ADS\_S2

2111

2024-05-27

knitr::opts\_chunk$set(echo = TRUE, message = FALSE)  
library(readr)  
library(ggplot2)  
library(dplyr)

##   
## 载入程辑包：'dplyr'

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

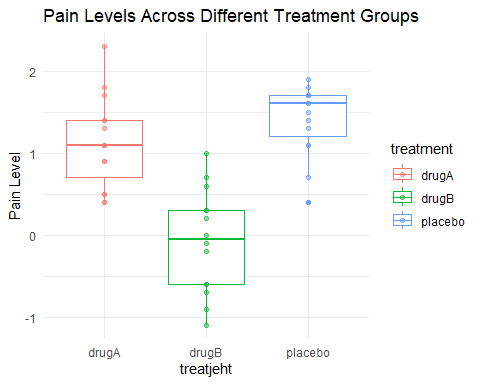
## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(tidyr)  
library(RColorBrewer)

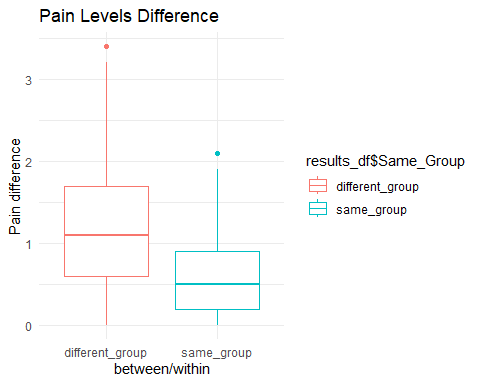
## Week1 multiple means

**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()

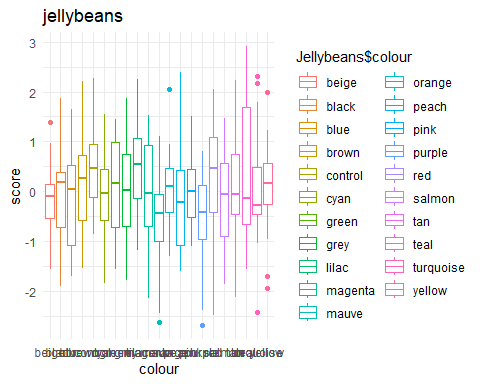


#2. H0 and HA  
#1) (H0): The mean pain levels for all three treatment groups are equal.  
# (HA): At least one treatment group has a different mean pain level compared to the others.  
#2) (H0): Drawing two data points at random from different groups will be as different from each other as drawing two data points at random from the same group.  
# (HA): Drawing two data points at random from different groups will be more different from each other than drawing two data points at random from the same group.  
  
  
  
#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$Same\_Group)) +  
 geom\_boxplot() +  
 labs(x = "between/within", y = "Pain difference", title = "Pain Levels Difference") +  
 theme\_minimal()

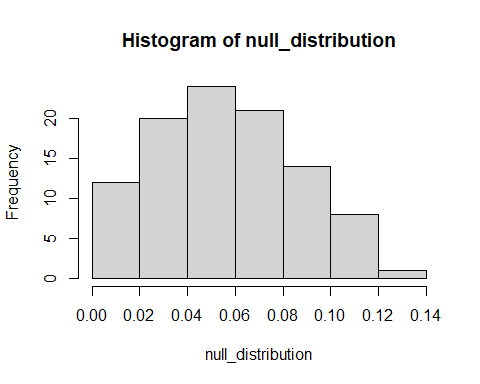


#a t-test is not appropriate in this case because the assumption of normality   
#may not hold for the distribution of pain differences  
  
wilcox\_test <- wilcox.test(Pain\_Difference ~ Same\_Group, data = results\_df)

# 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$colour)  
 colours <- sample(list\_colours, 2)  
 # Make spearate lists for each group  
 # (not strictly necessary, but helpful)  
 firstgroup <- Jellybeans[Jellybeans$colour == colours[1], "score"]  
 secondgroup <- Jellybeans[Jellybeans$colour == colours[1], "score"]  
 # draw samples  
 s1s2 <- sample(firstgroup$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)

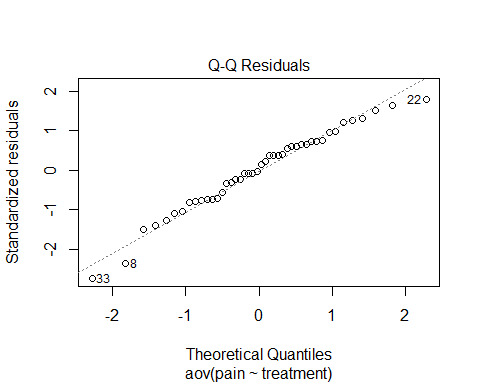


## Week2 ANOVA

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

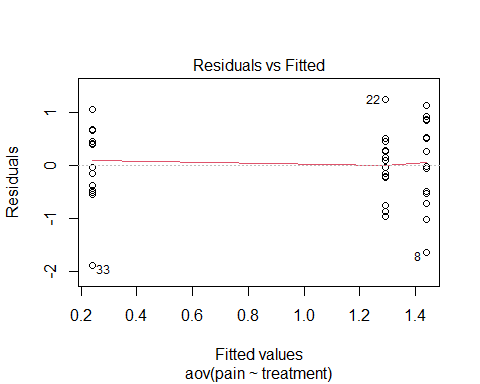
###  
dt <- read\_csv("C:/Users/10755/Desktop/Biooooo/ADS/S2/practical/Week2-drug\_trial(2).csv")  
  
#1. check for ANOVA assumption and run an ANOVA  
model = aov(pain~treatment, data=dt)  
plot(model, 2) # using the plot



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



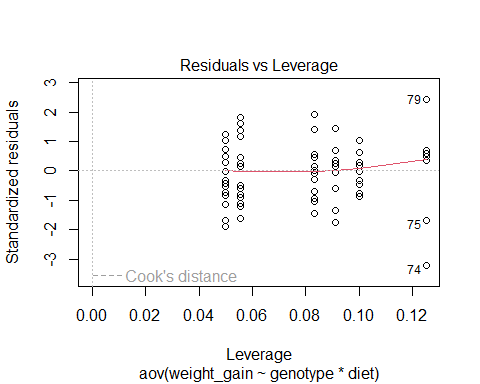
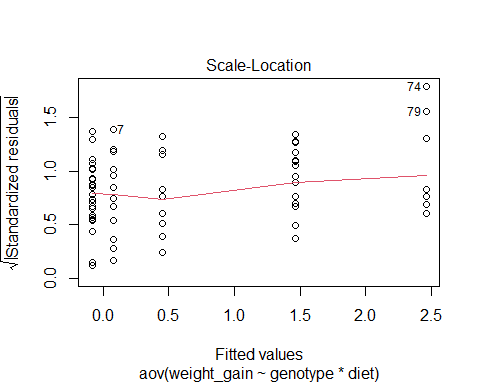
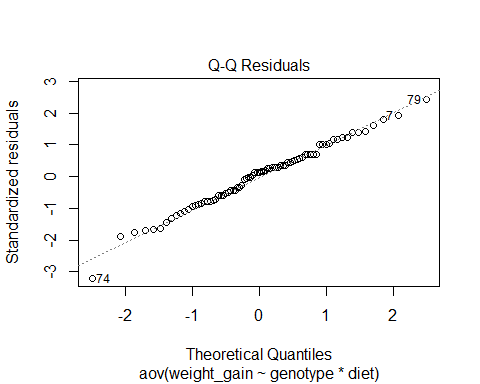
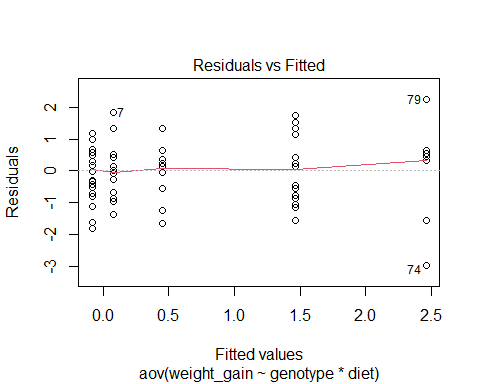
summary(model)

## 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

#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

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



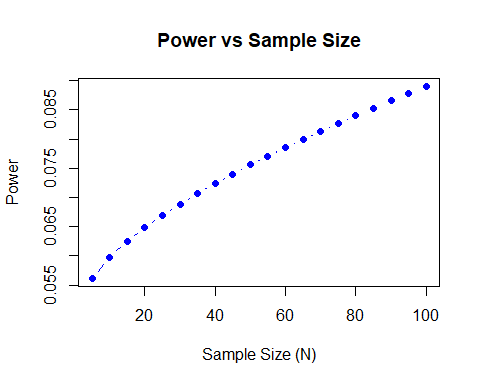
TukeyHSD(model2)

## 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’

# 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  
  
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")



#####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  
  
# 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

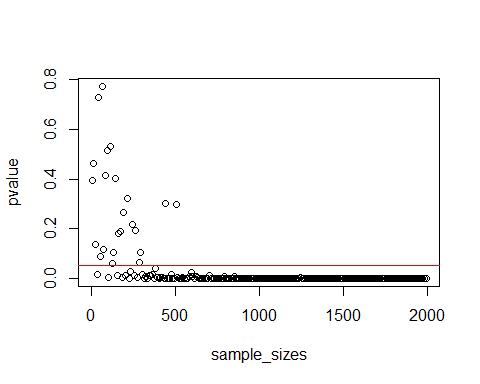
power\_analysis <- function(n) {  
 power\_result <- power.t.test(n = n, delta = effect\_size/100 \* sigma, sd = sigma, sig.level = 0.05, type = "two.sample", alternative = "two.sided")  
 return(power\_result$power)  
}  
  
# Calculate power for different sample sizes  
sample\_sizes <- seq(20, 200, 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

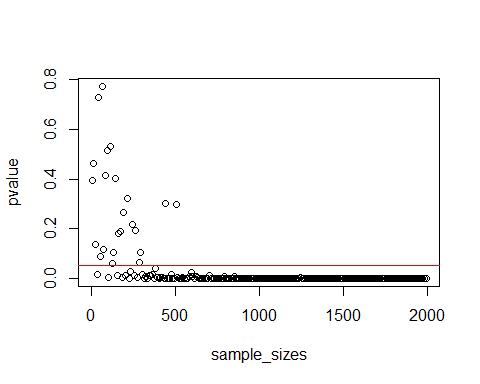
power\_analysis\_paired <- function(n) {  
 power\_result <- power.t.test(n = n, delta = effect\_size/100 \* sigma, sd = sigma, sig.level = 0.05, type = "paired", alternative = "two.sided")  
 return(power\_result$power)  
}  
  
# Calculate power for different sample sizes  
sample\_sizes\_paired <- 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 the desired power  
  
print(required\_sample\_size\_paired)

## [1] 200

# 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\_or\_cutoff, type = "two.sample", alternative = "two.sided")  
 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")



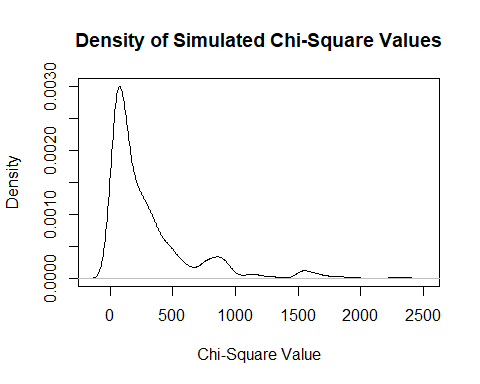
set.seed(13)  
sample\_sizes = seq(2, 2000, by = 10)  
a = list() #initialize a, why I use list to initialize a here?  
b = list() #initialize b  
pvalue = rep(0, length(sample\_sizes))  
for (i in 1:length(sample\_sizes)) {  
a[[i]] = rnorm(sample\_sizes[i], mean = 10, sd = 5)  
b[[i]] = rnorm(sample\_sizes[i], 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

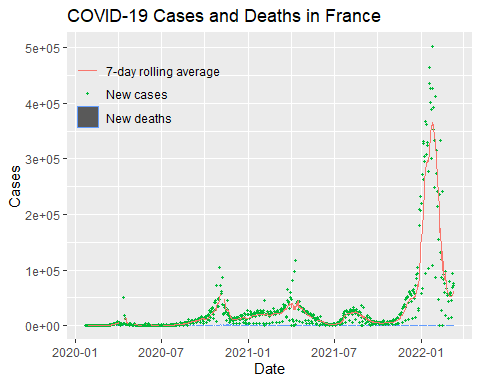
•Goodness-of-fit tests: The chi-square goodness-of-fit test is used to determine whether a sample of data comes from a population with a specific distribution. For example, it can test whether observed frequencies differ significantly from expected frequencies. •Test for independence: In a contingency 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 Value")

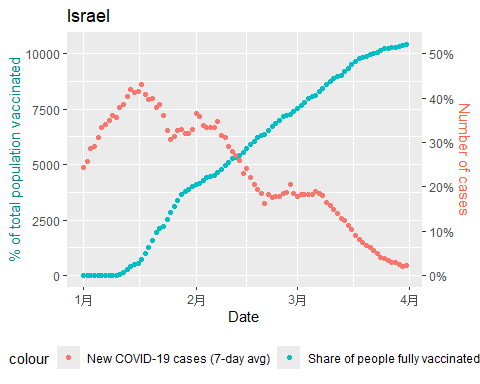


##Week6 Correlation and Linear Regressions A simple linear regression describes the association between an independent variable and a dependent one

# 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")  
 # 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))



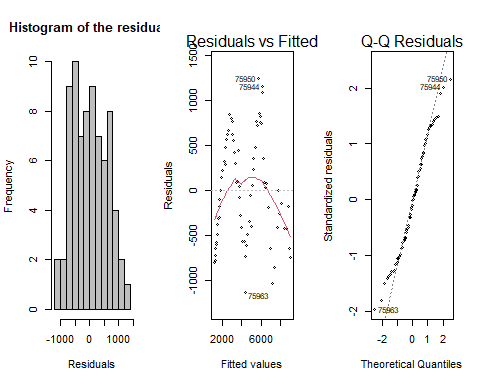
# Finding all the days from 2021, as they probably contain vaccination data  
dates\_from\_2021 <- seq(as.Date("2021-01-01"), as.Date("2021-12-31"), by = "days")  
  
one\_country <- "Israel"  
# Re-calculating the vector with 7-day rolling average of new COVID-19 cases  
cases\_means\_df <- generate\_rolling\_avg(subcovid, 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))



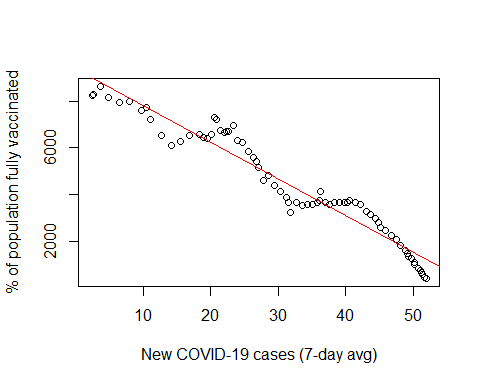
covid\_onecountry <- subcovid[which(subcovid$location == "Israel" &  
 subcovid$date >= as.Date("2021-01-15") &  
 subcovid$date < as.Date("2021-03-31")), ]  
# Calculating the Correlation Coefficient  
cor(covid\_onecountry$new\_cases\_avg,  
 covid\_onecountry$share\_fully\_vaccinated,  
 use = "complete.obs")

## [1] -0.9692676

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)



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")



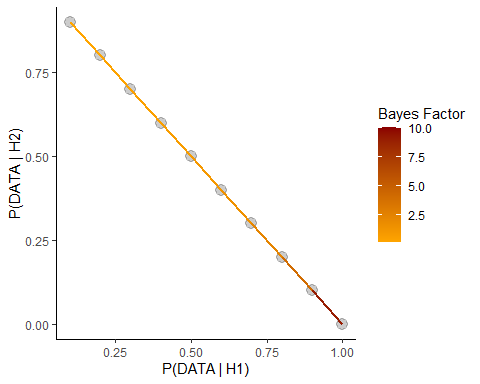
##Week7 Conditional Probabilities Bayes’ theorem P(A) × P(B|A) = P(B) × P(A|B) P(A|B) = P(A) × P(B|A)/P(B)

# 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\_probability))  
 lie\_detector\_results <- sapply(true\_status, function(status) {  
 if (status == 1) { # 如果员工是小偷  
 sample(c(0, 1), 1, prob = c(1 - detector\_accuracy, detector\_accuracy))  
 } else { # 如果员工不是小偷  
 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)  
 }  
}  
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  
 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?

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:

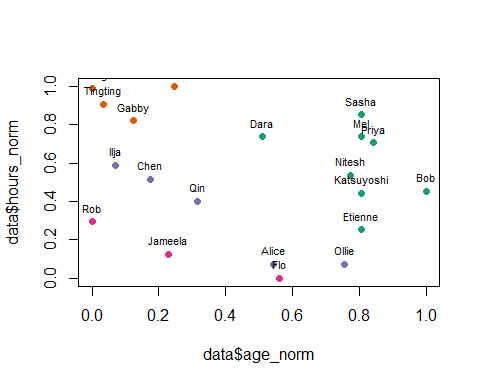
## [1] 5.816456e-06

##week 9&10 supervised

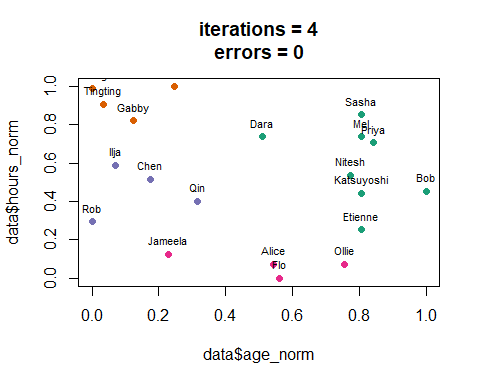
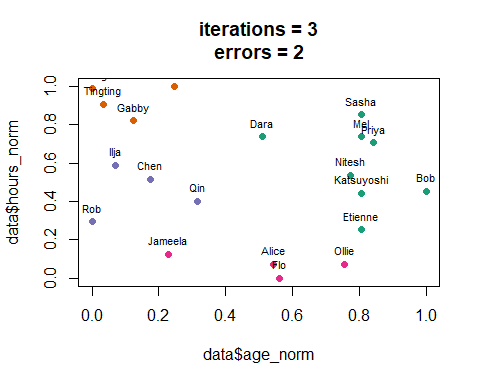
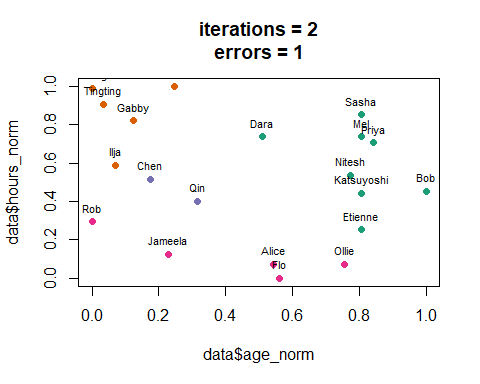
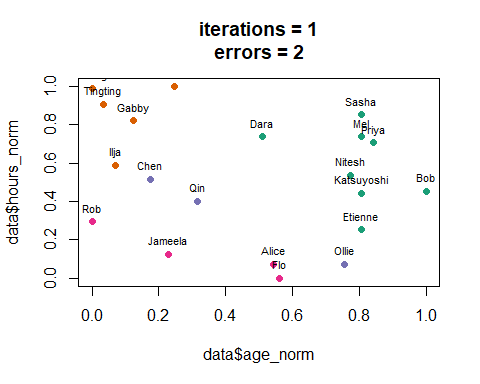
# read in data  
mnist\_raw <- read\_csv("../practical/Week9-mnist\_train.csv",col\_names = FALSE)  
mnist\_raw[1:10, 1:10]  
  
pixels\_gathered <- mnist\_raw %>%  
 head(1000) %>%  
 rename(label = X1) %>%  
 mutate(instance = row\_number()) %>%  
 gather(pixel, value, -label, -instance) %>%  
 tidyr::extract(pixel, "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)  
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

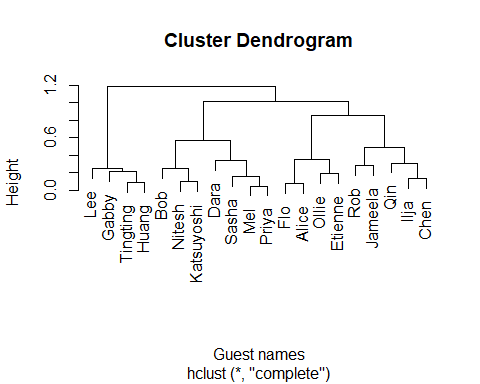
colours <- brewer.pal(4, "Dark2")  
  
  
  
data = read.csv("../practical/Week13-guest.csv", stringsAsFactors = T)  
p <- ggplot(data, aes(x = age\_norm, y = hours\_norm)) +  
 geom\_point(color = "blue") +  
 geom\_text(aes(label = names), vjust = -1, hjust = 0.5) +  
 labs(title = "Distribution of 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)  
}

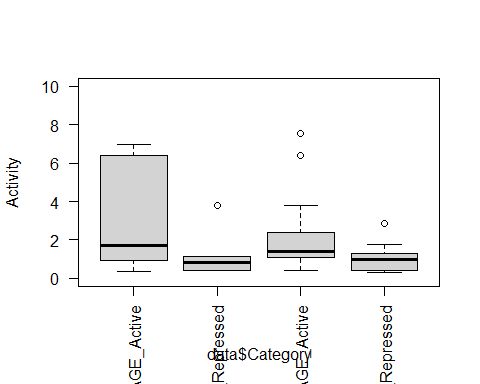


## Hierarchical clustering  
# cluster the data  
h\_cluster <- hclust(dist(data[, 2:3]))  
# plot the data  
plot(h\_cluster, xlab = "Guest names", labels = data$names)



##week14 bootstrapping

# 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")



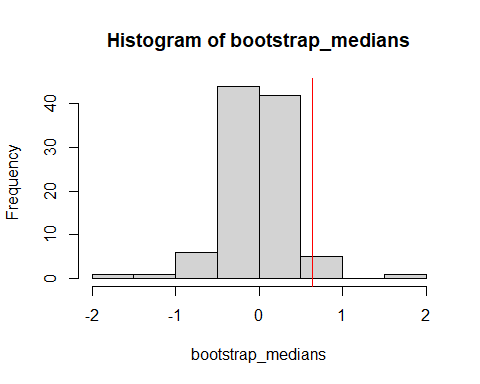
# calculate the difference between the means of activan d repressed  
active\_activity<-median(subset(data$ave, data$Epigenetic\_status == "Active"))  
repressed\_activity<-median(subset(data$ave, data$Epigenetic\_status == "Repressed"))  
median\_diff<-active\_activity - repressed\_activity  
median\_diff

## [1] 0.6368301

# 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

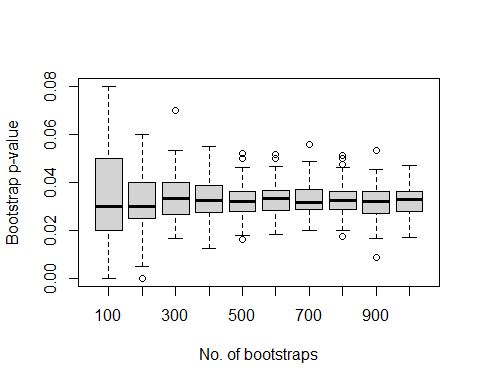
# Generate a large number of bootstraps  
set.seed(45)  
bootstrap\_medians<-vector()  
for (a in 1:100){  
 bootstrap\_active<-median(sample(data$ave, length\_active, replace = TRUE))  
 bootstrap\_repressed<-median(sample(data$ave, length\_repressed, replace = TRUE))  
 bootstrap\_median<-bootstrap\_active - bootstrap\_repressed  
 bootstrap\_medians<-c(bootstrap\_medians, bootstrap\_median)  
}  
hist(bootstrap\_medians)  
abline(v = median\_diff, col = 'red')



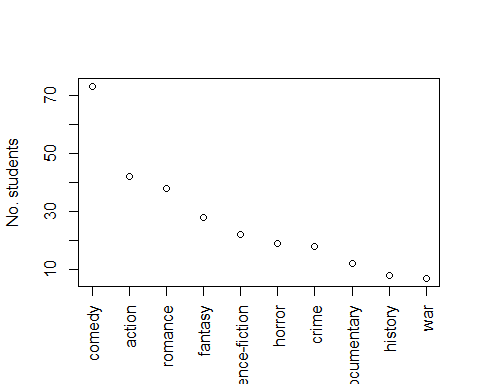
# Make a statistical inference  
sig\_bootstraps = length(subset(bootstrap\_medians, bootstrap\_medians >= median\_diff))  
sig\_bootstraps/100

## [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  
 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")



# 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)



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)

