ADS_S2

2111

2024-05-27

```
knitr::opts_chunk$set(echo = TRUE, message = FALSE)
library(readr)
library(ggplot2)
library(dplyr)
```

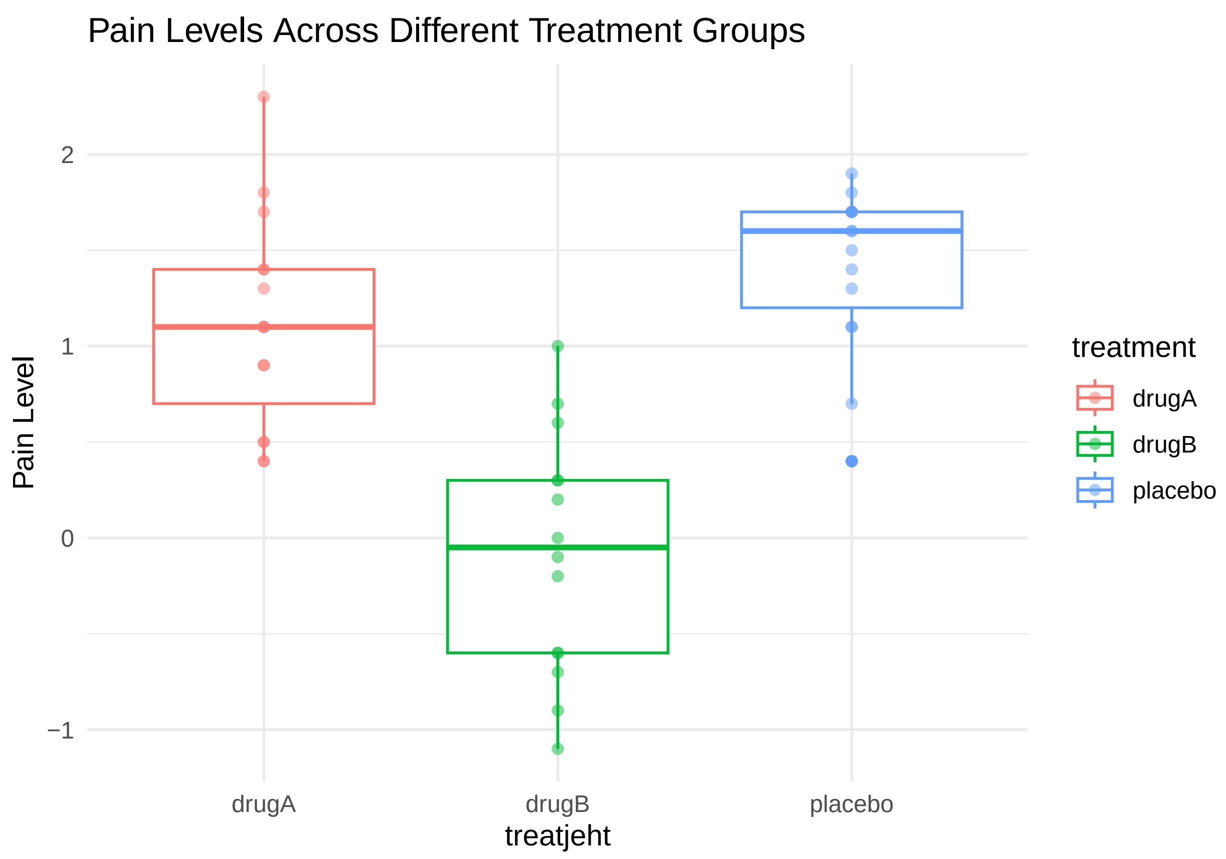
```
##
##    'dplyr'

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

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

```
library(tidyr)
library(RColorBrewer)
```

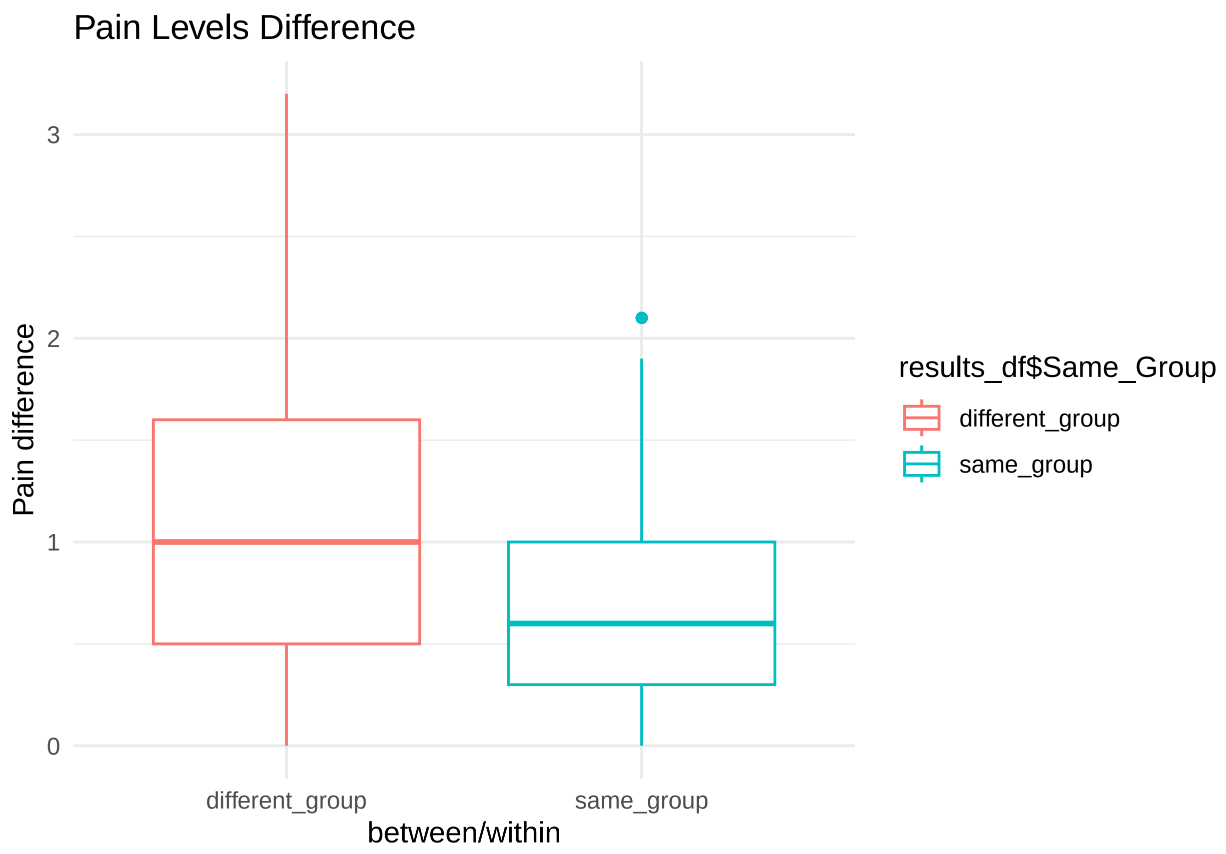
Week1 multiple means

limitation of t-test several times P(at least 1 false positive will be very large)

```
#1. import the dataset and overview it
dt <- read_csv("C:/Users/10755/Desktop/Biooooo/ADS/S2/practical/Week1-drug_trial.csv")

ggplot(dt, aes(x = treatment, y = pain, color = treatment)) +
  geom_boxplot() +
  geom_point(alpha = 0.5) +
  labs(x = "treatjeht", y = "Pain Level", title = "Pain Levels Across Different Treatment Groups") +
  theme_minimal()
```

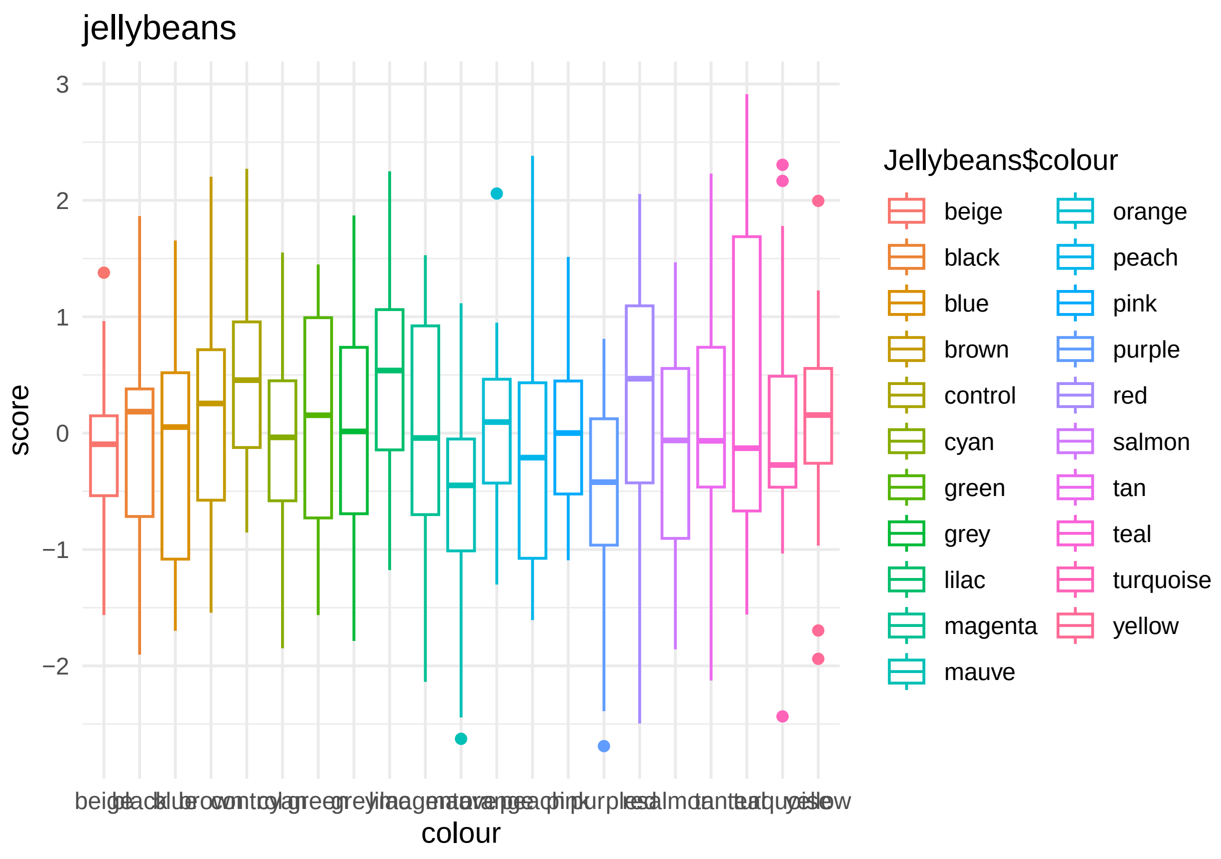


```r
#2. H0 and HA
#1) (H0): The mean pain levels for all three treatment groups are equal.
#   (HA): At least one treatment group has a different mean pain level compared to the others.
#2) (H0): Drawing two data points at random from different groups will be as different from each other 
#   (HA): Drawing two data points at random from different groups will be more different from each othe



#3. get the differences using loop
# Number of iterations
num_iterations <- 1000

# Initialize empty data frame to store results
results_df <- data.frame(Pain_Difference = numeric(num_iterations),
                         Same_Group = character(num_iterations))

# Loop over the iterations
for (i in 1:num_iterations) {
  row1 <- sample(1:nrow(dt), 1)
  row2 <- sample(1:nrow(dt), 1)
  while(row2 == row1) {
    row2 <- sample(1:nrow(dt), 1)
  }
  sample_data <- dt[c(row1, row2), ]
  pain_difference <- abs(sample_data$pain[1] - sample_data$pain[2])
  if (sample_data$treatment[1] == sample_data$treatment[2]) {
    same_group <- "same_group"
  }else{
    same_group <- "different_group"
  }
  results_df[i, ] <- list(Pain_Difference = pain_difference,
                          Same_Group = same_group)
}

mean_between <-mean(results_df$Pain_Difference[results_df$Same_Group == "different_group"])
mean_within <-mean(results_df$Pain_Difference[results_df$Same_Group == "same_group"])

ggplot(results_df, aes(x = results_df$Same_Group, y = results_df$Pain_Difference, color = results_df$Sa
  geom_boxplot() +
  labs(x = "between/within", y = "Pain difference", title = "Pain Levels Difference") +
  theme_minimal()
```



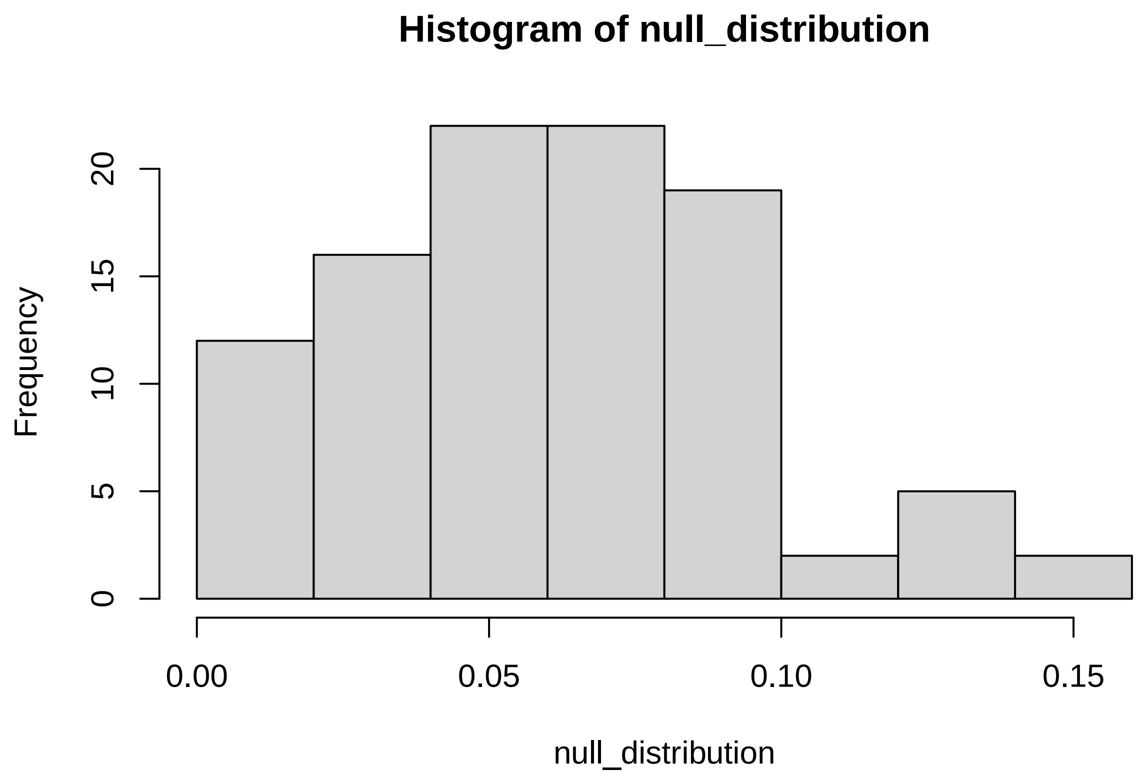
```
#a t-test is not appropriate in this case because the assumption of normality
#may not hold for the distribution of pain differences

wilcox_test <- wilcox.test(Pain_Difference ~ Same_Group, data = results_df)
```

```
# import data
Jellybeans <- read_csv("C:/Users/10755/Desktop/Biooooo/ADS/S2/prob/Week1_jellybeans.csv")

ggplot(Jellybeans, aes(x = colour, y = score, color = Jellybeans$colour)) +
  geom_boxplot() +
  labs(x = "colour", y = "score", title = "jellybeans") +
  theme_minimal()
```



```
# create a function that randommly pick up two samples and compare their difference.
# but in this case, we need to split it into two part: within and between groups.

# create compare_pairs function.
# Inputs: name of dataset, number of draws
# draws random pairs of sample points that are either in the same group or in different groups
# computes their absolute distance
# computes the mean distances in the same group and the mean of distances in different groups
# returns the difference between those mean distances
compare_pairs2 <- function(dataset, ndraws) {
  same_group <- {}
  between_group <- {}
  for (i in 1:ndraws) {
    # determine condition (colour) to sample from
    list_colours <- unique(Jellybeans$colour)
    colours <- sample(list_colours, 2)
    # Make speարate lists for each group
    # (not strictly necessary, but helpful)
    firstgroup <- Jellybeans[Jellybeans$colour == colours[1], "score"]
    secondgroup <- Jellybeans[Jellybeans$colour == colours[1], "score"]
    # draw samples
```

```r
    s1s2 <- sample(firstgroup$score, 2)
    s3 <- sample(secondgroup$score, 1)
    # compute same group and between group differences, add to list
    same_group <- c(same_group, abs(s1s2[1] - s1s2[2]))
    between_group <- c(between_group, abs(s1s2[2] - s3))
  }
  # compute means of same-group and between-group differences
  mean_same <- mean(same_group)
  mean_between <- mean(between_group)
  # compute absolute difference between those means
  diffmeans = abs(mean_same-mean_between)
  return(diffmeans)
}


our_experiment <- compare_pairs2(Jellybeans, 1000)

null_distribution <- {}
for (j in 1:100) {
  # make new dataframe called random_experiment
  # and shuffle colour column
  random_experiment <- Jellybeans
  random_experiment$colour <- sample(Jellybeans$colour,
                                    nrow(Jellybeans), replace=FALSE)
  # use compare_pairs function on random_experiment
  random_meandiff2 <- compare_pairs2(random_experiment, 1000)
  # add the result to the Null distribution
  null_distribution <- c(null_distribution, random_meandiff2)
}
hist(null_distribution)
```

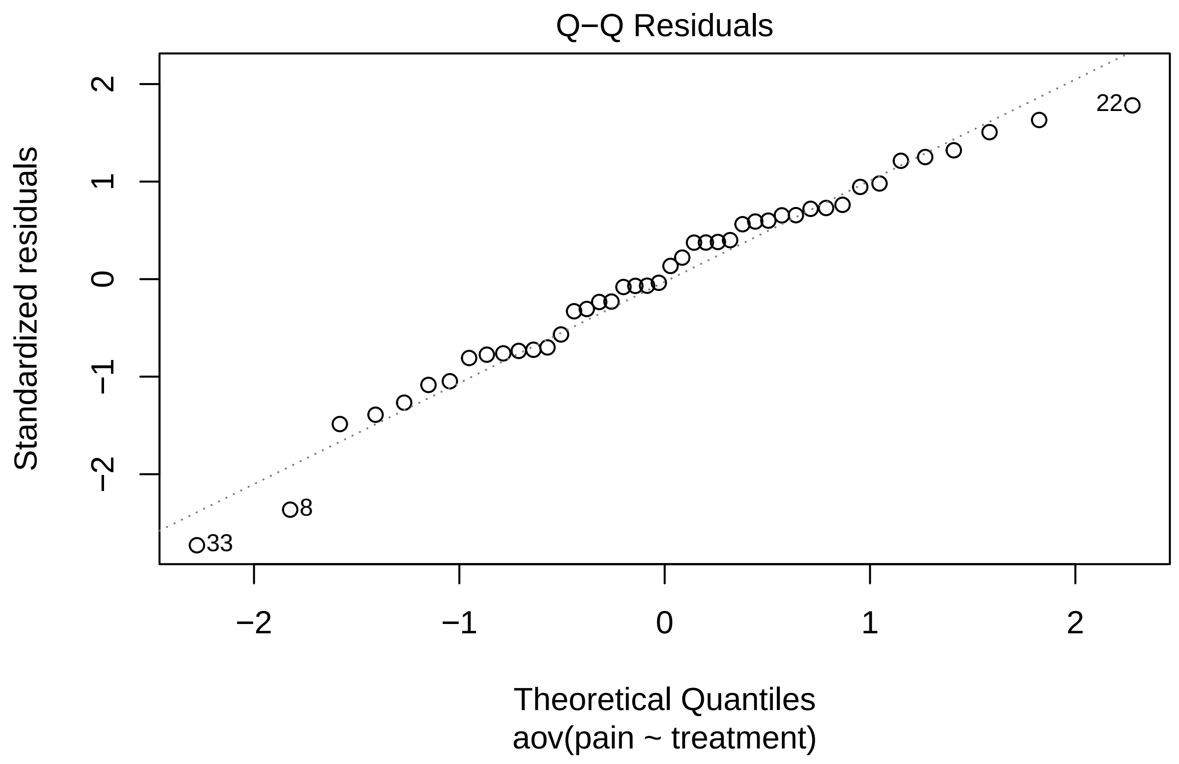


Week2 ANOVA

assumption of annova: • Independent random sampling • Normality of residuals (distances from group mean) • Equality of Variances

Post-hoc tests:Tukey's HSD test • Honestly Signifcant Difference

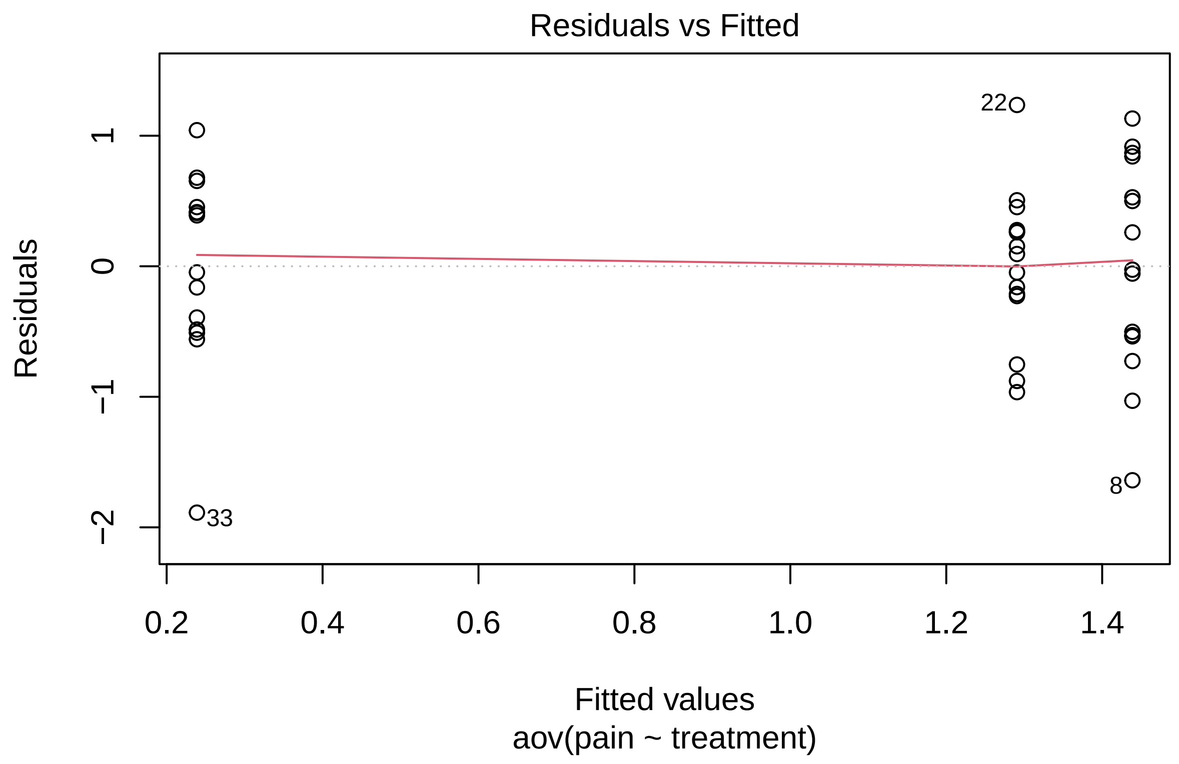
```
###
dt <- read_csv("C:/Users/10755/Desktop/Biooooo/ADS/S2/practical/Week2-drug_trial(2).csv")

#1. check for ANOVA assumption and run an ANOVA
model = aov(pain~treatment, data=dt)
plot(model, 2) # using the plot
```



```
shapiro.test(resid(model)) # using a formal test
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(model)
## W = 0.97317, p-value = 0.3895
```

```
plot(model, 1) # check for same variance
```



```r
summary(model)
```

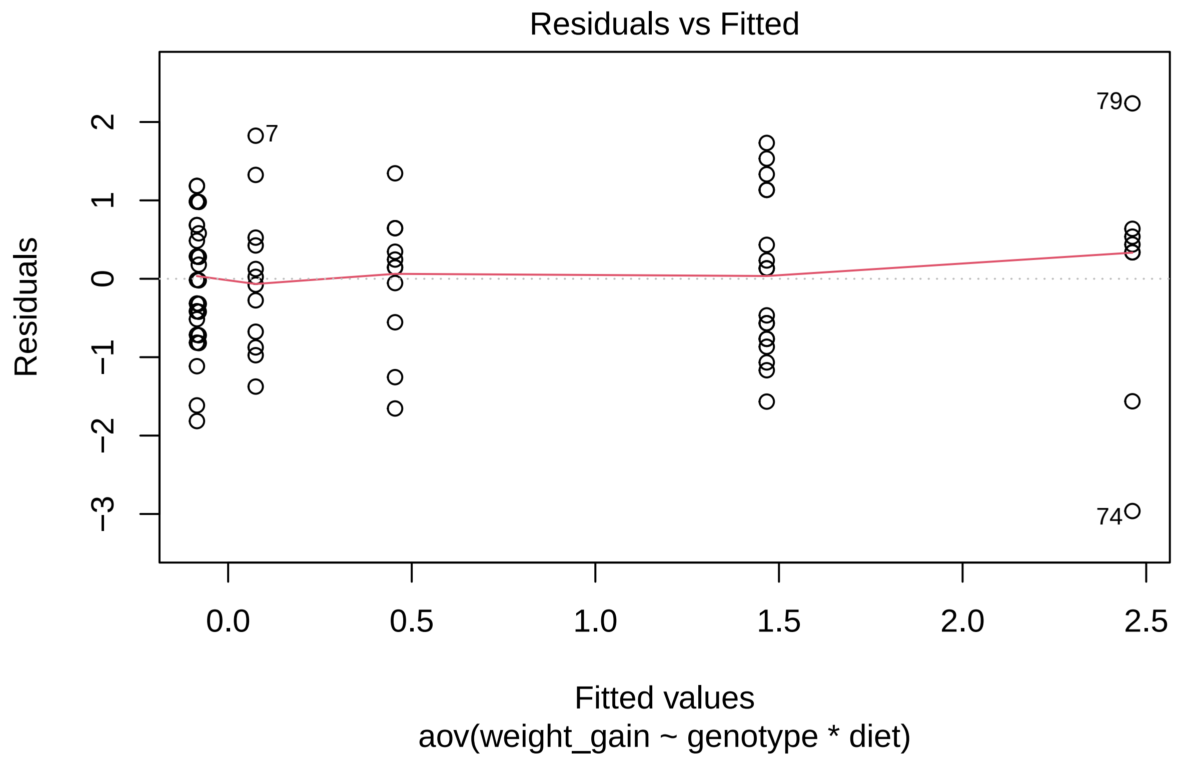
```
##             Df Sum Sq Mean Sq F value  Pr(>F)    
## treatment    2  12.27   6.133    11.9 8.4e-05 ***
## Residuals   41  21.13   0.515                    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

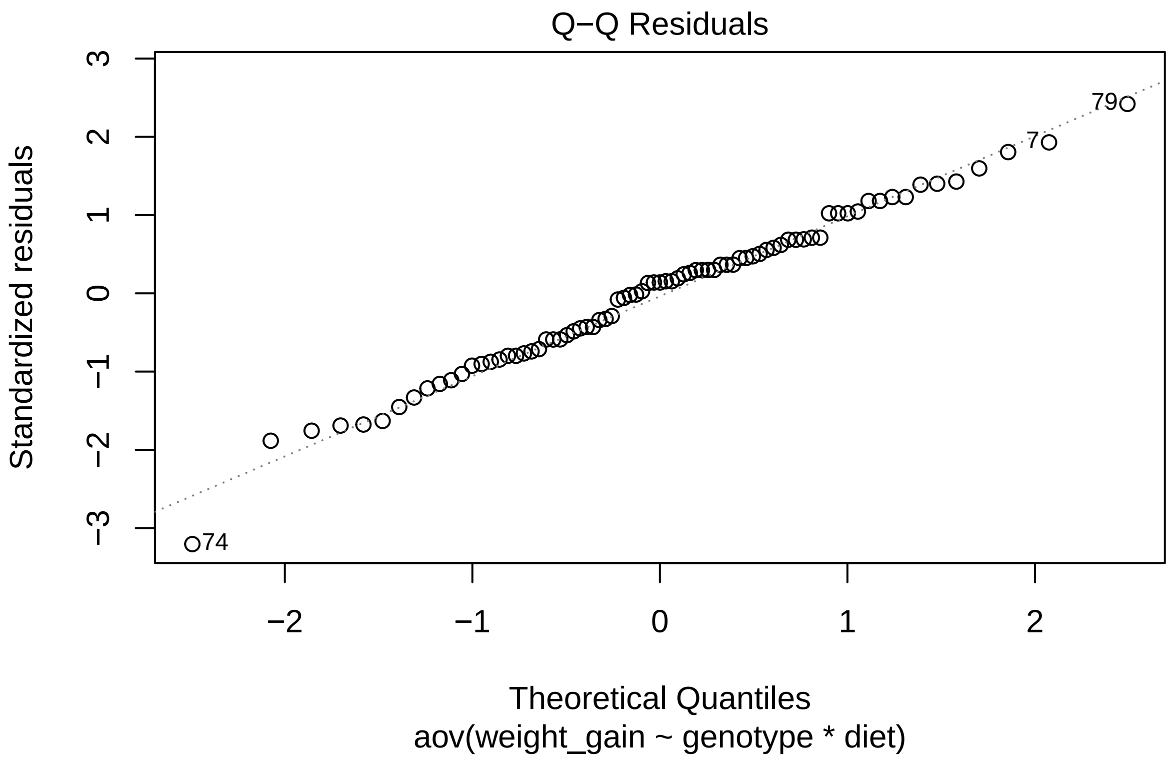
```r
#2. post-hoc analysis
TukeyHSD(model)
```

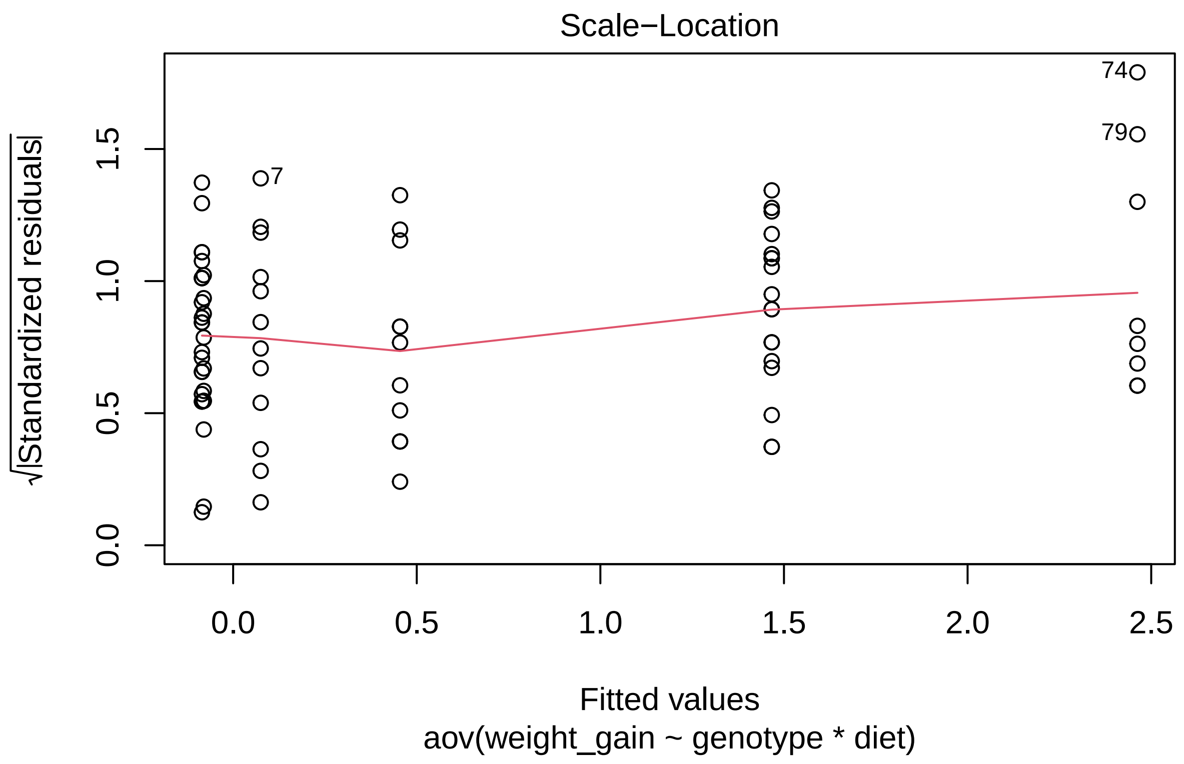
```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
## 
## Fit: aov(formula = pain ~ treatment, data = dt)
## 
## $treatment
##                     diff        lwr        upr     p adj
## drugB-drugA   -1.0519098 -1.7005747 -0.4032449 0.0008789
## placebo-drugA  0.1480734 -0.4893095  0.7854563 0.8393996
## placebo-drugB  1.1999832  0.5513183  1.8486481 0.0001610
```

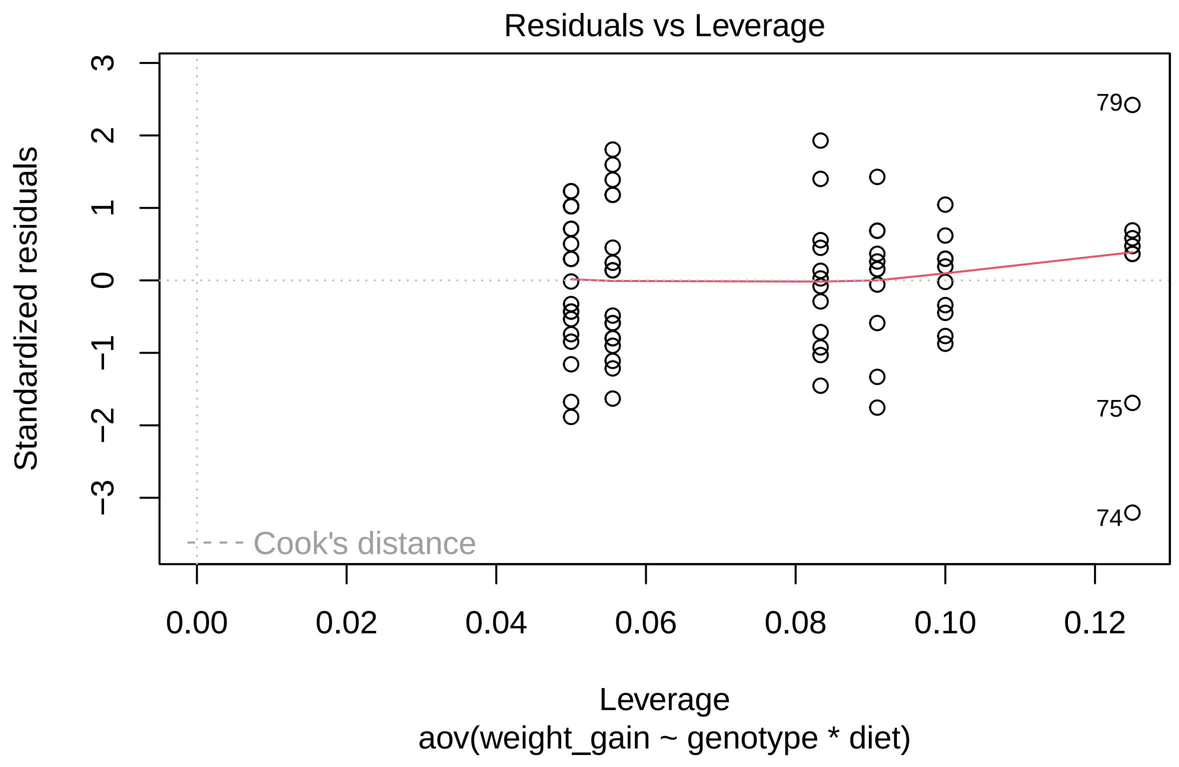
```r
#3. draw a conclusion: the drug B is significantly convinced to be effective
```

```r
###
me <- read_csv("C:/Users/10755/Desktop/Biooooo/ADS/S2/practical/Week2-mouse_experiment(1).csv")

model2 = aov(weight_gain~genotype*diet, data=me)
plot(model2)
```




Q−Q Residuals
Standardized residuals
Theoretical Quantiles
aov(weight_gain ~ genotype * diet)
−3
−2
−1
0
1
2
3
−2
−1
0
1
2
74
7
79


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
0.0
0.5
1.0
1.5
2.0
2.5
Fitted values
aov(weight_gain ~ genotype * diet)
74
79
7



```
TukeyHSD(model2)
```

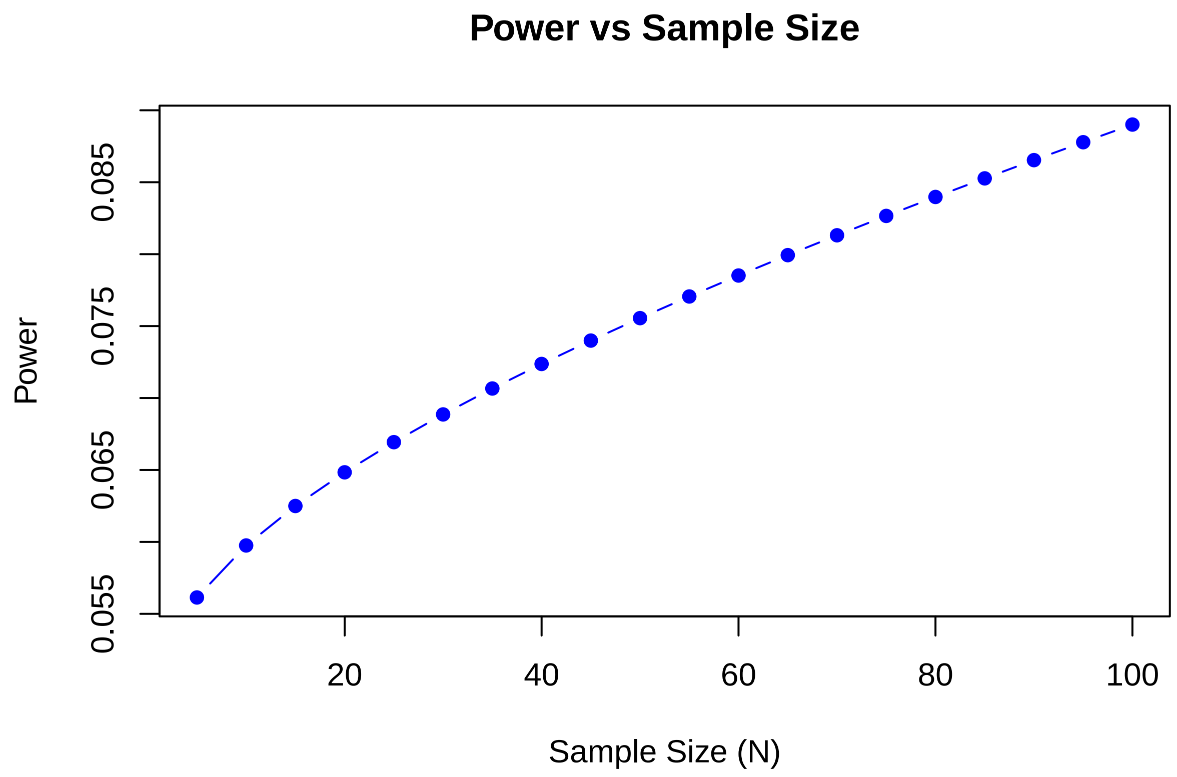
```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = weight_gain ~ genotype * diet, data = me)
##
## $genotype
##            diff        lwr      upr     p adj
## AB-AA 0.3934783 -0.2311222 1.018079 0.2935392
## BB-AA 0.7934783  0.0494578 1.537499 0.0338875
## BB-AB 0.4000000 -0.2764864 1.076486 0.3387304
##
## $diet
##                             diff       lwr      upr p adj
## unrestricted-restricted 1.432486 0.9884195 1.876552     0
##
## $`genotype:diet`
##                                  diff         lwr       upr     p adj
## AB:restricted-AA:restricted     -0.1600000 -1.21611661 0.8961166 0.9977516
## BB:restricted-AA:restricted     -0.1550000 -1.39340649 1.0834065 0.9991025
## AA:unrestricted-AA:restricted    0.3795455 -0.82776637 1.5868573 0.9401208
## AB:unrestricted-AA:restricted    1.3916667  0.31377217 2.4695612 0.0041690
## BB:unrestricted-AA:restricted    2.3875000  1.06735424 3.7076458 0.0000176
## BB:restricted-AB:restricted      0.0050000 -1.11518082 1.1251808 1.0000000
## AA:unrestricted-AB:restricted    0.5395455 -0.54615997 1.6252509 0.6937938
## AB:unrestricted-AB:restricted    1.5516667  0.61198003 2.4913533 0.0001032
## BB:unrestricted-AB:restricted    2.5475000  1.33756643 3.7574336 0.0000005
## AA:unrestricted-BB:restricted    0.5345455 -0.72918895 1.7982799 0.8167215
## AB:unrestricted-BB:restricted    1.5466667  0.40593036 2.6874030 0.0022395
## BB:unrestricted-BB:restricted    2.5425000  1.17056429 3.9144357 0.0000105
## AB:unrestricted-AA:unrestricted  1.0121212 -0.09478011 2.1190225 0.0925815
## BB:unrestricted-AA:unrestricted  2.0079545  0.66402047 3.3518886 0.0005552
## BB:unrestricted-AB:unrestricted  0.9958333 -0.23315548 2.2248221 0.1800034
```

Week4 Power and sample size

Define statistical power • The probability of rejecting the null hypothesis when it is false Explain how power relates to sample size • The bigger the sample size, the higher the power Discuss ethical issues around power and sample size • Under-powered or over-powered studied can be unethical, due to sample size-choices Use a simulation-based approach to compute power • For a given effect size (and standard deviation), power, and alpha, you can simulate to find the minimum 'n'

```r
# Set seed for reproducibility
set.seed(42)

# Define parameters
population_mean <- 175
population_sd <- 10

sample_mean <- 178  # Hypothetical mean height of college students
num_simulations <- 10000

# Initialize vector to store p-values
p_values <- numeric(num_simulations)

stimulation <- function(s_size, s_time) {
  p_values <- numeric(s_time)
  for (i in 1:s_time) {
  # Simulate data for heights of 10 male students
    heights <- rnorm(n = s_size, mean = sample_mean, sd = population_sd)
    t_test <- t.test(heights, mu = population_mean, alternative = "greater")

  # Extract the p-value and store it
    p_values[i] <- t_test$p.value
  }
  return(p_values)
}

sim1 <- stimulation(10, 10000)
cutoff <- 0.05
percentage_greater_than_cutoff1 <- sum(sim1 > cutoff)/100
sim2 <- stimulation(50, 10000)
percentage_greater_than_cutoff2 <- sum(sim2 > cutoff)/100


#3
```

```r
Ns <- seq(5, 100, by = 5)
powers <- numeric(length(Ns))

for (i in 1:length(Ns)) {
  power_result <- power.t.test(n = Ns[i], delta = (178 - 175) / 10, sd = 10,
                               sig.level = 0.05, type = "one.sample",
                               alternative = "one.sided")
  powers[i] <- power_result$power
}
plot(Ns, powers, type = "b", pch = 16, col = "blue", xlab = "Sample Size (N)",
     ylab = "Power", main = "Power vs Sample Size")
```



```r
#####2--------------------------------------------------------------------------

# Set parameters
n_simulations <- 10^5
p_value_cutoff <- 0.05
effect_size <- 10 # 10% weight loss
n <- 20 # Number of volunteers
mu <- 130 # Mean weight in the normal population
sigma <- 30 # Standard deviation of weight in the normal population

# Initialize counter
not_significant_count <- 0
```

```r
# Perform simulations
for (i in 1:n_simulations) {
  # Generate data for placebo and drug groups
  placebo_weights <- rnorm(n, mean = mu, sd = sigma)
  drug_weights <- rnorm(n, mean = mu - effect_size, sd = sigma)

  # Perform t-test
  t_test_result <- t.test(placebo_weights, drug_weights)

  # Check if p-value is greater than cutoff
  if (t_test_result$p.value > p_value_cutoff) {
    not_significant_count <- not_significant_count + 1
  }
}

# Calculate probability
probability_not_significant <- not_significant_count / n_simulations
print(probability_not_significant)
```

```
## [1] 0.8225
```

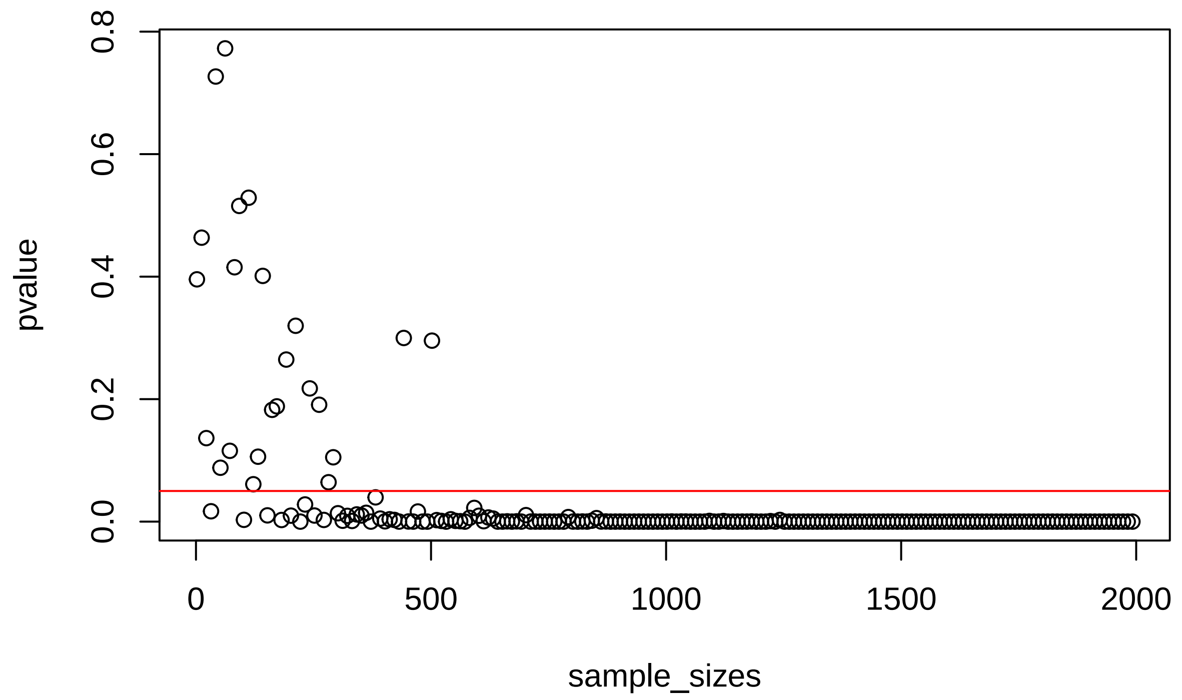
```r
power_analysis <- function(n) {
  power_result <- power.t.test(n = n, delta = effect_size/100 * sigma, sd = sigma, sig.level = 0.05, ty
  return(power_result$power)
}

# Calculate power for different sample sizes
sample_sizes <- seq(20, 200, by = 5)
powers <- sapply(sample_sizes, power_analysis)

# Find sample size that achieves desired power
required_sample_size <- sample_sizes[which.min(abs(powers - 0.8))] # Adjust 0.8 to the desired power

print(required_sample_size)
```

```
## [1] 200
```

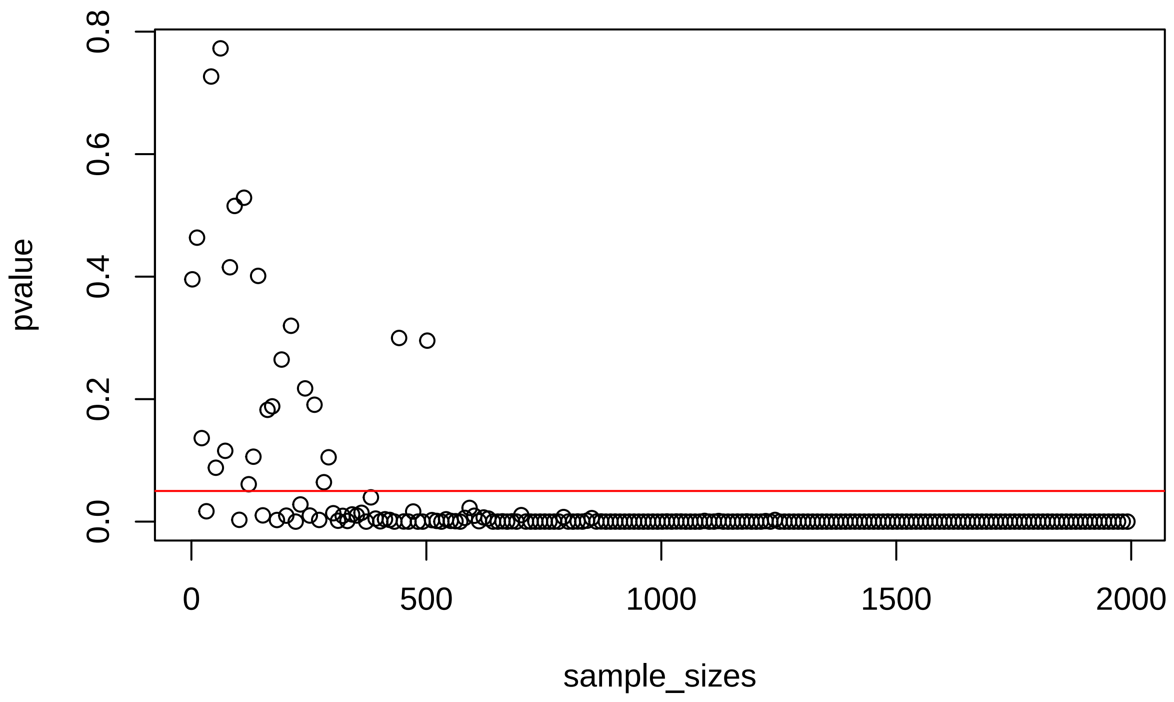
```r
power_analysis_paired <- function(n) {
  power_result <- power.t.test(n = n, delta = effect_size/100 * sigma, sd = sigma, sig.level = 0.05, ty
  return(power_result$power)
}

# Calculate power for different sample sizes
sample_sizes_paired <- seq(20, 200, by = 5)
powers_paired <- sapply(sample_sizes_paired, power_analysis_paired)

# Find sample size that achieves desired power
required_sample_size_paired <- sample_sizes_paired[which.min(abs(powers_paired - 0.8))] # Adjust 0.8 to

print(required_sample_size_paired)
```

```
## [1] 200
```

```r
# Define function to calculate required sample size for given power or p-value cutoff
calculate_required_sample_size <- function(power_or_cutoff) {
  power_analysis <- function(n) {
    power_result <- power.t.test(n = n, delta = effect_size/100 * sigma, sd = sigma, sig.level = power_o
    return(power_result$power)
  }

  # Calculate power for different sample sizes
  sample_sizes <- seq(20, 200, by = 5)
  powers <- sapply(sample_sizes, power_analysis)

  # Find sample size that achieves desired power
  required_sample_size <- sample_sizes[which.min(abs(powers - 0.8))] # Adjust 0.8 to the desired power

  return(required_sample_size)
}

set.seed(13)
sample_sizes = seq(2, 2000, by = 10)
a = list() #initialize a, why I use list to initialize a here?
b = list() #initialize b
pvalue = rep(0, length(sample_sizes))
for (i in 1:length(sample_sizes)) {
a[[i]] = rnorm(sample_sizes[i], mean = 10, sd = 5)
b[[i]] = rnorm(sample_sizes[i], mean = 11, sd = 5)
pvalue[i] = t.test(a[[i]], b[[i]])$p.value
}
plot(x = sample_sizes, y = pvalue)
abline(h = 0.05, col = "red")
```



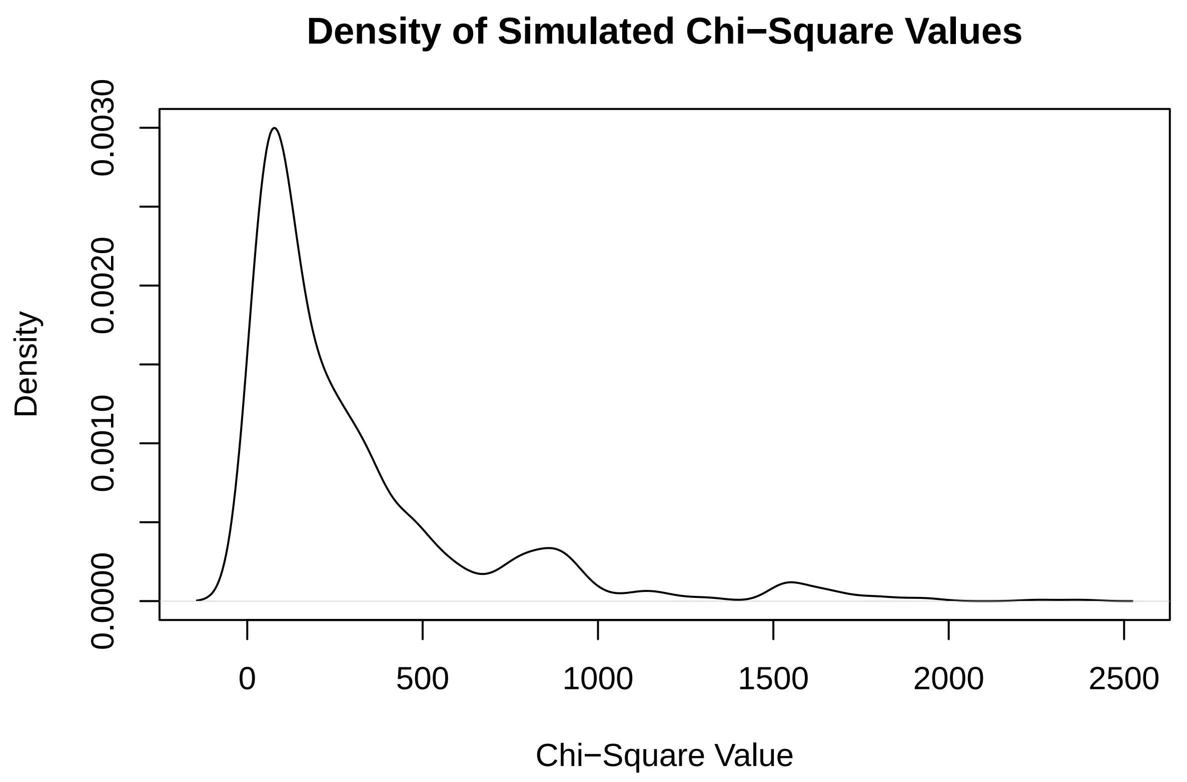
```r
set.seed(13)
sample_sizes = seq(2, 2000, by = 10)
a = list() #initialize a, why I use list to initialize a here?
b = list() #initialize b
pvalue = rep(0, length(sample_sizes))
for (i in 1:length(sample_sizes)) {
a[[i]] = rnorm(sample_sizes[i], mean = 10, sd = 5)
b[[i]] = rnorm(sample_sizes[i], mean = 11, sd = 5)
pvalue[i] = t.test(a[[i]], b[[i]])$p.value
}
plot(x = sample_sizes, y = pvalue)
abline(h = 0.05, col = "red")
```



Week5 Categorical Data

• Goodness-of-fit tests: The chi-square goodness-of-fit test is used to determine whether a sample of data comes from a population with a specific distribution. For example, it can test whether observed frequencies differ significantly from expected frequencies. • Test for independence: In a contingency table, the chi-square test for independence can determine whether two categorical variables are independent of each other.

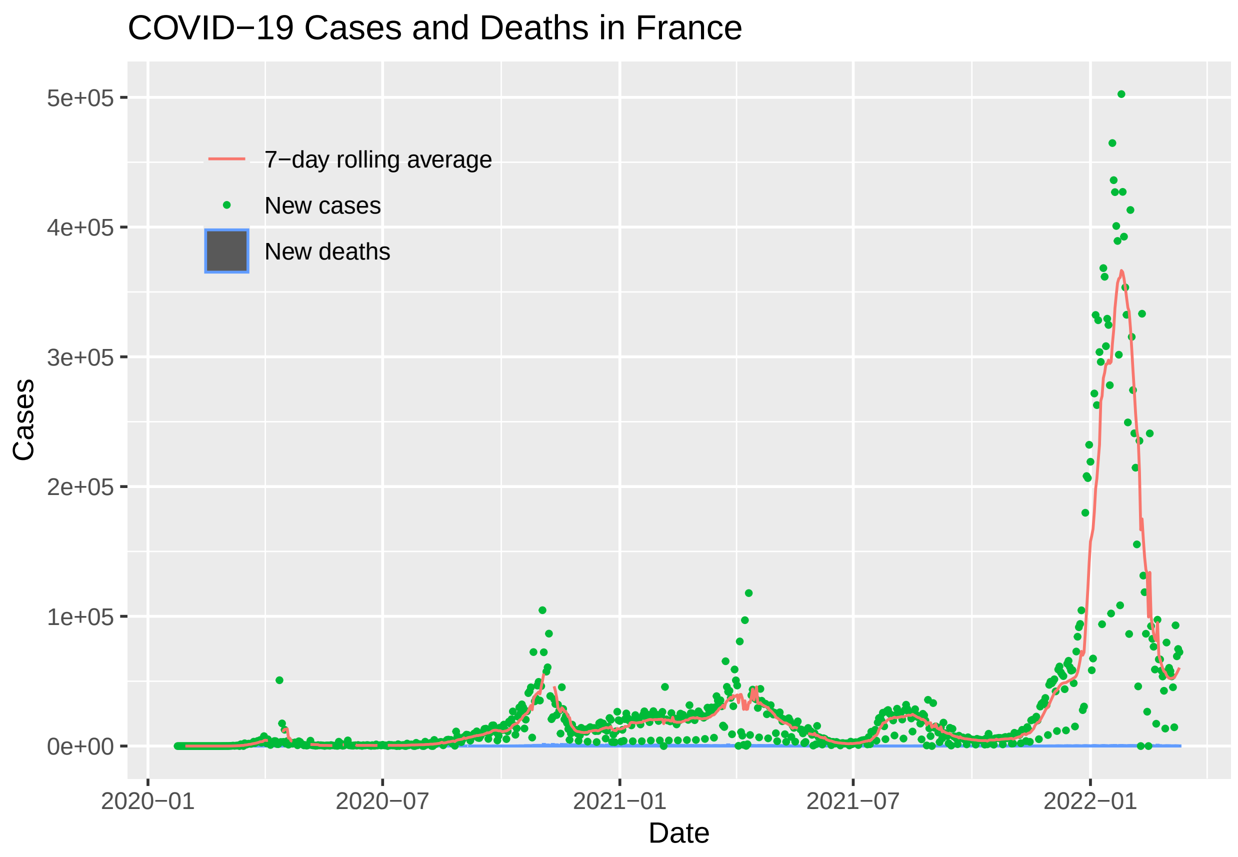
```
### 1. ---------------------------------------------------------------------
Poll_seasons <- data.frame(Spring = 40, Summer = 30, Autumn = 18, Winter = 28)

# Expected equal preferences
expected_preferences <- sum(Poll_seasons) * 0.25

# Function to calculate chi-square
calculate_chi_square <- function(observed, expected) {
  chi_square <- sum((observed - expected)^2 / expected)
  return(chi_square)
}

# Function to simulate chi-square values
simulate_chi_square <- function(n_simulations, observed, expected) {
  chi_square_values <- replicate(n_simulations, {
    sample_expected <- sample(expected, size = sum(observed), replace = TRUE)
    chi_square <- calculate_chi_square(observed, sample_expected)
    return(chi_square)
  })
  return(chi_square_values)
}

# Simulate chi-square values
simulated_chi_square <- simulate_chi_square(1000, Poll_seasons, expected_preferences)
# Calculate observed chi-square
observed_chi_square <- calculate_chi_square(Poll_seasons, expected_preferences)

# Calculate p-value
p_value <- mean(simulated_chi_square >= observed_chi_square)

# Plot density of simulated chi-square values
plot(density(simulated_chi_square), main = "Density of Simulated Chi-Square Values", xlab = "Chi-Square
```



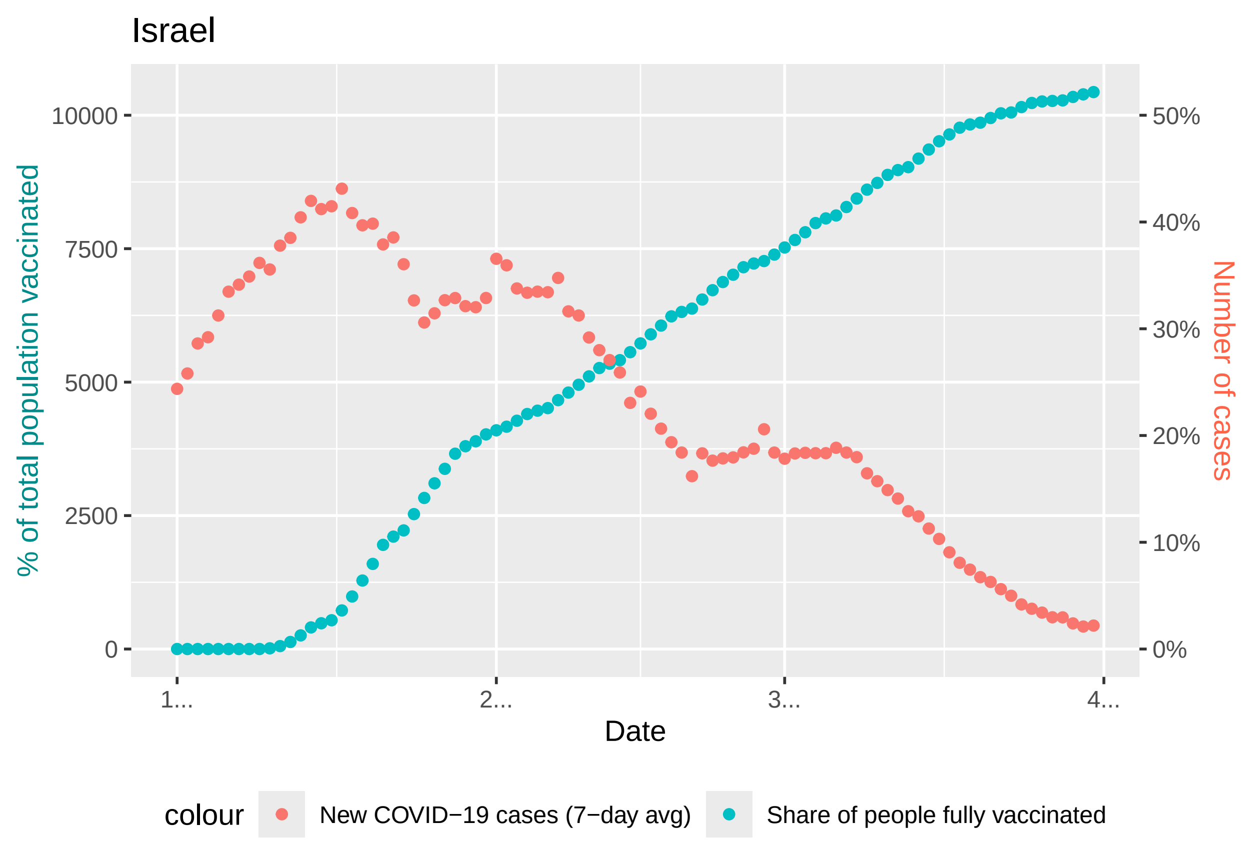
##Week6 Correlation and Linear Regressions A simple linear regression describes the association between an independent variable and a dependent one

```
# Function that generates a vector with x-days rolling average for a given country and a given variable
# USAGE: generate_rolling_avg(subcovid, "France", "new_cases", 7)
# one_country = "France"; one_variable = "new_cases"; days = 7
generate_rolling_avg <- function(subcovid, one_country, one_variable, days = 7){
  range_days_in_one_country <- range(subcovid$date[which(subcovid$location == one_country)])
  # Identifying the dates present in subcovid
  dates_included <- seq(range_days_in_one_country[1], range_days_in_one_country[2],
                        by = "days")
```

```r
  # Calculating 7-day rolling mean
  variable_means <- sapply(dates_included[-(1:6)], function(end_of_the_week){
    # end_of_the_week <- dates_included[7]
    x_days_cases <- sapply(-6:0, function(y){
      # y <- -6
      subcovid[which(subcovid$location == one_country & subcovid$date == (end_of_the_week + y)),
               one_variable]
    })
    mean(x_days_cases)
  })
  variable_means_df <- data.frame(Dates = dates_included[-(1:6)],
                                  new_variable_avg = variable_means)
}


trying <- try(covid <- read.csv("owid-covid-data.txt", header = TRUE))
if(is(trying, "try-error")){
  download.file(url = paste0("https://github.com/hugocarlos/covid-19-data/blob/master/",
                             "public/data/owid-covid-data.csv?raw=true"),
                destfile = "owid-covid-data.txt")
  covid <- read.csv("owid-covid-data.txt", header = TRUE)
}
# Selecting some columns
subcovid <- covid %>%
  select(iso_code, location, date, new_cases, new_deaths, new_cases_per_million,
         total_cases_per_million, new_vaccinations, people_fully_vaccinated,
         aged_65_older, aged_70_older, gdp_per_capita, extreme_poverty,
         cardiovasc_death_rate, diabetes_prevalence, life_expectancy,
         human_development_index)

# To date format
subcovid$date <- as.Date(subcovid$date)

# Setting one country
one_country <- "France"

# Calculating the 7-days window average for new cases of COVID-19
cases_means_df <- generate_rolling_avg(subcovid, one_country, "new_cases", 7)

# Merging cases_means_df to subcovid
subcovid$new_cases_avg <- NA
for(i in 1:nrow(cases_means_df)){
  # i <- 1
  subcovid$new_cases_avg[which(subcovid$location == one_country &
                               subcovid$date == cases_means_df$Dates[i])] <-
    cases_means_df$new_variable_avg[i]
}

# Plot
ggplot() +
  geom_bar(stat = "identity",
           aes(x = subcovid$date[which(subcovid$location == one_country)],
               y = subcovid$new_deaths[which(subcovid$location == one_country)],
               colour = "New deaths")) +
  geom_point(aes(x = subcovid$date[which(subcovid$location == one_country)],
                y = subcovid$new_cases[which(subcovid$location == one_country)],
                colour = "New cases"), size = 0.7) +
  geom_line(aes(x = subcovid$date[which(subcovid$location == one_country)],
               y = subcovid$new_cases_avg[which(subcovid$location == one_country)],
               colour = "7-day rolling average")) +
  labs(x = "Date", y = "Cases") +
  ggtitle(paste0("COVID-19 Cases and Deaths in ", one_country)) +
  theme(legend.position = c(0.2, 0.8),
        legend.title = element_blank(),
        legend.background = element_blank(),
        legend.key = element_rect(fill = NULL, color = NULL))
```



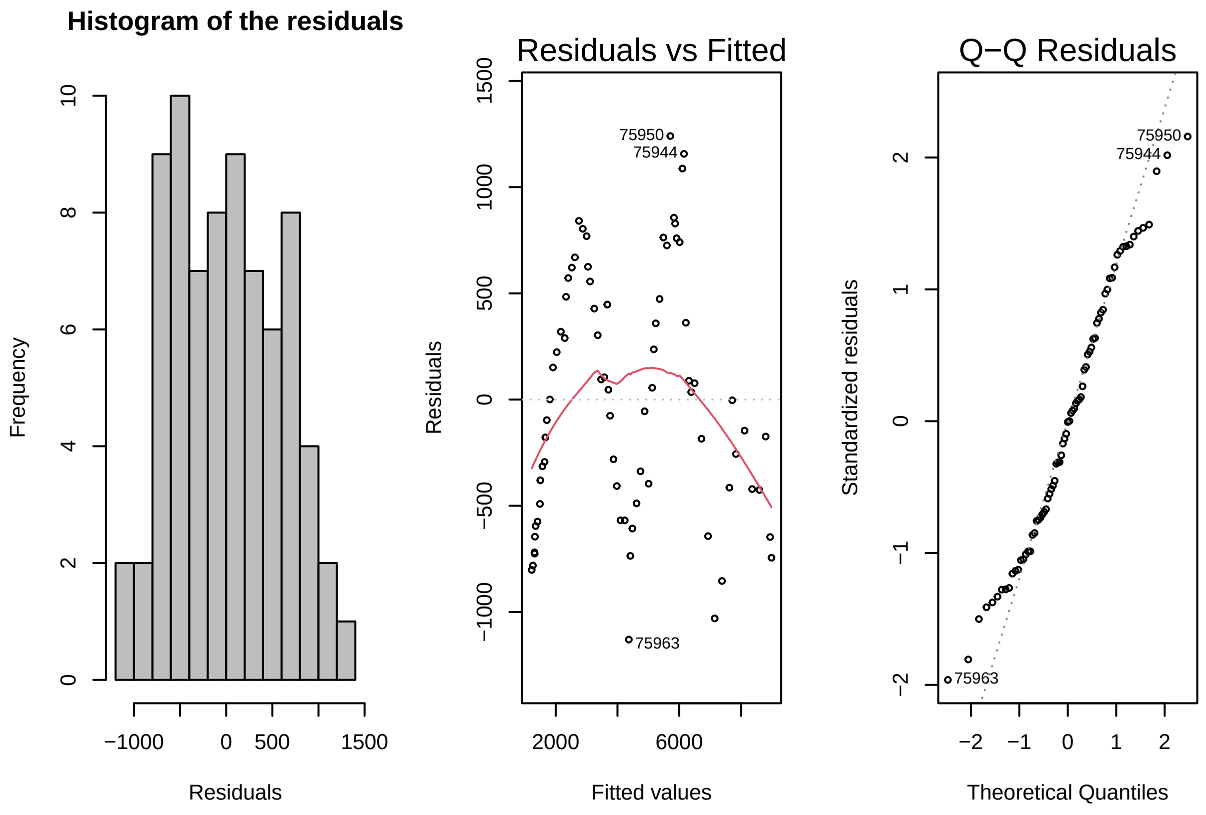
```
# Finding all the days from 2021, as they probably contain vaccination data
dates_from_2021 <- seq(as.Date("2021-01-01"), as.Date("2021-12-31"), by = "days")

one_country <- "Israel"
# Re-calculating the vector with 7-day rolling average of new COVID-19 cases
cases_means_df <- generate_rolling_avg(subcovid, one_country, "new_cases", 7)

# Attaching the file with the population
trying <- try(population <- read.csv("WBpopulation.csv", header = TRUE, sep = "\t"))
if(is(trying, "try-error")){
  download.file(url = paste0("https://raw.githubusercontent.com/hugocarlos/public_scripts/",
                              "master/teaching/WBpopulation.csv"),
                destfile = "WBpopulation.csv")
  population <- read.csv("WBpopulation.csv", header = TRUE, sep = "\t")
}

# Calculating the percentage of the population fully vaccinated
Israel_population <- population$X2020[which(population$Country.Name == one_country)]
subcovid$share_fully_vaccinated <- subcovid$people_fully_vaccinated * 100 / Israel_population

# Merging cases_means_df to subcovid
subcovid$new_cases_avg <- NA
for(i in 1:nrow(cases_means_df)){
  # i <- 1
  subcovid$new_cases_avg[which(subcovid$location == one_country &
                                 subcovid$date == cases_means_df$Dates[i])] <-
    cases_means_df$new_variable_avg[i]
}

subcovid %>%
  filter(location == one_country) %>%
  filter(date >= dates_from_2021[1] & date < as.Date("2021-04-01")) %>%
  ggplot() +
  geom_point(aes(x = date, y = share_fully_vaccinated * 200,
                 colour = "Share of people fully vaccinated")) +
  geom_point(aes(x = date, y = new_cases_avg, colour = "New COVID-19 cases (7-day avg)")) +
  scale_y_continuous(name = "% of total population vaccinated",
                     sec.axis = sec_axis(~./200, name = "Number of cases",
                                         labels = function(b){
                                           paste0(b, "%")
                                         })) +
  xlab("Date") +
  theme(axis.title.y = element_text(color = "cyan4"),
        axis.title.y.right = element_text(color = "tomato"),
        legend.position = "bottom") +
  ggtitle(paste0(one_country))
```



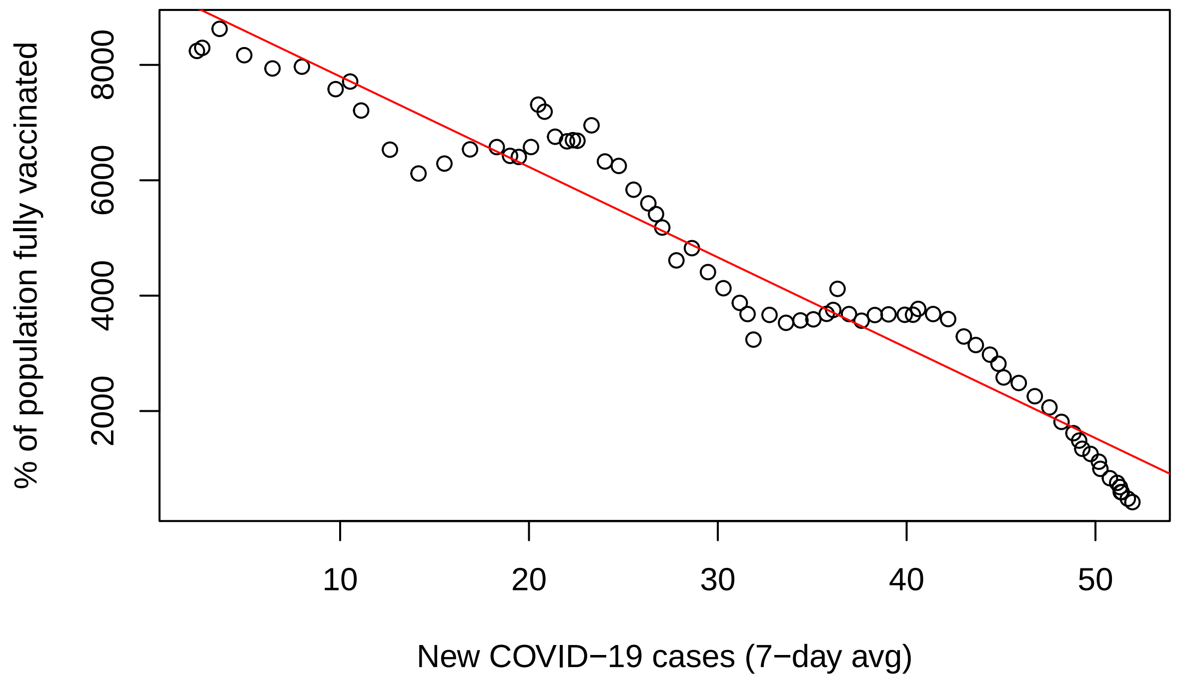
```r
covid_onecountry <- subcovid[which(subcovid$location == "Israel" &
                                    subcovid$date >= as.Date("2021-01-15") &
                                    subcovid$date < as.Date("2021-03-31")), ]
# Calculating the Correlation Coefficient
cor(covid_onecountry$new_cases_avg,
    covid_onecountry$share_fully_vaccinated,
    use = "complete.obs")
```

```
## [1] -0.9692676
```

```r
lm_Israel <- lm(new_cases_avg ~ share_fully_vaccinated, covid_onecountry)
par(mfrow = c(1, 3))
hist(residuals(lm_Israel), breaks = 15, col = "gray",
     main = "Histogram of the residuals", xlab = "Residuals", cex = 0.6)
plot(lm_Israel, which = c(1, 2), cex = 0.6)
```



```r
par(mfrow = c(1, 1))
plot(x = covid_onecountry$share_fully_vaccinated, y = covid_onecountry$new_cases_avg,
     xlab = "New COVID-19 cases (7-day avg)",
     ylab = "% of population fully vaccinated")
abline(lm_Israel, col = "red")
```



##Week7 Conditional Probabilities Bayes' theorem P(A) × P(B|A) = P(B) × P(A|B) P(A|B) = P(A) × P(B|A)/P(B)

```
# Q1:
set.seed(123)

num_employees <- 1000
thief_probability <- 0.10
detector_accuracy <- 0.80
num_simulations <- 1000

run_simulation <- function() {
  true_status <- sample(c(0, 1), num_employees, replace = TRUE, prob = c(1 - thief_probability, thief_pr
  lie_detector_results <- sapply(true_status, function(status) {
    if (status == 1) {  #
      sample(c(0, 1), 1, prob = c(1 - detector_accuracy, detector_accuracy))
    } else {  #
      sample(c(0, 1), 1, prob = c(detector_accuracy, 1 - detector_accuracy))
    }
  })
  num_liars <- sum(lie_detector_results)
  if (num_liars == 50) {
    num_thieves_among_liars <- sum(true_status[lie_detector_results == 1])
    return(num_thieves_among_liars)
  }
}
```

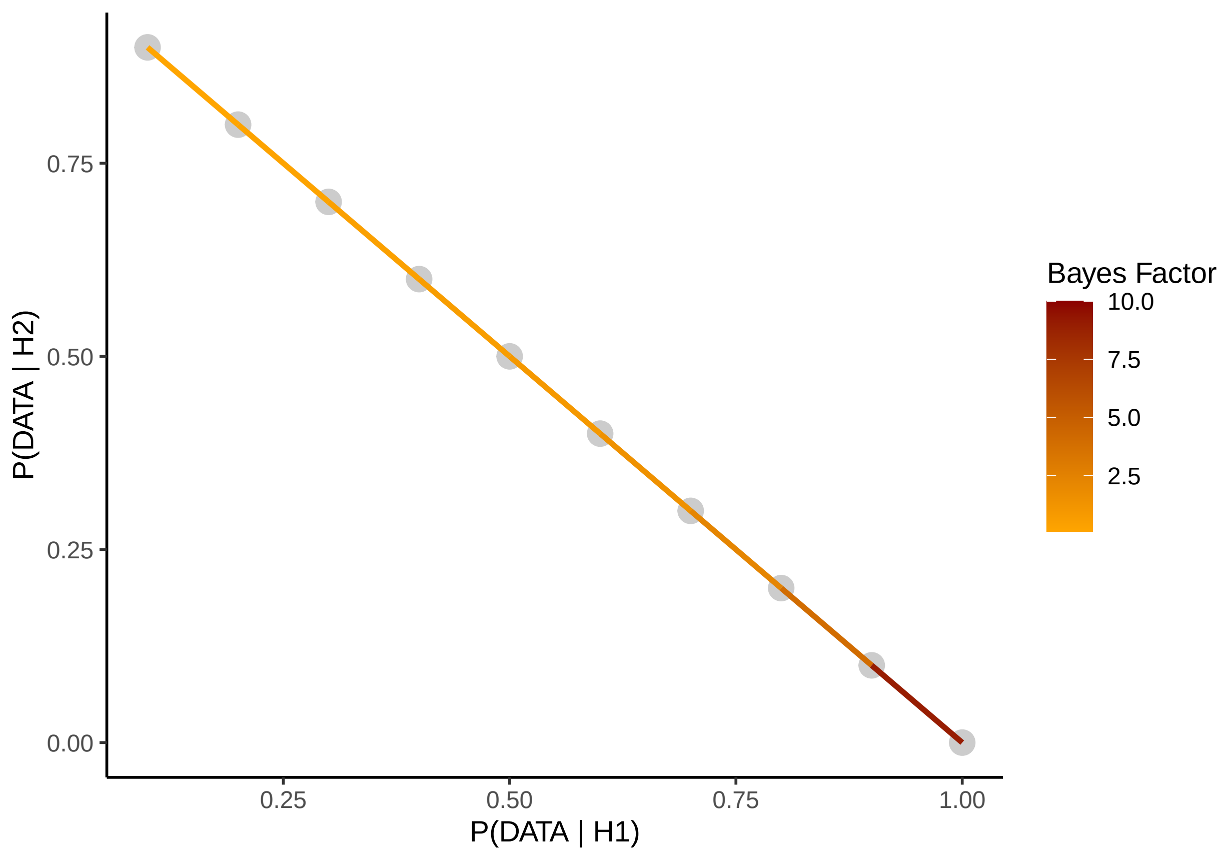
```r
thieves_among_liars <- replicate(num_simulations, run_simulation())
sequenceWanted <- c("H", "T", "T", "H") %>% paste(collapse = "")
simulate_HTTH <- function(sequenceWanted) {
  STEP <- 1
  sequencecurrent <- c(sample(x = c("H", "T"), size = 1, prob = c(0.5, 0.5)))
  #print(sequencecurrent) # Uncomment to see the progress

  while(sequencecurrent != sequenceWanted){
    if(nchar(sequencecurrent) == 1 & sequencecurrent != "H"){# Proofread the first toss
      # And wait till I get H
      while(sequencecurrent != "H"){
        STEP <- STEP + 1
        sequencecurrent <- sample(x = c("H", "T"), size = 1, prob = c(0.5, 0.5))
        #print(sequencecurrent) # Uncomment to see the progress
      }
    } else { # Otherwise, go to the second step
      STEP <- STEP + 1
      sequencecurrent <- c(sequencecurrent,
                           sample(x = c("H", "T"),
                                  size = 1,
                                  prob = c(0.5, 0.5))) %>%
        paste(collapse = "")
      #print(sequencecurrent) # Uncomment to see the progress
    }
    if(sequencecurrent != "HT"){ # Check the second step and proofread it
      sequencecurrent <- substr(sequencecurrent,
                                nchar(sequencecurrent),
                                nchar(sequencecurrent))
      #print(sequencecurrent) # Uncomment to see the progress
    } else { # Otherwise, proceed to the third step
      STEP <- STEP + 1
      sequencecurrent <- c(sequencecurrent,
                           sample(x = c("H", "T"),
                                  size = 1,
                                  prob = c(0.5, 0.5))) %>%
        paste(collapse = "")
      #print(sequencecurrent) # Uncomment to see the progress
    }
    if(nchar(sequencecurrent) == 3 & sequencecurrent != "HTT"){# proofread the third step
      sequencecurrent <- substr(sequencecurrent,
                                nchar(sequencecurrent),
                                nchar(sequencecurrent))
      #print(sequencecurrent) # Uncomment to see the progress
    } else { # otherwise, go to the fourth step
      STEP <- STEP + 1
      sequencecurrent <- c(sequencecurrent,
                           sample(x = c("H", "T"),
                                  size = 1,
                                  prob = c(0.5, 0.5))) %>%
        paste(collapse = "")
      #print(sequencecurrent) # Uncomment to see the progress
    }
    if(sequencecurrent != "HTTH"){ # restart the whole chain
```

```r
      sequencecurrent <- substr(sequencecurrent,
                                nchar(sequencecurrent),
                                nchar(sequencecurrent))
      STEP <- STEP + 1
      #print(sequencecurrent) # Uncomment to see the progress
    } else { # otherwise, finish the whole sequence
      #print(sequencecurrent) # Uncomment to see the progress
      #print("Sequence complete") # Uncomment to see the progress
    }
  }
  return(STEP)
}
# Run the simulation 1000 times and store the steps in a list
steps_list <- numeric(1000)
for (i in 1:1000) {
  steps_list[i] <- simulate_HTTH("HTTH")
}

sum(steps_list)/1000
```

```
## [1] 54.775
```

##Week8 Bayesian statistics is quite different. Include probabilistic methods and reasoning to the parameters: want to estimate the chance something is true and the extent of our belief in the estimate. Can be updated with new knowledge, just like scientists operate normally. Truly explicit/scrutable - did you notice the lack of assumptions in today's lecture?

```r
bfs = data.frame('H1' = seq(0.1, 1, 0.1),
                 'H2' = 1 - seq(0.1, 1, 0.1)) %>%
  mutate('bf' = H1 / H2) %>%
  mutate('bf' = ifelse(bf == 'Inf', 10, bf))
ggplot(bfs, aes(H1, H2)) +
  theme_classic() +
  geom_point(size = 4, alpha = 0.2) +
  geom_line(aes(colour = bf), size = 1) +
  labs(x = 'P(DATA | H1)', y = 'P(DATA | H2)') +
  scale_color_gradient(low = 'orange',
                       high = 'darkred',
                       name = 'Bayes Factor')
```



```
## 1 vs 5 sixes per die
Probabilities_sixes <- c(1 / 6, 2 / 6, 3 / 6, 4 / 6, 5 / 6)
P_givenData_Expectation <- dbinom(x = 7, size = 20, prob = Probabilities_sixes)

dice <- cbind(Probabilities_sixes, P_givenData_Expectation)
P_hypotheses <- rep(0.2, 5) # As far as we are not sure in any hypothesis, we set it to 0.2
P_givenData <- sum(P_hypotheses * dice[, 2])
Probabilities_sixes # Statistical models for our data == chances of success
```

```
## [1] 0.1666667 0.3333333 0.5000000 0.6666667 0.8333333
```

```
P_hypotheses # Our prior beliefs in each hypothesis
```

```
## [1] 0.2 0.2 0.2 0.2 0.2
```

```
P_givenData # The probability of getting 7 sixes out of 20 trials according to each hypothesis
```

```
## [1] 0.05695742
```

```
P_givenData_Expectation # The likelihood of each hypothesis
```

```
## [1] 2.588206e-02 1.821288e-01 7.392883e-02 2.845762e-03 1.656452e-06
```

```
(P_givenData_Expectation[5] * P_hypotheses[5]) / P_givenData # The probability of H1 being correct:
```

```
## [1] 5.816456e-06
```

##week 9&10 supervised

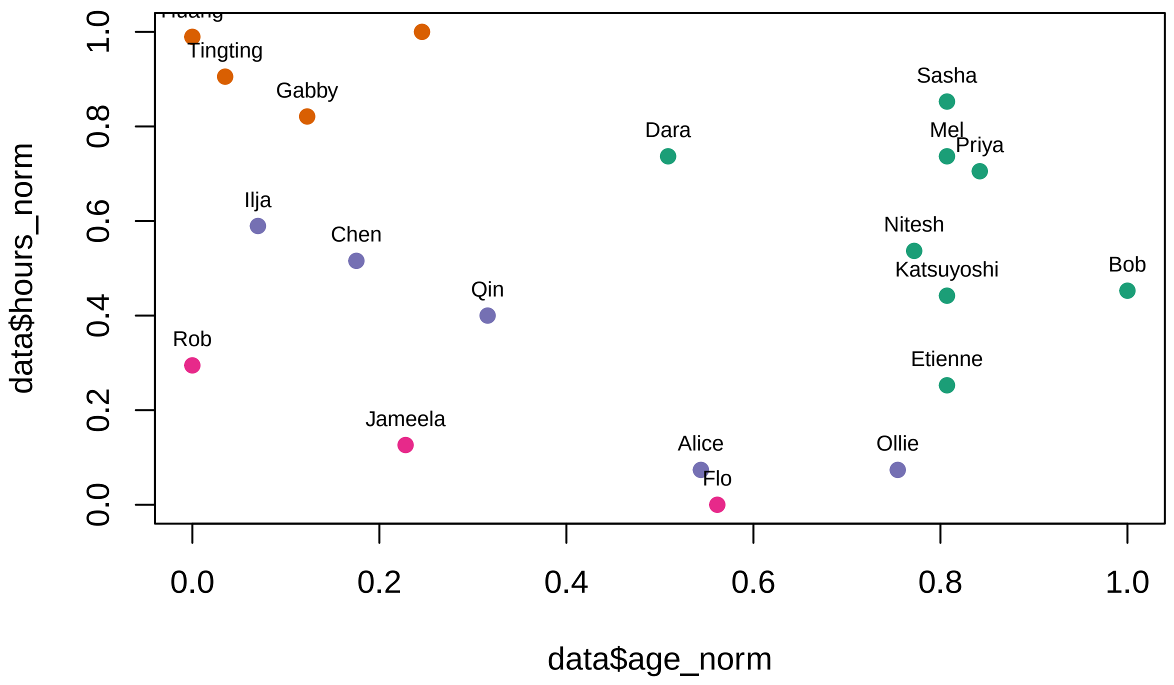
```
# read in data
mnist_raw <- read_csv("../practical/Week9-mnist_train.csv",col_names = FALSE)
mnist_raw[1:10, 1:10]

pixels_gathered <- mnist_raw %>%
  head(1000) %>%
  rename(label = X1) %>%
  mutate(instance = row_number()) %>%
  gather(pixel, value, -label, -instance) %>%
  tidyr::extract(pixel, "pixel", "(\\d+)", convert = TRUE) %>%
  mutate(pixel = pixel - 2,
         x = pixel %% 28,
         y = 28 - pixel %/% 28)

# We now have one row for each pixel in each image. This is a useful format
# because it lets us visualize the data along the way. For example, we can
# visualize the first 12 instances with a couple lines of ggplot2.
theme_set(theme_light())
#pixels_gathered %>%
#  filter(instance <= 12) %>%
#  ggplot(aes(x, y, fill = value)) +
#  geom_tile() +
#  facet_wrap(~ instance + label)

# Compute features that substantially reduce dimensionality from 784,
# while preserving essential information

# Based on the features you selected, create and compute a features dataframe,
# where rows are examples and variables/columns are different features (e.g. 56 of them)

features = data.frame(label = mnist_raw$X1[1:1000])

for (i in 1:56)
  features = cbind(features, fi = c(1:1000)*0)

for (i in 1:28)
  # loop over 28 rows and 28 columns
  for (j in 1:1000)
    # compute row & column means for each digit example using pixels_gathered
  { features[j,i+1] = mean(pixels_gathered$value[pixels_gathered$instance==j & pixels_gathered$y==i]);
  # first 28 features: row means (each row has fixed y)
  features[j,i+29] = mean(pixels_gathered$value[pixels_gathered$instance==j & pixels_gathered$x==i-1]);
  # next 28 features: column means (each row has fixed x)
  }

# First let's compute means for each label and feature:
fstats = matrix(1:560, nrow = 10, ncol = 56)
```

```r
for (i in 1:10)
  for (j in 1:56)
    fstats[i,j] = mean(features[features$label==i-1,j+1]);

par(mfrow=c(5,2))
for (i in 1:10)
{ plot(fstats[i,], ylab = "Value", xlab = "Feature index")
  title(paste("Feature values for digit", toString(i-1)))
}

#supervised machine learning
library(nnet)
# separat the data into training group and validation gorup
rows <- sample(1:1000, 700)
train_labels <- features[rows, 1]
valid_labels <- features[-rows, 1]
train_data <- features[rows, -1]
valid_data <- features[-rows, -1]
# normalizing the data
train_data = train_data/255
valid_data = valid_data/255
# generate a N*10 matrix
train_labels_matrix = class.ind(train_labels)
head(train_labels_matrix)

#now conduct nueral network=
nn = nnet(train_data, train_labels_matrix, size = 4, softmax = TRUE)
pred_train = predict(nn, train_data, type="class")
pred_valid = predict(nn, valid_data, type="class")

# Now experiment training your model with a different number of parameters
# (e.g. 1 to 12 hidden layer neurons).
# initialising vectors for storing training and validation performance
trainerrs = 1:12
validerrs = 1:12
# training the networks for each number of hidden layer neurons
for (i in 1:12){
  nn = nnet(train_data, train_labels_matrix, size = i, softmax = TRUE)
  pred_train = predict(nn, train_data, type="class")
  pred_valid = predict(nn, valid_data, type="class")
  trainerrs[i] = mean(pred_train == train_labels)
  validerrs[i] = mean(pred_valid == valid_labels)
}

plot(trainerrs, xlab='Number of hidden layer neurons', ylab='Classification
performance')
lines(validerrs)
legend ("bottomright", c("training set", "validation set"), pch="o-")
title("Performance of a default neural network with 100 iterations for digit
classification")
```

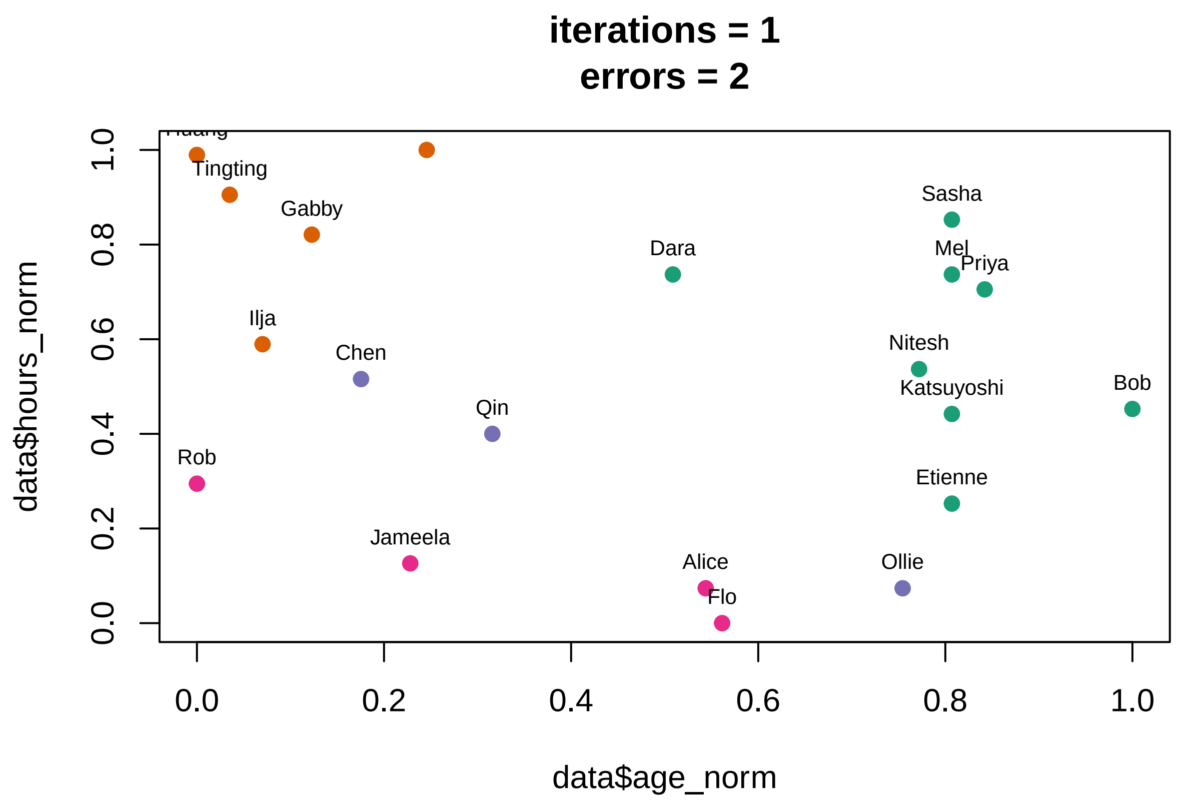
##week13

```r
colours <- brewer.pal(4, "Dark2")



data = read.csv("../practical/Week13-guest.csv", stringsAsFactors = T)
p <- ggplot(data, aes(x = age_norm, y = hours_norm)) +
  geom_point(color = "blue") +
  geom_text(aes(label = names), vjust = -1, hjust = 0.5) +
  labs(title = "Distribution of Age vs. Travel Hours",
       x = "Normalized Age",
       y = "Normalized Travel Hours") +
  theme_minimal()


# 1step k-means
# reorgnize the data frame
data$cluster <- NA
data$distance_1 <- NA
data$distance_2 <- NA
data$distance_3 <- NA
data$distance_4 <- NA
data$colour <- "black"


# assign 4 initial centroid
set.seed(123)
initial_centroids <- sample(data$names, 4)
centroids <- subset(data, names %in% initial_centroids)
p <- p + geom_point(data = centroids, aes(x = age_norm, y = hours_norm),
                    shape = 1, size = 5, colour = "red")


# that cluster
for (a in 1:nrow(data)) {
  data_age <- data[a, 2]
  data_hour <- data[a, 3]
  for (b in 1:length(initial_centroids)) {
    centroid_age <- subset(data$age_norm, data$names == initial_centroids[b])
    centroid_hour <- subset(data$hours_norm, data$names ==
                              initial_centroids[b])
    distance <- dist(matrix(c(data_age, centroid_age, data_hour,
                              centroid_hour), ncol = 2))
    data[a, (b + 4)] <- distance
  }
  cluster_name <- which(data[a, 5:8] == min(data[a, 5:8]))
  data[a, 4] <- cluster_name
}

# add the colours based on the cluster
for (a in 1:4) {
  data$colour[data$cluster == a] <- colours[a]
}
```

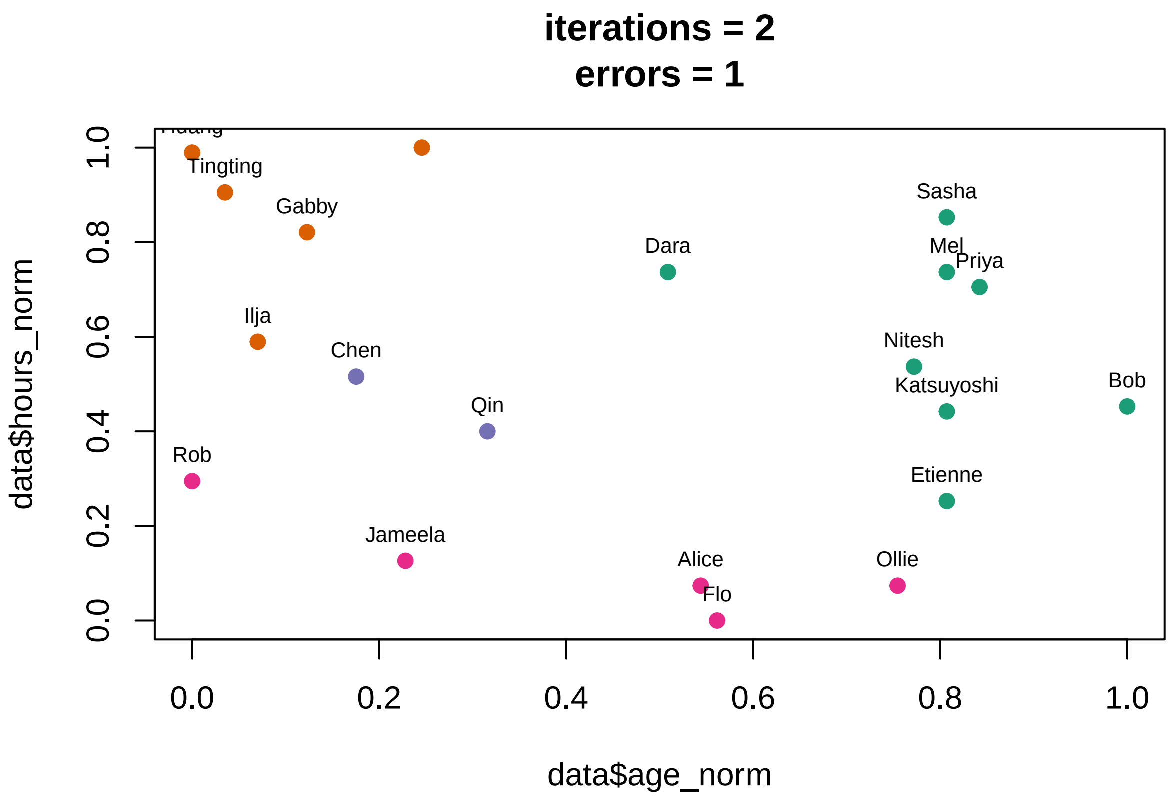
```
plot(data$hours_norm ~ data$age_norm, col = data$colour, pch = 19)
text(data$age_norm, data$hours_norm, labels = data$name, cex = 0.7,
     pos = 3)
```

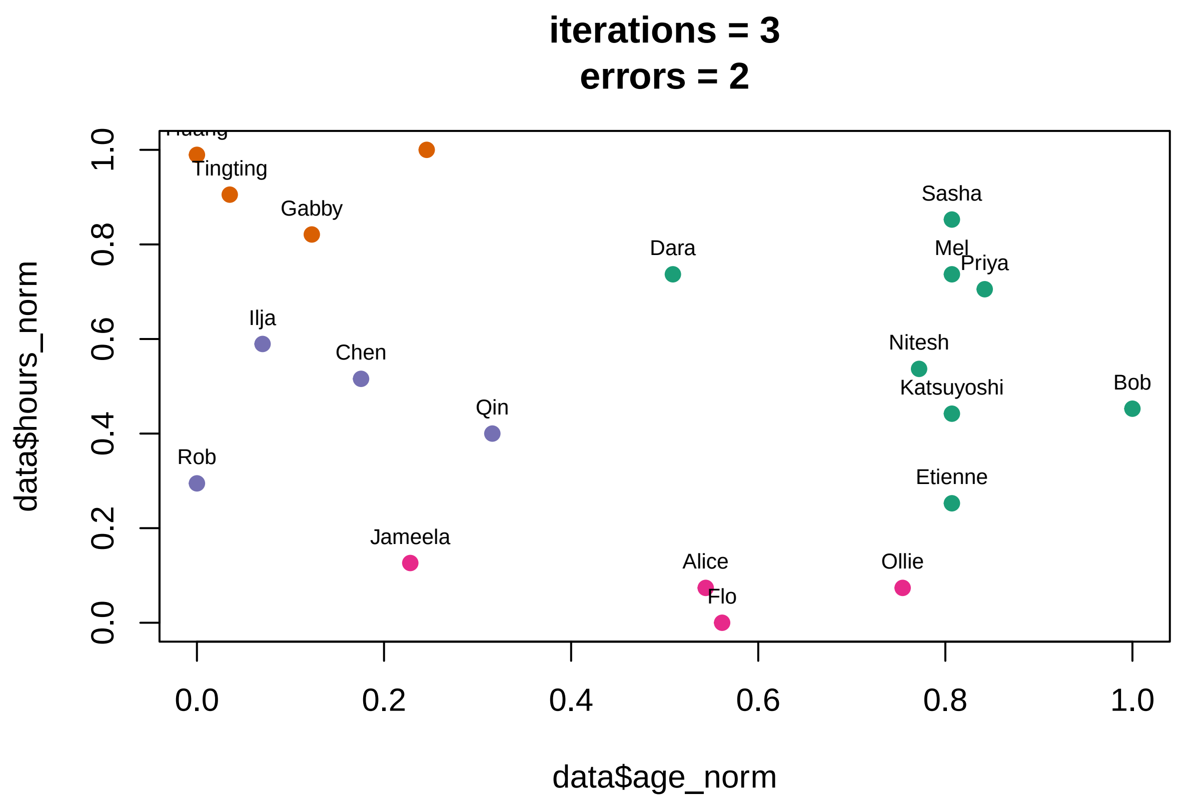


```
##Repeat
# calculate new clusters for each iterations keep the old
# clusters in old_cluster
# set the number of errors to 1 to initialise the loop
errors <- 1
# keep a record of the number of iterations of the loop
iterations <- 0
while (errors > 0) {
  # store the last round of cluster assignments
  data$old_cluster <- data$cluster
  # reinitialise everything
  data$distance_1 <- NA
  data$distance_2 <- NA
  data$distance_3 <- NA
  data$distance_4 <- NA
  data$cluster <- NA
  data$colour <- NA
  for (a in 1:nrow(data)) {
    data_age <- data[a, 2]
    data_hour <- data[a, 3]
    for (b in 1:4) {
      # calculate the new mean position for the
      # cluster
      cluster_data <- subset(data, data$old_cluster ==
                               b)
      centroid_age <- mean(cluster_data$age_norm)
      centroid_hour <- mean(cluster_data$hours_norm)
      # recalculate the distances
      distance <- dist(matrix(c(data_age, centroid_age,
                                data_hour, centroid_hour), ncol = 2))
      data[a, (b + 4)] <- distance
    }
    # cluster reassignment
    cluster_name <- which(data[a, 5:8] == min(data[a, 5:8]))
    data[a, 4] <- cluster_name
  }
  # calculate the error rates note that this is not the
  # number of datapoints which change cluster, but it
  # will be zero if nothing changes
  errors <- sum(abs(data$cluster - data$old_cluster))
  iterations <- iterations + 1
  # add the colours based on the cluster
  for (a in 1:4) {
    data$colour[data$cluster == a] <- colours[a]
  }
  plot(data$hours_norm ~ data$age_norm, col = data$colour,
       pch = 19, main = paste("iterations = ", iterations, "\nerrors = ",
                              errors, sep = ""))
  text(data$age_norm, data$hours_norm, labels = data$name,
       cex = 0.7, pos = 3)
}
```

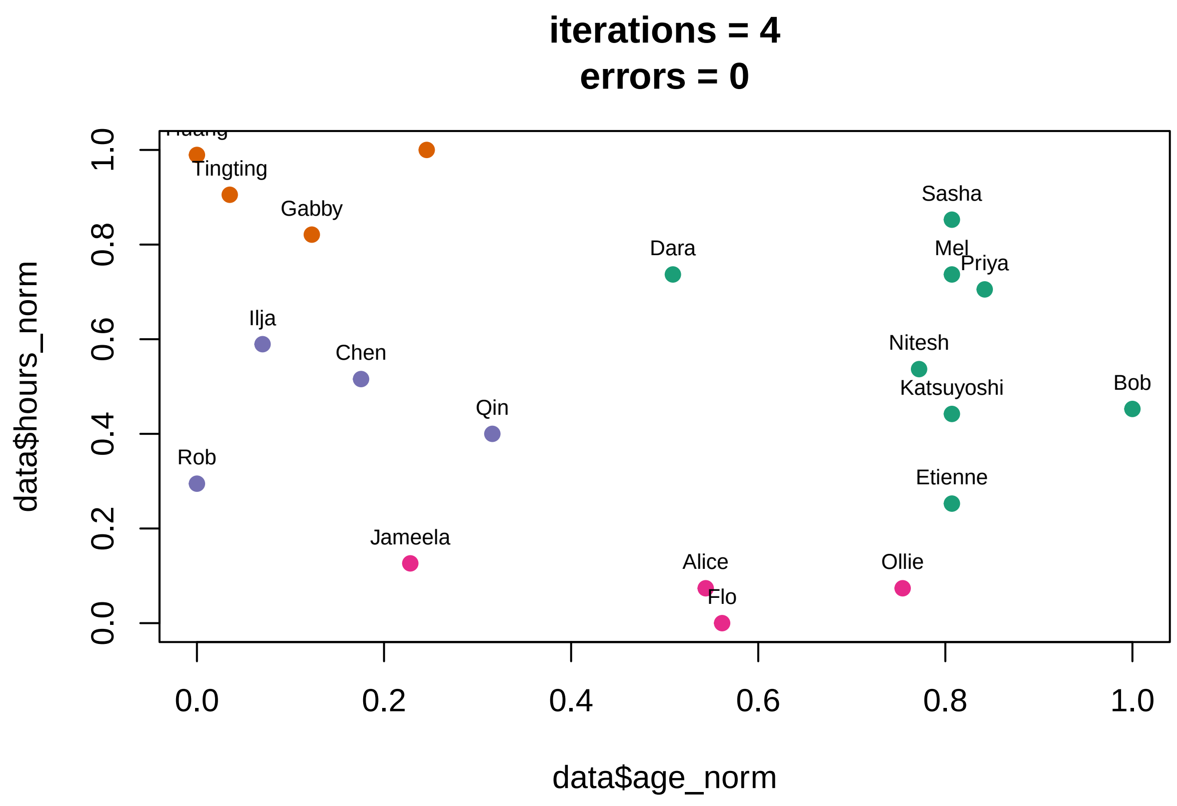


iterations = 1
errors = 2
data$hours_norm
data$age_norm
0.0
0.2
0.4
0.6
0.8
1.0
Huang
Tingting
Gabby
Ilja
Chen
Qin
Rob
Jameela
Dara
Alice
Flo
Ollie
Sasha
Mel
Priya
Nitesh
Katsuyoshi
Bob
Etienne

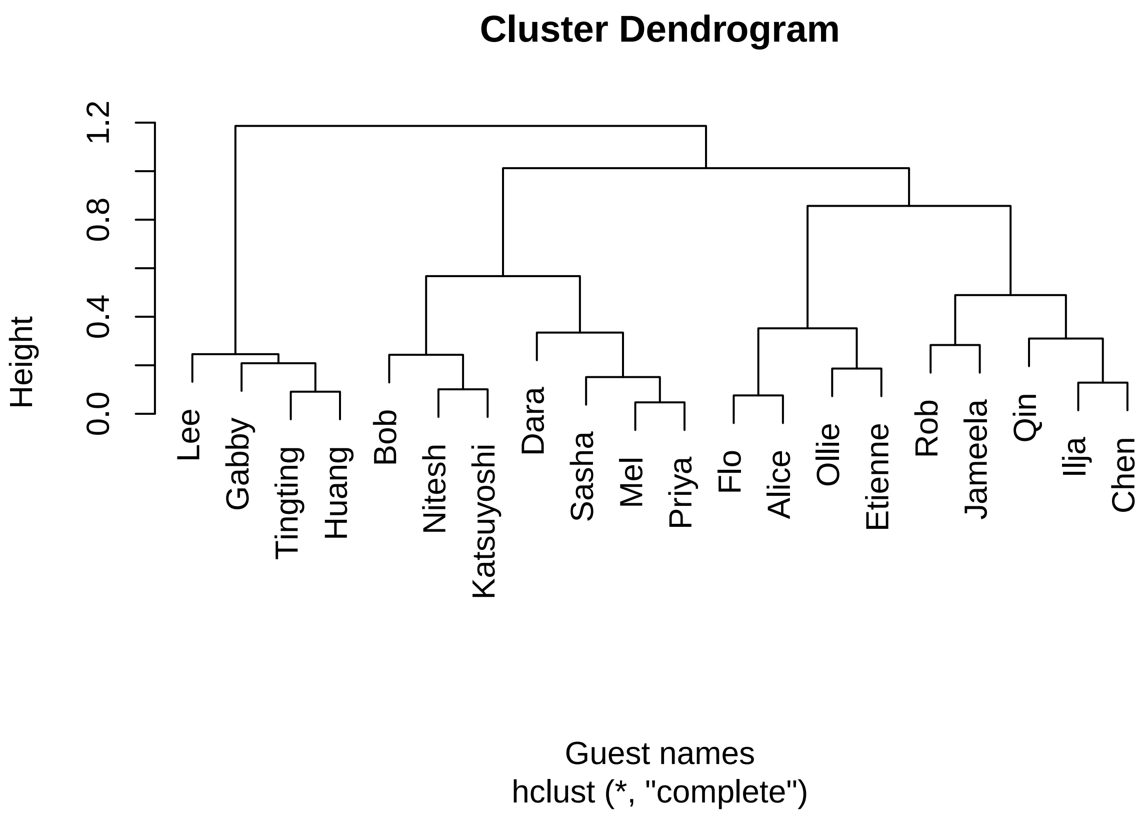

iterations = 2
errors = 1
Huang
Tingting
Gabby
Ilja
Chen
Qin
Rob
Jameela
Dara
Alice
Flo
Ollie
Sasha
Mel
Priya
Nitesh
Katsuyoshi
Bob
Etienne
data$hours_norm
data$age_norm
0.0
0.2
0.4
0.6
0.8
1.0



iterations = 3
errors = 2
Tingting
Gabby
Sasha
Dara
Mel
Priya
Ilja
Chen
Nitesh
Katsuyoshi
Bob
Qin
Rob
Etienne
Jameela
Alice
Ollie
Flo
data$hours_norm
data$age_norm
0.0
0.2
0.4
0.6
0.8
1.0

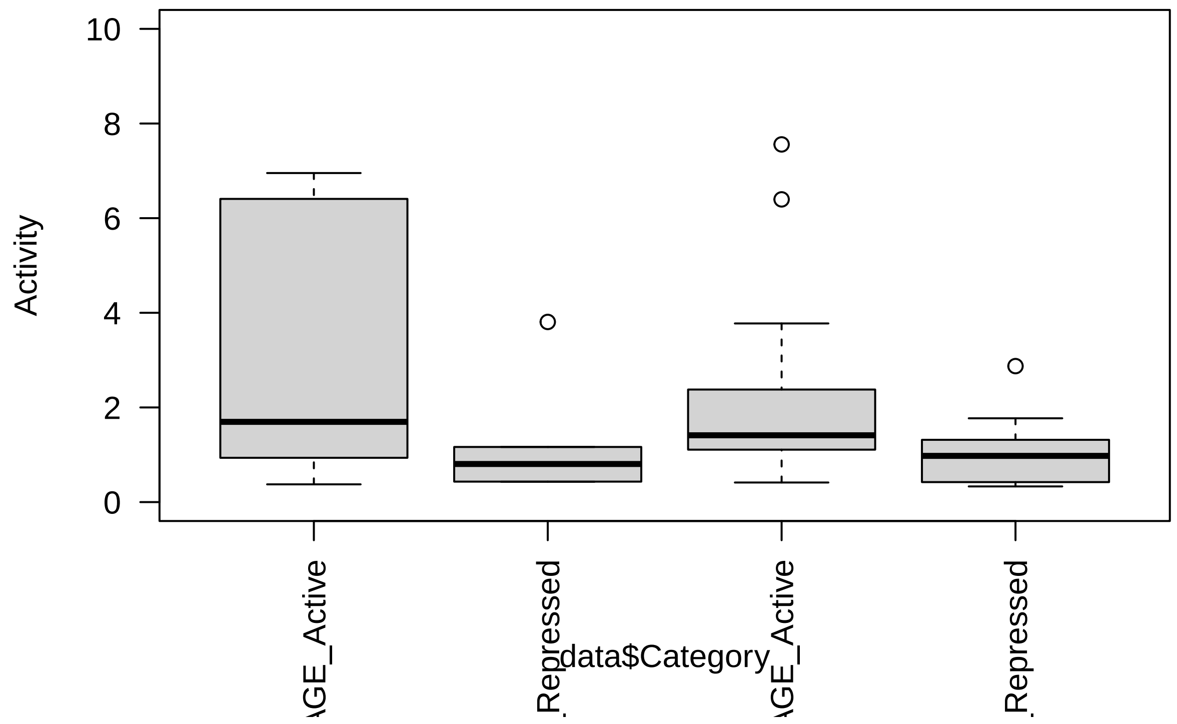


```
## Hierarchical clustering
# cluster the data
h_cluster <- hclust(dist(data[, 2:3]))
# plot the data
plot(h_cluster, xlab = "Guest names", labels = data$names)
```



##week14 bootstrapping

```r
# readin the txt file and plot
data <- read.table("../practical/Week14-Reporter_assay_4-1-15.txt", header = TRUE)
boxplot(data$ave~data$Category, ylim = c(0,10), las = 2, ylab = "Activity")
```



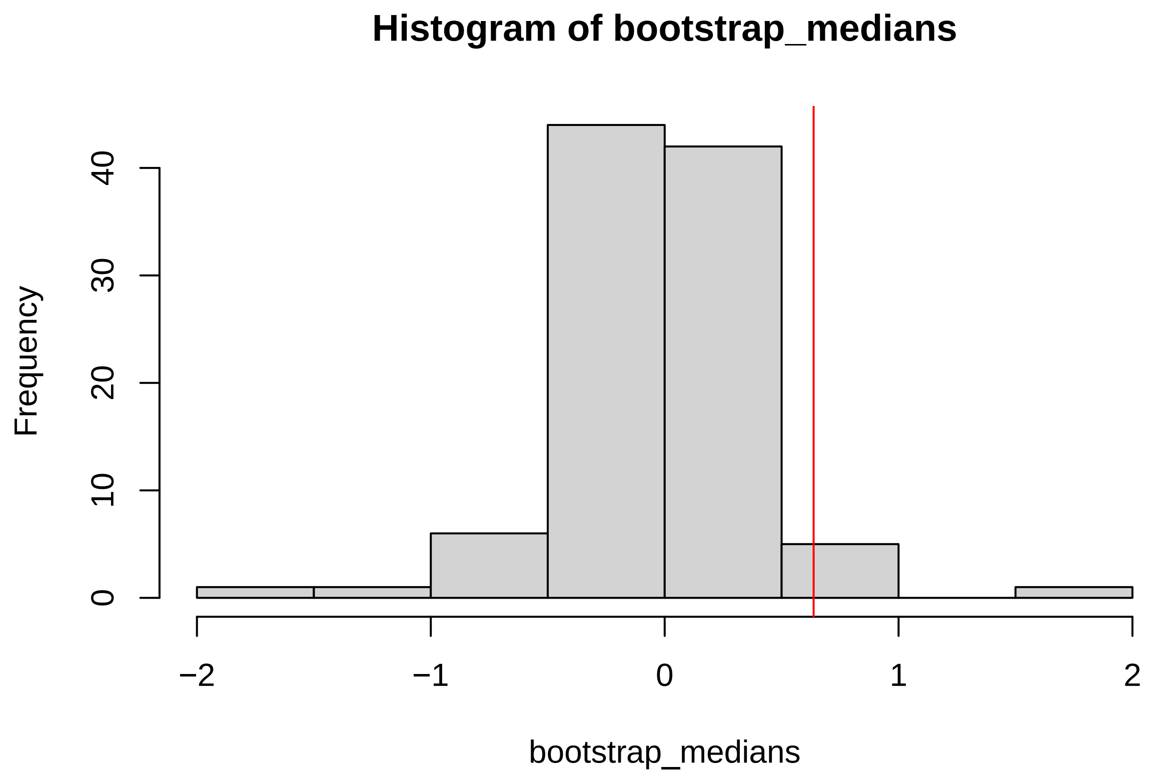
```r
# calculate the difference between the means of activan d repressed
active_activity<-median(subset(data$ave, data$Epigenetic_status == "Active"))
repressed_activity<-median(subset(data$ave, data$Epigenetic_status == "Repressed"))
median_diff<-active_activity - repressed_activity
median_diff
```

```
## [1] 0.6368301
```

```r
# generate one bootstrap sample of a median difference
length_active<-nrow(subset(data, data$Epigenetic_status == "Active"))
length_repressed<-nrow(subset(data, data$Epigenetic_status == "Repressed"))
bootstrap_active<-median(sample(data$ave, length_active, replace = TRUE))
bootstrap_repressed<-median(sample(data$ave, length_repressed, replace = TRUE))
bootstrap_median<-bootstrap_active - bootstrap_repressed
bootstrap_median
```

```
## [1] 0.0103623
```

```r
# Generate a large number of bootstraps
set.seed(45)
bootstrap_medians<-vector()
for (a in 1:100){
  bootstrap_active<-median(sample(data$ave, length_active, replace = TRUE))
  bootstrap_repressed<-median(sample(data$ave, length_repressed, replace = TRUE))
  bootstrap_median<-bootstrap_active - bootstrap_repressed
  bootstrap_medians<-c(bootstrap_medians, bootstrap_median)
}
hist(bootstrap_medians)
abline(v = median_diff, col = 'red')
```

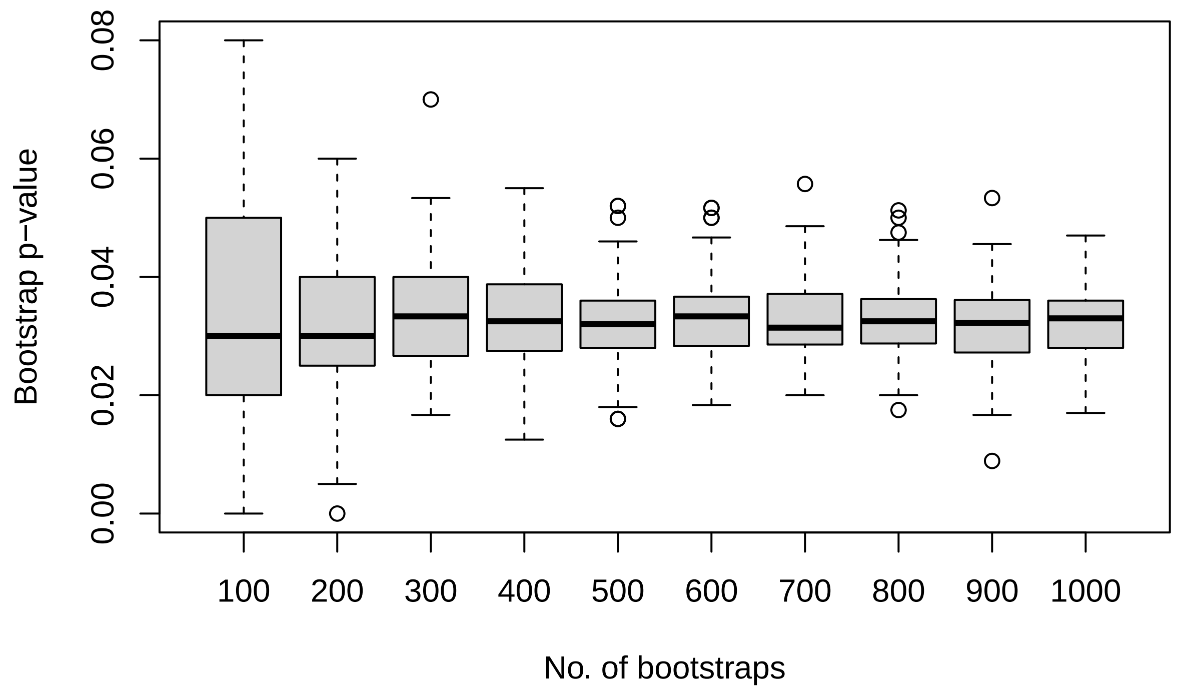


```
# Make a statistical inference
sig_bootstraps = length(subset(bootstrap_medians, bootstrap_medians >= median_diff))
sig_bootstraps/100
```

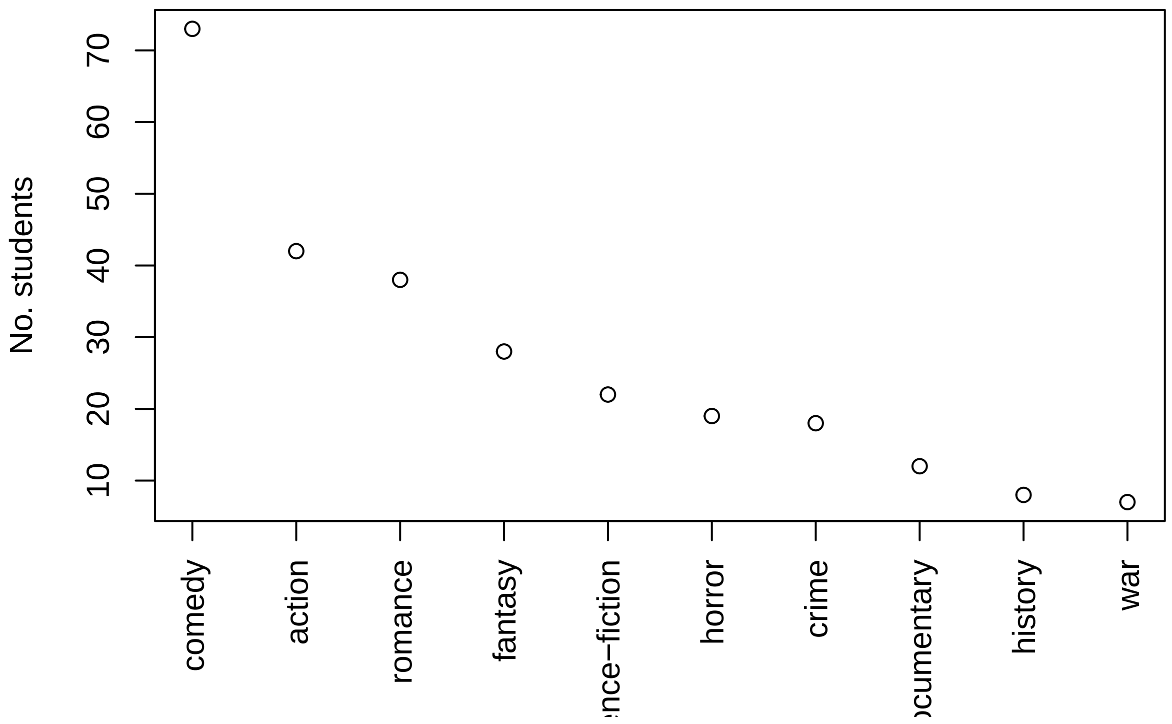
```
## [1] 0.03
```

```
# Explore the number of replicates
all_sig_results <- vector()
all_bootstrap_replicates <- vector()
for (bootstrap_replicate in seq(100, 1000, 100)) {
  for (a in 1:100) {
    sig_bootstraps <- 0
    for (b in 1:bootstrap_replicate) {
      bootstrap_active <- median(sample(data$ave, length_active,
                                        replace = TRUE))
      bootstrap_repressed <- median(sample(data$ave, length_repressed,
                                           replace = TRUE))
      bootstrap_median <- bootstrap_active - bootstrap_repressed
```

```r
      if (bootstrap_median >= median_diff) {
        sig_bootstraps <- sig_bootstraps + 1
      }
    }
    sig_result <- sig_bootstraps/bootstrap_replicate
    all_sig_results <- c(all_sig_results, sig_result)
    all_bootstrap_replicates <- c(all_bootstrap_replicates,
                                  bootstrap_replicate)
  }
}
boxplot(all_sig_results ~ factor(all_bootstrap_replicates), ylab = "Bootstrap p-value",
        xlab = "No. of bootstraps")
```

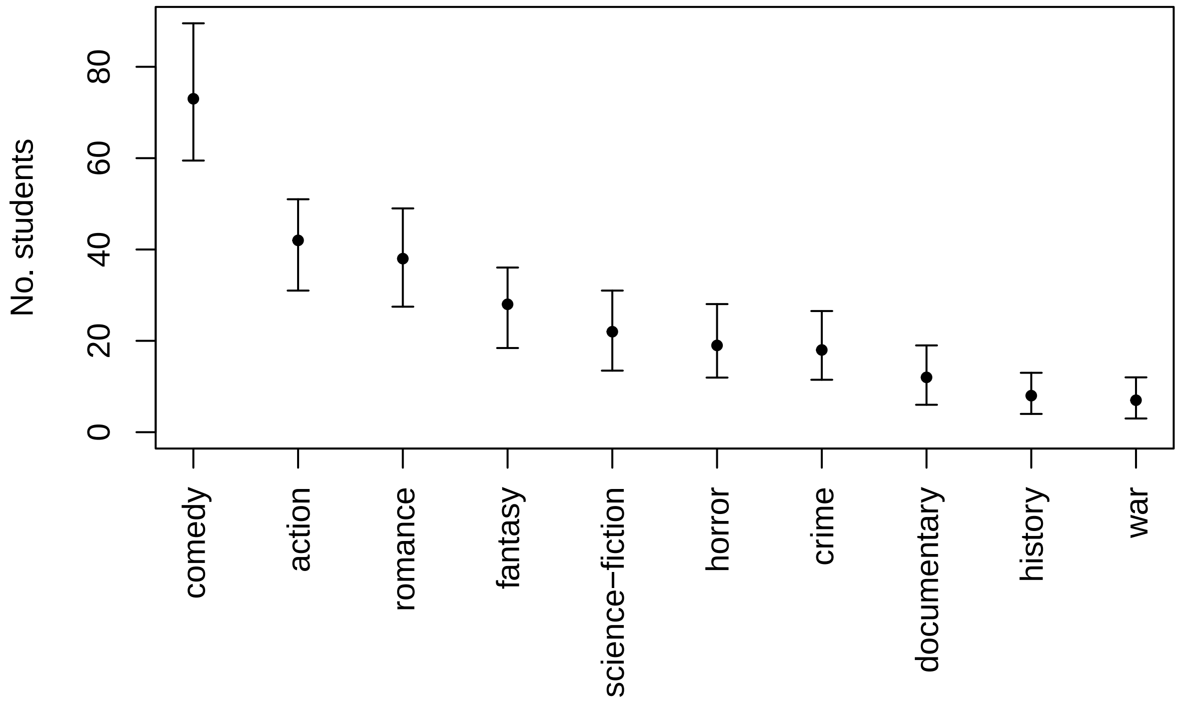


```r
# see whether you can repeat this procedure by looking at the 'Transcription_status' variable.


###------------------------------------------------------------------------
# movie question

movie_data<-read.table("../practical/Week14-movie_data.txt", header = T)
plot(movie_data$students, ylab = "No. students", xaxt = "n", xlab ='')
axis(side = 1, at = seq(1,nrow(movie_data),1), labels = movie_data$genre, las = 2)
```



```r
comedy_fans<-73
total_fans<-267
not_comedy_fans<-total_fans - comedy_fans
obs_values <- vector()
lower_cis <- vector()
upper_cis <- vector()
for (a in 1:nrow(movie_data)) {
  genre <- movie_data[a, 1]
  observed_fans <- movie_data[a, 2]
  not_observed_fans <- total_fans - observed_fans
  obs_sample <- c(rep(genre, observed_fans), rep("not_genre",
                                                  not_observed_fans))
  bootstrap_new <- vector()
  for (b in 1:100) {
    bootstrap_sample <- sample(obs_sample, length(obs_sample),
                               replace = T)
    bootstrap_new <- c(bootstrap_new, length(subset(bootstrap_sample,
                                                    bootstrap_sample == genre)))
  }
  lower_ci <- quantile(bootstrap_new, 0.025)
  upper_ci <- quantile(bootstrap_new, 0.975)
  obs_values <- c(obs_values, observed_fans)
  lower_cis <- c(lower_cis, lower_ci)
  upper_cis <- c(upper_cis, upper_ci)
}
```

```
ymax<-ceiling(max(upper_cis)*100)/100
par(mar=c(7,4,4,2))
plot(obs_values, xaxt = "n", ylim = c(0,ymax), xlab = "", pch = ".", ylab = "No. students")
axis(side = 1, at = seq(1,nrow(movie_data),1), labels = movie_data$genre, las = 2)
for (a in 1:length(lower_cis)){
  lines(x = c(a,a), y = c(lower_cis[a], upper_cis[a]))
  lines(x = c(a-0.1,a+0.1), y = c(lower_cis[a], lower_cis[a]))
  lines(x = c(a-0.1,a+0.1), y = c(upper_cis[a], upper_cis[a]))
}
points(x = seq(1,nrow(movie_data),1), y = obs_values, pch = 20)
```


No. students
comedy
action
romance
fantasy
science−fiction
horror
crime
documentary
history
war