

Assignment-2

Manuel Bottino

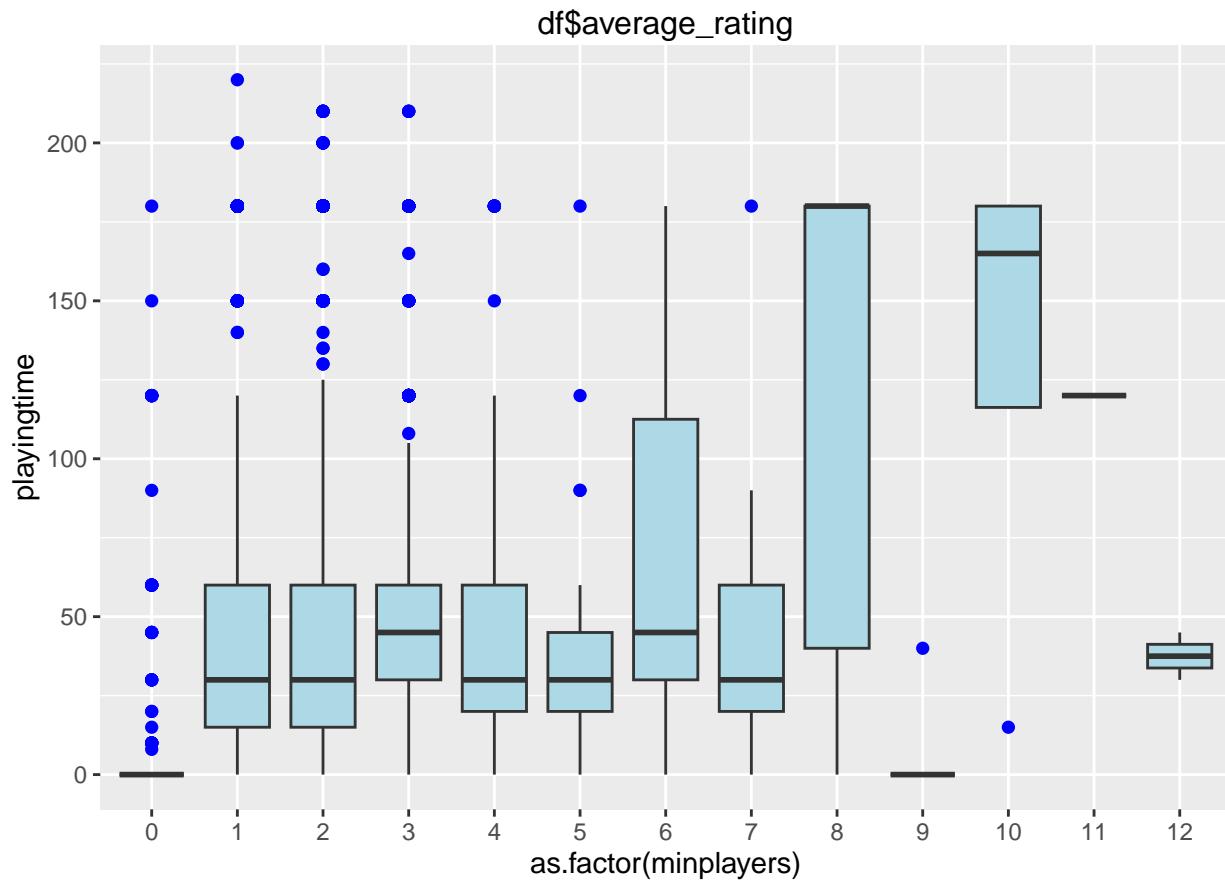
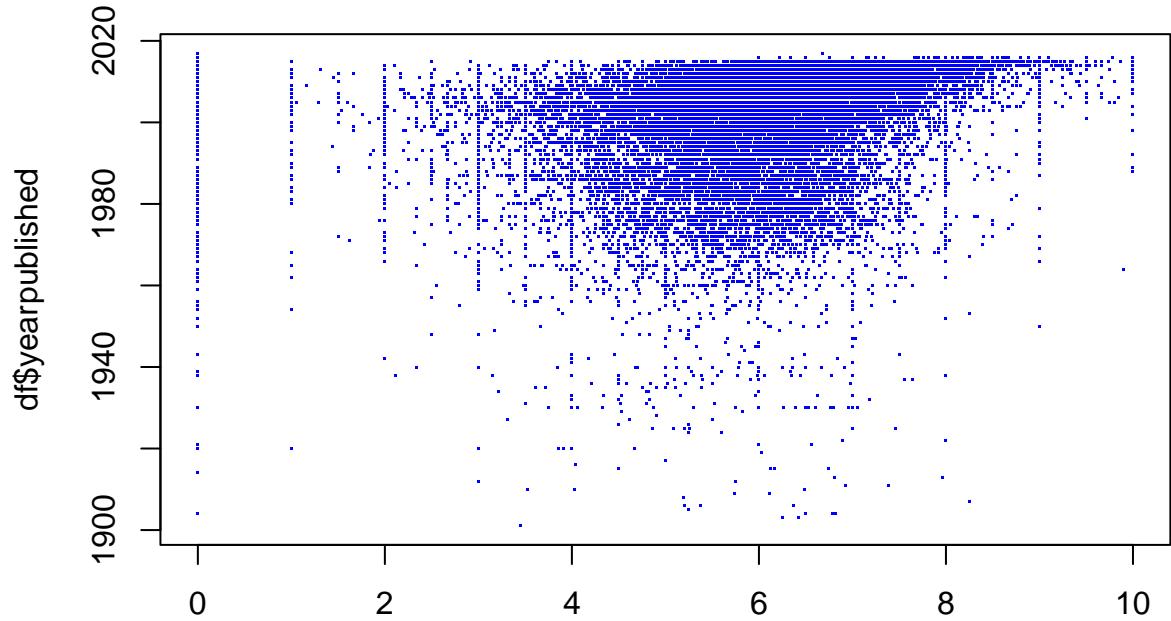
2023-06-02

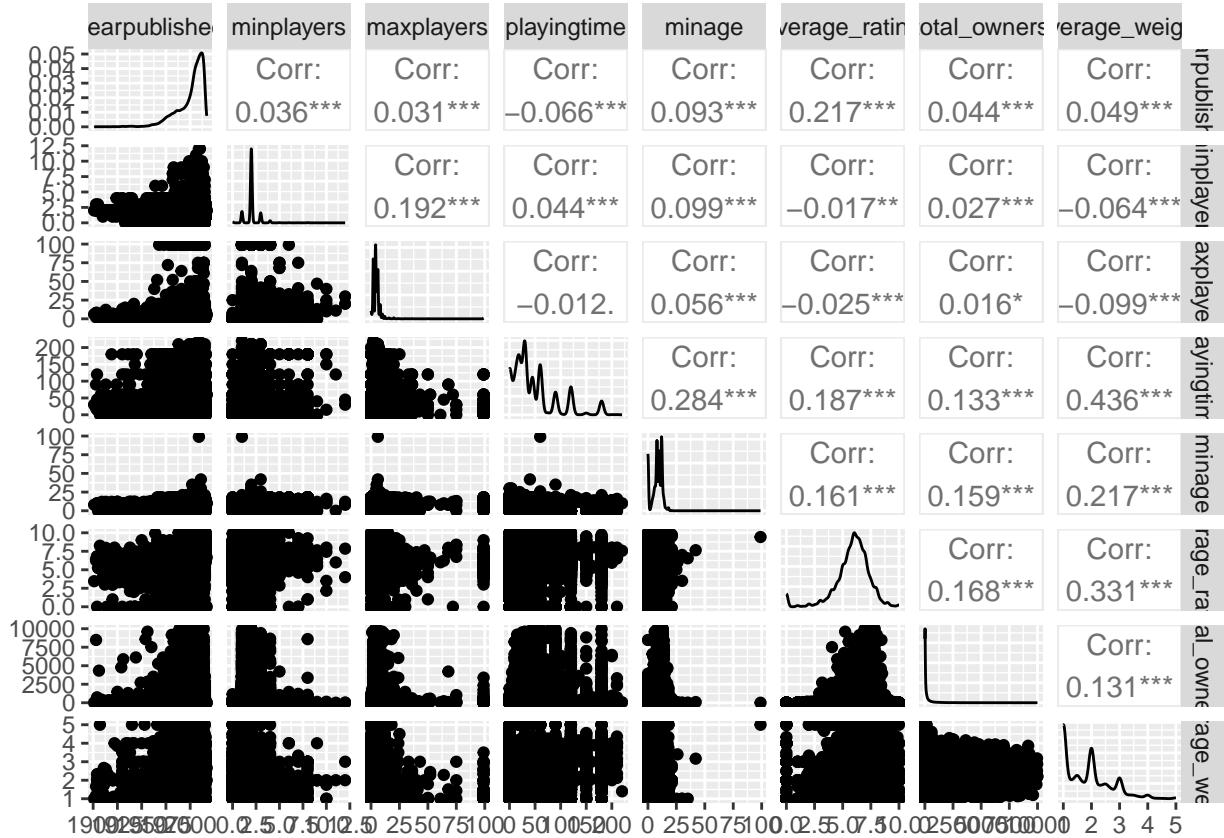
Contents

1 Excercise 1 - Risiko! is back, with friends	2
1.1 Point a)	2
1.2 Point b)	3
1.3 Point c)	4
1.4 Point d)	6
1.5 Point e)	6
1.6 Point f)	7
1.7 Point g)	7
1.8 Point h)	12
2 Exercise 2 - We need some music!	14
2.1 Point a)	14
2.2 Point b	17
2.3 Point c)	19
2.4 Point d)	20
2.5 Point e)	21
2.6 Point f)	22
2.7 Point g)	22

1 Excercise 1 - Risiko! is back, with friends

1.1 Point a)





Some comment: - There is not much correlation between the variables, we can have high confidence on this (they are all significant)

1.2 Point b)

```
df = df[,-1:-2]
df = data.frame(scale(df))
```

```
sample_cov=cov(df)
a = cor(df)
# We can get the same result using cov2cor
kable(cov2cor(sample_cov))
```

	yearpublished	minplayers	maxplayers	playingtime	minage	average_rating	total_owners
yearpublished	1.0000000	0.0358563	0.0311254	-0.0657260	0.0931492	0.2165733	0.0444172
minplayers	0.0358563	1.0000000	0.1916686	0.0442961	0.0985169	-0.0170554	0.0272407
maxplayers	0.0311254	0.1916686	1.0000000	-0.0120495	0.0562662	-0.0250944	0.0157420
playingtime	-0.0657260	0.0442961	-0.0120495	1.0000000	0.2840740	0.1869474	0.1328122
minage	0.0931492	0.0985169	0.0562662	0.2840740	1.0000000	0.1609671	0.1592991
average_rating	0.2165733	-0.0170554	-0.0250944	0.1869474	0.1609671	1.0000000	0.1681943
total_owners	0.0444172	0.0272407	0.0157420	0.1328122	0.1592991	0.1681943	1.0000000
average_weight	0.0489981	-0.0635317	-0.0991836	0.4364520	0.2168594	0.3309004	0.1310639

Since we have scaled all the columns with variance equal to 1, it implies that the covariance-variance matrix is equal to the correlation matrix. Fonte: <https://math.stackexchange.com/questions/3780344/under-what-conditions-will-the-covariance-matrix-be-identical-to-the-correlation>

1.3 Point c)

```
B=100

cov_var_sample=array(NA, dim=c(ncol(df),ncol(df),B))

for(i in 1:B){
  cov_var_sample[,,i]=cov(slice_sample(df,n=nrow(df), replace=TRUE))
}

theta_hat_bar=apply(cov_var_sample, c(1, 2), mean)
se_cov_var=apply(cov_var_sample, c(1, 2), sd)
```

Confidence intervals 1. A first method consist on using the fact that, as the sample size n grows large
 $\frac{\hat{\theta} - \theta}{SE_B(\hat{\theta})} \approx N(0, 1)$

```
lower_bound=sample_cov+qnorm(c(0.025), mean=0, sd=1)*se_cov_var
upper_bound=sample_cov+qnorm(c(0.975), mean=0, sd=1)*se_cov_var
```

2. Use the quantile of the bootstrap sampling distribution, this method is considered more robust than the one before

```
boot_sample_variance <- data.frame(matrix(nrow = 8, ncol = 8))
colnames(boot_sample_variance) <- c("yearpublished", "minplayers", "maxplayers",
                                    "playingtime", "minage", "average_rating", "total_owners",
                                    "average_weight")

for(i in 1:8){
  for(j in 1:8){
    boot_sample_variance[i,j]=list(list(cov_var_sample[i,j,]))
  }
}
boot_sample_variance_ci <- data.frame(matrix(nrow = 8, ncol = 8))
colnames(boot_sample_variance_ci) <- c("yearpublished", "minplayers", "maxplayers",
                                       "playingtime", "minage", "average_rating", "total_owners", "averag
```

```
for(i in 1:8){
  for(j in 1:8){
    boot_sample_variance_ci[i,j]=list(list(quantile(boot_sample_variance[i,j][[1]], probs=c(0.025,0.975)))
  }
}
```

3. Using as assumption the fact that the behavior of the bias $\theta - \hat{\theta}$ is the same as $\hat{\theta} - \hat{\theta}^*$, we can compute the confidence interval of the bias-corrected estimate

```
boot_sample_variance_ci_bias <- data.frame(matrix(nrow = 6, ncol = 6))
colnames(boot_sample_variance_ci) <- c("yearpublished", "minplayers", "maxplayers", "playingtime", "minage", "average_rating", "total_owners", "average_weight")

for(i in 1:6){
  for(j in 1:6){
    boot_sample_variance_ci_bias[i,j]=list(list(2*sample_cov[i,j]-quantile(boot_sample_variance[i,j][[1]], probs=c(0.025,0.975))))
  }
}
```

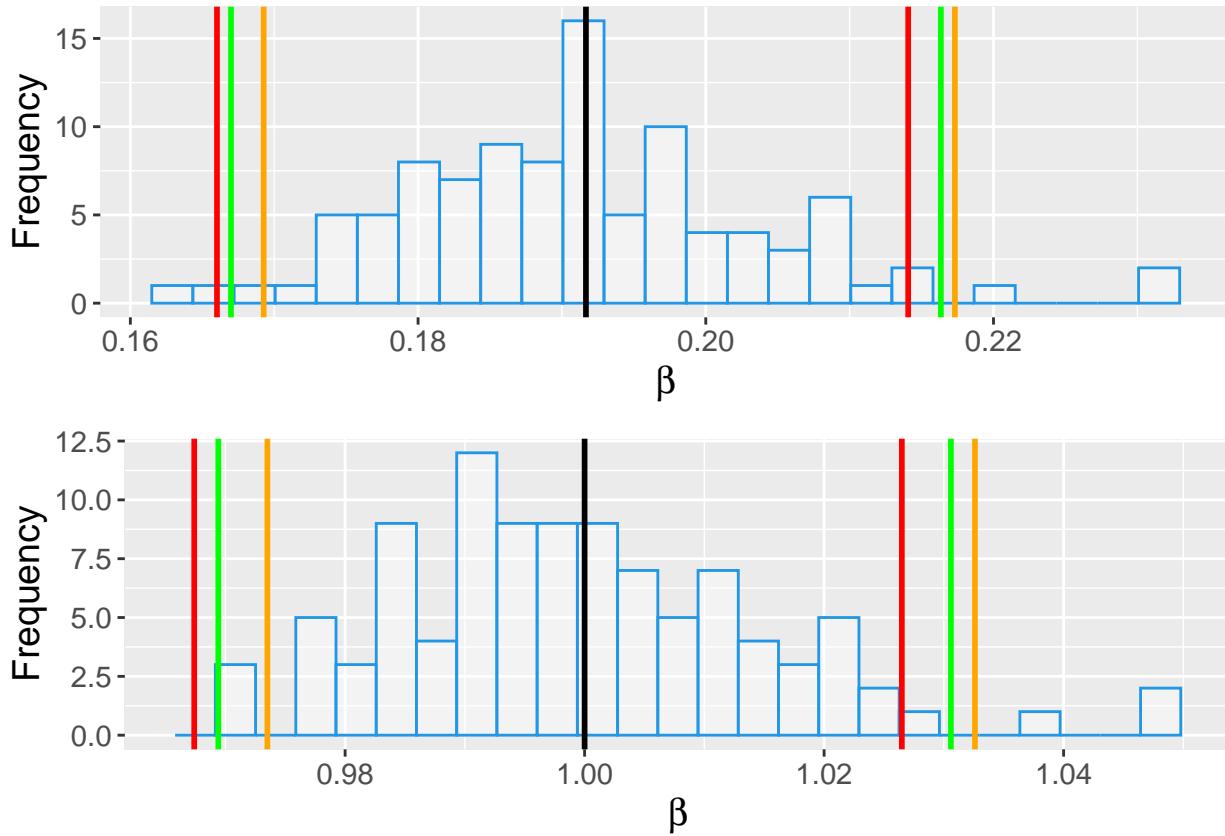
```

    }
}

plot_boot_sample_ci = function(row, column){
  ggplot()+
    geom_histogram(aes(x=boot_sample_variance[[row, column]]), alpha=.4, col=4, fill="white", bins=25) +
    geom_vline(aes(xintercept = sample_cov[[row, column]]), col=1, lwd = 1) +
    geom_vline(aes(xintercept = boot_sample_variance_ci[[row, column]][1]), col="orange", lwd = 1) +
    geom_vline(aes(xintercept = boot_sample_variance_ci[[row, column]][2]), col="orange", lwd = 1) +
    geom_vline(aes(xintercept = boot_sample_variance_ci_bias[[row, column]][1]), col="red", lwd = 1) +
    geom_vline(aes(xintercept = boot_sample_variance_ci_bias[[row, column]][2]), col="red", lwd = 1) +
    geom_vline(aes(xintercept = lower_bound[[row, column]]), col="green", lwd = 1) +
    geom_vline(aes(xintercept = upper_bound[[row, column]]), col="green", lwd = 1) +
    scale_linetype_manual( guide = guide_legend()) +
    xlab(TeX("$\\beta$")) +
    ylab("Frequency") +
    theme(text = element_text(size=14), legend.position = "none")
}

grid.arrange(plot_boot_sample_ci(2,3),plot_boot_sample_ci(1,1))

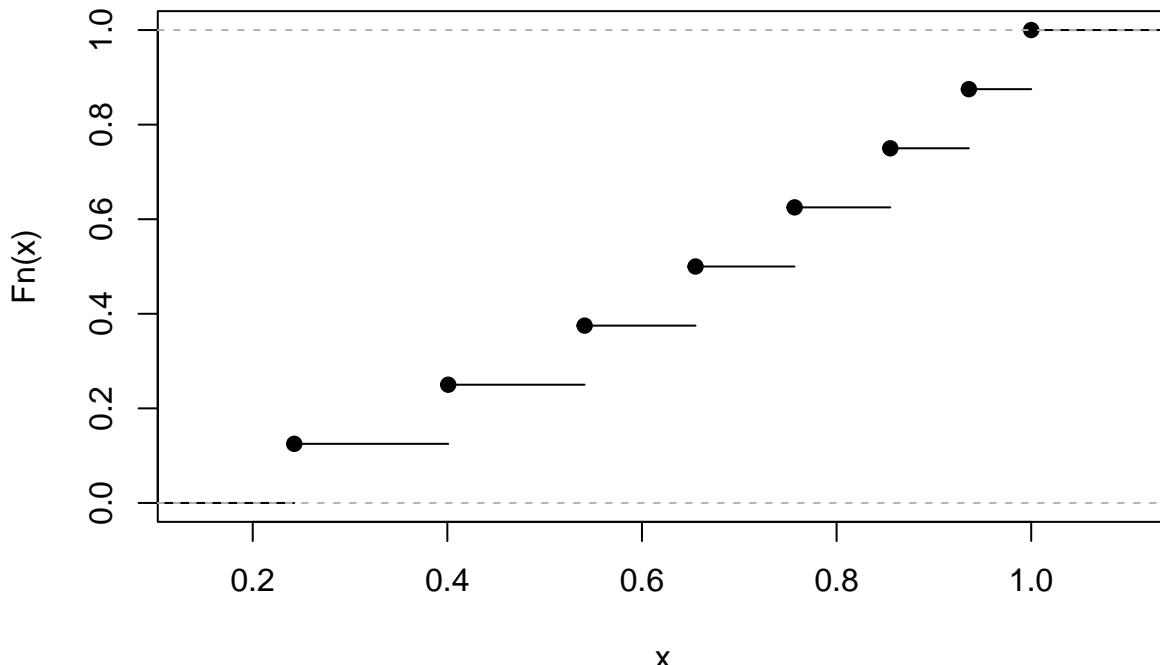
```



1.4 Point d)

```
#We know that these two values are equal  
sum(eigen(sample_cov)$values) & sum(diag(sample_cov))  
  
## [1] TRUE  
  
j_star=6  
theta_j_star=sum(eigen(sample_cov)$values[1:j_star])/sum(eigen(sample_cov)$values)  
  
theta=numeric(ncol(df))  
for( i in 1:length(theta)){  
  theta[i]=sum(eigen(sample_cov)$values[1:i])/sum(eigen(sample_cov)$values)  
}  
  
plot(ecdf(theta))
```

ecdf(theta)



1.5 Point e)

```
B=1000  
sort(sample(theta, replace=TRUE))  
  
## [1] 0.2426729 0.5410761 0.6550491 0.6550491 0.6550491 0.8551548 0.8551548  
## [8] 1.0000000
```

```

sample_theta=replicate(B, sample(theta, replace=TRUE))
sample_theta=apply(sample_theta, 2, sort)
theta_hat_boot=apply(sample_theta, 1, mean)
se_hat_boot=apply(sample_theta, 1, sd)

bias=theta-theta_hat_boot

t = data.frame(which(sample_theta>0.76, arr.ind=TRUE)) %>% # Computing the first column when we reach
# the condition
group_by(col) %>%
filter(row_number()==1)

cat("Bootstrap estimate:", theta_hat_boot[6], "\n Bootstrap standard error:", se_hat_boot[6] )

## Bootstrap estimate: 0.8258131
## Bootstrap standard error: 0.115321

```

1.6 Point f)

Make things more feasible

```

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

lm_game=lm(data=sub_data, average_rating ~ .)

res_sub_data=matrix(NA, 1, ncol(sub_data)+2)
res_sub_data[1,]=c(lm_game$coefficients, summary(lm_game)$r.squared,
                  max((lm_game$coef["playingtime"]-lm_game$coef["minage"])/(lm_game$coef["minplayers"]+lm_game$coef["maxplayers"])))

colnames(res_sub_data)=c(colnames(sub_data), "R2" , "theta_hat")
res_sub_data[, "yearpublished"]-1

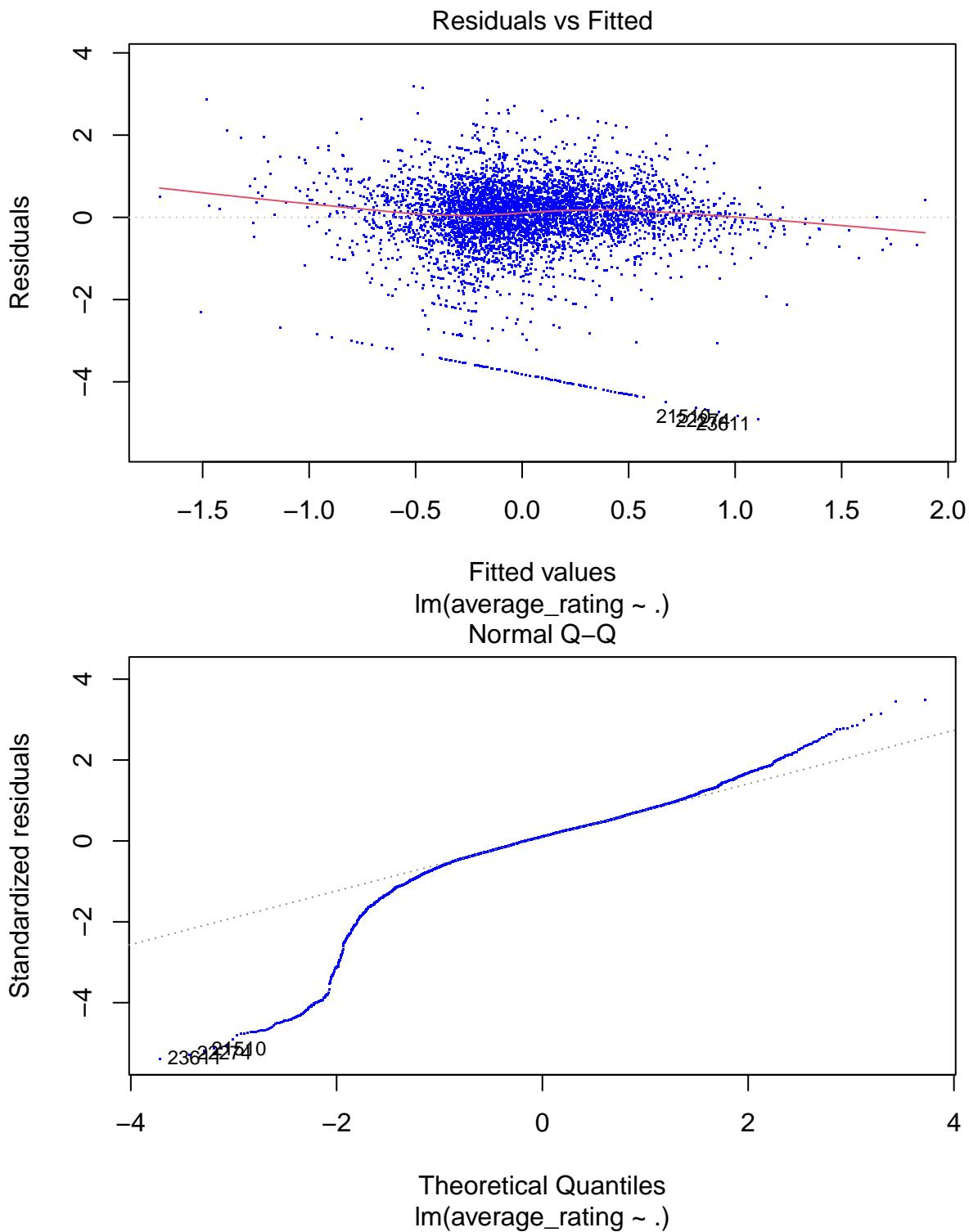
## yearpublished
##      -0.9916079

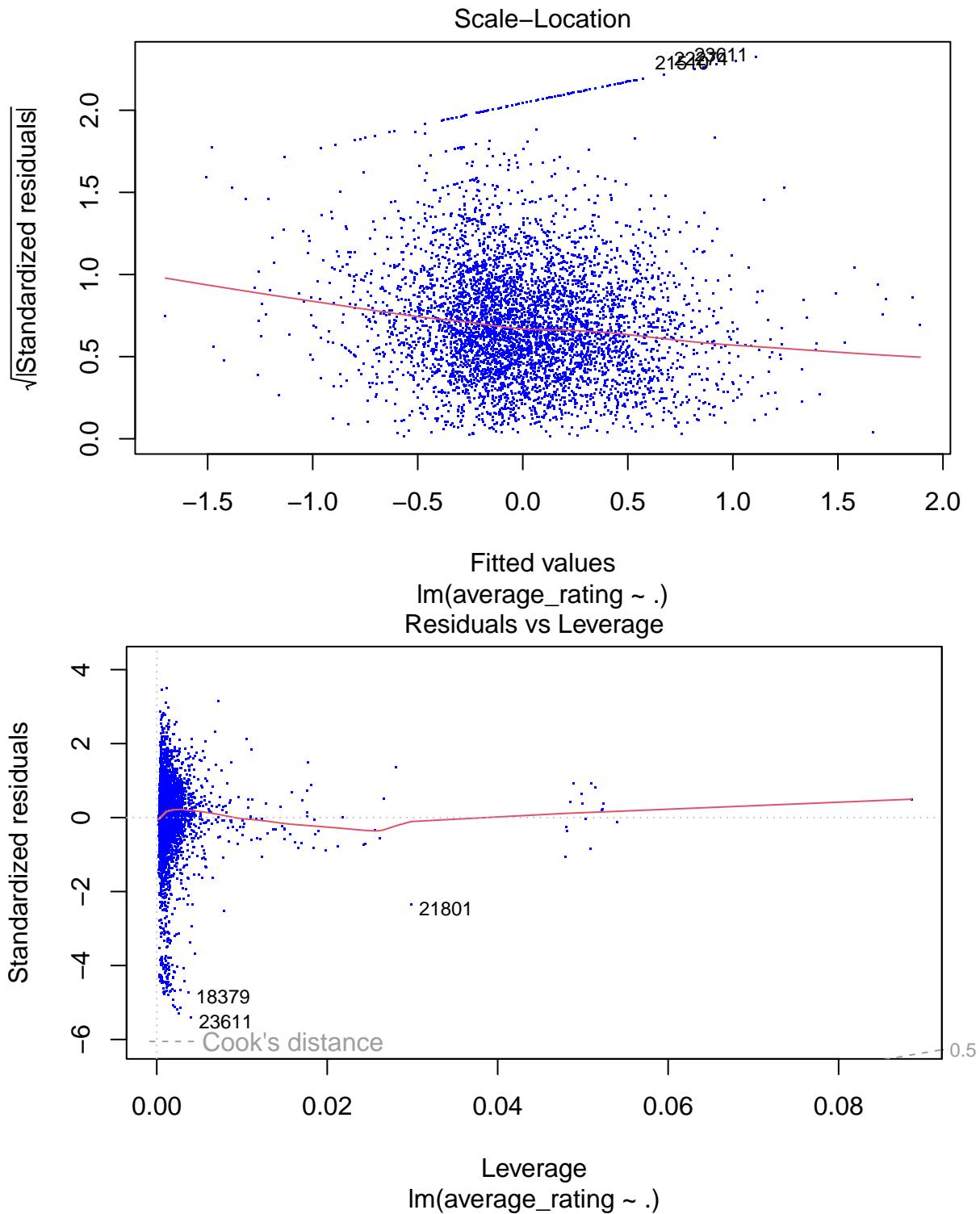
```

COMMENT

1.7 Point g)

```
plot(lm_game, col="blue", pch=".")
```





We can notice some features from this plots, useful for the choice of which bootstrap would be a sensitive choice. First of all the assumption of constant variance is not realistic in this case. Also the assumption of the errors distributed as a normal has problems, since we have a long tailed distribution. From this quick analysis of the model we can choose the bootstrap method more suited and explain why the others are not.

1. shuffle the errors → this method relies on assumption that the chosen model fits the data well and the residuals have constant variance
2. Parametric bootstrap for logistic regression → we are assuming that we

know the distribution of the population which we don't 3. Resample the line of the two datasets → this approach violates the assumption of constant (fixed) design matrix. however this applies perfectly to our case

```
boot_reg_game <- function(data, B=200){

  n0 <- nrow(data)

  res <- replicate(B,{
    ind <- sample(1:n0, n0, TRUE)
    Dstar <- data[ind,]
    Mstar <- lm(data=Dstar, average_rating ~ .)

    c(Mstar$coef, summary(Mstar)$r.squared, max((Mstar$coef["playingtime"]-Mstar$coef["minage"])/(Mstar$intercept)))
  })

  t(res)
}

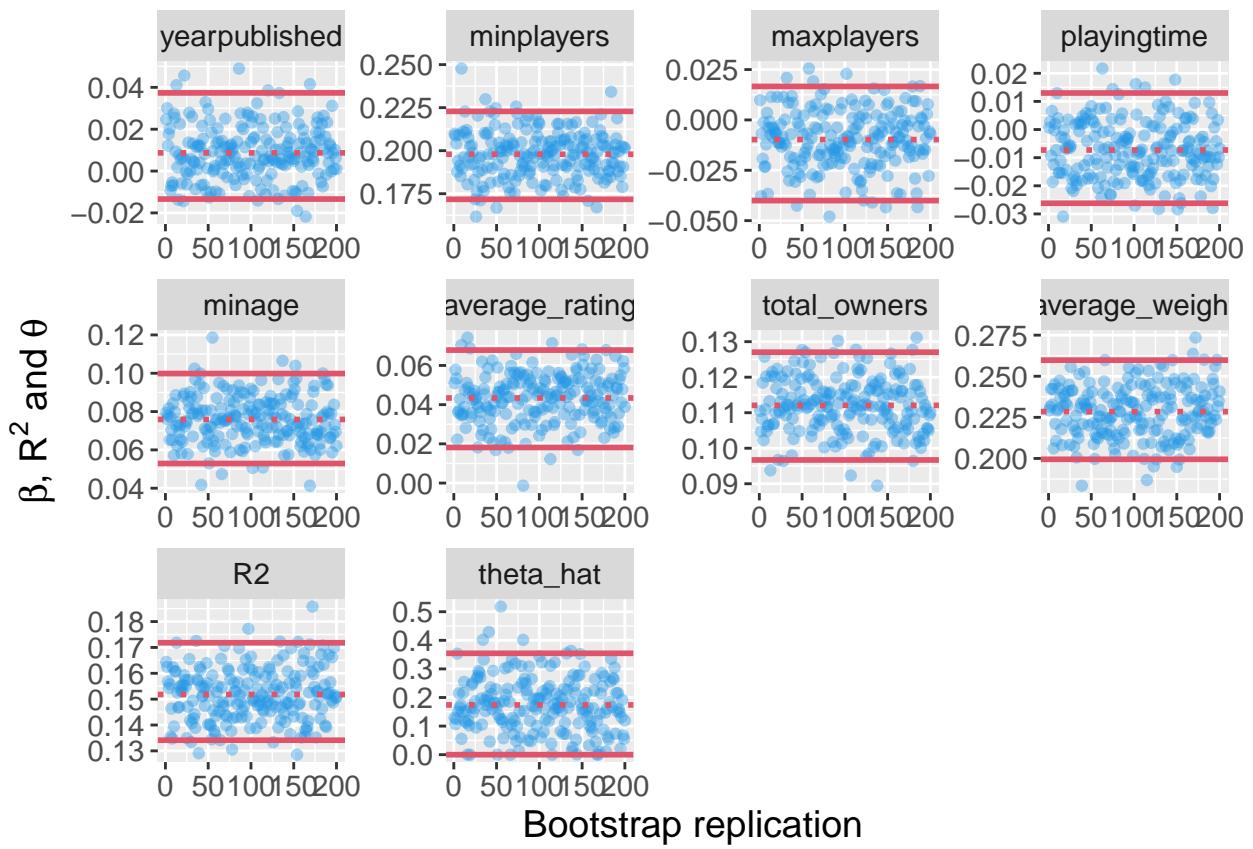
res_game_boot=boot_reg_game(data=sub_data)
colnames(res_game_boot)=c(colnames(sub_data), "R2" , "theta_hat")
```

Compute the estimates

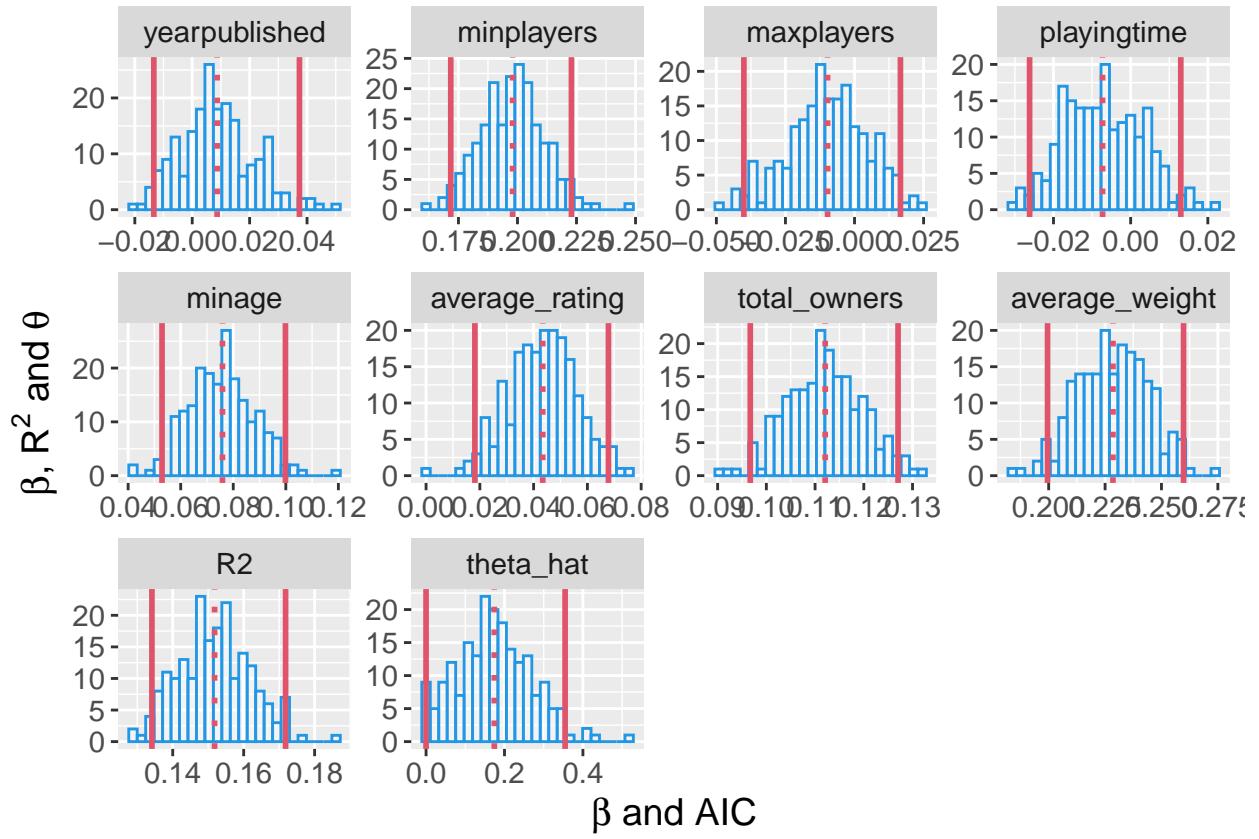
```
melted <- reshape2::melt(res_game_boot)
melted <- melted %>% group_by(Var2) %>% mutate(
  boot_sd = sd(value),
  boot_lowerbound = quantile(value, .025), boot_upperbound = quantile(value, .975),
  boot_mean = mean(value), bias=boot_mean-res_sub_data[,as.character(1)])
```

Plot the results

```
ggplot( melted )+
  geom_point(aes(x=Var1,y=value),alpha=.4,col=4)+
  geom_hline(aes(yintercept = boot_upperbound) ,col=2, lwd = 1)+
  geom_hline(aes(yintercept = boot_lowerbound) ,col=2, lwd = 1)+
  geom_hline(aes(yintercept = boot_mean) ,col=2, lwd = 1, lty=3)+
  facet_wrap(~Var2,scales = "free")+
  xlab("Bootstrap replication")+
  ylab(TeX("$\beta$, $R^2$ and $\theta$")) +
  theme(text = element_text(size=14), legend.position = "none")+
  scale_fill_manual(values="lightblue")
```



```
ggplot( melted )+
  geom_histogram(aes(x=value),alpha=.4,col=4,fill="white",bins=25) +
  geom_vline(aes(xintercept = boot_upperbound ),col=2, lwd = 1) +
  geom_vline(aes(xintercept = boot_lowerbound ),col=2, lwd = 1) +
  geom_vline(aes(xintercept = boot_mean) ,col=2, lwd = 1, lty=3) +
  facet_wrap(~Var2,scales = "free") +
  xlab(TeX("\beta$ and AIC")) +
  ylab(TeX("$\beta$, $R^2$ and $\theta$")) +
  theme(text = element_text(size=14), legend.position = "none")
```



1.8 Point h)

Jackknife approach

```
jack_reg_game = function(data, B){

  n0 <- nrow(data)

  res=matrix(NA, n0, ncol(data)+2)
  for(i in 1:n0){
    Mjack=lm(data[-i,], average_rating ~ .)
    res[i,]=c(Mjack$coef, summary(Mjack)$r.squared, max((Mjack$coef["playingtime"]-Mjack$coef["minage"])/
  }

  return(res)
}

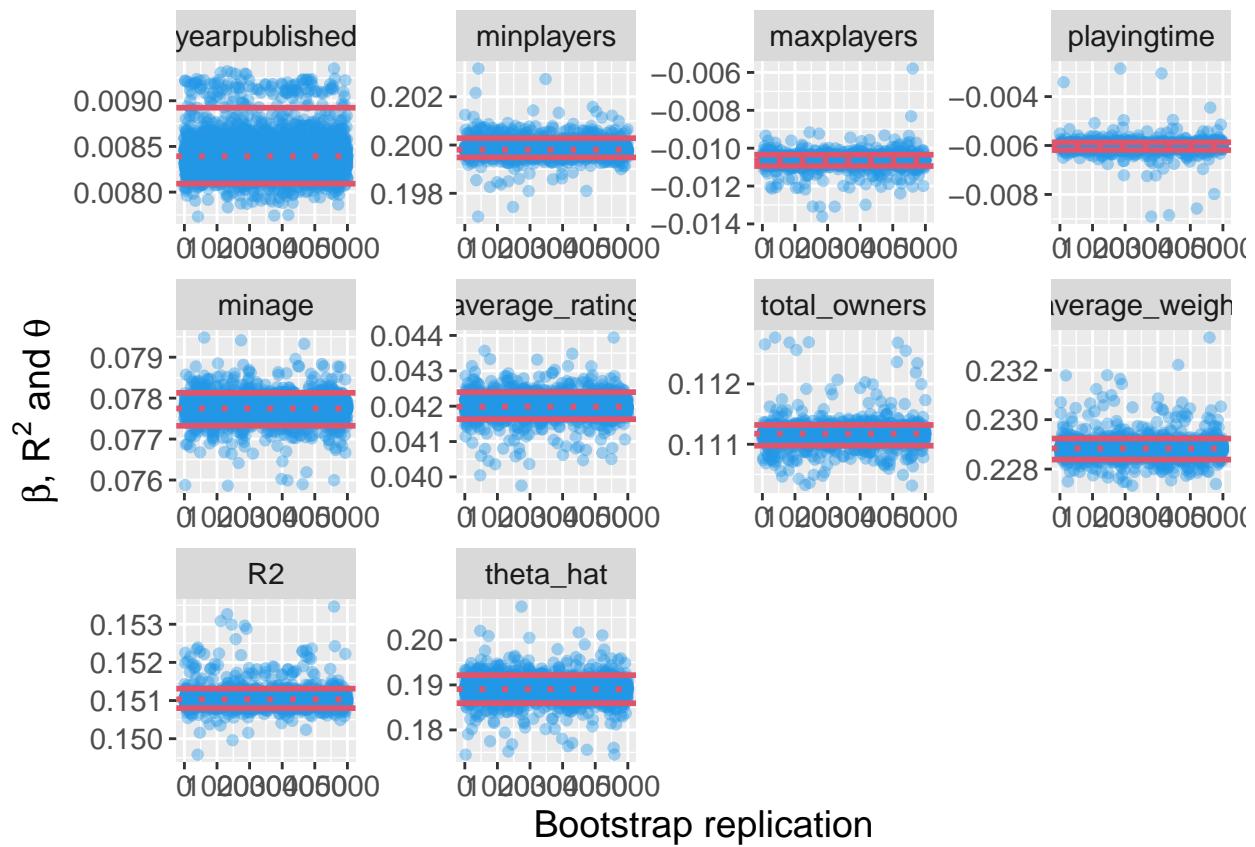
res_game_jack=jack_reg_game(sub_data, 10)
colnames(res_game_jack)=c(colnames(sub_data), "R2" , "theta_hat")
```

Compute the jackknife estimate, bias-corrected estimate and standard error

```
melted <- reshape2::melt(res_game_jack)
melted <- melted %>% group_by(Var2) %>% mutate( boot_sd = sd(value),
  boot_lowerbound = quantile(value, .025), boot_upperbound = quantile(value, .975),
  boot_mean = mean(value), bias=boot_mean-res_sub_data[,as.character(Var2)])
```

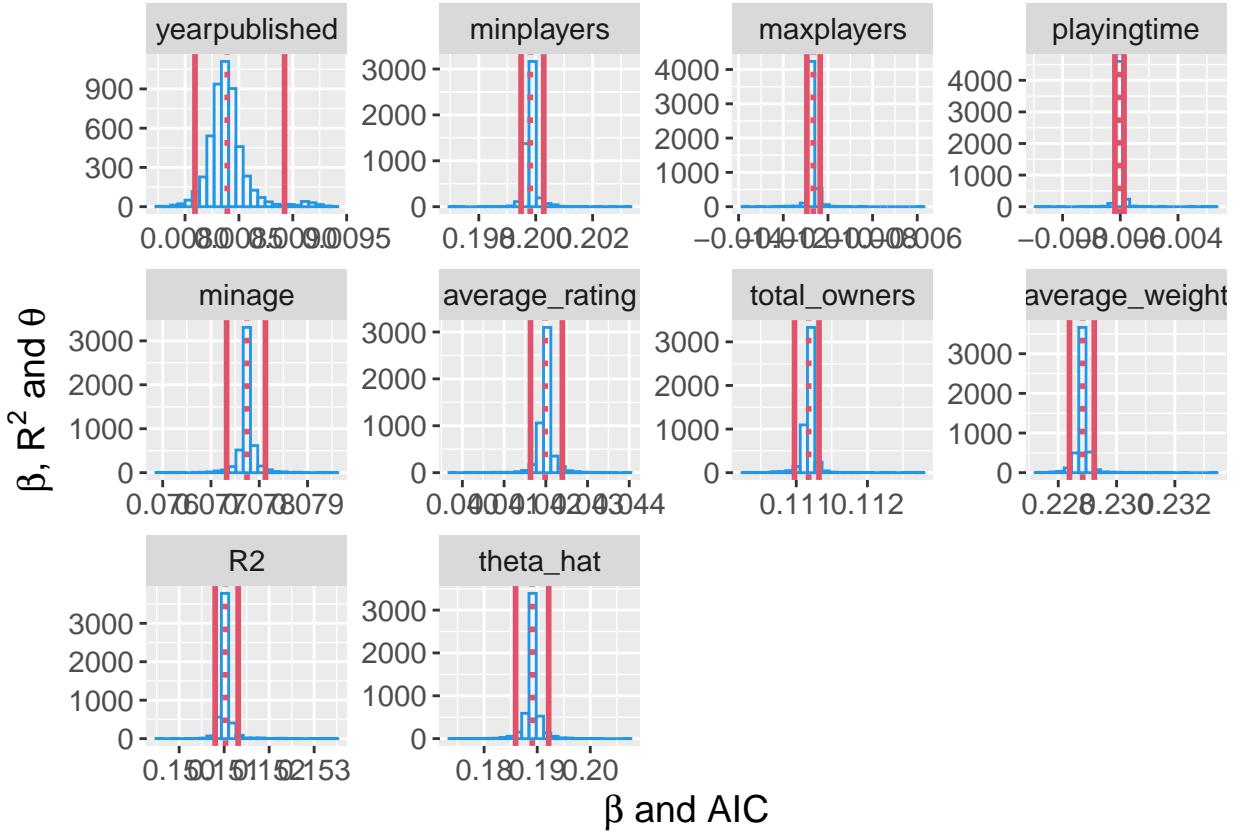
Plot the results

```
ggplot( melted )+
  geom_point(aes(x=Var1,y=value),alpha=.4,col=4)+
  geom_hline(aes(yintercept = boot_upperbound) ,col=2, lwd = 1)+
  geom_hline(aes(yintercept = boot_lowerbound) ,col=2, lwd = 1)+
  geom_hline(aes(yintercept = boot_mean) ,col=2, lwd = 1, lty=3)+
  facet_wrap(~Var2,scales = "free")+
  xlab("Bootstrap replication")+
  ylab(TeX("$\\beta$, $R^2$ and $\\theta$")) +
  theme(text = element_text(size=14), legend.position = "none")
```



```
ggplot( melted )+
  geom_histogram(aes(x=value),alpha=.4,col=4,fill="white",bins=25)+
```

 $\text{geom_vline}(\text{aes}(xintercept = boot_upperbound) ,\text{col}=2, \text{lwd} = 1)+$
 $\text{geom_vline}(\text{aes}(xintercept = boot_lowerbound) ,\text{col}=2, \text{lwd} = 1)+$
 $\text{geom_vline}(\text{aes}(xintercept = boot_mean) ,\text{col}=2, \text{lwd} = 1, \text{lty}=3)+$
 $\text{facet_wrap}(\sim \text{Var2}, \text{scales} = \text{"free"})+$
 $\text{xlab}(\text{TeX}("$\\beta$ and AIC"))+$
 $\text{ylab}(\text{TeX}("$\\beta$, R^2 and $\\theta$")) +$
 $\text{theme}(\text{text} = \text{element_text}(\text{size}=14), \text{legend.position} = \text{"none"})$



2 Exercise 2 - We need some music!

2.1 Point a)

The dataset we're working with is a nested from a bigger version comprehending 2072 observations for 18 variables providing data of a Spotify tracks sample. Data are provided by the Spotify for Developers API (source: <https://developer.spotify.com/documentation/web-api/reference/get-audio-features>).

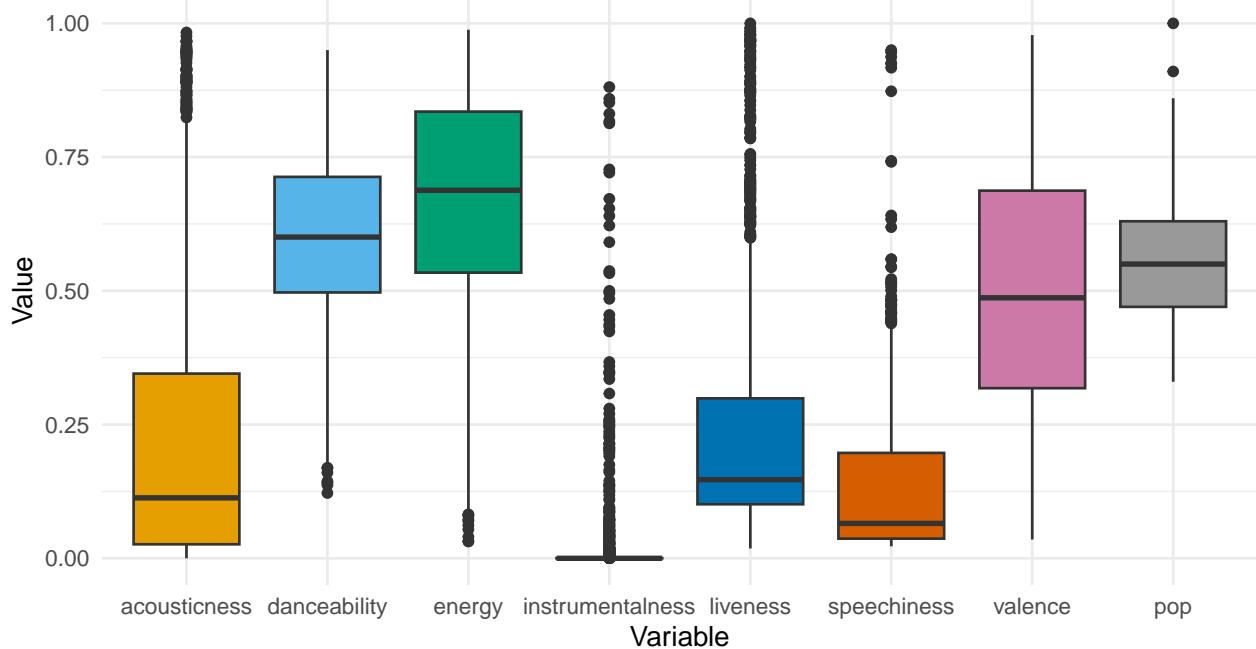
acousticness	danceability	energy	instrumentalness	liveness	speechiness
Min. :0.0000063	Min. :0.1220	Min. :0.0316	Min. :0.0000000	Min. :0.0184	Min. :0.02250
1st Qu.:0.0262000	1st Qu.:0.4970	1st Qu.:0.5340	1st Qu.:0.0000000	1st Qu.:0.1010	1st Qu.:0.03678
Median :0.1130000	Median :0.6005	Median :0.6880	Median :0.0000013	Median :0.1470	Median :0.06520
Mean :0.2224369	Mean :0.5968	Mean :0.6678	Mean :0.0141941	Mean :0.2267	Mean :0.12895
3rd Qu.:0.3452500	3rd Qu.:0.7130	3rd Qu.:0.8350	3rd Qu.:0.0001810	3rd Qu.:0.2990	3rd Qu.:0.19700
Max. :0.9830000	Max. :0.9500	Max. :0.9880	Max. :0.8810000	Max. :1.0000	Max. :0.95000

valence	pop	loudness	duration_ms	tempo	n
Min. :0.0352	Min. :0.3300	Min. :-24.383	Min. : 31360	Min. : 46.17	Min. : 11.00
1st Qu.:0.3180	1st Qu.:0.4700	1st Qu.: -8.141	1st Qu.:200723	1st Qu.: 95.28	1st Qu.: 44.00
Median :0.4870	Median :0.5500	Median : -6.285	Median :234886	Median :117.96	Median :102.00
Mean :0.5019	Mean :0.5545	Mean : -6.798	Mean :241978	Mean :121.07	Mean : 88.41
3rd Qu.:0.6873	3rd Qu.:0.6300	3rd Qu.: -4.773	3rd Qu.:273056	3rd Qu.:141.91	3rd Qu.:126.00
Max. :0.9780	Max. :1.0000	Max. : -0.564	Max. :768640	Max. :220.07	Max. :142.00

Note that factors and characters are excluded from the summary. The first 8 variables are all synthetic float indexes from 0 to 1 describing one feature of the track, sometime based on musical features like path,

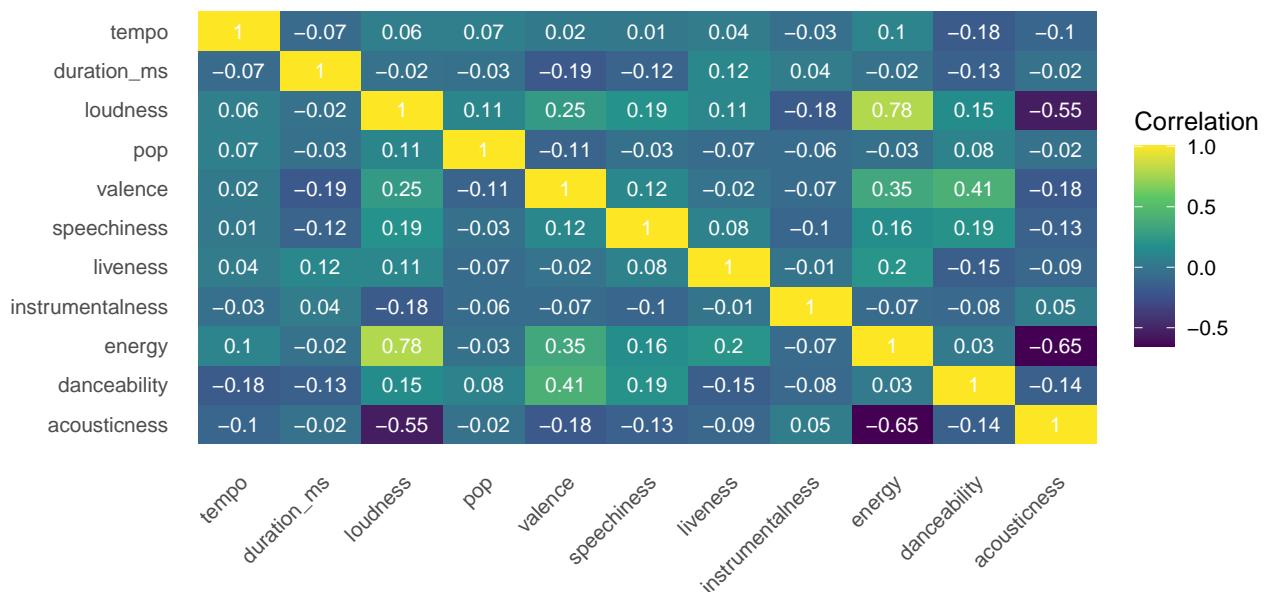
tempo, beat, stability, strength, regularity, and sometimes based on elements which may come out during the analysis (background noise for liveness, sound for acousticness). These ones measure for example danceability or energy as well as liveness or acousticness. Then there are three numerical variables which are measures of duration, loudness and the tempo of the song and a factor which tells if the track is explicit (0) or not (1). Four qualitative variables then tell us the original author, the artist who actually published the track, and an ID of the track itself. Variable n is eventually the number of songs by the same artist in this dataset.

Boxplots of synthetic indexes

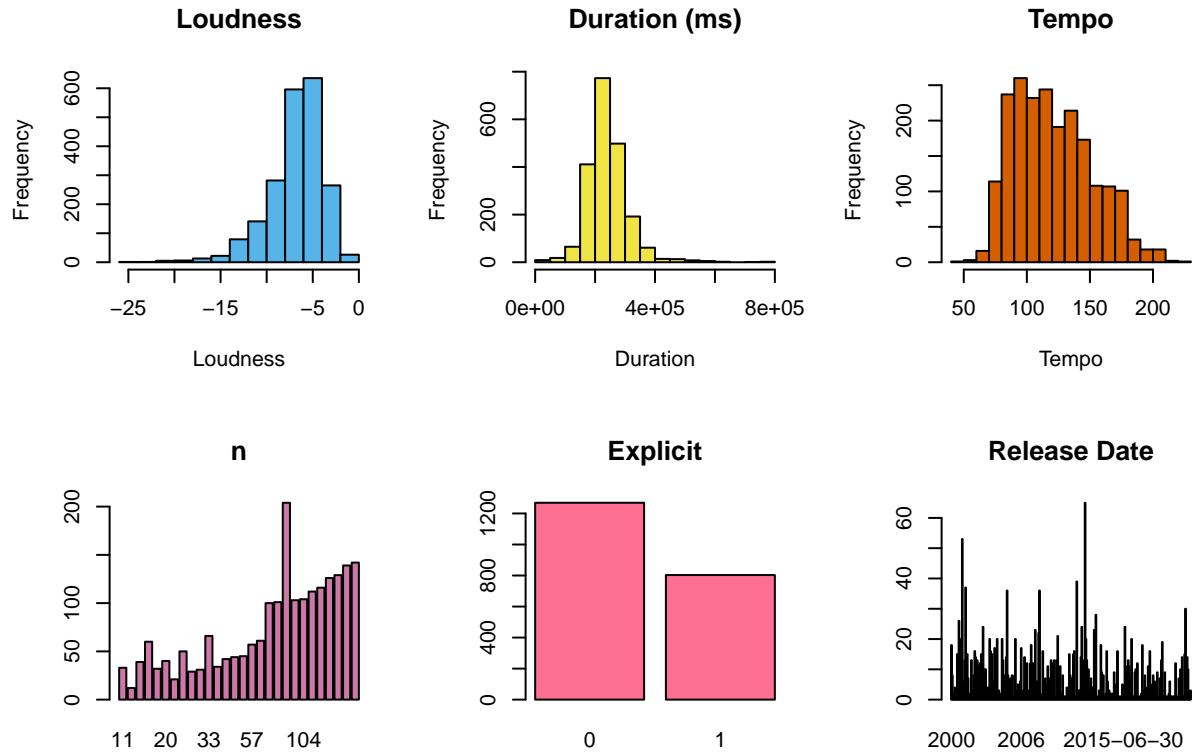


From these boxplots we can have an idea of the distributions of each feature among the tracks. For example we can say that most songs are likely to be non live, acoustic or spoken. Danceability, energy and popularity are distributed a little above 0.5 and the first two present a fatter left tail and viceversa for the third. The variable valence seems to be the most symmetric around the middle value, with equally weighted tails.

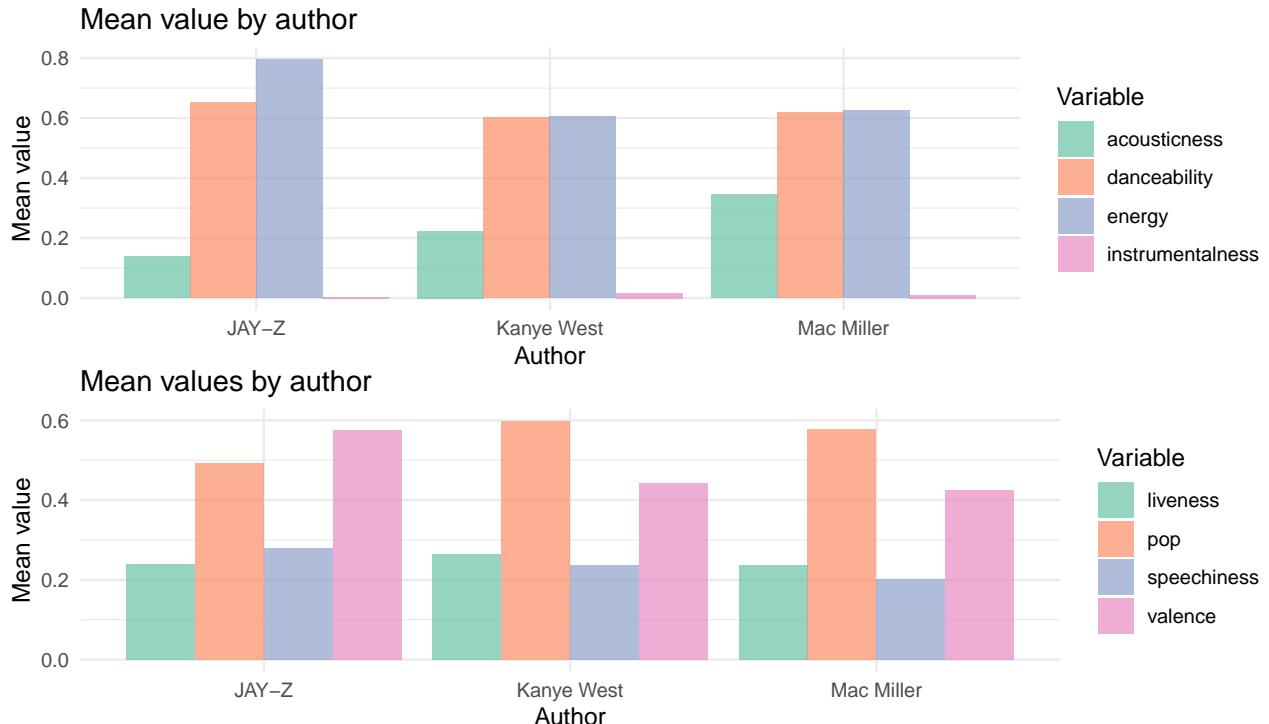
Correlation Matrix



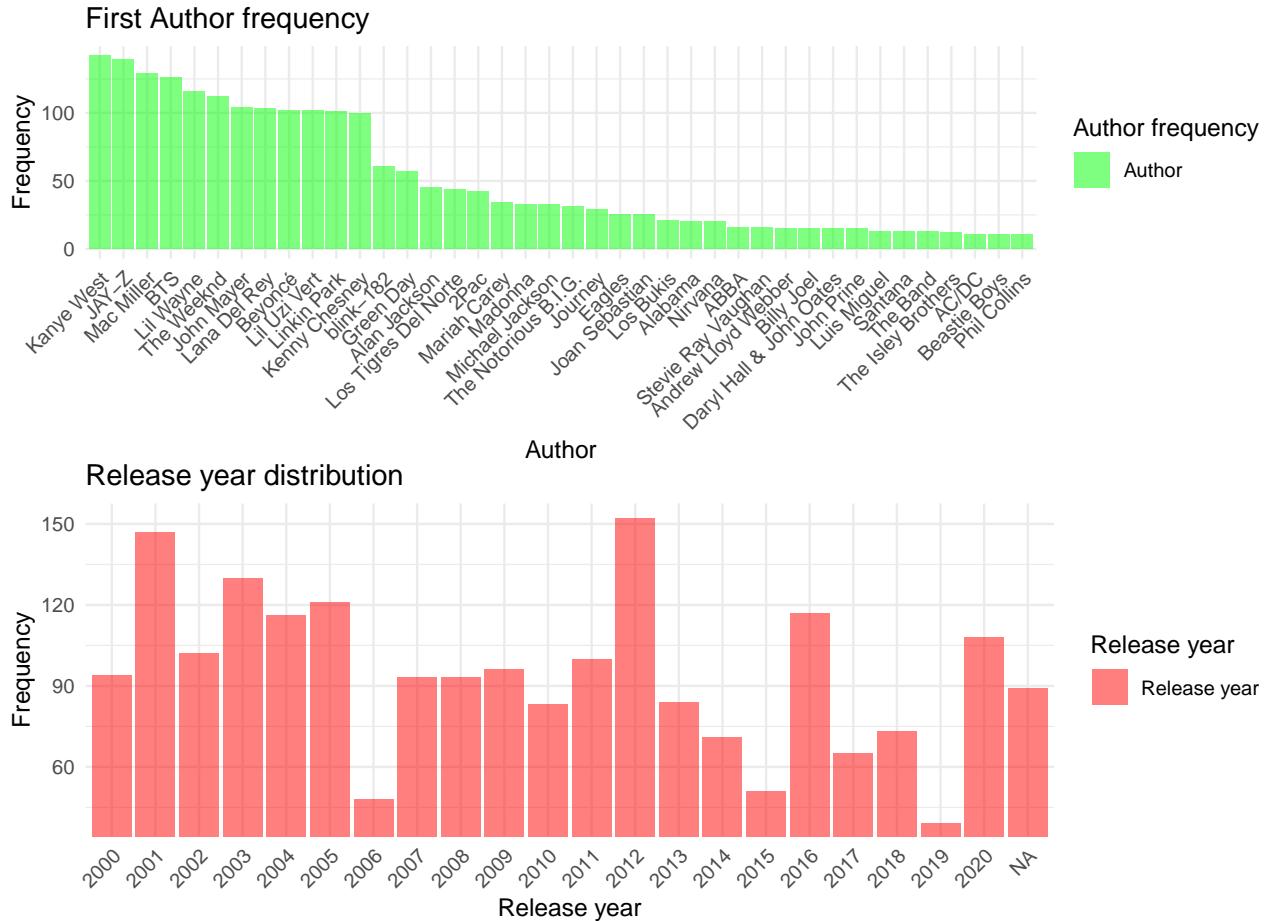
The correlation matrix plotted on a heatmap shows that most variables are almost uncorrelated with each other. The variable loudness is strongly positive related with energy and negative related with acousticness, which perfectly makes sense as well as the fact that energy and acousticness are negative related. Another reasonable result is that valence (negativity or positivity of the song) has some correlation with danceability and energy.



The histograms show the distribution for numerical variables.



Histograms above show the mean value of our indexes for the top three artists (based on number of songs in the data) to have a qualitative idea of at least the three authors that contributed the most to the data.



We plotted the frequencies of authors in the dataset and frequencies of release per year, to have an idea of how the two most relevant qualitative variables are distributed, aggregating tracks among their author and over the years of release.

2.2 Point b

```
# Function definition
pop_est = function(data, auth_X, auth_Y) {
  Xauth_pop = data %>%
    filter(first_auth == auth_X) %>%
    summarise(med_log = median(logit(pop)))
  Yauth_pop = data %>%
    filter(first_auth == auth_Y) %>%
    summarise(med_log = median(logit(pop)))
  pop_diff = abs(Xauth_pop$med_log - Yauth_pop$med_log)
  return(pop_diff)
}
# Apply the function
diff <- pop_est(df2, "AC/DC", "Kanye West")
print(diff)
```

```

## [1] 0.04149973

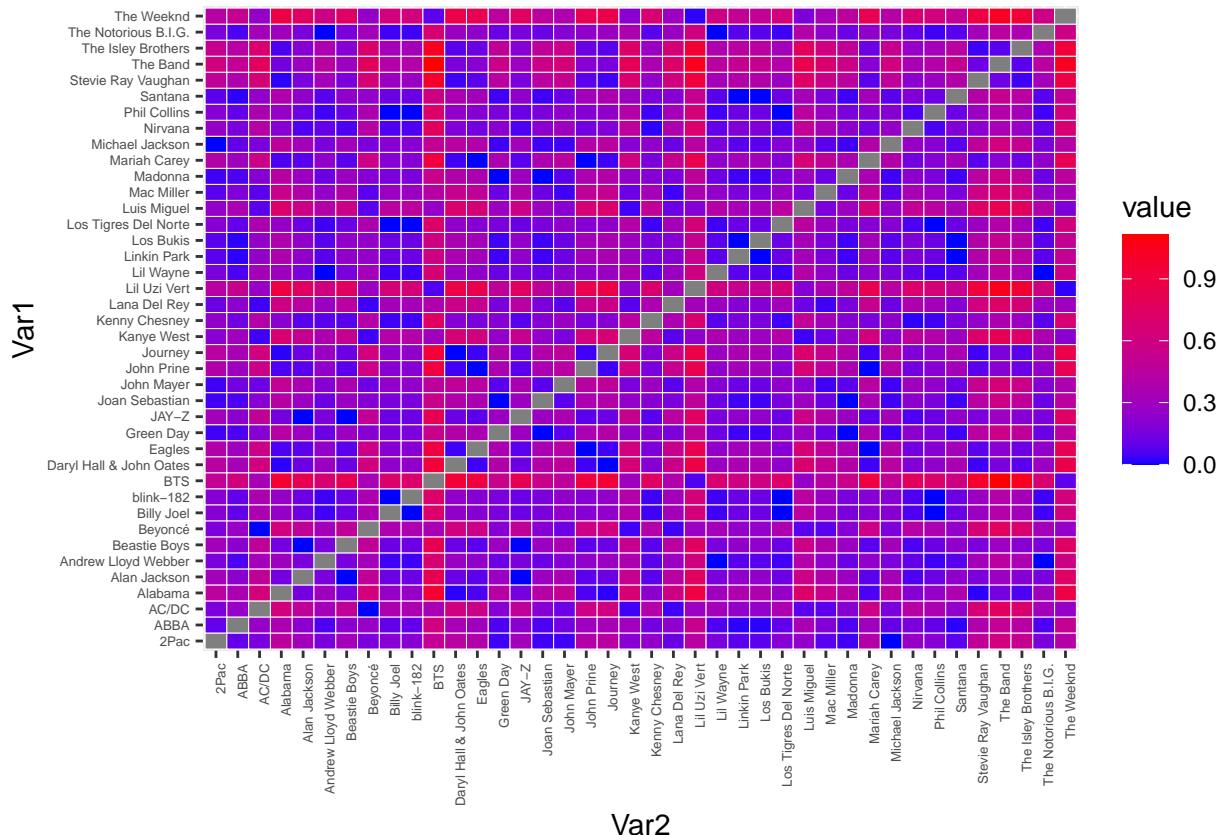
# Lista degli autori unici nel dataset
auth <- unique(df2$first_auth)

# Creazione della matrice delle comparazioni
comp <- matrix(NA, nrow = length(auth), ncol = length(auth), dimnames = list(auth, auth))

# Loop per calcolare le comparazioni
for (i in 1:length(auth)) {
  for (j in 1:length(auth)) {
    if (i != j) {
      diff <- pop_est(df2, auth[i], auth[j])
      comp[i, j] <- diff
    }
  }
}
}

heatmap <- ggplot(data = melt(comp)) +
  geom_tile(aes(x = Var2, y = Var1, fill = value), color = "white") +
  scale_fill_gradient(low = "blue", high = "red") +
  theme(axis.text.x = element_text(angle=90, hjust = 1, size=5)) +
  theme(axis.text.y = element_text(hjust = 1, size=5))
heatmap

```



2.3 Point c)

The formal testing hypothesis system would be the following:

- 1) Null Hypothesis (H0): There is no difference in popularity between two artists, X and Y. In other words, the median log-odds of popularity for X is equal to the median log-odds of popularity for Y.

$$H_0 : \left| \text{median} \left(\frac{\log(X)}{\log(1-X)} \right) - \text{median} \left(\frac{\log(Y)}{\log(1-Y)} \right) \right|$$

- 2) Alternative Hypothesis (HA): There is a difference in popularity between two artists, X and Y. In other words, the median log-odds of popularity for X is not equal to the median log-odds of popularity for Y. $H_0 : \left| \text{median} \left(\frac{\log(X)}{\log(1-X)} \right) - \text{median} \left(\frac{\log(Y)}{\log(1-Y)} \right) \right|$

```
# re defining pop_est
pop_est <- function(df, auth_X, auth_Y, songs) {
  # Authors selection
  songs_X <- df$pop[df$first_auth == auth_X]
  songs_Y <- df$pop[df$first_auth == auth_Y]

  # Songs vector
  if (!missing(songs)) {
    songs_X <- songs[1:length(songs_X)]
    songs_Y <- songs[(length(songs_X) + 1):(length(songs_X) + length(songs_Y))]
  }

  # formulas
  median_X <- median(logit(songs_X))
  median_Y <- median(logit(songs_Y))

  # Statistic
  return(abs(median_X - median_Y))
}

# Variables
first_auth <- df2$first_auth
pop <- df2$pop

# Initialize matrix
p_value_matrix <- matrix(NA, nrow = length(unique(first_auth)), ncol = length(unique(first_auth)))

# P-values
authors <- unique(first_auth)
for (i in 1:(length(authors)-1)) {
  for (j in (i+1):length(authors)) {
    author1 <- authors[i]
    author2 <- authors[j]

    # Tracks of authors
    songs_author1 <- pop[first_auth == author1]
    songs_author2 <- pop[first_auth == author2]

    # Obs_stat
```

```

observed_statistic <- abs(pop_est(df2, author1, author2))

# Permutations
num_permutations <- 1000
permutation_stats <- rep(NA, num_permutations)

for (k in 1:num_permutations) {
  permuted_songs <- sample(c(songs_author1, songs_author2))

  # Permutations of current stat
  permutation_stats[k] <- abs(pop_est(df2, author1, author2, permuted_songs))
}

# P-values
p_value <- sum(permutation_stats >= observed_statistic) / num_permutations

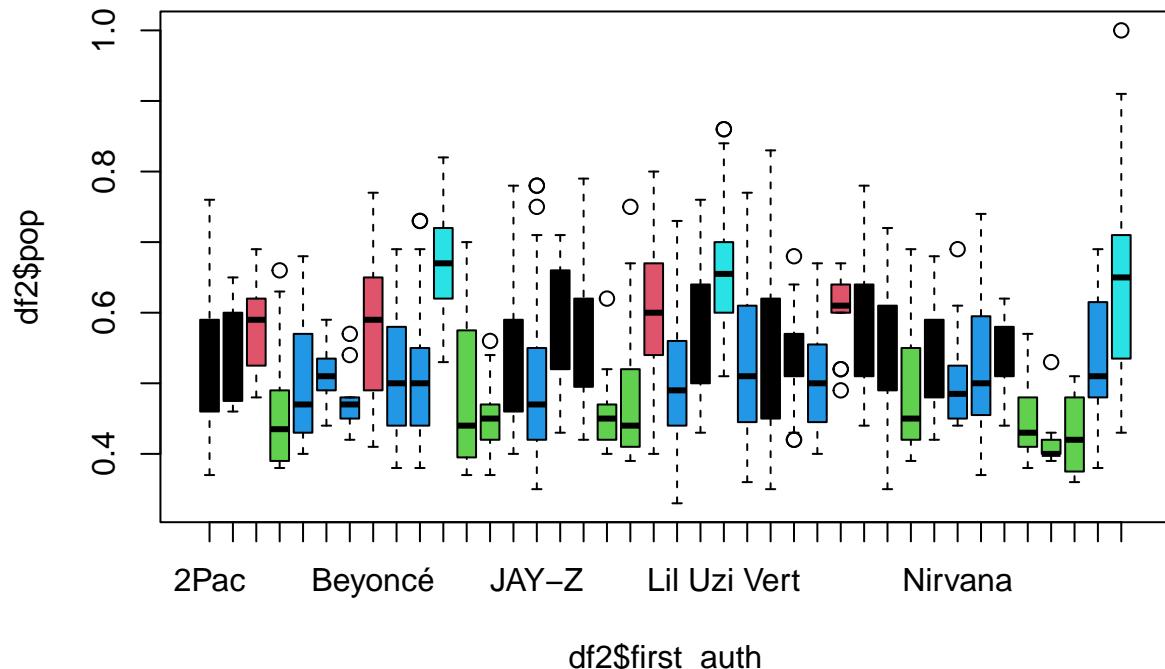
# Update matrix
p_value_matrix[i, j] <- p_value
p_value_matrix[j, i] <- p_value
}
}

pval_mat = p_value_matrix

```

2.4 Point d)

```
boxplot(df2$pop ~ df2$first_auth, col = cutree(hclust(as.dist(1-pval_mat))), 5))
```



The code above is meant to plot boxplots to show the distribution of the pop variable for each level of the first_auth variable. “col” is an argument for painting the plots. The composed function inside this argument can be disentangled as follows: First of all, complements to 1 are applied to the p-value matrix in order

to switch the values into a divergency measure in terms of distance. This may be because it's divergency (distance) is used in hierarchical clustering. Then the function “as.dist” is applied to our new matrix of complements to transform it into a distance matrix. This function operates assuming a symmetric square matrix so that it takes the lower half and computes the distance between the rows of the input matrix. So in this case the function compute the distances and store them into a matrix where the i, j component corresponds to the distance between row i and row j of the input matrix.

Proceeding the code analysis, the “hclust” function is applied to the result above. This function is used to perform agglomerative hierarchical clustering on a distance matrix. The hierarchical clustering algorithm constructs a hierarchy of clusters by iteratively merging rows or columns based on their similarity or dissimilarity. Hierarchical clustering produces a cluster tree, known as a dendrogram, which shows how the rows or columns were grouped during the clustering process. The dendrogram can be used to select a certain number of clusters or to identify the structure of groups based on the heights of the tree nodes. Finally The “cutree” function in R is used to partition the dendrogram generated by the “hclust” function into a specified number of groups, in our case, 5. It takes an object of class “hclust” representing the hierarchical cluster tree as input and returns a vector of labels indicating the membership of each observation to the generated groups. In our context, the cutree function is used to assign colors (as it is an argument to “col”) to the groups obtained from hierarchical clustering based on the distance matrix. The hierarchical tree is thus divided into 5 distinct groups, and cutree returns a vector of labels assigning each observation a corresponding group membership label. Once the label vector is obtained from cutree, it's used to color the groups in the boxplot graph.

To sum up, by running this code boxplots will be generated for the “pop” (popularity) variable, divided into groups corresponding to the song authors. Each boxplot represents the distribution of song popularity for a specific author. By coloring the boxplots based on the groups obtained from hierarchical clustering, you will obtain a visualization where similar author groups (with similar popularity) are represented with similar colors. This can help identify if there are groups of authors that have significantly different popularity levels. For example, observing that boxplots in one group (color) have similar medians and ranges, while boxplots in another group have different medians and ranges, could indicate significant differences in song popularity between the two author groups.

2.5 Point e)

```
df_t_test= df2 %>%
  mutate( pop = scale(pop))
artists=unique(df_t_test$first_auth)

t_art=c()
for(i in artists){
  t_art=c(t_art, t.test(df_t_test$pop[df_t_test$first_auth == i], mu=0, alternative="greater")$p.value)
}
alpha=0.1
t_art[which(t_art<alpha)]

## [1] 2.209962e-03 6.182956e-41 4.837052e-02 4.268830e-08 8.943500e-03
## [6] 2.415517e-23 5.805835e-03 6.148819e-04 2.176939e-10

# Probability of the global null to be violated
(1-alpha)^40

## [1] 0.01478088
```

2.6 Point f)

```
function(method, df){  
  p=p.adjust(df, method=  
}  
  
## function(method, df){  
##   p=p.adjust(df, method= method)  
##  
## }  
  
t_art_bonf=p.adjust(t_art, method= "bonferroni")  
t_art_holm=p.adjust(t_art, method= "holm")  
t_art_bh=p.adjust(t_art, method= "BH")
```

Difference between the 3 approaches: - Bonferroni → we are controlling for the number of false discoveries, in particular the probability of getting even one false discovery is less than half - Holm → similar to the Bonferroni but with more power - Benjamini-Hockberg → the parameter which is controlled is the false discovery rate, but since we do not know V, we can take the estimate and set it less than or equal to alpha

2.7 Point g)