R for Data Science & Intelligent Analytics Lab Compilation

Nethchie Dürregger

5/20/2020

Table of Contents

The motivation of this post is to compile all the labs i have in my Statistics and Algorithmic for Data Science. When i started the master study, i almost had no idea about R and every time i am analyzing projects i realized that R is a friend. I have trouble understanding the programming and now that i have understanding about them, i do not want to forget it and this is the reason why i am compiling everything i have done so far in my laboratory class for easy access if i want to review something.

# Loading libraries

The pacman::p\_load() package allows you to list down all the libraries all at once without using the library function all the time.

# Clear the workspace  
rm(list = ls())

knitr::opts\_chunk$set(eval = FALSE)

pacman::p\_load(  
 tidyverse,   
 caret,   
 tibble,  
 dplyr,  
 e1071,  
 neuralnet,  
 leaps,  
 mlbench, #BreastCancer data   
 skimr  
   
 )

# Loading Dataset

## Loading .RData (Churn)

load("./data/churn.RData")

## Loading Package Data (Boston)

df = MASS::Boston

## Loadding mlbench package data (BreastCancer)

data(BreastCancer)

## Loading CSV File

# Import data: library(tidyverse)  
#hmeq <- read\_csv("http://www.creditriskanalytics.net/uploads/1/9/5/1/19511601/hmeq.csv")  
  
#datatable(head(hmeq), rownames = FALSE, options = list(pageLength = 6, scrollX = TRUE))  
  
#read spotify dataset  
spotify\_songs <- read\_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-01-21/spotify\_songs.csv')  
  
weatherAUS <- read.csv("data/weatherAUS.csv")

# Data Understanding

## Checking of class data is balance

#breast\_cancer <- bcancer\_data %>% count(Class) %>%group\_by(Class) %>%summarize(percent\_dis= round((n / 569)\* 100, 2))

## Excluding Variables in the Analysis

bcancer\_data <- na.omit(bcancer\_data)  
bcancer\_data <- bcancer\_data[-1]

library(FNN)  
#Separating Outcome variable from the dataset  
Ozone\_outcome\_train <- trn\_data %>% select(Ozone)  
Ozone\_outcome\_test <- tst\_data %>% select(Ozone)  
  
#KNN dataset without outcome variable  
knn\_train\_data <- trn\_data %>% select(-Ozone)  
knn\_test\_data <- tst\_data %>% select(-Ozone)

# Explorative Data Analysis (Through Graphs)

## Distribution of all variables in the Dataset

# fix\_windows\_histograms() Run this once to show histogram in rmd html  
#skimr::skim(hmeq)

#-------------- Distributions ------------#  
# xray::distributions tries to analyze the distribution of your variables, so you  
# can understand how each variable is statistically structured. It also returns a  
# percentiles table of numeric variables as a result, which can inform you of the  
# shape of the data.  
  
#xray::distributions(hmeq)

## Sampling and Distribution

### Bernoulli Distribution

data.binom <- rbinom(100,1,0.5)   
# Plot distribution  
hist(data.binom)

### Binomial Distribution

data.binom2 <- rbinom(1000,10, 0.8)  
  
# Plot the distribution  
hist(data.binom2)  
  
# Assign and print probability of 8 or less successes  
prob1.sol <- pbinom(8,size=10, prob=0.8) # 0.6241904  
  
# Assign and print probability of all 10 successes  
prob2.sol <- dbinom(10,size = 10, prob = 0.8) #0.1073742

### Normal Distribution

# Generate normal data  
data.norm <- rnorm(1000)  
  
# Plot distribution  
hist(data.norm)  
  
# Compute and print true probability for greater than 2  
true\_prob.sol <- 1 - pnorm(2)  
  
# Compute and print sample probability for greater than 2  
sample\_prob.sol <- sum(data.norm > 2) / length(data.norm)

### Confidence Interval

z\_score <- 2.7764451051977987  
nums <- c(1, 2, 3, 4, 5)  
confidence <- 0.95  
  
std <- function(x) sd(x)/sqrt(length(x))  
std\_err.sol <- std(nums)  
  
#computation of the standard error of the mean  
sem<-sd(nums)/sqrt(length(nums))  
margin\_error <- z\_score \* std\_err.sol  
  
#95% confidence intervals of the mean  
c(mean(nums)-2\*sem,mean(nums)+2\*sem)

### Application of Confidence Interval

heads.sol <- rbinom(1, 50, 0.5)  
confidence\_int.sol <- prop.test(heads.sol, 50, conf.level=0.99)$conf.int # 99% conf - 0.3026493 0.6622787  
confidence\_int.9.sol <- prop.test(heads.sol, 50, conf.level=0.9)$conf.int # 90% Conf - 0.3584495 0.6037760  
  
# repeat 10 times  
heads = rbinom(n = 10, size = 50, prob = 0.5)  
for (i in 1:10) {  
 print(paste("untere Grenze: ",  
 prop.test(heads[i], n = 50, conf.level = 0.90)$conf.int[1],  
 ", Obere Grenze: ",   
 prop.test(heads[i], n = 50, conf.level = 0.90)$conf.int[2]  
 )  
 )  
}

### Samples from a rolled die

set.seed(1234)  
# Create a sample of 10 die rolls  
small.sol <- sample(1:6, 10, replace=T)  
  
# Calculate and print the mean of the sample  
small\_mean.sol <- sum(small.sol)/10 #ca. 3.9  
print(small\_mean.sol)  
  
  
# Create a sample of 1000 die rolls  
large.sol <- sample(1:6, 1000, replace=T)  
  
# Calculate and print the mean of the large sample  
large\_mean.sol <- sum(large.sol)/1000 #ca. 3.554  
print(large\_mean.sol)

### Simulation Cenral Limit Theorem

set.seed(1234)  
# Create a vector of 1000 sample means of size 30  
means.sol <- c()  
for(i in 1:1000){  
 means.sol<- c(means.sol,sample(1:6, 30, replace = T) %>% mean())  
   
}  
  
means\_100.sol <- c()  
for(i in 1:100){  
 means\_100.sol<- c(means\_100.sol,sample(1:6, 30, replace = T) %>% mean())  
   
}  
ggplot(data.frame(means=means.sol), aes(x=means))+geom\_histogram()  
  
ggplot(data.frame(means=means\_100.sol), aes(x=means))+geom\_histogram()

## Geom\_Bar

#Plot most popular genres in Spotify  
most\_popular <- ggplot(  
 data = as.data.frame(spotify\_songs %>% count(playlist\_genre)),   
 mapping = aes(x = reorder(playlist\_genre, n),n, fill=playlist\_genre)) +   
 geom\_bar(stat="identity") +  
 ggtitle("Most Popular genres in Spotify") +  
 theme(plot.title = element\_text(hjust = 0.5)) +   
 xlab("Playlist\_Genre") +  
 ylab("Frequency")   
  
  
# Plot top 1 artist of playlist genres   
top1\_artist <- ggplot(  
 data = spotify\_songs %>%   
 select(playlist\_genre, track\_artist, track\_popularity) %>%   
 group\_by(playlist\_genre,track\_artist) %>% summarise(n = n()) %>%   
 top\_n(1, n),  
 mapping = aes(x = reorder(playlist\_genre, n),n, fill=track\_artist)) +  
 geom\_bar(stat="identity") +  
 ggtitle("Top 1 artist of playlist\_genres") +  
 theme(plot.title = element\_text(hjust = 0.5)) +   
 xlab("Playlist\_Genre") +  
 ylab("Frequency")   
  
#geom\_bar filtered laptop   
ggplot(laptops\_filtered, aes(x=Company))+  
 geom\_bar()  
  
#Simple Geom\_bar  
ggplot(laptops\_filtered, aes(x=Company)) +  
 geom\_bar()

## Geom\_Boxplot

#Plot to see the variation of Acousticness between genres  
acousticness <- ggplot(data = spotify\_songs) +  
 geom\_boxplot(mapping= aes( x = acousticness, y = playlist\_genre, fill = playlist\_genre)) +  
 ggtitle("Variation of Acousticness between Playlist\_Genres") +  
 theme(plot.title = element\_text(hjust = 0.5))  
  
#Plot to see the variation of energy between genres  
energy <- ggplot(data = spotify\_songs) +  
 geom\_boxplot(mapping= aes( x = energy, y = playlist\_genre, fill = playlist\_genre)) +  
 ggtitle("Variation of Energy between Playlist\_Genres") +  
 theme(plot.title = element\_text(hjust = 0.5))   
  
#grid.arrange(acousticness,energy)  
  
  
#Simple geom\_boxplot  
ggplot(laptops\_filtered, aes(x=Company, y=Price\_euros)) +   
 geom\_boxplot()  
  
  
#1.Colored Boxplot  
boxplot(PlantGrowth$weight~PlantGrowth$group,  
 col=rainbow(4),   
 xlab = "Treatment",   
 ylab ="Weight",   
 main = "Plant Growth")

## Geom\_Historgram

ggplot(weatherAUS, aes(x=Temp3pm))+  
 geom\_histogram(bins=30)

## Geom\_Density

#Plot of Energy density of EDM sub-genres  
edm <- ggplot(data = spotify\_songs %>% filter(playlist\_genre == "edm"),  
 mapping = aes(x=energy, y = ..density..)) +  
 geom\_freqpoly(mapping=aes(color=playlist\_subgenre)) +  
 ggtitle("Energy density of EDM sub-genres") +  
 theme(plot.title = element\_text(hjust = 0.5))   
  
#Plot of Energy density of R&B sub-genres  
rnb <- ggplot(data = spotify\_songs %>% filter(playlist\_genre == "r&b"),  
 mapping = aes(x=energy, y = ..density..)) +  
 geom\_freqpoly(mapping=aes(color=playlist\_subgenre)) +  
 ggtitle("Energy density of R&B sub-genres") +  
 theme(plot.title = element\_text(hjust = 0.5))   
  
#grid.arrange(edm, rnb)

## Corrplot

library(corrplot)  
options(repr.plot.width = 20, repr.plot.height = 15)  
spotify\_sliced <- spotify\_songs[sapply(spotify\_songs, is.numeric)]  
corr <- cor(spotify\_sliced)  
  
corrplot <- corrplot(corr,method ="number")  
  
#Another type of corrplot  
#correlation matrix of the wisconsin breast cancer data  
corrplot(cor(wdbc[2:11]), method = "circle", diag = F, type = "lower", tl.cex = 0.7)

## Simple Scatterplot

ggplot(weatherAUS, aes(x=MinTemp, y=MaxTemp)) +  
 geom\_point()

## Anonnated Scatterplot

ggplot(data = spotify\_songs %>% sample\_frac(0.01), aes(x = valence, y = energy)) +  
 geom\_jitter() +  
 geom\_vline(xintercept = 0.5) +  
 geom\_hline(yintercept = 0.5) +  
 scale\_x\_continuous(expand = c(0, 0), limits = c(0, 1)) +  
 scale\_y\_continuous(expand = c(0, 0), limits = c(0, 1)) +  
 annotate('text', 0.25 / 2, 0.95, label = "Angry music", fontface =  
 "bold") +  
 annotate('text', 1.75 / 2, 0.95, label = "Joyful music", fontface = "bold") +  
 annotate('text', 1.75 / 2, 0.05, label = "Peaceful music", fontface =  
 "bold") +  
 annotate('text', 0.25 / 2, 0.05, label = "Depressing music", fontface =  
 "bold") +  
 ggtitle("Emotion of the Song in Spotify") +  
 theme(plot.title = element\_text(hjust = 0.5))

## Interactive Plot

library(tidyverse)  
library(ggrepel)  
#For a better view to show relationship between Ozone and Solar radiation in the training dataset. There is a strong negative relationship between Wind and Ozone, that is, the increase in Wind speed, it decreases the ozone.   
ggplot(data = trn\_data, mapping = aes(x = Wind, y = Ozone)) +   
 geom\_point(mapping = aes(color = Month )) +  
 geom\_smooth(color = "red") +   
 labs(  
 x = "Average Wind Speed (mph)",  
 y = " Maximum Daily Temperature (°F)",  
 color = "Month"  
 )

ggplot(data = trn\_data ) +  
 geom\_point(aes(x = Temp, y = Ozone), color = 'blue') +   
 geom\_point(aes(x = 89, y = predict(mod\_3)), color= "green", size=2) +  
 geom\_line(aes(x = Temp, predict(mod\_3)), color = 'red') +  
 xlab('Temperature') + ylab('Ozone')

df\_mod\_knn\_pred <- data.frame(pred=mod\_knn$pred, Temp = tst\_data$Temp)  
df\_mod\_4 <- data.frame( pred = pred\_tst\_4,Temp = tst\_data$Temp)  
df\_mod\_5 <- data.frame( pred = pred\_tst\_5,Temp = tst\_data$Temp)  
  
plot\_models <- ggplot() +   
 geom\_point(data = tst\_data, aes(Temp, Ozone), size = 3, color = "blue") +   
 geom\_line(data = df\_mod\_knn\_pred, aes(Temp, pred), color = "green") +   
 geom\_line(data = df\_mod\_5, aes(Temp, pred), color = "red") +   
 geom\_line(data = df\_mod\_4, aes(Temp, pred), color = "purple")

#The spiral Graph  
fviz\_pca\_ind(wdbc.prcomp, geom.ind = "point", pointshape = 21,   
 pointsize = 2,   
 fill.ind = wdbc$class,   
 col.ind = "red",   
 palette = "jco",   
 addEllipses = TRUE,  
 label = "var",  
 col.var = "red",  
 repel = TRUE,  
 legend.title = "Class") +  
 theme(plot.title = element\_text(hjust = 0.5))

# Plot predictions  
par(mfrow=c(1,3))  
  
#plot NN   
plot(test\_boston$medv,  
 nn\_boston\_pred\_,  
 col='red',  
 main='Real vs predicted NN',  
 pch=18,  
 cex=0.7)  
  
legend('bottomright',  
 legend='NN',  
 pch=18,  
 col='red',   
 bty='n')  
  
# plot LM  
plot(test\_boston$medv,  
 lm\_pred,  
 col='blue',  
 main='Real vs predicted lm',  
 pch=18,   
 cex=0.7)  
  
legend('bottomright',  
 legend='LM',  
 pch=18,  
 col='blue',   
 bty='n',   
 cex=.95)  
  
# Plot GLM   
plot(test\_boston$medv,   
 glm\_pred,  
 col='blue',  
 main='Real vs predicted glm',  
 pch=18, cex=0.7)  
  
legend('bottomright',  
 legend='LM',  
 pch=18,  
 col='blue',   
 bty='n',   
 cex=.95)

## Pairplot

weatherAUS %>% select(Temp3pm, Temp9am, Humidity9am ) %>% na.omit() %>%   
ggpairs()

library(GGally)  
library(ggplot2)  
library(dplyr)  
#diagram that shows all possible scatterplots of two variables in the training data set  
plot(trn\_data, col = "darkgrey")

## geom\_freqpoly

#Plot of Energy density of EDM sub-genres  
edm <- ggplot(data = spotify\_songs %>% filter(playlist\_genre == "edm"),  
 mapping = aes(x=energy, y = ..density..)) +  
 geom\_freqpoly(mapping=aes(color=playlist\_subgenre)) +  
 ggtitle("Energy density of EDM sub-genres") +  
 theme(plot.title = element\_text(hjust = 0.5))   
  
#Plot of Energy density of R&B sub-genres  
rnb <- ggplot(data = spotify\_songs %>% filter(playlist\_genre == "r&b"),  
 mapping = aes(x=energy, y = ..density..)) +  
 geom\_freqpoly(mapping=aes(color=playlist\_subgenre)) +  
 ggtitle("Energy density of R&B sub-genres") +  
 theme(plot.title = element\_text(hjust = 0.5))   
  
grid.arrange(edm, rnb)

# Data Processing

## Filtering

#Filtering only playlist genres "edm" and r&b   
spotify\_songs2 <- spotify\_songs %>%  
 filter(playlist\_genre %in% c("edm", "r&b")) %>%  
 mutate(playlist\_genre = recode(playlist\_genre, "r&b" = "RnB", "edm" = "EDM"))

laptops\_filtered <- laptops %>% filter(Company %in% c("Acer", "Asus", "Toshiba"))

## Selecting

#my selected predictors   
predictors <- names(spotify\_songs2)[12:23]  
  
#creating a new dataset so that i only have my selected predictors and classification variables for the analysis  
spotify\_songs2 <- spotify\_songs2 %>%  
 select(playlist\_genre, predictors)  
  
#I converted genres from character to factor type  
spotify\_songs2$playlist\_genre = as.factor(spotify\_songs2$playlist\_genre)

## Mean of Each Numerical COlumn

#Explorative Data Analysis  
#Count of B  
b <- sum(wdbc$class == "B")  
  
#count og M  
m <- sum(wdbc$class == "M")  
  
#Mean of each numerical columns  
num\_var\_mean <- apply(wdbc[,2:11],2,mean)

## Get the 10th observation of the dataset

#Get the 10th observation of the dataset   
  
test\_10th\_obs <- test\_bcancer[10,]

## Identfying & Handling Outliers

laptop <- get(load("./data/laptops\_unclean.RData"))  
  
# Calculate the mean and std  
laptops\_price\_mean <- mean(laptop$Price\_euros, na.rm = TRUE)  
laptops\_price\_std <- sd(laptop$Price\_euros , na.rm = TRUE)  
  
# Compute and print the upper and lower threshold  
lowerThreshold = laptops\_price\_mean - (laptops\_price\_std \* 3)  
upperThreshold = laptops\_price\_mean + (laptops\_price\_std \* 3)  
  
# Identify and print rows with outliers  
outliers <- boxplot(laptop$Price\_euros, plot = FALSE)$out   
  
# Drop the rows from the dataset  
laptop\_new <- laptop[-which(laptop$Price\_euros %in% outliers),]

## Getting Rid of Outlier observation

AnscombeX <- read.csv(file = "data/anscombex.csv")  
ggplot(AnscombeX, aes(x=x, y=y)) +geom\_point()  
cor(AnscombeX$x, AnscombeX$y)  
Anscombe\_f <- AnscombeX %>% filter(y<10)  
cor(Anscombe\_f$x,Anscombe\_f$y)

## Missing Value Handling and data pre-processing

library(statistics4ds)  
library(dplyr)  
laptop <- get(load("./data/laptops\_unclean.RData"))  
  
# Identify and print the rows with null values  
col\_index = which(is.na(laptop), arr.ind=TRUE)  
laptop\_null <- filter(laptop, is.na(Price\_euros)) %>% head()  
  
# Impute constant value 0 and print the head  
laptop\_null[is.na(laptop\_null)] = 0  
laptops\_impute\_0 <- laptop\_null   
  
# Impute median price and print the head  
laptop\_null\_med <- filter(laptop, is.na(Price\_euros))  
med\_price = median(laptop$Price\_euros, na.rm = TRUE)  
laptop\_null\_med[is.na(laptop\_null\_med)] = med\_price  
laptops\_impute\_med <- laptop\_null\_med %>% head()  
  
# Drop each row with a null value and print the head  
null\_row\_deleted <- na.omit(laptop) %>% head()

airquality\_cleaned = airQualityData[complete.cases(airQualityData[,1:6]),]

#deleting all missing values in the dataset  
churn <- na.omit(churn)   
  
#-------------- Anomaly detection ------------#  
# xray::anomalies analyzes all your columns for anomalies, whether they are NAs,  
# Zeroes, Infinite, etc, and warns you if it detects variables with at least 80%  
# of rows with those anomalies. It also warns you when all rows have the same  
# value.  
xray::anomalies(hmeq)  
  
# Stage 1 - Impute missing data: library(missRanger)  
hmeq\_imputed <- missRanger(hmeq, seed = 29)  
  
# Stage 2 - Normalize 0-1 features:  
df\_final <- hmeq\_imputed %>% mutate(BAD = case\_when(BAD == 1 ~ "Bad", TRUE ~ "Good")) %>%   
 mutate\_if(is.character, as.factor) %>% mutate\_if(is.numeric, function(x) {  
 (x - min(x))/(max(x) - min(x))  
})  
hmeq\_imputed %>% missing\_plot()  
hmeq\_imputed %>% inspect\_na %>% show\_plot

## Categorize numeric variable

# 'cut' function to categorize variable into categories   
churn['tenure'] <- as.factor(cut(churn$tenure, breaks=c(0,12,24,48,60)))

## Data Scaling

### Min-Max Scaling

# Get the min/max value of each variables  
maxs <- apply(df, 2, max)   
mins <- apply(df, 2, min)  
scaled <- as.data.frame(scale(df, center = mins, scale = maxs - mins))

## One-Hot Encoding

#read in laptops  
laptops <- read.csv("data/laptops.csv", stringsAsFactors = F)  
  
#convert factor  
laptops$Company <- laptops$Company %>% as.factor()  
  
# dummy coding  
dmy<-dummyVars(" ~ Company", data=laptops)   
laptops\_onehot <- data.frame(predict(dmy, newdata=laptops))  
laptops\_onehot %>% head() %>% data.table()

## Train-Test-Split

### Using Index Sample

set.seed(1910837166)  
  
#defining index for the partition  
index = sample(2, nrow(churn), replace=TRUE, prob=c(0.70,0.30))  
  
#Train and test split  
churn\_train <- churn[index==1,]  
churn\_test <- churn[index==2,]

set.seed(1910837166) #id Number  
  
# Sample size and sample index generation  
sample\_size = floor(0.70 \* nrow(airquality\_cleaned))   
trn\_index = sample(seq\_len(nrow(airquality\_cleaned)), size = sample\_size) #row index of a needed sample size  
  
#This is my train data having 70% from the total airquality\_cleaned observations  
trn\_data <- airquality\_cleaned[trn\_index, ]  
  
#This is the 30% data from the total airquality observations  
tst\_data <- airquality\_cleaned[-trn\_index, ]

# Statistical Experiment and Significance Test

## One-Sided Z-test

statistics4ds::ab\_test %>% head()  
  
#1. Mean of each group  
ab\_test\_summary <- statistics4ds::ab\_test %>%   
 group\_by(group) %>%   
 summarise(n = n(),  
 sum=sum(converted),  
 mean=mean(converted))  
  
#2. Total number of control group and the total number of trials of the control group  
num\_control <- ab\_test\_summary %>%   
 filter(group=="control") %>%   
 pull(sum)  
  
#3. pulling total number of control  
total\_control <- ab\_test\_summary %>%   
 filter(group=="control") %>%   
 pull(n)  
  
#4. Total number of treatment group and the total number of trials of the treatment group  
num\_treat <- ab\_test\_summary %>%   
 filter(group =="treatment") %>%   
 pull(sum)  
  
#5. Pulling total number of treatment  
total\_treat <- ab\_test\_summary %>%   
 filter(group =="treatment") %>%   
 pull(n)  
  
#6. z-Test using prop.test() function   
x <- c(num\_control,num\_treat) # number of success for each group  
n <- c(total\_control,total\_treat) # number of trials for each group  
prop.test(x , n , alternative="greater")

## Two-sided T-Test

#Load Laptop Data  
laptop <- read.csv("laptops.csv") %>% group\_by(Company)  
  
#1. Average price for each group of laptops Asus and Toshiba  
avg\_price <- laptop %>%   
 group\_by(Company) %>%   
 filter(Company =="Asus" | Company=="Toshiba") %>%   
 summarise(Avg.Price = mean(Price\_euros))  
  
#2. Summary of each group  
laptop\_summary <- laptop %>%   
 group\_by(Company) %>%   
 filter(Company =="Asus" | Company=="Toshiba") %>%   
 summarise( n = n(),   
 sum = sum(Price\_euros),   
 average = mean(Price\_euros))   
  
#3.The two-tailed t.test has an assumption of normality, before performing we perform normality test using shapiro. The null hypothesis of the shapiro test is that “sample distribution is normal”. If the test is significant, the distribution is non-normal.  
x <- laptop %>% filter(Company == "Asus") %>% pull(Price\_euros)  
y <- laptop %>% filter(Company == "Toshiba") %>% pull(Price\_euros)  
  
#At 5% level, the p-value = 2.281e-08 leads to conclusion that Asus prices are not normaliy distributed  
shapiro.test(x)  
  
#At 5% level, the p-value = 0.2571 we can assume normality distribution of Toshiba prices  
shapiro.test(y)  
  
# Parametric statistical two tailed t.test   
t\_test <- t.test(x, y, alternative = "two.sided")   
  
#Non-parametic statistical Wilcoxon test is an alternative t.test if the data is not normaly distributed  
wilcox.test(x,y, alternative = "two.sided")

## Power Analyses

library(pwr)  
# 1. Computes the effect size of the conversion rate from 20% to 25% success  
h <- ES.h(0.25,0.20)  
  
#2. pwr.\*.test(): \* refers to the statistical test of interest. It could be c("p = power calculation for proportions test", "t = power calculation for t.test", "r = effect size for correlation", "anov = power calculation for ANOVA", "chisq = power calculation for chi-squared test", "f2 = power calculation for GLM")  
sample95power <- pwr.2p.test(h = h, sig.level = 0.05, power = .95)  
  
#3. Required sample with 80% power  
sample80power <- pwr.2p.test(h = h, sig.level = 0.05, power = .80)  
  
#4. The required sample size having the effect size of the Websites's conversion rate from 20% to 25% success with 95% power is approx. 1808 samples.However it only requires approx. 1092 samples with 80% power this is because the accuracy decreases with lesser power.  
  
library(pwr2)  
sample\_sizes = seq(5, 100)  
effect\_sizes = c(0.2, 0.5, 0.8)  
  
#1. Graph that shows the relationship between Performance and Sample size  
pwr.plot(n=sample\_sizes, k=2, f=effect\_sizes, alpha=0.05)  
  
#2. The plot shows the impact on statistical power for three different effect sizes as the sample size is increased.We can see that if we are interested in a large effect that a point of diminishing returns in terms of statistical power occurs at approximately 20 observations.

## Multiple Test

#1. Print error rate for 60 tests with 5% significance  
error\_rate\_60 <- 1-(0.95^60)  
  
#2. Print error rate for 30 tests with 5% significance  
error\_rate\_30 <- 1-(0.95^30)  
  
#3. Print error rate for 10 tests with 5% significance  
error\_rate\_10 <- 1-(0.95^10)  
  
#4. Conclusion: If we run a series of 60, 30 and 10 significance tests respectively, the probability that at least one predictor will falsely test significant is 95.39%, 78.54% and 40.13% respectively.This issue is related to the problem of “fitting the model to the noise.” The more test we do, or the more models we run, the greater the probability that something will emerge as “significant” just by chance

## Bonferroni-Korrektur

pvals = c(.01, .05, .10, .50, .99)  
  
#1. Bonferroni Corrector for pvals  
pvals\_adjust <- p.adjust(p = pvals , method = "bonferroni")  
  
#2. Bonferroni Corrector multiple tests  
mult\_adjust <- p.adjust(c(0.95^60, 0.95^30, 0.95^10), "bonferroni")  
  
#3. Bonferroni Corrector   
my\_pvals = c(0.2, 0.3, 0.01, 0.003)  
my\_pvals\_adjust <- p.adjust(my\_pvals, "bonferroni" )

## ANOVA Testing

PlantGrowth %>% head()  
  
#1.Boxplot  
boxplot(PlantGrowth$weight~PlantGrowth$group,  
 col=rainbow(4),   
 xlab = "Treatment",   
 ylab ="Weight",   
 main = "Plant Growth")   
  
#2. Compute the analysis of variance  
anova <- aov(weight ~ group, data=PlantGrowth)  
  
#4. Tukey multiple pairwise-comparisons  
TukeyHSD(anova,"group")  
  
# 6a.Homogeneity of variances. Points 17, 15, 4 are detected as outliers, which can severely affect normality and homogeneity of variance. It can be useful to remove outliers to meet the test assumptions  
plot(anova, 1)  
  
#Here i use Barlett's test to check the homogeneity of variances. The p-value = 0.2371 is greater than the significance level of 0.05.Thus, there is no evidence to suggest that the variance across groups is significantly different.We can assume the homogeneity of variances in the different treatment groups  
bartlett.test(PlantGrowth$weight~PlantGrowth$group)  
  
# 5b. Using QQplot, the data is normally distributed as all the points fall approximately along the reference line  
plot(anova,2)  
  
# Here i use the Shapiro-Wilk test on the ANOVA residuals to support the normality graph.The p-value = 0.4379 is greater than alpha = 0.05 which is not signifcant. Therefore, we can assume normality of PlantGrowth data  
shapiro.test(residuals(object = anova))

# Modelling

## Regression and Prediction

### Linear Regression

library(tidyverse)  
#loading weather dataset and transforming into clean data  
weather <- read.csv("./data/weatherAUS.csv")  
weather\_cleaned <- weather[complete.cases(weather[ , 1:24]),]  
  
# Create and fit your linear regression model  
linear\_model <- lm(Humidity3pm ~ Humidity9am , data = weather\_cleaned)  
  
# Assign and print predictions  
preds <- data.frame(predict(linear\_model))  
print(head(preds))  
  
# Plot the fit of the model. The graphs shows that there is a positive relationship between humidity 9am and humidity 3pm   
ggplot(weather\_cleaned, aes(x=Humidity9am, y=Humidity3pm)) +  
 geom\_point() +  
 geom\_smooth(method= "lm")  
  
# Assign and print coefficient   
coef\_lm <- coef(linear\_model)  
  
# Model Evaluation  
mean\_Humidity3pm = mean(weather\_cleaned$Humidity3pm)  
actual = weather\_cleaned$Humidity3pm  
predicted = predict(linear\_model)  
error = actual - predicted  
  
#1. R-squared score  
r2 <- 1 - (sum((error^2))/sum((actual - mean\_Humidity3pm)^2))  
  
#2. Mean squared error   
mse <- mean(error^2)  
  
#3. Mean absolute error  
mae <- mean(abs(error))

### Polynomial Regression Model

#Interpretation: Model 1 implies an increase of Temprerature leads to a significantly increase of 2.41 in Ozone  
mod\_1 <- lm(Ozone ~ Temp, data = trn\_data)  
  
#Interpretation: Model 2 better describes their relationship.The R-squared indicates that 50.26% of the variability of the data is explained by a polynomial of degree 2 instead of 47.13% with a polynomial of degree 1. An increase in tempereature is a significantly increase of 0.058 in Ozone given polynomial of degree 2  
mod\_2 <- lm(Ozone ~ Temp + I(Temp^2), data = trn\_data)  
  
#Interpretation: The R-squared in model 3 has a variability 50.95% which is almost equal to the polynomial of degree 2. The residual standard error of this model is exactly the same as the previous one which implies that this model brings no improvement.   
mod\_3 <- lm(Ozone ~ Temp + I(Temp^2) + I(Temp^3), data = trn\_data)  
  
#Interpretation: Model 4 explains the 52.55% variability which gives a higher weight compared to the previous model.However, this improvment is really too small with RSE 24.58 compared to the previous ones  
mod\_4 <- lm(Ozone ~ Temp + I(Temp^2) + I(Temp^3) + I(Temp^4), data = trn\_data)  
  
#Interpretation: Although model 5 improves only a bit to the previous models, it is the best model having 52.81% variability that explains the data and it has the least residual standard error among all models.  
mod\_5 <- lm(Ozone ~ Temp + I(Temp^2) + I(Temp^3) + I(Temp^4) + I(Temp^5), data = trn\_data)  
  
  
#Root Mean Square Error of all polynomial models for train data  
rmse\_train\_1 <- sqrt(mean((trn\_data$Ozone - predict(mod\_1))^2))  
rmse\_train\_2 <- sqrt(mean((trn\_data$Ozone - predict(mod\_2))^2))  
rmse\_train\_3 <- sqrt(mean((trn\_data$Ozone - predict(mod\_3))^2))  
rmse\_train\_4 <- sqrt(mean((trn\_data$Ozone - predict(mod\_4))^2))  
rmse\_train\_5 <- sqrt(mean((trn\_data$Ozone - predict(mod\_5))^2))  
  
#Root Mean Square Error of all polynomial models for test data  
rmse\_tst\_1 <- sqrt(mean((tst\_data$Ozone - pred\_tst\_1)^2))  
rmse\_tst\_2 <- sqrt(mean((tst\_data$Ozone - pred\_tst\_2)^2))  
rmse\_tst\_3 <- sqrt(mean((tst\_data$Ozone - pred\_tst\_3)^2))  
rmse\_tst\_4 <- sqrt(mean((tst\_data$Ozone - pred\_tst\_4)^2))  
rmse\_tst\_5 <- sqrt(mean((tst\_data$Ozone - pred\_tst\_5)^2))  
rmse\_tst\_knn <- sqrt(mean((tst\_data$Ozone - mod\_knn$pred)^2))  
  
# R-Squared Score of the all models for train data  
r2\_trn\_1 <- 1 - (sum((trn\_data$Ozone - predict(mod\_1))^2) / sum((trn\_data$Ozone - mean(trn\_data$Ozone))^2))  
r2\_trn\_2 <- 1 - (sum((trn\_data$Ozone - predict(mod\_2))^2) / sum((trn\_data$Ozone - mean(trn\_data$Ozone))^2))  
r2\_trn\_3 <- 1 - (sum((trn\_data$Ozone - predict(mod\_3))^2) / sum((trn\_data$Ozone - mean(trn\_data$Ozone))^2))  
r2\_trn\_4 <- 1 - (sum((trn\_data$Ozone - predict(mod\_4))^2) / sum((trn\_data$Ozone - mean(trn\_data$Ozone))^2))  
r2\_trn\_5 <- 1 - (sum((trn\_data$Ozone - predict(mod\_5))^2) / sum((trn\_data$Ozone - mean(trn\_data$Ozone))^2))  
  
#R-Squared Score of the all models for train data  
r2\_tst\_1 <- 1 - (sum((tst\_data$Ozone - pred\_tst\_1)^2) / sum((tst\_data$Ozone - mean(tst\_data$Ozone))^2))  
r2\_tst\_2 <- 1 - (sum((tst\_data$Ozone - pred\_tst\_2)^2) / sum((tst\_data$Ozone - mean(tst\_data$Ozone))^2))  
r2\_tst\_3 <- 1 - (sum((tst\_data$Ozone - pred\_tst\_3)^2) / sum((tst\_data$Ozone - mean(tst\_data$Ozone))^2))  
r2\_tst\_4 <- 1 - (sum((tst\_data$Ozone - pred\_tst\_4)^2) / sum((tst\_data$Ozone - mean(tst\_data$Ozone))^2))  
r2\_tst\_5 <- 1 - (sum((tst\_data$Ozone - pred\_tst\_5)^2) / sum((tst\_data$Ozone - mean(tst\_data$Ozone))^2))  
r2\_test\_knn <- 1 - (sum((tst\_data$Ozone - mod\_knn$pred)^2) / sum((tst\_data$Ozone - mean(tst\_data$Ozone))^2))

### Multiple Linear Regression

#Multiple Regression model predicting Ozone from the other variables  
add\_mod = lm(Ozone ~ Solar.R + Wind + Temp + Month + Day, data = trn\_data)  
summary(add\_mod)  
  
#Interpretation: The multiple R-squared is given by 0.6223, approximately 62% of variation in Ozone can be expained by solar radiation, wind and temperature. The intercept -46.88 is the estimate Ozone when solar radiation, wind and temperature are all 0. In this case the intercept doesn't really have a meaningful interpretation. But there is significantly decrease of -4.206 in Ozone if the wind speed increases. There is also significantly increase of 1.674 in Ozone if temperature gets higher  
  
#Calculates the root mean square error of the model using test data  
actual = tst\_data$Ozone  
predicted = predict(add\_mod, tst\_data )  
error = actual - predicted   
calc\_rmse <- sqrt(mean(error^2))  
print(calc\_rmse)  
  
#Interpretation: The root mean square error 18.9511 is the standard deviation of the unexplained variance from the actual values in the test data

## K-Nearest Neighbor

### Cancer Detection with KNN

library(statistics4ds)  
library(tidyverse)  
library(plotly)  
  
wisc\_trn <- read.csv('data/wisc-trn.csv')  
wisc\_tst <- read.csv('data/wisc-tst.csv')  
  
#Modelling for unscaled data  
pred <-knn(train = data.frame(wisc\_trn$radius, wisc\_trn$symmetry, wisc\_trn$texture), test=data.frame(wisc\_tst$radius, wisc\_tst$symmetry, wisc\_tst$texture), cl=wisc\_trn$class, k=5)  
cf <- confusionMatrix(pred, wisc\_tst$class, positive = 'M')  
  
  
#Modelling for Scaled data  
wisc\_trnsc <- wisc\_trn %>% mutate\_if(is.numeric, scale)  
wisc\_tstsc <- wisc\_tst %>% mutate\_if(is.numeric, scale)  
  
pred\_sc <-knn(train = data.frame(wisc\_trnsc$radius, wisc\_trnsc$symmetry, wisc\_trnsc$texture), test=data.frame(wisc\_tstsc$radius, wisc\_tstsc$symmetry, wisc\_tstsc$texture), cl=wisc\_trnsc$class, k=5)  
cfsc <- confusionMatrix(pred\_sc, wisc\_tst$class, positive = 'M')  
  
# define function for class error  
calc\_class\_err = function(actual, predicted) {  
 mean(actual != predicted)  
}  
  
# define function for class error  
calc\_class\_err = function(actual, predicted) {  
 mean(actual != predicted)  
}  
  
# loop over k  
k\_to\_try = 1:200  
err\_k = tibble(k=k\_to\_try, err=rep(x = 0, times = length(k\_to\_try)),  
 err\_sc=rep(x = 0, times = length(k\_to\_try)))  
  
for (i in seq\_along(k\_to\_try)) {  
 pred = knn(train = data.frame(wisc\_trn$radius, wisc\_trn$symmetry, wisc\_trn$texture),  
 test = data.frame(wisc\_tst$radius, wisc\_tst$symmetry, wisc\_tst$texture),  
 cl=wisc\_trn$class,   
 k = k\_to\_try[i])  
 err\_k$err[i] = calc\_class\_err(wisc\_tst$class, pred)  
}  
  
# loop over k scaled  
err\_k\_sc = rep(x = 0, times = length(k\_to\_try))  
  
for (i in seq\_along(k\_to\_try)) {  
 pred = knn(train = data.frame(wisc\_trnsc$radius, wisc\_trnsc$symmetry, wisc\_trnsc$texture),  
 test = data.frame(wisc\_tstsc$radius, wisc\_tstsc$symmetry, wisc\_tstsc$texture),  
 cl=wisc\_trnsc$class,   
 k = k\_to\_try[i])  
 err\_k$err\_sc[i] = calc\_class\_err(wisc\_tstsc$class, pred)  
}  
  
ggplot(err\_k, aes(x=k)) +  
geom\_line(aes(y=err,col= "not scaled") ) +  
geom\_line(aes(y=err\_sc, col="scaled variables")) +  
xlab('k, number of nearest neighbours') +  
ylab('classification error') +  
scale\_color\_manual(name = "Train data",   
 values = c("not scaled" = "orange", "scaled variables" = "green"))

### Classical KNN-Classification

#loading library  
pacman::p\_load(  
 caret,  
 tibble,   
 dplyr,  
 e1071,  
 caTools,  
 leaps  
)  
  
# load("...../churn.RData")  
load("./churn.RData")   
str(churn)  
  
#Min and max value of tenure  
print(min(churn$tenure))  
print(max(churn$tenure))  
  
#Categorizing variable to 5 categories. “0–12 Month”, “12–24 Month”, “24–48 Months”, “48–60 Month”, “> 60 Month”  
churn['tenure'] <- as.factor(cut(churn$tenure, breaks=c(0,12,24,48,60)))   
  
#Include on complete case in the analysis  
churn <- na.omit(churn)   
  
#Train-test split  
#set seed for reproducible partition  
set.seed(1910837166)  
  
#defining index for the partition  
index = sample(2, nrow(churn), replace=TRUE, prob=c(0.70,0.30))  
  
#Train and test split  
churn\_train <- churn[index==1,]  
churn\_test <- churn[index==2,]   
  
  
# fit model   
churn\_knn <- train(  
 Churn ~ .,   
 data = churn\_train,   
 method = "knn"  
 )  
  
#Prediction  
churn\_knn\_pred <- predict(  
 churn\_knn,   
 newdata = churn\_test  
 )  
  
# Model Evaluation - Confusion Matrix   
knn\_confmat <- confusionMatrix(  
 churn\_knn\_pred,   
 as.factor(churn\_test$Churn)  
 )

### Resampled KNN-Classification

#Defining a resampling method with 10 fold cross-validation and repeasting 3 times  
knn\_control <- trainControl(  
 method = "repeatedcv",  
 number = 10,  
 repeats = 3,  
 summaryFunction = defaultSummary,  
 classProbs = TRUE,  
 verboseIter = FALSE  
)  
  
# fit resampled KNN-Model   
improved\_churn\_knn <- train(  
 Churn ~ .,  
 data = churn\_train,  
 method = "knn",  
 trControl = knn\_control,  
 preProcess = c("center","scale"),  
 tuneLength = 10  
 )  
  
# Prediction   
knn\_pred <- predict(  
 improved\_churn\_knn,   
 newdata = churn\_test  
 )  
  
# Model Evaluation   
improved\_confMat\_knn <- confusionMatrix(  
 knn\_pred,   
 churn\_test$Churn  
 )

### Tuned KNN-Classification

# Create a resampling method with 5-fold Cross-Validation  
fit\_control <- trainControl(  
 method = "cv",  
 number = 5,  
 summaryFunction = defaultSummary,  
 classProbs = TRUE,  
 verboseIter = FALSE  
)  
  
# Create a tuning parameter grid search  
knn\_grid <- expand.grid(  
 k = floor(seq(1, 101, length.out = 49))  
)  
  
#Prediction  
cancer\_knn\_pred <- predict(  
 bcancer\_knn,   
 newdata = test\_bcancer  
 )  
  
# Model Evaluation - confusion matrix   
cancer\_knn\_confmat <- confusionMatrix(  
 cancer\_knn\_pred,   
 test\_bcancer$Class  
 )  
  
# plot of the cross validated accuracy  
trellis.par.set(caretTheme())  
plot(bcancer\_knn, col="red")

library(mlbench)  
set.seed(1337)   
  
#loading breastcancer data from mlbench  
data(BreastCancer)  
  
#Assigning BreastCancer data to bcancer\_data variable  
bcancer\_data <- BreastCancer  
  
#percentage of class  
bcancer\_data %>%   
 count(Class) %>%  
 group\_by(Class) %>%  
 summarize(percent\_dis= round((n / 569)\* 100, 2))  
  
bcancer\_data <- na.omit(bcancer\_data)  
bcancer\_data <- bcancer\_data[-1]  
  
# reproducible partition  
set.seed(1337)  
  
#defining index for the partition  
index = sample(2, nrow(bcancer\_data), replace=TRUE, prob=c(0.70,0.30))  
  
#Train and test split  
train\_bcancer <- bcancer\_data[index==1,]  
test\_bcancer <- bcancer\_data[index==2,] # your tibble  
  
library(caret)  
set.seed(1337)  
  
# Create a resampling method with 5-fold Cross-Validation  
fit\_control <- trainControl(  
 method = "cv",  
 number = 5,  
 summaryFunction = defaultSummary,  
 classProbs = TRUE,  
 verboseIter = FALSE  
)  
  
# Create a tuning parameter grid search  
knn\_grid <- expand.grid(  
 k = floor(seq(1, 101, length.out = 49))  
)  
  
# Fit knn model and perform grid search  
bcancer\_knn <- train(  
 Class ~ .,  
 data = train\_bcancer,  
 method = "knn",  
 trControl = fit\_control,  
 tuneGrid = knn\_grid,  
 metric = "accuray"  
 )  
  
#Plot of the cross-validated accuracy as a function of a tuning parameter  
trellis.par.set(caretTheme())  
plot(bcancer\_knn, col="red")   
  
bcancer\_knn$finalModel  
  
cancer\_knn\_pred <- predict(  
 bcancer\_knn,   
 newdata = test\_bcancer  
 )  
  
cancer\_knn\_confmat <- confusionMatrix(  
 cancer\_knn\_pred,   
 test\_bcancer$Class  
 )

### Normal KNN

library(FNN)  
#Separating Outcome variable from the dataset  
Ozone\_outcome\_train <- trn\_data %>% select(Ozone)  
Ozone\_outcome\_test <- tst\_data %>% select(Ozone)  
  
#KNN dataset without outcome variable  
knn\_train\_data <- trn\_data %>% select(-Ozone)  
knn\_test\_data <- tst\_data %>% select(-Ozone)  
  
#Generating KNN predictions  
mod\_knn <- knn.reg(knn\_train\_data, test = knn\_test\_data , Ozone\_outcome\_train , k = 5) %>% head()  
print(head(mod\_knn$pred))

## Logistic

### Cancer Detection with Logistic Regression

library(statistics4ds)  
library(tidyverse)  
library(plotly)  
  
wisc\_trn <- read.csv('data/wisc-trn.csv')  
wisc\_tst <- read.csv('data/wisc-tst.csv')  
  
model\_glm = glm(class ~ radius + symmetry, data = wisc\_trn, family = "binomial")  
  
# use tst for prediction!  
predglm\_c1 = ifelse(predict(model\_glm, type = "link") > 0.1 , "M", "B")  
predglm\_c2 = ifelse(predict(model\_glm, type = "link") > 0.5 , "M", "B")  
predglm\_c3 = ifelse(predict(model\_glm, type = "link") > 0.9 , "M", "B")  
  
train\_tab\_c1 = table(predicted = predglm\_c1, actual = wisc\_trn$class)  
train\_tab\_c2 = table(predicted = predglm\_c2, actual = wisc\_trn$class)  
train\_tab\_c3 = table(predicted = predglm\_c3, actual = wisc\_trn$class)  
  
  
library(caret)  
train\_con\_mat\_c1 = confusionMatrix(train\_tab\_c1, positive='M')  
train\_con\_mat\_c2 = confusionMatrix(train\_tab\_c2, positive='M')  
train\_con\_mat\_c3 = confusionMatrix(train\_tab\_c3, positive='M')  
  
result <- data.frame(  
c1 = c(train\_con\_mat\_c1$overall["Accuracy"],   
 c1= train\_con\_mat\_c1$byClass["Sensitivity"],   
 c1= train\_con\_mat\_c1$byClass["Specificity"]),  
c5 = c(train\_con\_mat\_c2$overall["Accuracy"],   
 train\_con\_mat\_c2$byClass["Sensitivity"],   
 train\_con\_mat\_c2$byClass["Specificity"]),  
c9 = c(train\_con\_mat\_c3$overall["Accuracy"],   
 train\_con\_mat\_c3$byClass["Sensitivity"],   
 train\_con\_mat\_c3$byClass["Specificity"]))  
  
knitr::kable(result)

### Weather Classification

library(caret)  
set.seed(3456)  
data(weatherAUS)  
weatherAUS\_n <- weatherAUS[sample(nrow(weatherAUS), 1000),]  
trainIndex <- createDataPartition(weatherAUS\_n$RainTomorrow, p = .75,   
 list = FALSE,   
 times = 1)  
train <- weatherAUS\_n[ trainIndex, ]# %>% select(RainTomorrow, Humidity9am, Humidity3pm) %>% na.omit()  
test <- weatherAUS\_n[-trainIndex, ] # %>% select(RainTomorrow, Humidity9am, Humidity3pm) %>% na.omit  
  
train <- weatherAUS\_n[ trainIndex, ] %>% select(RainTomorrow, Humidity9am, Humidity3pm) %>% na.omit()  
logistic\_model <- glm(RainTomorrow~., data=train, family=binomial(link='logit'))  
  
pred = predict(logistic\_model, type="response")  
accuracy <- table(pred>0.5, train$RainTomorrow)  
sum(diag(accuracy))/sum(accuracy)  
  
# Evaluation  
logistic\_pred <- predict(logistic\_model, newdata = test, type="response")  
logistic\_pred <- (logistic\_pred>0.5)   
logistic\_pred <- logistic\_pred %>% as.factor()  
levels(logistic\_pred) <- c("No", "Yes")  
xtab <- table(logistic\_pred, test$RainTomorrow)  
confusionMatrix(xtab) ## 181 38

### Weather Classification Comparison

# redefine train/test dataset  
train <- weatherAUS\_n[ trainIndex, ] %>% na.omit()  
test <- weatherAUS\_n[-trainIndex, ] %>% na.omit()  
# pair plot  
library(GGally)  
train <- within(train, rm('Date','Location','Temp9am','Temp3pm','RISK\_MM','Cloud9am','Humidity9am','Pressure9am','WindGustDir','WindDir9am','WindDir3pm','WindGustSpeed','WindSpeed9am','WindSpeed3pm'))  
test <- within(test, rm('Date','Location','Temp9am','Temp3pm','RISK\_MM','Cloud9am','Humidity9am','Pressure9am','WindGustDir','WindDir9am','WindDir3pm','WindGustSpeed','WindSpeed9am','WindSpeed3pm'))  
library(corrplot)  
correlations <- cor(train[,1:8])  
corrplot(correlations, method="circle")  
# helper function for RMSE  
calc\_err = function(actual, predicted) {  
 mean(actual != predicted)  
}  
  
  
# define flat prior  
flat = c(1, 1) / 2  
  
mod\_intercept <- glm(RainTomorrow ~ 1, data = train, family = "binomial")  
mod\_simple <- glm(RainTomorrow ~ RainToday, data = train, family = "binomial")  
mod\_multiple <- glm(RainTomorrow ~ ., data = train, family = "binomial")  
mod\_additive <- gam(RainTomorrow ~ Evaporation + MinTemp + Sunshine + Humidity3pm + Pressure3pm + RainToday, data = train, family = "binomial", na.omit=TRUE) # careful on this one!  
mod\_interaction <- glm(RainTomorrow ~ (Evaporation + MinTemp + Sunshine + Sunshine + Humidity3pm + Pressure3pm + RainToday)^2 , data = train, family = "binomial")  
  
mod\_lda <- lda(RainTomorrow ~ .,data=train)  
mod\_lda\_flat <- lda(RainTomorrow ~ ., data = train, prior = flat)  
  
mod\_nb <- naiveBayes(RainTomorrow ~ ., data = train)  
  
# model list  
mod\_list <- c('Intercept','Simple','Multiple','Additive','Interaction','LDA','LDA flat','Naive Bayes')  
  
# calculate train error  
train\_error <- c(  
 calc\_err(train$RainTomorrow, ifelse(predict(mod\_intercept, newdata = train, type = 'response') > 0.5, "Yes", "No")),  
 calc\_err(train$RainTomorrow, ifelse(predict(mod\_simple, newdata = train, type = 'response') > 0.5, "Yes", "No")),  
 calc\_err(train$RainTomorrow, ifelse(predict(mod\_multiple, newdata = train, type = 'response') > 0.5, "Yes", "No")),  
 calc\_err(train$RainTomorrow, ifelse(predict(mod\_additive, newdata = train, type = 'response') > 0.5, "Yes", "No")),  
 calc\_err(train$RainTomorrow, ifelse(predict(mod\_interaction, newdata = train, type = 'response') > 0.5, "Yes", "No")),  
 calc\_err(train$RainTomorrow, predict(mod\_lda, newdata = train)$class),  
 calc\_err(train$RainTomorrow, predict(mod\_lda\_flat, newdata = train)$class),  
 calc\_err(train$RainTomorrow, predict(mod\_nb, newdata = train))  
)  
# calculate test error  
test\_error <- c(  
 calc\_err(test$RainTomorrow, ifelse(predict(mod\_intercept, newdata = test, type = 'response') > 0.5, "Yes", "No")),  
 calc\_err(test$RainTomorrow, ifelse(predict(mod\_simple, newdata = test, type = 'response') > 0.5, "Yes", "No")),  
 calc\_err(test$RainTomorrow, ifelse(predict(mod\_multiple, newdata = test, type = 'response') > 0.5, "Yes", "No")),  
 calc\_err(test$RainTomorrow, ifelse(predict(mod\_additive, newdata = test, type = 'response') > 0.5, "Yes", "No")),  
 calc\_err(test$RainTomorrow, ifelse(predict(mod\_interaction, newdata = test, type = 'response') > 0.5, "Yes", "No")),  
 calc\_err(test$RainTomorrow, predict(mod\_lda, newdata = test)$class),  
 calc\_err(test$RainTomorrow, predict(mod\_lda\_flat, newdata = test)$class),  
 calc\_err(test$RainTomorrow, predict(mod\_nb, newdata = test))  
)  
# generate table  
  
weather\_aus\_models <- data.frame(  
 mod\_list, train\_error, test\_error  
)  
  
knitr::kable(weather\_aus\_models)  
plot\_ly(df, x = ~mod\_list, y = ~train\_error, type = 'bar', name = 'Train Error') %>%  
 add\_trace(y = ~test\_error, name = 'Test Error') %>%  
 layout(yaxis = list(title = 'Error Rate'), xaxis = list(title = 'Model'), title = 'Model Errors of Classification',  
 barmode = 'group')

### Bias-Variance Trade-off Logistic Regression

make\_sim\_data = function(n\_obs = 100) {  
 x1 = runif(n = n\_obs, min = 0, max = 2)  
 x2 = runif(n = n\_obs, min = 0, max = 4)  
 prob = exp(1 + 2 \* x1 - 1 \* x2) / (1 + exp(1 + 2 \* x1 - 1 \* x2))  
 y = rbinom(n = n\_obs, size = 1, prob = prob)  
 data.frame(y, x1, x2)  
}  
  
sim\_data = make\_sim\_data()  
  
set.seed(123456789)  
create\_simulated\_data <- function(sample\_size = 30) {  
 x\_1 = runif(n = sample\_size, min = 0, max = 2)  
 x\_2 = runif(n = sample\_size, min = 0, max = 4)  
 prob = exp(1 + 2 \* x\_1 - 1 \* x\_2) / (1 + exp(1 + 2 \* x\_1 - 1 \* x\_2))  
 y = rbinom(n = sample\_size, size = 1, prob = prob)  
 data.frame(y, x\_1, x\_2)  
}  
  
simulated\_data <- create\_simulated\_data()  
  
  
  
count\_simulations <- 2000  
count\_models <- 4  
sample\_size <- 30  
x <- data.frame(x\_1 = 0.5, x\_2 = 0.75)  
  
predictions <- matrix(0, nrow = count\_simulations, ncol = count\_models)  
  
  
for (simulation in 1:count\_simulations) {  
  
  
 simulated\_data = create\_simulated\_data(sample\_size)  
   
  
 fit\_1 = glm(y ~ 1, data = simulated\_data, family = "binomial")  
 fit\_2 = glm(y ~ ., data = simulated\_data, family = "binomial")  
 fit\_3 = glm(y ~ x\_1 \* x\_2, data = simulated\_data, family = "binomial")  
 fit\_4 = glm(y ~ x\_1 \* x\_2 + I(x\_1 ^ 2) + I(x\_1 ^ 2), data = simulated\_data, family = "binomial")  
  
 predictions[simulation, 1] = predict(fit\_1, x, type = "response")  
 predictions[simulation, 2] = predict(fit\_2, x, type = "response")  
 predictions[simulation, 3] = predict(fit\_3, x, type = "response")  
 predictions[simulation, 4] = predict(fit\_4, x, type = "response")  
}  
  
  
  
# functions from R4SL  
  
get\_var = function(estimate) {  
 mean((estimate - mean(estimate)) ^ 2)  
}  
  
get\_bias = function(estimate, truth) {  
 mean(estimate) - truth  
}  
  
get\_mse = function(truth, estimate) {  
 mean((estimate - truth) ^ 2)  
}  
  
  
  
# true funciton p(x)   
p = function(x) {  
 with(x,  
 exp(1 + 2 \* x\_1 - 1 \* x\_2) / (1 + exp(1 + 2 \* x\_1 - 1 \* x\_2))  
 )  
}  
  
p(x = x)  
  
  
# calculate bias, variance, and mse of predictions for each logistic regression  
bias = apply(predictions, 2, get\_bias, truth = p(x))  
variance = apply(predictions, 2, get\_var)  
mse = apply(predictions, 2, get\_mse, truth = p(x))  
  
# summarize results  
results = data.frame(  
 c("Intercept Only", "Additive", "Interaction", "Full Second Order"),  
 round(mse, 5),  
 round(bias ^ 2, 5),  
 round(variance, 5)  
)  
colnames(results) = c("Logistic Regression Model",   
 "Mean Squared Error",   
 "Bias Squared",   
 "Variance")  
rownames(results) = NULL  
print(results)

### Logistic-Classification

# logisitc model   
glm\_control <- trainControl(  
 method = "repeatedcv",  
 number = 10,  
 repeats = 3,  
 summaryFunction = defaultSummary,  
 classProbs = TRUE,  
 verboseIter = FALSE  
)  
  
# Fit logistic Model  
churn\_glm <- train(  
 Churn ~ .,   
 data = churn\_train,  
 method = "glm",  
 trControl = glm\_control,  
 )  
  
# Prediction   
churn\_glm\_pred <- predict(  
 churn\_glm,   
 data = churn\_test  
 )  
  
#Model Evaluation / Confusion Matrix  
confmat\_churn\_glm <- confusionMatrix(  
 churn\_knn\_pred,   
 as.factor(churn\_test$Churn)  
 )

## Neural Network Model

### NN-Classification

#Get the Boston dataset  
df = MASS::Boston  
glimpse(df)  
  
# Get the min/max value of each variables  
maxs <- apply(df, 2, max)   
mins <- apply(df, 2, min)  
scaled <- as.data.frame(scale(df, center = mins, scale = maxs - mins))  
  
#set seed for reproducible partition  
set.seed(1910837166)  
  
#defining index for the partition  
index = sample(2, nrow(scaled), replace=TRUE, prob=c(0.70,0.30))  
  
#Train and test split  
train\_boston <- scaled[index==1,]  
test\_boston <- scaled[index==2,]   
  
library(neuralnet)  
#get the variable names of the dataset  
boston\_var <- names(train\_boston)  
  
#define formula to for the neural network  
formula <- as.formula(paste("medv ~", paste(boston\_var[!boston\_var %in% "medv"], collapse = " + ")))  
  
#train data using neuralnet function  
nn\_boston <- neuralnet(  
 formula,  
 data=train\_boston,  
 hidden=c(1,10),  
 linear.output=T  
 )  
  
# Prediction   
nn\_boston\_pred<- compute(nn\_boston, test\_boston[,1:13]) #1:13 is the number of predictors  
  
# Plot neural network  
nn\_plot <- plot(nn\_boston, rep = "best")  
  
# Denormalize the data for true predictions  
nn\_boston\_pred\_ <- nn\_boston\_pred$net.result \*   
 (max(df$medv)-min(df$medv)) +   
 min(df$medv)  
  
test\_boston\_r <- (test\_boston$medv) \*   
 (max(df$medv)-min(df$medv)) +   
 min(df$medv)  
  
#Calculate MSE   
nn\_sc\_mse <- sum((test\_boston\_r - nn\_boston\_pred\_)^2)/nrow(test\_boston)

## OLS-Model

### Linear

#fit lm model   
m\_mod <- lm(medv ~., data = train\_boston)  
  
#prediction   
lm\_pred <- predict(lm\_mod, test\_boston)  
  
# Model Evaluation - rmse  
lm\_rmse <- sum((lm\_pred - test\_boston$medv)^2)/nrow(test\_boston)

### GLM

# fit GLM model  
glm\_mod<- glm(medv ~ ., data = train\_boston)  
  
# Prediction  
glm\_pred <- predict(glm\_mod, test\_boston)  
  
# M  
glm\_rmse <- sum((lm\_pred - test\_boston$medv)^2)/nrow(test\_boston)

## Tree-Based Models

### Random Forest

#### Tuned Random Forest

# defining grid search mtry = 1 to 10  
rf\_grid <- expand.grid(mtry=(1:10))  
  
#rf fit control 5 fold cross validation  
rf\_control <- trainControl(  
 method = "cv",  
 number = 5)  
  
# fit the rf model   
fit\_rf <- train(as.factor(Class) ~ .,   
 data = train\_bcancer,   
 method = "rf",   
 metric = "Accuracy",   
 tuneGrid= rf\_grid,  
 trControl = rf\_control)  
  
#predict  
rf\_pred <- predict(fit\_rf, newdata = test\_bcancer)  
  
#confusion matrix  
rf\_bcancer\_confmat <- confusionMatrix(  
 rf\_pred,  
 test\_bcancer$Class  
 )  
  
#Fit final model  
fit\_rf$finalModel  
  
#plot the rf model  
plot(fit\_rf)

## Logistic Regression

### Tuned Logistic

#Train a logistic model using 5-k folds cross validation resample method  
glm\_cv\_time = system.time({  
 sim\_glm\_cv = train(  
 class ~ .,  
 data = sim\_trn,  
 trControl = cv\_5,  
 method = "glm")  
})

## Support Vector Machine

### SVM-Linear

library(rsample)  
uin = 1910837166  
set.seed(uin)  
oj\_split = initial\_split(OJ, p = 0.5)  
oj\_trn = oj\_split %>% training()  
oj\_tst = oj\_split %>% testing()  
  
#cost parameter for the SVM model  
cost\_param <- (C = c(2 ^ (-5:5)))  
  
#Linear kernerl grid  
lin\_grid = expand.grid(C = c(2 ^ (-5:5)))  
  
# 5-k folds cross validation to control when fitting svm model  
fit\_control = trainControl(method="cv", number=5)  
  
#Fit SVM model with linear kernel   
svm\_linear <- train(  
 Purchase ~ .,  
 data = oj\_trn,  
 method = 'svmLinear',  
 trControl = fit\_control,  
 preProcess = c('center', 'scale'),  
 tuneGrid = lin\_grid  
 )  
  
#Predict  
svm\_lin\_pred <- predict(svm\_linear, oj\_tst)  
  
#accuracy of the SVM model with linear kernel   
svm\_lin\_acc <- mean(svm\_lin\_pred == oj\_tst$Purchase)  
  
#Best tune   
svm\_linear\_best <- svm\_linear$bestTune

### SVM-Polynomial

library(rsample)  
uin = 1910837166  
set.seed(uin)  
oj\_split = initial\_split(OJ, p = 0.5)  
oj\_trn = oj\_split %>% training()  
oj\_tst = oj\_split %>% testing()  
  
#cost parameter for the SVM model  
cost\_param <- (C = c(2 ^ (-5:5)))  
  
#Linear kernerl grid  
lin\_grid = expand.grid(C = c(2 ^ (-5:5)))  
  
#5k folds cross-validation resampling method  
fit\_control = trainControl(method="cv", number=5)  
  
#fit the svm model with polynomial kernel without tuning it  
svm\_poly\_1 <- train(  
 Purchase ~ .,  
 data = oj\_trn,  
 method = 'svmPoly',  
 trControl = fit\_control,  
 preProcess = c('center', 'scale')  
 )  
  
#Predict  
svm\_poly\_pred <- predict(svm\_poly\_1, oj\_tst)  
  
#accuracy of the svm model with polynomial kernel without tuning  
svm\_poly\_acc <- mean(svm\_poly\_pred == oj\_tst$Purchase)  
  
#Best tune   
svm\_poly\_1\_best <- svm\_poly\_1$bestTune

### SVM-Radial

library(rsample)  
uin = 1910837166  
set.seed(uin)  
oj\_split = initial\_split(OJ, p = 0.5)  
oj\_trn = oj\_split %>% training()  
oj\_tst = oj\_split %>% testing()  
  
#cost parameter for the SVM model  
cost\_param <- (C = c(2 ^ (-5:5)))  
  
#Linear kernerl grid  
lin\_grid = expand.grid(C = c(2 ^ (-5:5)))  
  
#defining grid search for svm model with radial kernel  
rad\_grid = expand.grid(C = c(2 ^ (-2:3)), sigma = c(2 ^ (-3:1)))   
  
#5k folds cross-validation resampling method will be applied  
fit\_control = trainControl(method="cv", number=5)  
  
#fit the svm model with radian kernel  
svm\_radial <- train(  
 Purchase ~ .,  
 data = oj\_trn,  
 method = "svmRadial",  
 trControl = fit\_control,  
 preProcess = c('center', 'scale'),  
 tuneGrid = rad\_grid  
)  
  
#predict  
svm\_radial\_pred <- predict(svm\_radial, oj\_tst)  
  
# get the accuracy in the test data   
svm\_radial\_acc <- mean(svm\_radial\_pred == oj\_tst$Purchase)  
  
  
#Best tune  
svm\_radial\_best <- svm\_radial$bestTune

## Random Forest

### Random Forest with OBB resampling method

set.seed(1910837166)  
  
#defining grid search   
rf\_oob\_grid <- expand.grid(mtry = 1:(ncol(hit\_trn) - 1))  
  
#OBB resampling method to fit control the model  
fit\_control <- trainControl(method='oob')  
  
#fit RF model with log tranforming Salary  
hitter\_rflog\_model <- train(  
 log(Salary) ~ .,   
 data = hit\_trn,   
 method = 'rf',   
 tuneGrid= rf\_oob\_grid,   
 trControl = fit\_control,  
 importance = TRUE,  
 verbose = FALSE  
 )  
  
log\_rf\_rmse <- RMSE(  
 predict(hitter\_rflog\_model,hit\_tst),  
 hit\_tst$Salary)  
  
log\_rf\_resamp\_rmse <- min(hitter\_rflog\_model$results$RMSE)  
  
#Plotting the data  
hitter\_rflog\_imp <- varImp(hitter\_rflog\_model)  
plot(hitter\_rflog\_imp)  
  
#tuning parameter  
plot(hitter\_gbm\_model, main = "Gradient Boosting: Error vs Number of Trees")  
  
# Variable important gradient boosting  
summary(hitter\_gbm\_model, main = "Boosting Variable Importance")

#5k fold cross validation  
rf\_control = trainControl(method="cv", number=5)  
  
#  
mtry = floor((ncol(oj\_trn)-1)/3)  
rf\_grid = expand.grid(mtry=(mtry))  
  
rf\_model <- train(  
 Purchase ~ .,  
 data = oj\_trn,  
 method = "rf",  
 trControl = rf\_control,  
 preProcess = c('center', 'scale'),  
 tuneGrid = rf\_grid  
)  
  
#predict   
rf\_model\_pred <- predict(rf\_model, oj\_tst)  
  
# get the accuracy in the test data  
rf\_acc <- mean(rf\_model\_pred == oj\_tst$Purchase)  
  
#best tune   
rf\_model\_best <- rf\_model$bestTune

## K-Means Clustering

set.seed(1910837166)  
  
#Defining k ranging from 1 to 15  
k\_max = 15  
  
#calculate tot.withinss of the k's   
optimal\_k <- sapply(  
 1:k\_max,  
 function(k) kmeans(  
 clust\_data,  
 centers = k,  
 nstart = 10)$tot.withinss  
)  
  
#store tot.withinss as dataframe  
tot\_withinss <- as.data.frame(optimal\_k)  
  
  
#plot the result to find the optimal k  
cluster\_plot <- plot(  
 1:k\_max,  
 optimal\_k,  
 xlab = "Number of clusters k",  
 ylab = "Total WSS",  
 main = "Optimal number of clusters"  
   
)  
abline(  
 v=4,   
 lty=2,   
 col = "red"  
 )  
  
#get the value of the tot.withinss  
kmeans\_4 <- kmeans(clust\_data,4)  
tot\_withinss\_4 <- kmeans\_4$tot.withinss  
  
# Get the observation size of each clusters  
cluster\_sizes <- kmeans\_4$size  
  
  
library(cluster)  
library(fpc)  
#Fetch the first 2 variables   
two\_vars <- clust\_data[, -(3:50)]   
  
#plot result and the 4 clusters defined earlier  
plot\_cluster <- plotcluster(  
 two\_vars,  
 kmeans\_4$cluster)  
  
  
  
# Verwenden Sie PCA, um diese Daten zu visualisieren  
plot\_cluster\_PCA <- clusplot(  
 clust\_data,  
 kmeans\_4$cluster,  
 color = TRUE,  
 col.p = kmeans\_4$cluster,  
 lines =0)  
  
# How many components we need in PCA to explain 95% of the variance  
pca\_clust <- prcomp(clust\_data)  
clust\_data\_var <- cumsum(pca\_clust$sdev ^ 2 / sum(pca\_clust$sdev ^ 2))

#Cluster of datapoints ingnoring their known classes  
wdbc\_cluster <- wdbc[-1]  
  
# Berechne die Ratio der Sum of Squares (Innerhalb/Zwischen Clustern) für k k=1,2,3,4,5,6,7  
kmeans\_7 <- kmeans(wdbc\_cluster, 7)  
ratio <- kmeans\_7$tot.withinss / kmeans\_7$betweenss  
print(ratio)  
  
#Interpretation: The ratio of sum-of-squares within compared to the sum-of-squares between the clusters is approximately 0.04287464. Since we have 7 clusters, 0.7 would be expected with separated clusters. But at 0.04287464 we cannot really expect a completey separated clusters.   
  
# Erstelle einen Screeplot - Wo liegt der "Ellbogen"  
#Computes the cluster SS  
kmean\_withinss <- function(k) {  
 cluster <- kmeans(wdbc\_cluster, k)  
 return (cluster$tot.withinss)  
}  
max\_k <-7 # Set maximum cluster  
wss <- sapply(2:max\_k, kmean\_withinss) # Run algorithm over a range of k  
elbow <- data.frame(2:max\_k, wss) # Create a data frame to plot the graph  
  
#Screeplot   
ggplot(elbow, aes(x = X2.max\_k, y = wss)) +  
 geom\_point() +  
 geom\_line() +  
 geom\_abline(h = 0.5, col="red", lty=100) +   
 scale\_x\_continuous(breaks = seq(1, 7, by = 1))  
  
#Interactive graph  
#The Elbow is at 4  
  
#Here just want to play around to take a look at the 7 clusters in relation to the principal component of the less correlated predictors  
ggplot(as.data.frame(wdbc.pr\_cor$x), aes(x=PC1, y=PC2, color=as.factor(kmeans\_7$cluster), shape = wdbc.pr\_cor$class)) +  
 geom\_point( alpha = 0.6, size = 3) +  
 theme\_minimal()+  
 theme(legend.position = "bottom") +  
 labs(title = "K-Means clusters against PCA", x = "PC1", y = "PC2", color = "Cluster", shape = "class")  
  
# noch einmal für scalierte daten  
wdbc\_cluster\_scaled <- scale(wdbc[-1])  
  
# k = 7 , scaled data  
wdbc\_kmeansscaled\_7 <- kmeans(wdbc\_cluster\_scaled,7)  
ratio <- wdbc\_kmeansscaled\_7$tot.withinss / wdbc\_kmeansscaled\_7$betweenss  
print(ratio)  
  
# k = 2, scaled data  
wdbc\_kmeansscaled\_2 <- kmeans(wdbc\_cluster\_scaled,2)  
ratio\_2 <- wdbc\_kmeansscaled\_2$tot.withinss / wdbc\_kmeansscaled\_2$betweenss  
print(ratio\_2)

clust\_data <- read\_csv("clust\_data.csv")  
  
#I am doing the anlysis with scaled data  
clust\_data <- scale(clust\_data)  
  
set.seed(1910837166)  
  
#Defining k ranging from 1 to 15  
k\_max = 15  
  
#calculate tot.withinss of the k's   
optimal\_k <- sapply(  
 1:k\_max,  
 function(k) kmeans(  
 clust\_data,  
 centers = k,  
 nstart = 10)$tot.withinss  
)  
  
#store tot.withinss as dataframe  
tot\_withinss <- as.data.frame(optimal\_k)  
  
  
#plot the result to find the optimal k  
plot(  
 1:k\_max,  
 optimal\_k,  
 xlab = "Number of clusters k",  
 ylab = "Total WSS",  
 main = "Optimal number of clusters"  
   
)  
abline(  
 v=4,   
 lty=2,   
 col = "red"  
 )  
  
#calculating tot.withins  
kmeans\_4 <- kmeans(clust\_data,4)  
tot\_withinss\_4 <- kmeans\_4$tot.withinss   
  
#determining cluster size  
cluster\_sizes <- kmeans\_4$size # your cluster sizes  
  
  
library(cluster)  
library(fpc)  
#Fetch the first 2 variables   
two\_vars <- clust\_data[, -(3:50)]   
  
#plot result and the 4 clusters defined earlier  
plotcluster(  
 two\_vars,  
 kmeans\_4$cluster)  
  
  
#another plot  
library(cluster)  
clusplot(  
 clust\_data,  
 kmeans\_4$cluster,  
 color = TRUE,  
 col.p = kmeans\_4$cluster,  
 lines =0)  
  
#Calculation of the variation  
pca\_clust <- prcomp(clust\_data)  
clust\_data\_var <- cumsum(pca\_clust$sdev ^ 2 / sum(pca\_clust$sdev ^ 2))

## Hierarchichal Clustering

#dataset  
us\_arrest <- USArrests  
  
#  
#Get Euclidean distance of the dataset  
us\_arrest\_dist <- dist(us\_arrest, method="euclidean")  
  
#create a "plain" hclust()-model  
us\_arrest\_hclust <- hclust(  
 us\_arrest\_dist,   
 method = "average"  
 )  
  
#plot the result   
pplot\_hierarchy <- plot(  
 us\_arrest\_hclust,   
 hang = -1,   
 cex = 0.2,  
 main = "Euclidean Average, Cutree = 4" # also good options "Euclidean Ward.D2, Cutree = 4"  
 )  
  
rect.hclust(  
 us\_arrest\_hclust,   
 k = 4,   
 border = c("red","green", "blue", "orange"),   
 which = c(1, 2, 3, 4)  
 )

library(dendextend)  
#Recode class to numeric and convert the data into matrix   
wdbc\_data <- as.matrix(mutate(wdbc, recoded\_class = as.numeric(wdbc$class == "M"))[,2:12])  
  
#Scale the Winsconsin data  
wdbc\_scaled <- scale(wdbc\_data)  
  
# distance matrix - (we use euclidean)  
wdbc\_distance <- dist(wdbc\_scaled, method = "euclidean")  
  
# create a "plain" hclust()-model for both matrices  
wdbc\_hclust <- hclust(wdbc\_distance, method = "complete")  
  
#Interactive Graph  
wdbc\_hclust\_cut <- cutree(wdbc\_hclust , 4)  
plot(wdbc\_hclust, labels = wdbc$class, hang = -1, cex = 0.2,main = "Cutree (k = 4)" )  
rect.hclust(wdbc\_hclust , k = 4, border = c("red","green", "blue", "orange"), which = c(1, 2, 3, 4))  
abline(h = 18, col = 'red', lwd=3, lty=2)

Another Hierarchical Case

us\_arrest\_sc <- scale(us\_arrest)  
  
us\_arrest\_sc\_dist <- dist(  
 us\_arrest\_sc,  
 method="euclidean"  
 )   
  
us\_arrest\_sc\_hclust <- hclust(  
 us\_arrest\_sc\_dist,  
 method="ward.D2"  
)  
  
plot(  
 us\_arrest\_sc\_hclust,   
 hang = -1,   
 cex = 0.2,  
 main = "Euclidean Ward.D2, Cutree = 4"   
 )  
  
rect.hclust(  
 us\_arrest\_sc\_hclust,   
 k = 4,   
 border = c("red","green", "blue", "orange"),   
 which = c(1, 2, 3, 4)  
 )

## Principal Component Analysis (PCA)

pacman::p\_load(tidyverse, ggfortify)  
library(dplyr)  
library(matrixStats)  
library(corrplot)  
library(factoextra)  
library(caret)  
library(GGally)  
library(ISLR)  
  
#Reading CSV dataset  
wdbc <- read\_csv(file = "data/wisc.csv")  
  
#Explorative Data Analysis  
#Count of B  
b <- sum(wdbc$class == "B")  
  
#count og M  
m <- sum(wdbc$class == "M")  
  
#Mean of each numerical columns  
num\_var\_mean <- apply(wdbc[,2:11],2,mean)  
  
#Standard deviation for each of the numeric columns  
num\_var\_sd <- apply(wdbc[,2:11],2,sd)  
  
#correlation matrix of the wisconsin breast cancer data  
corrplot(cor(wdbc[2:11]), method = "circle", diag = F, type = "lower", tl.cex = 0.7)   
  
# PCA Modelling  
wdbc.princomp <- princomp(~ radius + texture + perimeter +area + smoothness + compactness + concavity + concave + symmetry + fractal, data = wdbc)  
  
#Interpretation: The graph further shows that the first component has a very high standard deviation compared to the second component which is influenced by outliers. The first component has already account 100% variation in the data.   
biplot(wdbc.princomp)   
  
# The screeplot of the unsclaed data shows that only component 1 has 100% reflection of the variation in the data while the rest of the components show contant forming a horizontal line.   
screeplot(wdbc.princomp, type = "l", npcs = 10)  
abline(h = 1, col="red", lty=5)  
  
#PCA with correlation matrix  
dbc.prcomp <- prcomp(as.matrix(select\_if(wdbc, is.numeric)),center = TRUE, scale = TRUE)  
  
#screeplot again  
screeplot(wdbc.prcomp, type = "l", npcs = 10)  
abline(h = 1, col="red", lty=10)  
  
# Interactive Spiral graph   
fviz\_pca\_ind(wdbc.prcomp, geom.ind = "point", pointshape = 21,   
 pointsize = 2,   
 fill.ind = wdbc$class,   
 col.ind = "red",   
 palette = "jco",   
 addEllipses = TRUE,  
 label = "var",  
 col.var = "red",  
 repel = TRUE,  
 legend.title = "Class") +  
 theme(plot.title = element\_text(hjust = 0.5))   
  
# Calculation of the expplained variance  
wdbc\_var <- cumsum(wdbc.prcomp$sdev ^ 2 / sum(wdbc.prcomp$sdev ^ 2))   
  
#remove highly correlated predictors  
wdbc\_non\_cat <- wdbc[!names(wdbc) %in% "class"]  
cor <- cor(wdbc\_non\_cat)  
cut <- findCorrelation(cor > .95)  
rem <- colnames(cor)[cut]  
wdbc\_non\_highcor <- wdbc\_non\_cat[!names(wdbc\_non\_cat) %in% rem]  
wdbc.pr\_cor <- prcomp(wdbc\_non\_highcor,center = T, scale = T, )  
summary(wdbc.pr\_cor)  
  
#Screeplot for the high correlated predictors  
screeplot(wdbc.pr\_cor, type = "l", npcs = 8)  
abline(h = 0.5, col="red", lty=100)  
  
wdbc.princompcor <- princomp(wdbc\_non\_highcor )  
loadings(wdbc.princompcor)  
  
#Plotten wir die ersten beiden Hauptkomponenten als Scatterplot und “färben” nach der Klasse. Sind die Klassen separiert?  
fviz\_pca\_ind(wdbc.princompcor, geom.ind = "point", pointshape = 21,   
 pointsize = 2,   
 fill.ind = wdbc$class,   
 col.ind = "red",   
 palette = "jco",   
 addEllipses = TRUE,  
 label = "var",  
 col.var = "red",  
 repel = TRUE,  
 legend.title = "Class") +  
 theme(plot.title = element\_text(hjust = 0.5))

## Tidy Models (Ridge, Lasso, RF, Random Forest)

#my selected predictors   
predictors <- names(spotify\_songs2)[12:23]  
  
#creating a new dataset so that i only have my selected predictors and classification variables for the analysis  
spotify\_songs2 <- spotify\_songs2 %>%  
 select(playlist\_genre, predictors)  
  
#I converted genres from character to factor type  
spotify\_songs2$playlist\_genre = as.factor(spotify\_songs2$playlist\_genre)  
head(spotify\_songs2,2)  
  
  
library(tidymodels)  
library(glue)  
set.seed(2013)  
  
#train test split  
train\_test\_split <-  
 rsample::initial\_split(  
 data = spotify\_songs2,   
 prop = 0.80   
 )   
  
sptfy\_tr <- train\_test\_split %>% training()   
sptfy\_te <- train\_test\_split %>% testing()   
glue("train no. of rows: {nrow(sptfy\_tr)}  
 test no. of rows: {nrow(sptfy\_te)}")  
  
  
# preprocessing the data using recipe  
sptfy\_rec <- recipe(playlist\_genre ~ ., data = sptfy\_tr) %>%  
 step\_normalize(all\_numeric()) %>%  
 prep(data = sptfy\_tr)  
  
sptfy\_tr\_ready <- bake(sptfy\_rec, new\_data = sptfy\_tr)  
sptfy\_te\_ready <- bake(sptfy\_rec, new\_data = sptfy\_te )  
  
#Make sure that the data are normalize. It should now be mean=0 and variance=1  
glue("mean energy orig training: {format(mean(sptfy\_tr$energy))}", ", sd: {format(sd(sptfy\_tr$energy))}  
 mean energy bake training: {format(round(mean(sptfy\_tr\_ready$energy),1))}", ", sd: {format(round(sd(sptfy\_tr\_ready$energy),1))}")   
  
glue("mean of energy in orig training: {format(mean(sptfy\_te$energy))}", ", sd: {format(sd(sptfy\_te$energy))}  
 mean of energy in bake training: {format(round(mean(sptfy\_te\_ready$energy),1))}", ", sd: {format(round(sd(sptfy\_te\_ready$energy),1))}")   
  
  
#logistic model with ridge penalty spec,   
  
# fit logistic model, having playlist\_genre as independent variable of classification  
  
mod\_l1\_spec <- logistic\_reg(mixture=0, penalty=.001) %>% set\_engine("glmnet")  
  
mod\_l1 <- mod\_l1\_spec %>%  
 fit(as.factor(playlist\_genre) ~ ., data = sptfy\_tr\_ready)  
  
ridge <- mod\_l1 %>%  
 predict(new\_data = sptfy\_te\_ready, penalty = 0.001, type="prob") %>%  
 mutate(  
 truth = as.factor(sptfy\_te\_ready$playlist\_genre),   
 method = "Ridge"  
 )  
  
  
#logistic model with lasso lasso penalty spec  
#fit logistic model, having playlist\_genre as independent variable of classification  
mod\_l2\_spec <- logistic\_reg(mixture=1, penalty = .001) %>% set\_engine("glmnet")   
  
mod\_l2 <- mod\_l2\_spec %>%   
 fit(as.factor(playlist\_genre) ~ ., data = sptfy\_tr\_ready)  
  
lasso <- mod\_l2 %>%  
 predict(new\_data = sptfy\_te\_ready, penalty = 0.001, type="prob") %>%  
 mutate(  
 truth = as.factor(sptfy\_te\_ready$playlist\_genre),  
 method = "Lasso"  
 )  
  
# random forest classification model spec   
  
#fit random forest model, having playlist\_genre as independent variable of classification  
  
mod\_rf\_spec <- rand\_forest(mode="classification", mtry=4, trees =100, min\_n=50) %>%  
 set\_engine("randomForest")   
  
  
mod\_rf <- mod\_rf\_spec %>%   
 fit(as.factor(playlist\_genre) ~ ., data = sptfy\_tr\_ready)  
  
rf <- mod\_rf %>%  
 predict(new\_data = sptfy\_te\_ready, type="prob") %>%  
 mutate(  
 truth = as.factor(sptfy\_te\_ready$playlist\_genre),  
 method = "RF"  
 )  
  
  
# SVM Specs   
#fit SVM model, having playlist\_genre as independent variable of classification  
mod\_ksvm\_spec <- svm\_rbf(rbf\_sigma = 0.1) %>%  
 set\_engine("kernlab") %>%   
 set\_mode("classification")  
  
  
mod\_ksvm <- mod\_ksvm\_spec %>%   
 fit(as.factor(playlist\_genre) ~ ., data = sptfy\_tr\_ready)   
  
svm <- mod\_ksvm %>%  
 predict(new\_data = sptfy\_te\_ready, type="prob") %>%  
 mutate(  
 truth = as.factor(sptfy\_te\_ready$playlist\_genre),  
 method = "SVM"  
 )  
  
results\_test <- bind\_rows(ridge, lasso, rf, svm)  
  
  
results\_test %>%  
 group\_by(method) %>%  
 yardstick::roc\_auc(truth = truth, .pred\_RnB)   
  
  
# ROC Graph   
ridge <- roc\_curve((results\_test %>% group\_by(method) %>%   
 filter(method=="Ridge")), truth = truth, estimate = .pred\_RnB) %>%  
 ggplot(aes(x = 1 - specificity, y = sensitivity, color = "orange")) +  
 geom\_path() +  
 geom\_abline(lty = 3) +  
 coord\_equal() +  
 ggtitle("Ridge: ROC - 90.3%")   
   
  
lasso <- roc\_curve((results\_test %>% group\_by(method) %>%   
 filter(method=="Lasso")), truth = truth, estimate = .pred\_RnB) %>%  
 ggplot(aes(x = 1 - specificity, y=sensitivity, color = "orange"))+  
 geom\_path() +  
 geom\_abline(lty = 3) +  
 coord\_equal() +  
 ggtitle("Lasso: ROC - 90.3%")  
  
RF <- roc\_curve((results\_test %>%   
 group\_by(method) %>%  
 filter(method=="RF")), truth = truth, estimate = .pred\_RnB) %>%  
 ggplot(aes(x = 1 - specificity, y=sensitivity, color = "orange")) +  
 geom\_path() +  
 geom\_abline(lty = 3) +  
 coord\_equal() +  
 ggtitle("RF: ROC - 93.7%")  
  
SVM <- roc\_curve((results\_test %>%   
 group\_by(method) %>%  
 filter(method=="SVM")), truth = truth, estimate = .pred\_RnB) %>%  
 ggplot(aes(x = 1 - specificity, y=sensitivity, color = "orange")) +  
 geom\_path() +  
 geom\_abline(lty = 3) +  
 coord\_equal() +  
 ggtitle("SVM: ROC - 92.8%")  
  
  
library(gridExtra)  
grid.arrange(ridge, lasso, ncol=2, nrow=1)

## Competition Submission ( Tree-Based Models)

### Classification

# suggested packages  
pacman::p\_load(  
MASS,  
tidymodels,#caret  
tidyverse,  
knitr,  
kableExtra,  
mlbench,  
ISLR,  
ellipse,  
randomForest,  
gbm,  
glmnet,  
rpart,  
rpart.plot,  
klaR,  
gam,  
corrplot,  
e1071,  
ggplot2,  
caret,  
xray)  
  
# CLASSIFICATION   
  
#reading Dataset   
class\_trn = read\_csv("class-trn.csv")  
class\_tst = read\_csv("class-tst.csv")  
  
#EDA   
xray::distributions(class\_trn)  
  
#Train-test split  
train\_test\_split <- initial\_split(  
 data = class\_trn,  
 prop = 0.8  
)  
  
class\_trn\_train <- train\_test\_split %>% training()  
class\_trn\_test <- train\_test\_split %>% testing()  
  
#Modelling   
  
#fit model classical random forest model   
class\_rf\_model <- randomForest( as.factor(y) ~ .,data = class\_trn\_train)  
#predict  
class\_rf\_model\_pred <- predict(class\_rf\_model, class\_trn\_test)  
#confusionmatrix  
class\_confMat <- confusionMatrix(class\_rf\_model\_pred, as.factor(class\_trn\_test$y))  
  
  
# Tuned Random forest Model   
#defining mtry as the total number of predictors  
class\_rf\_grid <- expand.grid(mtry = 1:(ncol(class\_trn\_train) - 1))  
  
#resampled using cross validated method repeasted 10 times  
class\_rf\_control <- trainControl(method='cv', number=10)  
  
#model fit  
class\_rf\_model <- train(  
 as.factor(y) ~ .,  
 data = class\_trn\_train,  
 method = "rf",  
 metric = "Accuracy",  
 #preProcess = c("center", scale),   
 tuneGrid = class\_rf\_grid,  
 trControl = class\_rf\_control   
)  
#predict  
class\_rf\_pred <- predict(class\_rf\_model, class\_trn\_test)  
#confusionmatrix  
class\_tuned\_rf\_confMat <- confusionMatrix( class\_rf\_pred, as.factor(class\_trn\_test$y))  
  
  
#Tuned Boosting Model   
#defining grid search   
class\_gbm\_grid = expand.grid(  
 interaction.depth = c(1, 2),  
 n.trees = c( 1000, 1500),  
 shrinkage = c(0.001, 0.01, 0.1),  
 n.minobsinnode = 10  
 )  
  
#cross validated resampled repeated 10 times  
class\_gbm\_control = trainControl(method = "cv",number = 10)  
  
#fit the model  
class\_gbm\_model <- train(  
 y ~ .,   
 data = class\_trn\_train,   
 method = 'gbm',   
 #preProcess = c("center", scale),   
 tuneGrid= class\_gbm\_grid,  
 trControl = class\_gbm\_control,  
 verbose = FALSE  
 )  
  
#predict  
class\_gbm\_pred <- predict(class\_gbm\_model, class\_trn\_test)  
#confusion matrcix  
class\_gbm\_confMat <- confusionMatrix(class\_gbm\_pred, as.factor(class\_trn\_test$y))  
# Tuned Bagged Model   
class\_bag\_grid <- data.frame(mtry = (ncol(class\_trn\_train) - 1))  
bag\_fit\_control <- trainControl(method='cv', number = 10)  
  
class\_bag\_model <- train(  
 y ~ .,   
 data = class\_trn\_train,   
 method = 'rf',  
 #preProcess = c("center", scale),  
 tuneGrid= class\_bag\_grid,   
 trControl = bag\_fit\_control,  
 verbose = FALSE  
 )  
  
class\_bag\_pred <- predict(class\_bag\_model, class\_trn\_test)  
  
class\_bag\_confMat <- confusionMatrix(  
 class\_bag\_pred,   
 as.factor(class\_trn\_test$y)  
)  
  
# Model Comparison   
models <- c("classical RF", "Tuned RF", "Tuned Boost", "Tuned Bagged")  
summary\_accuracy <- bind\_rows(  
 a = class\_confMat$overall,   
 b = class\_tuned\_rf\_confMat$overall,   
 c = class\_gbm\_confMat$overall,  
 d = class\_bag\_confMat$overall  
 )  
  
table\_accuracy <- add\_column(summary\_accuracy, "Model" = models, .before = "Accuracy")  
table\_accuracy  
  
  
  
# Best Model   
#defining grid search   
class\_gbm\_grid = expand.grid(  
 interaction.depth = c(1, 2),  
 n.trees = c( 1000, 1500),  
 shrinkage = c(0.001, 0.01, 0.1),  
 n.minobsinnode = 10  
 )  
  
#cross validated resampled repeated 10 times  
class\_gbm\_control = trainControl(method = "cv",number = 10)  
  
#fit the model  
class\_gbm\_model <- train(  
 y ~ .,   
 data = class\_trn,   
 method = 'gbm',   
 tuneGrid= class\_gbm\_grid,  
 trControl = class\_gbm\_control,  
 verbose = FALSE  
 )  
  
# Submission  
# place code here that stores the test predictions that you submitted  
class\_pred = predict(class\_gbm\_model, newdata = class\_tst)  
write.table(class\_pred,file="class\_pred.csv",row.names = FALSE, col.names = c("Y"),sep = ",")  
  
#the methods above applies the same for regression and spam filter

## Model Comparison

### Using Data.Frame

a = "Without scaling, KNN-model final value is k = 3"  
b = "Without scaling, KNN-model has 94% accuracy"  
c = "Without scaling, KNN-model has 94% in the test data"  
d = "Scaled data, KNN-model final value is k = 1."  
e = "Scaled data, KNN-model has 93% accuracy"  
f = "Scaled data, KNN-model has 93% accuracy in the test data"  
g = "KNN better performs \*\*without\*\* scaling the data"  
h = "Without scaling, RF-model used mtry = 3, with scaled data it used mtry = 4"  
i = " The estimated probability that the 10th observation of the test data a cancerous tumor is 0% because the RF-model has correctly classified this patient to be 100% benign"  
j = "Without scaling, RF-model sensitivity in test set is approx. 95%, and 97% in train data"  
k = "without scaling, RF-model specificity in test set is approx. 94%, and 94% in train"  
l = "Random Forest is better than the KNN."  
  
results = data.frame(  
 part = LETTERS[1:12],  
 answer = c(a,b,c,d,e,f,g,h,i,j,k,l)  
)  
  
knitr::kable(results)

models\_mse <- data.frame(  
 lm\_model = lm\_mse,  
 glm\_model = glm\_mse,  
 NN\_model = nn\_sc\_mse)

### Using Tribble

summary\_results <- tribble(~model, ~method, ~accuracy,  
 "SVM", "Linear", svm\_lin\_acc,  
 "SVM", "Poly", svm\_poly\_acc,  
 "SVM", "Radial",svm\_radial\_acc,  
 "RF", "CV", rf\_acc)

### Using Bind\_Row

models <- c("classical RF", "Tuned RF", "Tuned Boost", "Tuned Bagged")  
summary\_accuracy <- bind\_rows(  
 a = class\_confMat$overall,   
 b = class\_tuned\_rf\_confMat$overall,   
 c = class\_gbm\_confMat$overall,  
 d = class\_bag\_confMat$overall  
 )  
  
table\_accuracy <- add\_column(summary\_accuracy, "Model" = models, .before = "Accuracy")  
table\_accuracy

models <- c("Logistic Regression", "KNN")  
summary\_accuracy <- bind\_rows(Logistic = confmat\_churn\_glm$overall, KNN = improved\_confMat\_knn$overall)  
table\_summary <- add\_column(summary\_accuracy, "Model" = models, .before = "Accuracy")  
data.frame(table\_summary)

### Using Kable

library(knitr)  
library(kableExtra)  
  
df <- data.frame(  
 Model = c("Linear","Model 2","Model 3","Model 4","Model 5", "KNN"),  
 RMSE\_train = c(round(rmse\_train\_1,3),  
 round(rmse\_train\_2,3),   
 round(rmse\_train\_3,3),   
 round(rmse\_train\_4,3),   
 round(rmse\_train\_5,3),   
 "none"),  
 RMSE\_test = c(rmse\_tst\_1,   
 rmse\_tst\_2,   
 rmse\_tst\_3,   
 rmse\_tst\_4,   
 rmse\_tst\_5,  
 rmse\_tst\_knn ),  
 R2\_train = c(round(r2\_trn\_1,3),   
 round(r2\_trn\_2,3),   
 round(r2\_trn\_3,3),   
 round(r2\_trn\_4,3),   
 round(r2\_trn\_5,3),   
 "none"),  
 R2\_test = c(r2\_tst\_1,   
 r2\_tst\_2,   
 r2\_tst\_3,   
 r2\_tst\_4,   
 r2\_tst\_5,  
 r2\_test\_knn)  
 )  
  
kable(df, digits = 3, align = "c", format = "markdown")

The motivation of this post is to show that data wrangling is significantly easier. DPLYR has a set of key verbs. Using these verbs, we can solve a wide range of data problems effectively in a shorter period of time. While doing all my lab exams in my statistics cours, i realized that approximately 80% of my time have been spent to data wrangling. The purpose of this document is to demonstrate how to execute the key dplyr verbs when manipulating data.

There are 6 key verbs in dplyr, there are listed in the following

filter : subset a dataframe according to condition(s) in a variable(s) select : choose a specific variable or set of variables arrange : order dataframe by index or variable group\_by : create a grouped dataframe summarise : reduce variable to summary variable (e.g. mean) mutate : transform dataframe by adding new variables

For demontration, i am using videogames dataset from kaggle.

library(tidyverse)  
  
#loading the dataset  
vgame\_data <- read\_csv("vgsales\_extended.csv")  
#head(vgame\_data)  
#summary(vgame\_data)

#Filter

#filter(vgame\_data, User\_Count > 10 & User\_Count< 81)  
#filter(vgame\_data, User\_Count == 'string')  
#vgame\_data %>% filter(User\_Count != 'string')  
#vgame\_data %>%group\_by() %>% filter(sum(User\_Count)>10)

# Select

#Selecting specific variables  
#select(vgame\_data, NA\_Sales, EU\_Sales, JP\_Sales, Other\_Sales, Global\_Sales)  
  
#Excluding specific variables  
select(vgame\_data, -Global\_Sales)

#Arrange: Order Dataframe by index or variable

#arranging the index with respect to JP\_Sales ascending  
#arrange(vgame\_data, JP\_Sales)  
  
#arrange the index with respect to JP\_Sales descending  
#arrange(vgame\_data, desc(JP\_Sales))

#Grouping

#vgame\_data %>% group\_by(group)  
  
#grouping by platform then filtering GB and DS  
#vgame\_data %>%   
# group\_by(Platform) %>%  
# filter(Platform=="GB" | Platform=="DS")  
  
#Ungrouping the dataset again   
#vgame\_data %>% ungroup()

#Summarize / aggregate dataframe by group

#taking the mean of the global sales of all platforms   
#mean\_platform <- vgame\_data %>%  
# group\_by(Platform) %>%  
# summarise(mean = mean(Global\_Sales))  
#mean\_platform  
  
#grouping two variables  
#vgame\_data %>%  
# group\_by(Platform, Genre) %>%  
# summarise(mean\_Globalsales = mean(Global\_Sales),  
# sum\_NAsales = sum(NA\_Sales),  
 # length\_pub = n())  
#

# Mutate

#vgame\_data %>% group\_by(Platform) %>% mutate(mean\_g\_sales = mean(Global\_Sales))

#Distinct

#vgame\_data %>% distinct()  
#vgame\_data %>% distinct(Platform)

# Sample

#sample\_n(vgame\_data, 100)  
#sample\_frac(vgame\_data, 0.5)

# Separating dates and filtering

#spotify\_songs\_year\_release <- spotify\_songs %>%   
# mutate(year\_of\_release = format(as.Date(track\_album\_release\_date, format="%Y-%m-%d"),"%Y")) %>%  
# arrange(desc(track\_popularity) %>% filter(track\_popularity > 97))   
#spotify\_songs\_year\_release  
  
#ggplot(data = spotify\_songs\_year\_release) +   
# geom\_line(mapping = aes(x=as.numeric(year\_of\_release), y=loudness) )

# treemap

#library("treemap")  
#top\_genre <- spotify\_songs\_3 %>% select(playlist\_genre, track\_artist, track\_popularity) %>% group\_by(playlist\_genre,track\_artist) #%>% summarise(n = n()) %>% top\_n(15, n)  
  
#tm <- treemap(top\_genre, index = c("playlist\_genre", "track\_artist"), vSize = "n", vColor = 'playlist\_genre', palette = viridis(6),title="Top 15 Track Artists within each Playlist Genre")

# Recoding element of variables

#spotify\_songs2 <- spotify\_songs %>%  
# filter(playlist\_genre %in% c("edm", "r&b")) %>%  
# mutate(playlist\_genre = recode(playlist\_genre, "r&b" = "RnB", "edm" = "EDM"))

#Imputing median

library(data.table)  
var\_num <- c('Adjusted net national income per capita (current US$)',  
 'Adjusted savings: net forest depletion (current US$)',  
 'Agricultural land (% of land area)',  
 'Agricultural methane emissions (% of total)',  
 'Agricultural nitrous oxide emissions (% of total)',  
 'CO2 emissions (kt)',  
 'Electric power consumption (kWh per capita)',  
 'Electricity production (kWh)',  
 'Forest area (% of land area)',  
 'GDP (current US$)',  
 'GDP per capita (current US$)',  
 'Organic water pollutant (BOD) emissions (kg per day)',  
 'Population (Total)',  
 'Tax revenue (% of GDP)'  
 )  
#iterrate through numeric columns in the dataset and impute median   
#for(k in names(reshaped\_african)){  
  
# if(k %in% var\_num){  
  
# # impute numeric variables with median  
# med <- median(reshaped\_african[[k]],na.rm = T)  
 # set(x = reshaped\_african, which(is.na(reshaped\_african[[k]])), k, med)  
# }  
#}