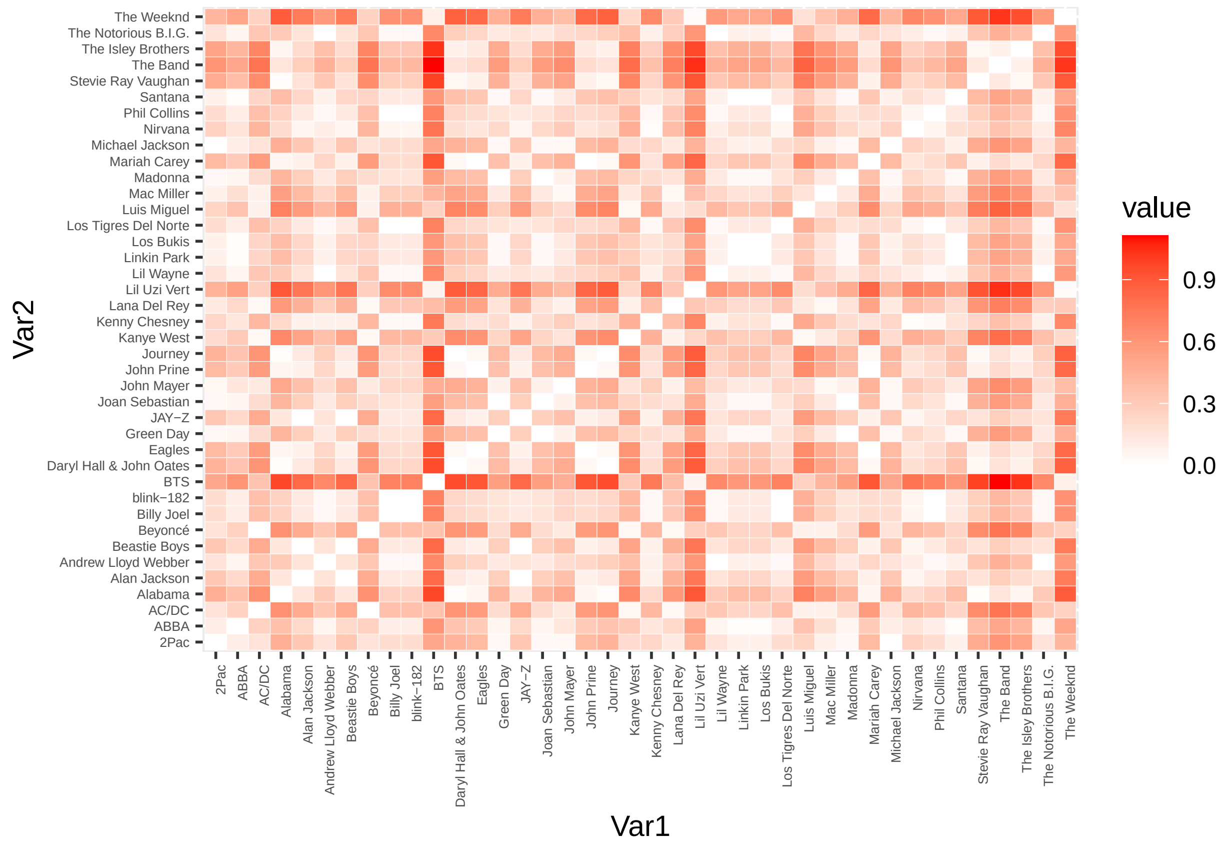
Focusing on “pop” and given two artists, we want to measure their different in popularity. Let X and Y denote the vectors of popularity measurements of the songs of two given artists. We measure their difference with:

$$T(X, Y) = |\text{median}(\text{logit}(X)) - \text{median}(\text{logit}(Y))|$$

After implementing this estimator to compare the popularity of AC/DC and Kanye West, we provide a matrix with all pairwise comparisons.

```
u_art<- unique(songs$first_auth)


X <- songs$pop[songs$first_auth == "AC/DC"]

# Extract popularity measurements for Artist Y
Y <- songs$pop[songs$first_auth == "Kanye West"]
```

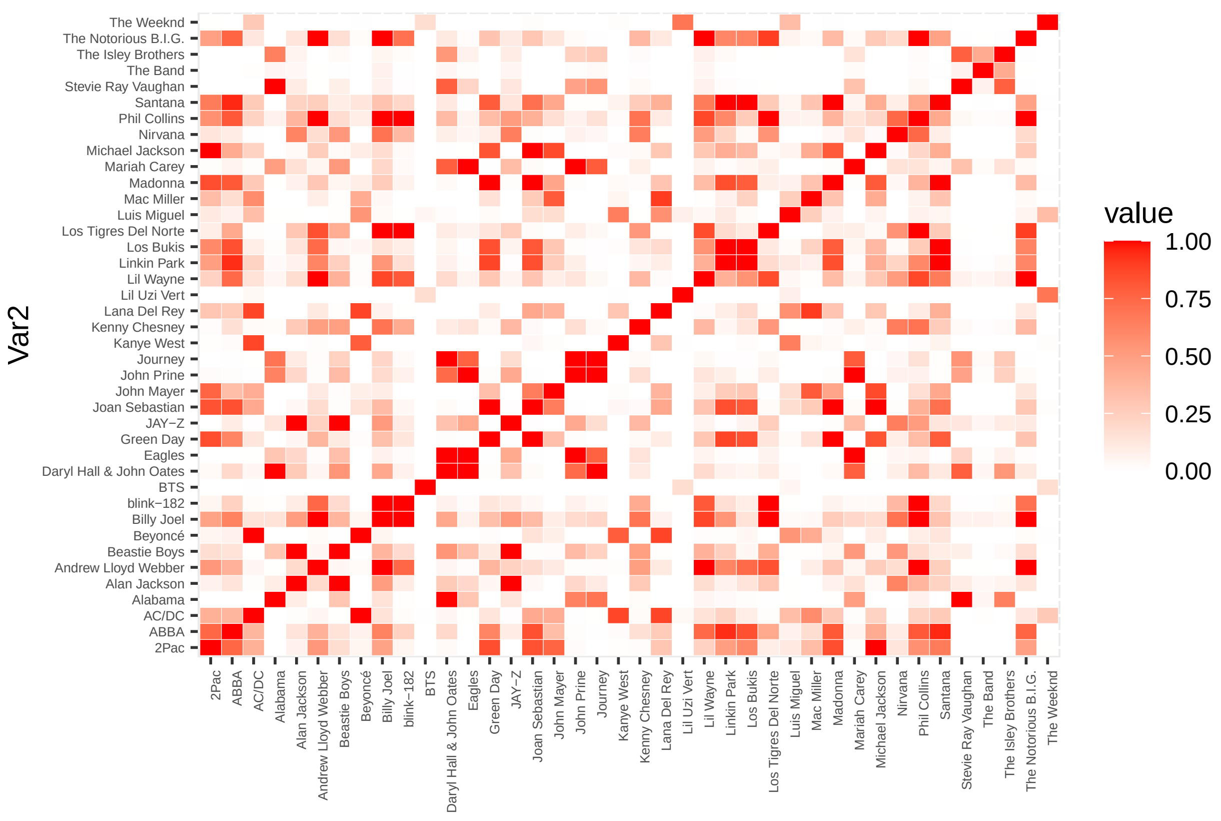
```r
med_logit_X <-median( logit(X))
med_logit_Y <-median( logit(Y))

T_XY <- abs(med_logit_X-med_logit_Y)

pop_diff_T <- function(){
  res<-sapply(1:40, function(x) {
    sapply(1:40, function(y) {
      abs(median(logit(songs$pop[songs$first_auth == u_art[x]]))-median(logit(songs$pop[songs$first_auth ==  u_art[
    })
  })
  return(res)
}

pop_diff<- pop_diff_T()

rownames(pop_diff) <- u_art
colnames(pop_diff) <- u_art

ggplot(melt(pop_diff),aes(Var1,Var2))+
  geom_tile(aes(fill = value), colour = "white")  +
  scale_fill_gradient(low = "white", high = "red")+
  theme(axis.text.x = element_text(angle=90,hjust = 1,siz=5),
        axis.text.y = element_text(hjust = 1,siz=5))
```



Point c

Now we perform a hypothesis testing to assess the absence of difference between the popularity of two artists measured via T(X,Y) and perform a sequence of pairwise permutation tests reporting the estimated p-values. The null hypothesis states that samples come from the same distribution i.e. the two playlist are similar. The alternative hypothesis stated that the samples are different, i.e. that the two playlist are not similar.

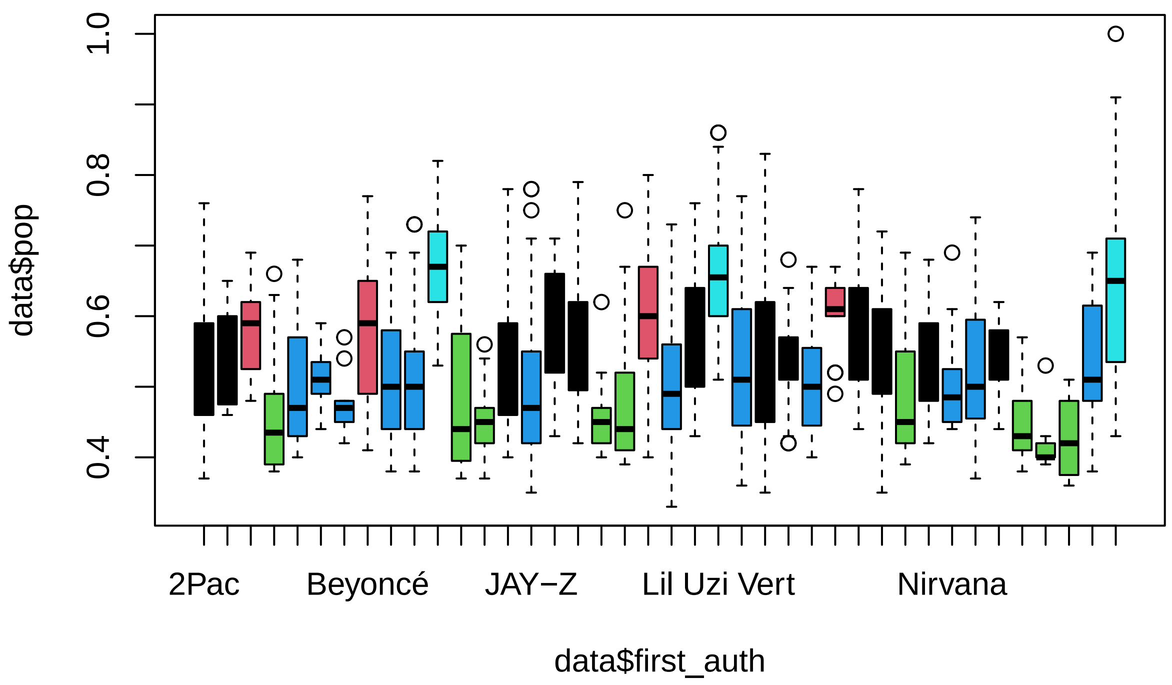
```r
data<-songs
p_values <- matrix(0,nrow = length(u_art),ncol = length(u_art))
for (i in 1:length(u_art)){
  for(j in 1:length(u_art)){
   if(i>=j){

      X <- data[data$first_auth ==  u_art[i],"pop"]
      Y <- data[data$first_auth ==  u_art[j],"pop"]
      observed_stat=abs(median(logit(songs$pop[songs$first_auth == u_art[i]]))-median(logit(songs$pop[songs$first_a



      num_perm<- 1000
      permutation_stat<- numeric(num_perm)
      for (k in 1:num_perm){
        combined<- union_all(X,Y)
        combined$pop <- sample(combined$pop,nrow(combined),replace = F)
        perm_X<- combined$pop[1:length(X$pop)]
        perm_Y<-combined$pop[(length(X$pop)+1):(length(X$pop)+length(Y$pop))]
        permutation_stat[k]<- abs(median(logit(perm_X))-median(logit(perm_Y)))

      }

      p_value <- mean(permutation_stat>=observed_stat)

      p_values[i,j]<- p_value
      p_values[j,i]<- p_value
    }
  }
}

rownames(p_values) <- u_art
colnames(p_values) <- u_art

ggplot(melt(p_values),aes(Var1,Var2))+
  geom_tile(aes(fill = value), colour = "white")  +
  scale_fill_gradient(low = "white", high = "red")+
  theme(axis.text.x = element_text(angle=90,hjust = 1,siz=5),
        axis.text.y = element_text(hjust = 1,siz=5))
```



Point d

The cutree() cut a hierarchical clustering tree into a 5 clusters. The clustering is performed on a distance matrix derived from the p-value matrix, pval_mat, by subtracting each value from 1 and converting it to a distance using as.dist(). The resulting clusters are used to assign colors to the boxes in the boxplot.

```
boxplot( data$pop ~ data$first_auth,
         col= cutree( hclust(as.dist(1-p_values)),5))
```



Point e

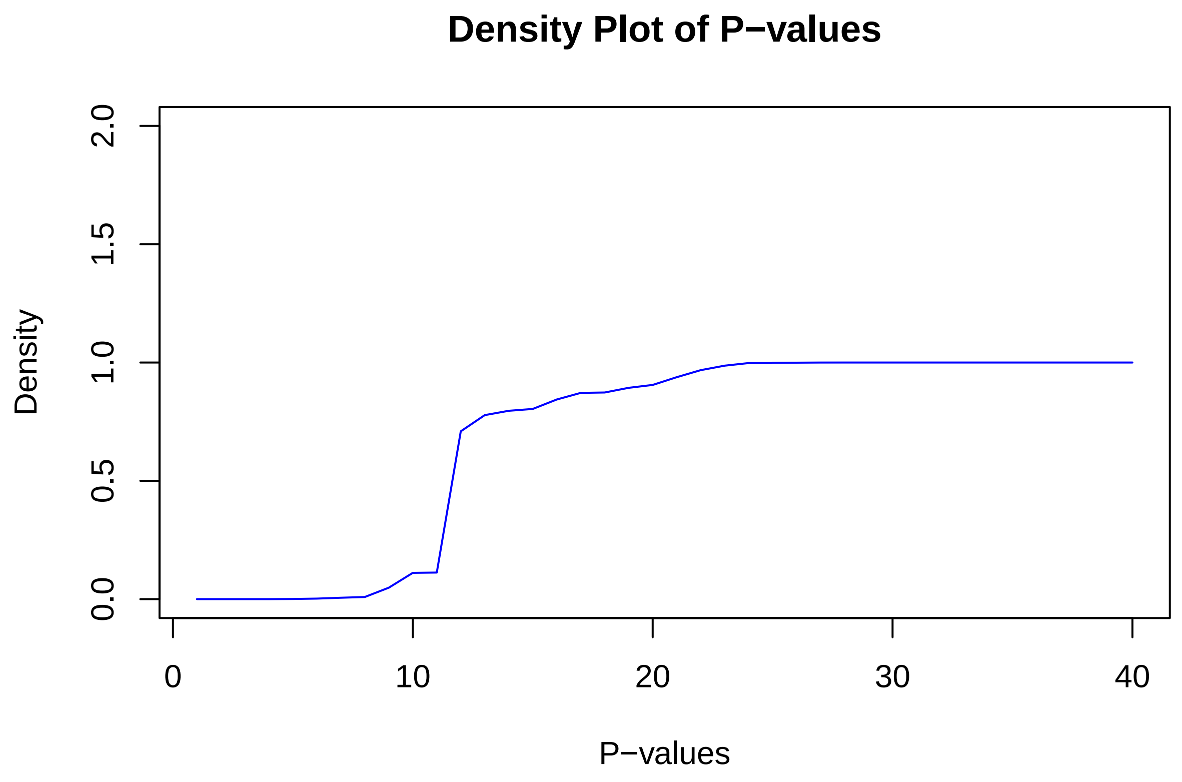
Now, we apply the scale() function to "pop" and perform, for each artist, a single-sample t-test to assess: $H_0 : \mu = 0$ VS $H_1 : \mu > 0$

```
pop_scaled <- scale(data$pop)
df<- data.frame(Artist = data$first_auth,pop_scaled)

t_test<- function(df,artist){
  x<- df[df$Artist==artist,"pop_scaled"]
  t_test<- t.test(x,mu=0,alternative = "greater")
  return(t_test$p.value)
}

p_values<- sapply(unique(df$Artist),t_test,df=df)

prob_violation <- sum(p_values<0.10)/length(p_values)
prob_violation #probability of the global null to be violated
```

```
## [1] 0.225
```

The plot below shows that the lies in the two extremes of probability i.e. 0 and 1. The ones with smaller values indicates a higher popularity index than the others.

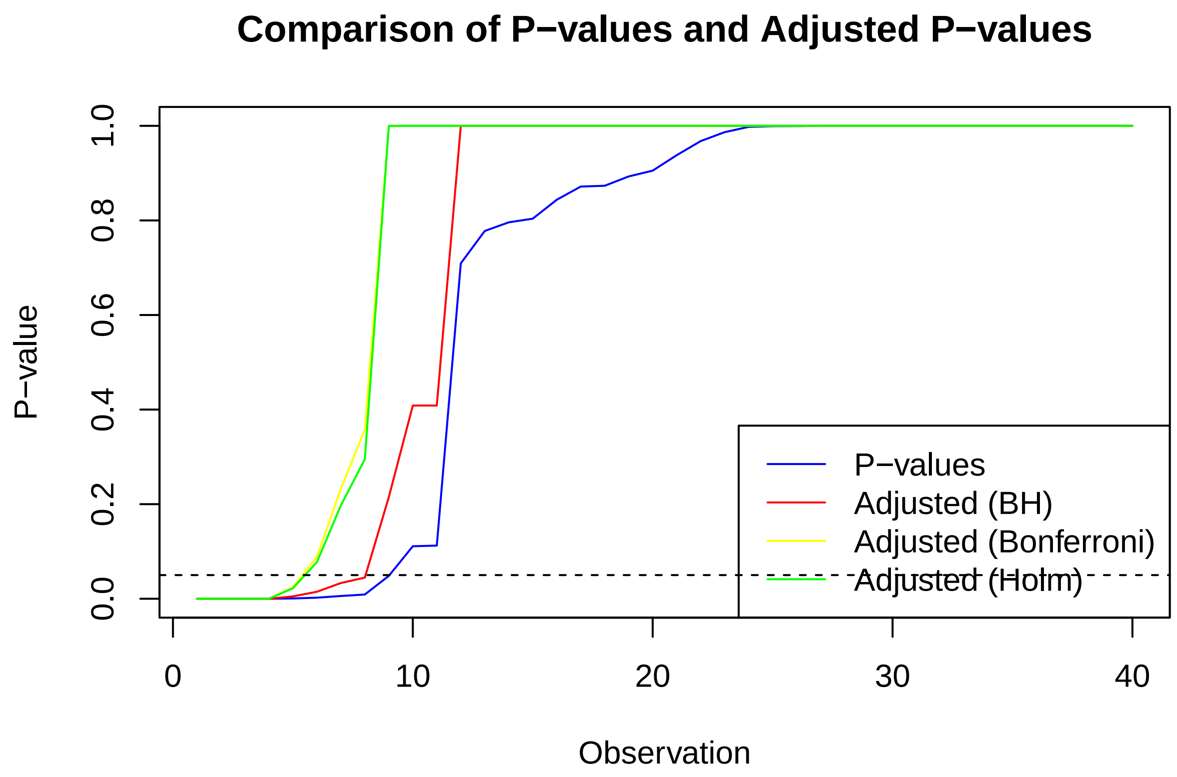
```
plot(sort(p_values), type="l",col = "blue", main = "Density Plot of P-values", xlab = "P-values", ylab = "Density",
```



Point f

The p.adjust function is used to adjust p-values for multiple comparisons. When conducting multiple statistical tests or comparing multiple groups, the probability of obtaining a significant result by chance increases. Multiple comparison procedures, such as adjusting p-values, are used to control for this increased risk of false positives (Type I errors).The p.adjust function takes a vector of p-values as input and applies various methods to adjust these p-values: Bonferroni adjustment,Benjamini-Hochberg and Holm. In summary, the p.adjust function helps in minimizing Type I errors by applying appropriate adjustments to p-values, depending on the chosen method.

```
adjusted_bh <- p.adjust(p_values, method = "BH")
adjusted_bonferroni <- p.adjust(p_values, method = "bonferroni")
adjusted_holm <- p.adjust(p_values, method = "holm")
results <- data.frame(p_values, adjusted_bh, adjusted_bonferroni,adjusted_holm)

# Plotting the p-values and adjusted p-values
plot(sort(p_values), type="l", col = "blue", xlab = "Observation", ylab = "P-value",
     main = "Comparison of P-values and Adjusted P-values")
lines(sort(adjusted_bh), col = "red")
lines(sort(adjusted_bonferroni),  col = "yellow")
lines(sort(adjusted_holm), col = "green")
abline(h = 0.05, lwd = 1, lty = 2)
legend("bottomright", legend = c("P-values", "Adjusted (BH)", "Adjusted (Bonferroni)","Adjusted (Holm)"),
       col = c("blue", "red","yellow","green"), lwd=1,)
```



Point g When the n hypotheses are tested seperately using tests with the level $\frac{\alpha}{n}$ it follows immediately from the Boole inequality that the probability of rejecting any true hypothesis is smaller then or equal to alpha. This constitutes a multiple test procedure with the multiple level of significance alpha for free combination, the classical Bonferroni multiple test procedure. As written in the paper "A simple sequentially rejective multiple test procedure" of Sture Holm, in the sequentially rejective Bonferroni test the obtained levels are compared to the numbers $\frac{\alpha}{n}, \frac{\alpha}{n-1}, \ldots, \frac{\alpha}{1}$, whereas in the classical Bonferroni test they are compared to $\frac{\alpha}{n}$. This means that the probability of rejecting any set of (false) hypotheses using the *classical* Bonferroni test is *smaller* than or equal to the same probability using the sequantially rejective Bonferroni test based on the same test statistics. So the classical Bonferroni test can be replaced by the corresponding sequentially rejective Bonferroni test without loosing any probability of rejecting false hypotheses.The power gain obtained by using a sequentially rejective Bonferroni test dependes very much upon the alternative.The power of a hypothesis test is the probability of rejecting the null hypothesis when the alternative hypothesis is the hypothesis that is true.As a consequence, the power gain is small if all the hypothese are "almost true".It may be considerable if a number of hypotheses are "completely wrong". So, it is clear that the treshold of Holm's procedure is equal or less than the thresholds of the Bonferroni correction: $\frac{\alpha}{n-j+1} \leq \frac{\alpha}{n}$. Since the power of the test and α are inversely correlated, we can conclude that the power of the Holm's procedure is greater than the power of the Bonferroni correction.