Conocimientos r

Ejecutar línea🡪control+enter

Cargar datos de csv:

getwd()

setwd("C:\\Users\\jmlozanoo\\Documents\\GitHub\\BIB-R")

*#leemos csv con header y corrigiendo vacios*

companies\_data <- read.csv("companies.csv",header = TRUE,na.strings = c(""))

Limpiar variables:

A screenshot of a computer

Description automatically generated

Comprobar datos cargados

head(companies\_data) *#primeros elementos*

summary(companies\_data) *#resumen de las variables->media y std*

str(companies\_data) *# ver valores que puede tomar*

names(companies\_data) *#nombre variables*

nrow(companies\_data) *#numero filas*

ncol(companies\_data) *#numero columnas*

Nombres columnas:

*#nombres columnas*

names(Chicago)

colnames(Chicago)

Filas con vacios

*#filas con algun atributo vacio*

companies\_data[!complete.cases(companies\_data),]

*#contamos esas filas*

nrow(companies\_data[!complete.cases(companies\_data),])

Convertir a factor un atributo:

*#convertir a factor un atributo*

companies\_data$Industry<-factor(companies\_data$Industry)

*#se muestran los valores diferentes que puede tomar*

str(companies\_data)

Valores diferentes de un factor:

*#valores diferentes del factor*

levels(companies\_data$Industry)

Valores diferentes de un atributo:

*#valores unicos de un atributo*

unicos\_estados<-unique(companies\_data$State)

unicos\_estados

*#los contamos*

cantidad\_estados <- length(unicos\_estados)

cantidad\_estados

Reemplazar cadena de texto en datos columna:

*#reemplazar cadenas*

head(companies\_data)

companies\_data$Expenses<-gsub(" Dollars","",companies\_data$Expenses)

head(companies\_data)

head(companies\_data)

*#\\ porque es especial el dolar*

companies\_data$Revenue<-gsub("\\$","",companies\_data$Revenue)

head(companies\_data)

Tipo de datos de una columna:

*#ver el tipo de una columna*

tipo\_columna <- class(companies\_data$Expenses)

tipo\_columna

Convertir a números una columna:

*#convertir a numerica una columna*

companies\_data$Expenses<-**as**.numeric(companies\_data$Expenses)

tipo\_columna <- class(companies\_data$Expenses)

tipo\_columna

Filtrar filas con un valor:

*#filtramos filas con determinado numero de empleados*

companies\_data[which(companies\_data$Employees==45),]

Quedarnos con las filas que tienen una columna totalmente informada:

*#cogemos las que no tienen el valor vacio*

companies\_data<-companies\_data[!**is**.na(companies\_data$Industry),]

nrow(companies\_data[!complete.cases(companies\_data),])

Reiniciar índices:

*#reiniciamos indices*

rownames(companies\_data)<-NULL

companies\_data

Poner en un atributo un valor filtrando:

*#ponemos en el atributo state el valor que corresponde*

companies\_data[**is**.na(companies\_data$State) & companies\_data$City=="New York","State"]<-"NY"

Ver filas:

*#vemos esas filas*

companies\_data[c(11,377),]

Mediana y media de una columna:

*#calculamos mediana de la columna. Los valores faltantes no los tiene en cuenta*

median(companies\_data[,"Employees"],na.rm=TRUE)

mean(companies\_data[,"Employees"],na.rm=TRUE)

*#media de la columna de beneficios*

mean(movies\_data$Profit)

Mediana de un atributo filtrando por otro:

*#mediana de la columna empleados para las filas de la industria retail*

med\_emp\_retail<-median(companies\_data[companies\_data$Industry=="Retail","Employees"],na.rm=TRUE)

Poner valor en atributo filtrando por atributo y si tiene valor vacio:

*#buscmos filas con empleados vacios y de la industria retail y ponemos el valor*

companies\_data[**is**.na(companies\_data$Employees) & companies\_data$Industry=="Retail",]

companies\_data[**is**.na(companies\_data$Employees) & companies\_data$Industry=="Retail","Employees"]<-med\_emp\_retail

Ver datos de una fila solo:

*#vemos la fila*

companies\_data[3,]

Completar valor a partir operaciones otras columnas:

*#el beneficio lo calculamos a partir de otros, filas que lo tienen vacio*

*#ponemos el calculo de beneficios-gastos*

companies\_data[**is**.na(companies\_data$Profit),]

companies\_data[**is**.na(companies\_data$Profit),"Profit"]<-companies\_data[**is**.na(companies\_data$Profit),"Revenue"]-companies\_data[**is**.na(companies\_data$Profit),"Expenses"]

Seleccionar filas con atributos informados:

*#nos quedamos solo con los que tienen todos los atributos informados*

companies\_data<-companies\_data[complete.cases(companies\_data),]

Instalar librería y cargar:

*#instalamos la libreria y la cargamos*

install.packages("ggplot2")

library(ggplot2)

Grafico de puntos con 4 dimensiones:

*#grafico puntos con 4 dimensiones. x,y,tamano y color*

p<-ggplot(data=companies\_data)

p+geom\_point(aes(x=Revenue,y=Expenses,colour=Industry,size=Profit))

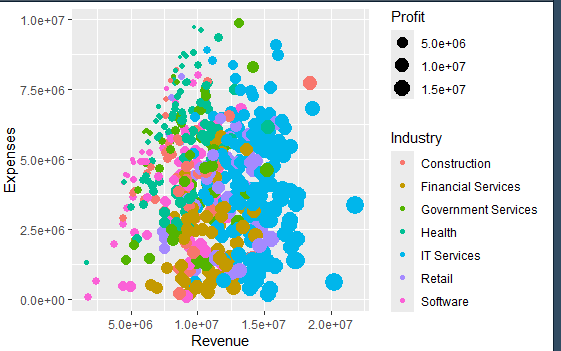


Grafico de puntos en dos grid:

*# de puntos en dos graficos*

p<-ggplot(data=movies\_data,aes(x=MediaRating,y=Profit,colour=Genre))

p+geom\_point(size=3)+facet\_grid(~aprobada)

A screenshot of a graph

Description automatically generated

Grafico con línea curva:

*#con linea curva*

d<-ggplot(data=companies\_data,aes(x=Revenue,y=Expenses,colour=Industry))

d+geom\_point()+geom\_smooth(fill=NA,size=1.2)

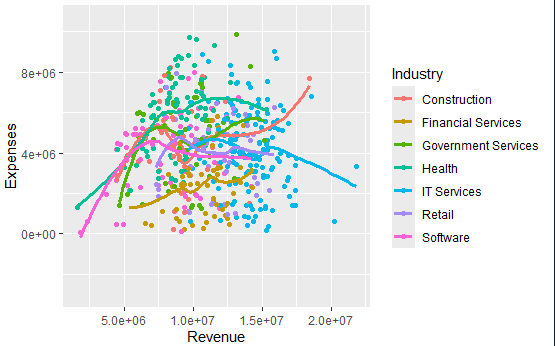


Grafico boxplot:

*#de caja y bigotes. x,y, y color*

f<-ggplot(data=companies\_data,aes(x=Industry,y=Growth,colour=Industry))

f+geom\_boxplot(size=1)

A graph with colorful squares and dots

Description automatically generated

Añadir puntos:

*#de caja con puntos*

f+geom\_jitter()+geom\_boxplot(size=1,alpha=0.5,outlier.color=NA)

A graph of different colored dots

Description automatically generated

Gráficos separando en grid:

*#separando varios graficos. x,y, color separando por industria*

p<-ggplot(data=companies\_data,aes(x=Revenue,y=Expenses,colour=State))

p+geom\_point(size=3)+facet\_grid(.~Industry)

A graph of different colored circles

Description automatically generated

Dplyr:

*#dplyr*

install.packages("dplyr")

library(dplyr)

Agrupar por atributo y calcular suma:

*#agrupa por industria, suma los beneficios y ordena descendente*

industria\_beneficios <- companies\_data %>%

group\_by(Industry) %>%

summarise(Suma\_Beneficios = sum(Profit)) %>%

arrange(desc(Suma\_Beneficios))

*#convertimos a dataframe y mostramos*

**print**(class(industria\_beneficios))

industria\_beneficios<- **as**.data.frame(industria\_beneficios)

industria\_beneficios

Agrupar por atributo y calcular media:

*#media de beneficios por studio*

library(dplyr)

movies\_data\_beneficios <- movies\_data %>%

group\_by(Studio) %>%

summarise(Media\_Beneficios = mean(Profit)) %>%

arrange(desc(Media\_Beneficios))

*#convertimos a dataframe y mostramos*

**print**(class(movies\_data\_beneficios))

movies\_data\_beneficios<- **as**.data.frame(movies\_data\_beneficios)

movies\_data\_beneficios

Obtener elemento con mayor valor:

*#studio con mayor media*

studio\_mas\_beneficios<-movies\_data\_beneficios[1,"Studio"]

**print**(paste("Studio con mayor media beneficios:", studio\_mas\_beneficios))

Buscar máximo:

*#pelicula con mayor beneficio*

fila\_max <- movies\_data[which.max(movies\_data$Profit), ]

**print**(fila\_max$MovieTitle)

Contar elementos agrupando:

*#cuenta cuantas companias son de cada industria*

frecuencia <- table(companies\_data$Industry)

frecuencia<- **as**.data.frame(frecuencia)

frecuencia

colnames(frecuencia)<-c("Industry","Cuenta")

frecuencia

Unir tablas por industria:

*#juntamos ambas tablas por la clave industria*

merged\_industry<-merge(industria\_beneficios, frecuencia, by.x="Industry", by.y="Industry")

merged\_industry

*# en el primer dataframe hay atributo Country.Code y en el segundo Code*

merged1960 <- merge(data1960, add1960, by.x="Country.Code", by.y="Code")

Insertar columna con calculo redondeando:

*#insertamos columna con calculo redondeando*

merged\_industry$Media<-round(merged\_industry$Suma\_Beneficios/merged\_industry$Cuenta,2)

merged\_industry

Grafico de barras con etiquetas:

*#beneficios por area, poniendo etiquetas*

ggplot(merged\_industry, aes(x = Industry, y = Cuenta)) +

geom\_bar(stat = "identity", fill = "skyblue") +

labs(title = "Gráfico de Barras de proporción beneficios", x = "Industria",

y = "Proporción")+

theme(plot.title = element\_text(hjust = 0.5))

A graph of blue bars

Description automatically generated

Imprimir tipo variable:

nombres\_datos<-names(movies\_data)

**print**(class(nombres\_datos)) *#imprimir tipo variable*

Tipo de una columna:

*#tipo de una columna*

**print**(class(movies\_data$Profitmill))

Contar nulos en una columna e imprimir junto a texto:

*#contamos nulos e imprimirmos en pantalla*

num\_na <- sum(**is**.na(movies\_data$Profitmill))

**print**(paste("NA en Profitmill:", num\_na))

Tener en cuenta na en la media de una columna:

*#media de la columna con 2 decimales*

media\_valor <- round(mean(movies\_data$Profitmill, na.rm = TRUE),2)

**print**(paste("Media de la columna Profitmill si na:", media\_valor))

*#si no eliminamos na, sale na*

media\_valor <- round(mean(movies\_data$Profitmill),2)

**print**(paste("Media de la columna Profitmill con na:", media\_valor))

Grafico de barras con etiquetas y cuenta en y sin definir:

*#grafico dias de estreno con etiquetas*

library(ggplot2)

p<-ggplot(data=movies\_data, aes(x=DayofWeek))

q<-p+ geom\_bar()

q <- q +

xlab("Dia de estreno") +

ylab("Cuenta") +

ggtitle("Distribución peliculas dia estreno")

q<- q+ theme(plot.title = element\_text(hjust = 0.5))

q

A graph of a number of days

Description automatically generated

Agrupar por datos y mostrar ordenados:

*#estudios con mas peliculas, contamos y ordenamos*

distribucion\_studio <- table(movies\_data$Studio)

distribucion\_studio <- sort(distribucion\_studio, decreasing = TRUE)

**print**(distribucion\_studio)

Nueva columna con cálculos de otras:

*#nueva columna con media puntuaciones*

movies\_data$MediaRating<-round((movies\_data$IMDbRating+movies\_data$MovieLensRating)/2,2)

head(movies\_data)

Eliminar columna:

*#eliminamos columna*

movies\_data$MovieLensRating<-NULL

head(movies\_data)

Añadir columna según condición:

*#añadir columna con valores segun condicion*

movies\_data <- movies\_data %>%mutate(aprobada = ifelse(MediaRating >5, "Aprobada", "No Aprobada"))

head(movies\_data)

Contar elementos agrupando y calcular porcentaje:

*#contamos aprobadas y las que no*

distribucion\_aprobadas <- table(movies\_data$aprobada)

**print**(distribucion\_aprobadas)

*#porcentaje de aprobadas*

distribucion\_aprobadas<-(distribucion\_aprobadas/nrow(movies\_data))\*100

**print**(distribucion\_aprobadas)

Tabla de frecuencias:

*#tabla de frecuencias con las notas*

intervalos <- seq(0, 10, by = 0.5)

movies\_data$IntervaloNota <- cut(movies\_data$MediaRating, breaks = intervalos,

include.lowest = TRUE)

tabla\_frecuencias <- table(movies\_data$IntervaloNota)

**print**(tabla\_frecuencias)

**print**(class(tabla\_frecuencias))

tabla\_frecuencias<-**as**.data.frame(tabla\_frecuencias)

tabla\_frecuencias

Grafico con qplot:

*#grafico rating, profit y genero*

qplot(data=movies\_data, x=MediaRating, y=Profit,

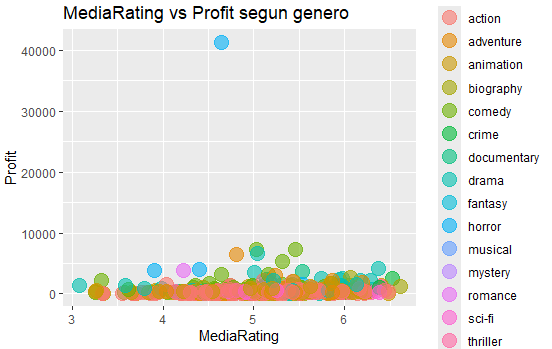
colour=Genre,

size=I(5),

alpha=I(0.6),

main="MediaRating vs Profit segun genero"

)



Filtrar filas por condiciones:

*#filtramos las que cumplen condicion*

filt\_studio <- ( (movies\_data$Studio== "WB") | (movies\_data$Studio== "Fox") | (movies\_data$Studio== "Universal") | (movies\_data$Studio== "Sony") )

filt\_nota<-movies\_data$MediaRating>6.4

*#que cumplan ambas condiciones*

filtradas <- movies\_data[filt\_studio & filt\_nota,]

filtradas

Otro formato boxplot con puntos:

*#grafico de caja y bigotes con puntos*

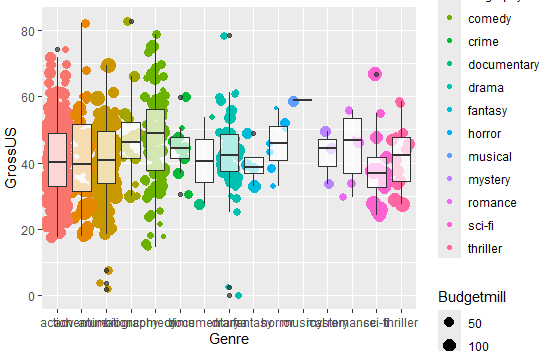
p <- ggplot(data=movies\_data, aes(x=Genre, y=GrossUS))

q<-p+

geom\_jitter(aes(size=Budgetmill,color=Genre))+

geom\_boxplot(alpha=0.7)

q



Cargar csv con nombres filas en primer valor:

*#leemos datos*

setwd("C:\\Users\\jmlozanoo\\Documents\\GitHub\\BIB-R\\Weather\_data")

Chicago<-read.csv("Chicago-F.csv",row.names=1)

NewYork<-read.csv("NewYork-F.csv",row.names=1)

Houston<-read.csv("Houston-F.csv",row.names=1)

SanFrancisco<-read.csv("SanFrancisco-F.csv",row.names=1)

Comprobar si es dataframe:

*#comprobar si es dataframe*

**is**.data.frame(Chicago)

Convertir valores a matriz:

*#conversion matriz*

Chicago<- **as**.matrix(Chicago)

NewYork<- **as**.matrix(NewYork)

Houston<- **as**.matrix(Houston)

SanFrancisco<- **as**.matrix(SanFrancisco)

Ver si un elemento es matriz:

*#ver si es matriz*

**is**.matrix(Chicago)

Crear lista agrupando matrices:

*#lista con matrices*

weather<-list(Chicago=Chicago,NewYork=NewYork,Houston=Houston,SanFrancisco=SanFrancisco)

weather

Acceder elemento de la matriz:

*#acceso elemento*

weather[1]

weather[[1]]

weather$Houston

Obtener valores de una fila:

Chicago[1,] *#obtener valores de la primera fila*

Apply, lapply y sapply:

apply: Esta función se utiliza para aplicar una función a filas o columnas de matrices o data frames. Puedes especificar si deseas aplicar la función a filas (MARGIN = 1), columnas (MARGIN = 2) o a ambos (MARGIN = c(1,2)).

lapply: Esta función aplica una función a cada elemento de una lista y devuelve una lista con los resultados. Es útil cuando deseas aplicar la misma operación a todos los elementos de una lista.

sapply: Similar a lapply, pero intenta simplificar el resultado si es posible. En lugar de devolver una lista, intenta convertir el resultado en un vector o una matriz si es apropiado. Es útil cuando quieres obtener un resultado simplificado en lugar de una lista.

Calcular media sobre los valores de las filas:

apply(Chicago,1,mean) *#calcula media sobre los valores de la primera fila*

mean(Chicago["DaysWithPrecip",]) *#calcula media sobre los valores de este campo*

Media sobre todas las filas:

*#media de las filas*

rowMeans(Chicago)

Media sobre las filas para todos los elementos de la matriz:

*#media de las filas sobre todos los elementos de la matriz*

lapply(weather,rowMeans)

Operación mas compleja:

*#otro tipo de operacion*

lapply(weather,function(z) round((z[1,]-z[2,])/z[2,],2))

resultados<-sapply(weather,function(z) round((z[1,]-z[2,])/z[2,],2))

resultados

class(resultados)

Convertir a dataframe:

*#convertimos a dataframe*

resultados<-**as**.data.frame(resultados)

resultados

Trasponer dataframe:

*#trasponemos dataframe*

library(dplyr)

resultados <- resultados %>%

mutate(mes = row.names(.))

resultados

Eliminar índices:

*#eliminamos indices*

rownames(resultados) <- NULL

resultados

Coger el máximo de cada fila:

*#cogemos el maximo*

max\_ciudades<-sapply(weather,"[",1,)

max\_ciudades<-**as**.data.frame(max\_ciudades)

max\_ciudades

Eliminar ultimo elemento vecotr:

ciudades<-colnames(max\_ciudades)

ciudades

*#eliminamos ultimo elemento*

ciudades <- head(ciudades, -1)

ciudades

Crear dataframe e ir insertando elementos:

*#creamos dataframe*

dataframe\_analisis<-data.frame(Ciudad=character(),Mes=character(),Maximo=numeric())

**for** (i **in** 1:nrow(max\_ciudades)) {

fila<-max\_ciudades[i,]*#cogemos la fila*

**for**(k **in** seq\_along(fila)){

**if**(k<5){

elemento\_anadir<-data.frame(Ciudad=ciudades[k],Mes=max\_ciudades[i,5],

Maximo=fila[[k]])

dataframe\_analisis <- rbind(dataframe\_analisis, elemento\_anadir)

}

}

}

Grafico de líneas:

*#grafico de lineas*

ggplot(dataframe\_analisis, aes(x = Mes, y = Maximo, color = Ciudad, group = Ciudad)) +

geom\_line() +

labs(title = "Temperaturas por ciudadd", x = "Mes", y = "Temperatura") +

scale\_color\_manual(values = c("SanFrancisco" = "red", "Chicago" = "blue",

"NewYork" = "green","Houston"="purple"))

A graph with different colored lines

Description automatically generated

Cargar datos poniendo nombres columnas:

*#carga de datos*

getwd()

setwd("C:\\Users\\jmlozanoo\\Documents\\GitHub\\BIB-R\\original curso")

nombres<-c("enero","febrero","marzo","abril","mayo","junio","julio","agosto","septiembre",

"octubre","noviembre","diciembre")

*#establecer nombres columnas*

datos <- read.csv("revenues-expenses.csv",header = FALSE,col.names=nombres)

head(datos)

Sumar todos los valores de una fila:

*#beneficios*

**print**(class(revenues))

**print**(class(expenses))

*#realizamos calculo*

profit <- revenues - expenses

**print**("beneficios")

profit

*#realizamos suma de todos los valores*

suma\_profits <- rowSums(profit)

mensaje\_profits<-paste("La suma de los profits es:",suma\_profits[1])

**print**(mensaje\_profits)

Completar vector con operaciones:

profit.after.tax<-NULL

*#quitando impuestos*

**for** (i **in** seq\_along(profit)) {

*#si el beneficio es negativo, los impuestos tambien son negativos y se suman.*

**if**(profit[i]<0){

profit.after.tax[i]=profit[i]+tax[i]

}**else**{

profit.after.tax[i]=profit[i]-tax[i]

}

}

profit.after.tax

Unir filas dataframes:

**print**(class(profit))

**print**(class(profit.after.tax))

**print**("Comparacion")

comparacion<-rbind(profit, profit.after.tax)*#une filas dataframes*

**print**(comparacion)

Calcular vector desde otros:

*#margen calculando desde otros vectores*

profit.margin <- round(profit.after.tax/ revenues, 2) \* 100

profit.margin

Contar elementos condicion:

*#meses con beneficios superiores a la media*

good.months <- profit.after.tax > mean\_pat

good.months *#devuelve true, false...*

cuenta\_good<-sum(good.months)

mensaje\_good<-paste("Meses superiores a la media:",cuenta\_good[1])

**print**(mensaje\_good)

Encontrar opuestos y contar:

*#meses con beneficios inferiores a la media*

bad.months <- !good.months*#opuestos a true, false...*

cuenta\_bad<-sum(bad.months)

mensaje\_bad<-paste("Meses inferiores a la media:",cuenta\_bad[1])

**print**(mensaje\_bad)

Encontrar nombre del maximo y minimo:

*#mejor mes*

mejor <- names(profit.after.tax)[which.max(profit.after.tax)]

**print**(paste("Mejor mes:", mejor))

*#peor mes*

peor <- names(profit.after.tax)[which.min(profit.after.tax)]

**print**(paste("Peor mes:", peor))

crear matriz desde datos:

*#datos inicio*

Seasons <- c("2005","2006","2007","2008","2009","2010","2011","2012","2013","2014")

Players <- c("KobeBryant","JoeJohnson","LeBronJames","CarmeloAnthony","DwightHoward","ChrisBosh","ChrisPaul","KevinDurant","DerrickRose","DwayneWade")

KobeBryant\_FT <- c(696,667,623,483,439,483,381,525,18,196)

JoeJohnson\_FT <- c(261,235,316,299,220,195,158,132,159,141)

LeBronJames\_FT <- c(601,489,549,594,593,503,387,403,439,375)

CarmeloAnthony\_FT <- c(573,459,464,371,508,507,295,425,459,189)

DwightHoward\_FT <- c(356,390,529,504,483,546,281,355,349,143)

ChrisBosh\_FT <- c(474,463,472,504,470,384,229,241,223,179)

ChrisPaul\_FT <- c(394,292,332,455,161,337,260,286,295,289)

KevinDurant\_FT <- c(209,209,391,452,756,594,431,679,703,146)

DerrickRose\_FT <- c(146,146,146,197,259,476,194,322,27,152)

DwayneWade\_FT <- c(629,432,354,590,534,494,235,308,189,284)

*#creamos la matriz*

FreeThrows <- rbind(KobeBryant\_FT, JoeJohnson\_FT, LeBronJames\_FT, CarmeloAnthony\_FT, DwightHoward\_FT, ChrisBosh\_FT, ChrisPaul\_FT, KevinDurant\_FT, DerrickRose\_FT, DwayneWade\_FT)

**print**(class(FreeThrows))

Eliminar vectoress:

*#eliminamos antiguas vectores*

rm(KobeBryant\_FT, JoeJohnson\_FT, CarmeloAnthony\_FT, DwightHoward\_FT, ChrisBosh\_FT, LeBronJames\_FT, ChrisPaul\_FT, DerrickRose\_FT, DwayneWade\_FT, KevinDurant\_FT)

Asingar nombres a las filas y columnas:

*#asignamos nombres a las filas y columnas*

colnames(FreeThrows) <- Seasons

rownames(FreeThrows) <- Players

Trasponer matriz:

FreeThrows

*#trasponer matriz*

t(FreeThrows[1:10,,drop=F])

Crear un grafico desde una funcion:

*#funcion para crear grafico*

myplot <- function(z, who=1:10) {

matplot(t(z[who,,drop=F]), type="b", pch=15:18, col=c(1:4,6),

main="Basketball Players Analysis")

legend("bottomleft",text.font = 1, cex = 0.5, inset=0.01,

legend=Players[who], col=c(1:4,6), pch=15:18, horiz=F)

}

myplot(FreeThrows)

Acceder a elementos de un dataframe:

*#acceder a elementos*

stats[3,3]

stats[3,"Birth.rate"]

stats$Internet.users*#toda la columna*

stats$Internet.users[2]

stats[,"Internet.users"]*#toda la columna*

Acceder a filas y columnas:

stats[1:10,]*#filas 1 a 10*

stats[c(4,100),] *#filas 4 y 10*

stats[,c(2,4)]*#columnas 2 y 4*

stats[,3:ncol(stats)]*#columna 3 hasta el final*

stats[3:nrow(stats),]*#filas 3 hasta final*

*#primera columna*

stats[,1,drop=F]

Filtrar filas:

*#filtro*

**print**(class(stats))

filter<-stats$Internet.users<2

filter*#devuelve true o false*

*#para obtener las filas*

stats[stats$Birth.rate>40 &stats$Internet.users<2,]

stats[stats$Country.Name=="Spain",]

Grafico boxplot con qplot:

qplot(data=stats, x=Income.Group,y=Birth.rate,geom="boxplot")

A graph with black and white lines

Description automatically generated

Grafico de puntos con qplot:

qplot(data=stats, x=Internet.users,y=Birth.rate,size=I(3),colour=Income.Group)

A graph with many colored dots

Description automatically generated

Crear dataframe desde vectores:

*#dataframe desde vectores del tipo c(2,3,4)*

**print**(class(Regions\_2012\_Dataset))

mydf<-data.frame(Countries\_2012\_Dataset,Codes\_2012\_Dataset,Regions\_2012\_Dataset)

head(mydf)

*#estableciendo los nombres de las columnas*

mydf<-data.frame(Country=Countries\_2012\_Dataset,Code=Codes\_2012\_Dataset,

Region=Regions\_2012\_Dataset)

Grafico de puntos poniendo triangulos:

*#grafico de puntos poniendo triangulos*

class(merged)

str(merged)

qplot(data=merged,x=Internet.users,y=Birth.rate,colour=Region,size=I(5),

shape=I(17),alpha=I(0.6),main="Birth rate vs Internet users")

A graph of different colored triangles

Description automatically generated

Mostrar ultimos elementos:

*#mostrar ultimos elementos*

tail(data, n=7)

Establecer nombres columnas:

*#nombres columnas*

colnames(movies)<-c("Film","Genre","CriticRating","AudienceRating","BudgetMillions","Year")

head(movies)

Crear columna calculando media dos atributos:

*#columna calculando media de dos atributos*

library(dplyr)

movies <- movies %>%

mutate(media\_col = (CriticRating + AudienceRating) / 2)

head(movies)

Histograma:

*#histograma con 2 dimensiones*

p<-ggplot(data=movies,aes(x=BudgetMillions))

p+geom\_histogram(binwidth=10,aes(fill=Genre),colour="Black")

A graph of a graph

Description automatically generated

Grafica de densidad:

*#grafica densidad*

p+geom\_density(aes(fill=Genre),position="Stack")

A graph of different colored lines

Description automatically generated

Otro tipo histograma:

*#histograma*

p<-ggplot(data=movies,aes(x=AudienceRating))

p+geom\_histogram(binwidth=10,fill="White",colour="Black")

A graph of a bar graph

Description automatically generated

Histogramas separados por atributo:

*#histogramas separados por atributo*

p<-ggplot(data=movies,aes(x=BudgetMillions))

p+geom\_histogram(binwidth=10,aes(fill=Genre),colour="Black")+facet\_grid(Genre~.,scales="free")

A graph of different colored lines

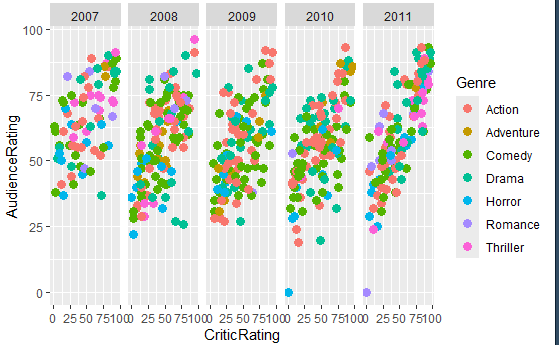
Description automatically generated

Grafico de puntos separados por atributo:

*#puntos separados por atributo*

p<-ggplot(data=movies,aes(x=CriticRating,y=AudienceRating,colour=Genre))

p+geom\_point(size=3)+facet\_grid(.~Year)



Por dos separaciones con linea curva:

*# con linea curva*

p+geom\_point(size=3)+facet\_grid(Genre~Year)+geom\_smooth()

A chart of different colored dots

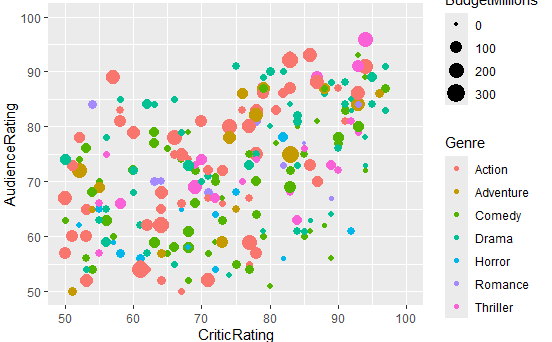
Description automatically generated with medium confidence

Limitando ejes:

*#puntos limitados*

p<-ggplot(data=movies,aes(x=CriticRating,y=AudienceRating,colour=Genre,size=BudgetMillions))

p+geom\_point()+xlim(50,100)+ylim(50,100)



Etiquetas en los graficos:

*#etiquetas*

p<-ggplot(data=movies,aes(x=BudgetMillions))

k<-p+geom\_histogram(binwidth=10,aes(fill=Genre),colour="Black")

k+xlab("Money analisis")+ylab("Number of movies")+ggtitle("Distribution")+theme(axis.title.x=element\_text(colour="DarkGreen",size=10),axis.title.y=element\_text(colour="Red",size=10),axis.text.x=element\_text(size=10),axis.text.y=element\_text(size=10),legend.title=element\_text(size=10),legend.text=element\_text(size=10),legend.position=c(1,1),legend.justification=c(1,1),plot.title = element\_text(colour="Darkblue",size=30))+theme(plot.title = element\_text(hjust = 0.5))

A graph of a distribution

Description automatically generated

Leer datos de excel:

*#leer excel*

install.packages("readxl")

library(readxl)

fin <- read\_excel("P3-Future-500-Marked-Up.xlsx")

head(fin)

Tipo de variables:

*#ver tipo de variables*

a<-c("23","33","334")

typeof(a)

Evaluar si hay na:

*#evaluar si hay na*

a<-c("112","23","45","3",NA,"33")

**is**.na(a)

Convertir fecha a posix:

*#convertimos fecha*

util$PosixTime<-**as**.POSIXct(util$Timestamp,format="%d/%m/%Y %H:%M")

head(util,12)

Reordenar columnas:

*#reodenamos columnas*

util<-util[,c(4,1,2,3)]

head(util,12)

Unir campos en un vector:

*#unimos campos en un vector*

util\_stats\_RL1<-c(min(RL1$Utilization,na.rm=T),

mean(RL1$Utilization,na.rm=T),

max(RL1$Utilization,na.rm=T))

util\_stats\_RL1

Contar filas cumplen filtro:

*#contamos filas cumplen filtro*

under\_rl1<-length(which(RL1$Utilization<0.90))

under\_rl1

Otra forma de crear lista con campos:

*#otra forma de crear una lista con campos*

list\_rl1<-list(Machine="RL1",Under\_09=under\_rl1,Stats=util\_stats\_RL1)

Acceder a elementos de la lista o añadir elementos:

*#acceso a los elementos*

list\_rl1[1]

list\_rl1[[1]]

list\_rl1$Machine

list\_rl1[[3]]

typeof(list\_rl1[[3]])

list\_rl1$Stats

list\_rl1$Stats[2]

list\_rl1[[3]][2]

*#añadir elemento*

list\_rl1[4]<-"New information"

*#anadimos elemento*

list\_rl1$Data<-RL1

Eliminar elemento de la lista:

*#eliminamos elemento*

list\_rl1[4]<-NULL

Accedemos a dos elementos:

*#accedemos dos elementos*

list\_rl1[c(1,4)]

Sublista y acceso a sus elementos:

*#sublista y su acceso*

sublist\_rl1<-list\_rl1[c("Machine","UnknownHours")]

sublist\_rl1

sublist\_rl1[[2]][2]

sublist\_rl1$UnknownHours[2]

Resumen elementos lista:

summary(list\_rl1)*#resumen de los elementos de la lista*

Grafico de lineas separando por atributo:

*#grafico de lineas separando por maquina*

p<-ggplot(data=util)

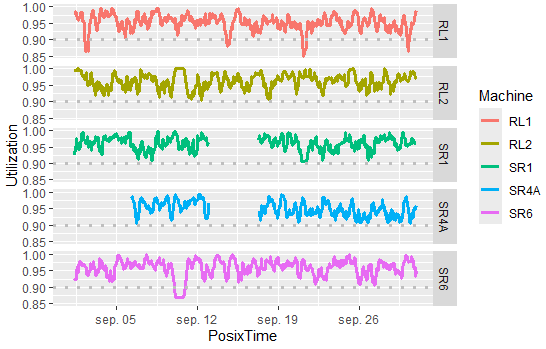
myplot<-p+geom\_line(aes(x=PosixTime,y=Utilization,colour=Machine),size=1.2)+

facet\_grid(Machine~.)+

geom\_hline(yintercept = 0.9,colour="Gray",size=1.2,linetype=3)

list\_rl1$Plot<-myplot

list\_rl1



Otra forma filtrar texto:

filt2<- mov$Studio %**in**% c("Buena Vista Studios","WB","Fox","Universal","Sony","Paramount Pictures")

Grafico boxplot con puntos y otras dimensiones:

*#boxplot con puntos y otras dimensiones*

q<-p+

geom\_jitter(aes(size=Budget...mill.,color=Studio))+

geom\_boxplot(alpha=0.7,outlier.colour = NA)

q

A screen shot of a chart

Description automatically generated

Añadir detalles sobre el grafico:

*#detalles sobre el grafico*

q <- q +

theme(

text = element\_text(family="Comic Sans MS"),

axis.title.x = element\_text(colour="Blue", size=10),

axis.title.y = element\_text(colour="Blue", size=10),

axis.text.x = element\_text(size=10),

axis.text.y = element\_text(size=10),

plot.title = element\_text(colour="Black",

size=20),

legend.title = element\_text(size=10),

legend.text = element\_text(size=10)

)

q

Maximo de cada fila:

*#maximo de cada fila*

apply(Chicago,1,max)

Media de los valores de cada columna:

*#media de los valores de cada columna*

apply(Chicago,2,mean)

Bucle que hace lo mismo:

*#media para cada fila*

output<-NULL

**for**(i **in** 1:5){

output[i]<-mean(Chicago[i,])

}

output

*#ponemos nombres*

names(output)<-rownames(Chicago)

output

Trasponer lista:

*#trasponer*

t(Chicago)

newlist<-lapply(weather,t)

newlist

Documentacion:

?lapply

Media de cada una de las filas de los elementos de una lista:

*#media de cada fila de cada uno de los elementos*

rowMeans(Chicago)

weather

lapply(weather,rowMeans)

Acceder elementos lista que contiene elementos:

*#acceder elemento*

weather[[1]][1]

lapply(weather,"[",1,1)*#primer valor primera fila*

lapply(weather,"[",1,)*#primera fils*

lapply(weather,"[",,3)*#tercera columna*

lapply con funcion:

*#lapply con funcion*

lapply(weather,function(x) x[1,])*#primera fila*

lapply(weather,function(x) x[,12])*#columna 12*

lapply(weather,function(z) z[1,]-z[2,])*#valores de la fila1 menos los de la fila2*

lapply(weather,function(z) round((z[1,]-z[2,])/z[2,],2))

Sapply:

*#lo convierte en matriz*

sapply(weather,"[",1,7)*#primera fila columna 7*

lapply(weather,"[",1,10:12)*#primera fila, columnas 10 a 12, devuelve en lista*

sapply(weather,"[",1,10:12)*#lo devuelve en matriz*

*#mismo ejemplo con funcion*

sapply(weather,function(z) round((z[1,]-z[2,])/z[2,],2))

Obtener maximo:

*#obtener el maximo*

lapply(weather,apply,1,max)

lapply(weather,function(x) apply(x,1,max))

sapply(weather,apply,1,max)

Nombre del maximo:

*#nombre del maximo*

names(which.max(Chicago[1,]))

apply(Chicago,1,function(x) names(which.max(x)))

lapply(weather,function(y) apply(y,1,function(x) names(which.max(x))))

sapply(weather,function(y) apply(y,1,function(x) names(which.max(x))))

Conjunto de datos de coches:

library(datasets)

*#conjunto de datos de coches*

head(mtcars)

cylinders <- table(mtcars$cyl)

cylinders

barplot(cylinders)

Listas y vectores:

mi\_vector <- c(10, 20, 30, 40, 50)*#vector:mismo tipo*

mi\_lista <- list(nombre = "Juan", edad = 30, notas = c(85, 90, 75))*#lista: diferente tipo*

Tipos basicos:

*#int*

n1 <- 15

typeof(n1)

*#double*

n2 <- 1.5

*#caracter*

c1 <- "c"

*#string*

c2 <- "a string of text"

*#boolean*

l1 <- TRUE

l2 <- F

Vector y matriz:

*#vector*

v1 <- c(1, 2, 3, 4, 5)

**is**.vector(v1)

*#matriz*

m1 <- matrix(c(T, T, F, F, T, F), nrow = 2)

m2 <- matrix(c("a", "b",

"c", "d"),

nrow = 2,

byrow = T)

Array:

*#array*

*# (rows, columns, tables)*

a1 <- array(c( 1:24), c(4, 3, 2))

Rbind y cbind:

*# Data frame*

vNumeric <- c(1, 2, 3)

vCharacter <- c("a", "b", "c")

vLogical <- c(T, F, T)

dfa <- cbind(vNumeric, vCharacter, vLogical)*#une en columnas->1acolumna: 1,2,3*

dfa1 <- rbind(vNumeric, vCharacter, vLogical)*#une en filas->1afila: 1,2,3*

df <- **as**.data.frame(cbind(vNumeric, vCharacter, vLogical))

Otra forma lista:

*#lista*

o1 <- c(1, 2, 3)

o2 <- c("a", "b", "c", "d")

o3 <- c(T, F, T, T, F)

list1 <- list(o1, o2, o3)

list2 <- list(o1, o2, o3, list1)

Convertir a int:

*#a int*

(coerce3 <- **as**.integer(5))

typeof(coerce3)

*#a numero*

(coerce5 <- **as**.numeric(c("1", "2", "3")))

typeof(coerce5)

Matriz a dataframe:

*#matriz a dataframe*

(coerce6 <- matrix(1:9, nrow= 3))

**is**.matrix(coerce6)

(coerce7 <- **as**.data.frame(matrix(1:9, nrow= 3)))

**is**.data.frame(coerce7)

Cargar iris y describir sus datos:

library(datasets)

head(iris)

describe(iris$Sepal.Length)

*#todo el dataframe*

describe(iris)

Seq, scan y rep:

*#con dos puntos*

x1 <- 0:10

x2 <- 10:0

*#seq*

(x3 <- seq(10))

(x4 <- seq(30, 0, by = -3))

*#scan*

x6 <- scan()

x6

*#rep*

x7 <- rep(TRUE, 5)

x8 <- rep(c(TRUE, FALSE), 5)

Combinar con cbind:

(x1 <- 1:3)

(y <- 1:9)

*#combinamos como columnas: primera fila 1,1*

(df1 <- cbind.data.frame(x1, y))

Unir con cbind de diferente tamaño:

*#como y es mas grande, se repite 1,2,3*

x3 <- c(1:3)

df3 <- cbind.data.frame(x3, y)

(df3$x3 <- factor(df3$x3,

levels = c(1, 2, 3)))

df3

Cbind reemplazando por etiqueta:

*#reemplaza numero por etiqueta*

x4 <- c(1:3)

df4 <- cbind.data.frame(x4, y)

df4$x4 <- factor(df4$x4,

levels = c(1, 2, 3),

labels = c("macOS", "Windows", "Linux"))

df4

*#igual pero pone las etiquetas ordenadas*

x5 <- c(1:3)

df5 <- cbind.data.frame(x5, y)

(df5$x5 <- ordered(df5$x5,

levels = c(3, 1, 2),

labels = c("No", "Maybe", "Yes")))

df5

Hierichal clustering:

library(datasets)

?mtcars

head(mtcars)

*#seleccionamos variables*

cars <- mtcars[, c(1:4, 6:7, 9:11)]

head(cars)

library(dplyr)

hc <- cars %>%

dist %>%

hclust

plot(hc)

*#añadir colores*

rect.hclust(hc, k = 2, border = "gray")

rect.hclust(hc, k = 3, border = "blue")

rect.hclust(hc, k = 4, border = "green4")

rect.hclust(hc, k = 5, border = "darkred")

A diagram of a cluster

Description automatically generated

Poner histrogramas juntos:

library(datasets)

*#cargamos iris*

?iris

head(iris)

hist(iris$Sepal.Length)

hist(iris$Sepal.Width)

hist(iris$Petal.Length)

hist(iris$Petal.Width)

*# poner graficos en 3 filas y una columna*

par(mfrow = c(3, 1))

hist(iris$Petal.Width [iris$Species == "setosa"],

xlim = c(0, 3),

breaks = 9,

main = "Petal Width for Setosa",

xlab = "",

col = "red")

hist(iris$Petal.Width [iris$Species == "versicolor"],

xlim = c(0, 3),

breaks = 9,

main = "Petal Width for Versicolor",

xlab = "",

col = "purple")

hist(iris$Petal.Width [iris$Species == "virginica"],

xlim = c(0, 3),

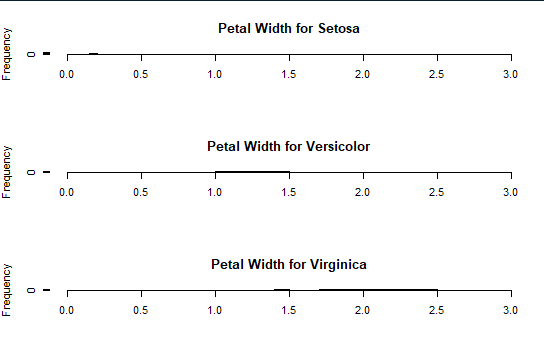
breaks = 9,

main = "Petal Width for Virginica",

xlab = "",

col = "blue")

par(mfrow = c(1, 1))



Diferentes formas de cargar los datos:

library(datasets)

*# CSV*

rio\_csv <- read.csv("C:\\Users\\jmlozanoo\\Documents\\GitHub\\BIB-R\\yt\\yt-exer\\ImportingData\_Datasets\\mbb.csv")

head(rio\_csv)

*# TXT*

rio\_txt <- read.csv("C:\\Users\\jmlozanoo\\Documents\\GitHub\\BIB-R\\yt\\yt-exer\\ImportingData\_Datasets\\mbb.txt", sep = "\t")

head(rio\_txt)

*# Excel XLSX*

library(readxl)

rio\_xlsx <- read\_excel("C:\\Users\\jmlozanoo\\Documents\\GitHub\\BIB-R\\yt\\yt-exer\\ImportingData\_Datasets\\mbb.xlsx")

head(rio\_xlsx)

*#ver datos cargados*

View(rio\_csv)

*#tabla*

r\_txt1 <- read.table("C:\\Users\\jmlozanoo\\Documents\\GitHub\\BIB-R\\yt\\yt-exer\\ImportingData\_Datasets\\mbb.txt", header = TRUE,

sep = "\t")

r\_txt1

Curvas de densidad:

library(datasets)

head(lynx)

hist(lynx)

hist(lynx,

breaks = 14,

freq = FALSE,

col = "thistle1",

main = paste("Histogram of Annual Canadian Lynx",

"Trappings, 1821-1934"),

xlab = "Number of Lynx Trapped")

*# Distribucion normal, superpuesto anterior grafico*

curve(dnorm(x, mean = mean(lynx), sd = sd(lynx)),

col = "thistle4",

lwd = 2,

add = TRUE)

*# curva de densidad estimada*

lines(density(lynx), col = "blue", lwd = 2)

lines(density(lynx, adjust = 3), col = "purple", lwd = 2)

*# Agrega pelos para ver la distribucion*

rug(lynx, lwd = 2, col = "gray")

A graph of a number of lynx trapped

Description automatically generated

Grafico de puntos con plot:

library(datasets)

head(iris)

plot(iris$Species) *# variable categorica--> de barras*

plot(iris$Petal.Length) *# variable cuantitativa--> de puntos*

plot(iris$Species, iris$Petal.Width) *# boxplot*

plot(iris$Petal.Length, iris$Petal.Width) *# de puntos*

plot(iris) *# con todas las parejas de variables*

*#grafico de puntos con plot*

plot(iris$Petal.Length, iris$Petal.Width,

col = "#cc0000",

pch = 19,

main = "Iris: Petal Length vs. Petal Width",

xlab = "Petal Length",

ylab = "Petal Width")

A diagram of a petal length

Description automatically generated

plot(mtcars$wt, mtcars$mpg,

pch = 19,

cex = 1.5,

col = "#cc0000",

main = "MPG as a Function of Weight of Cars",

xlab = "Weight (in 1000 pounds)",

ylab = "MPG")

A graph of weight loss

Description automatically generated

Diferentes tipos de regresion:

library(datasets)

install.packages("lars")

library(lars)

?USJudgeRatings

head(USJudgeRatings)

data <- USJudgeRatings

*#columnas menos la 12*

x <- **as**.matrix(data[, -12])

*#columna 12*

y <- data[, 12]

*#regresion*

reg1 <- lm(y ~ x)

*#especificando variables*

reg1 <- lm(RTEN ~ CONT + INTG + DMNR + DILG + CFMG +

DECI + PREP + FAMI + ORAL + WRIT + PHYS,

data = USJudgeRatings)

*# resultados*

reg1

summary(reg1)

*#otros datos*

anova(reg1)

coef(reg1)

confint(reg1)

resid(reg1)

hist(residuals(reg1))

*# Conventional stepwise regression*

stepwise <- lars(x, y, type = "stepwise")

*# Stagewise: stepwise pero con mejor generalizacion*

forward <- lars(x, y, type = "forward.stagewise")

*# LAR*

lar <- lars(x, y, type = "lar")

*# LASSO*

lasso <- lars(x, y, type = "lasso")

*# comparar los r2, cogiendolos y redondeando*

r2comp <- c(stepwise$R2[6], forward$R2[6],

lar$R2[6], lasso$R2[6]) %>%

round(2)

names(r2comp) <- c("stepwise", "forward", "lar", "lasso")

r2comp

Stepwise Regression: Método para seleccionar automáticamente variables explicativas relevantes al ajustar un modelo lineal, añadiendo o eliminando variables de manera iterativa basándose en criterios estadísticos como AIC o BIC.

Stagewise Regression: Similar a Stepwise pero las variables se añaden una a la vez de forma secuencial, ajustando el modelo en cada paso.

LAR (Least Angle Regression): Selecciona variables de forma similar a Stepwise pero maneja de forma más eficiente las correlaciones entre variables.

LASSO (Least Absolute Shrinkage and Selection Operator): Método de regularización que penaliza los coeficientes para favorecer la selección de variables y prevenir el overfitting, forzando algunos coeficientes a ser cero.

Histogramas con filtros:

library(datasets)

head(iris)

*#descripcion de los datos*

hist(iris$Petal.Length)

summary(iris$Petal.Length) *#media, 1o quartil...*

summary(iris$Species) *#cuenta cuantos de cada tipo*

*# solo con este tipo*

hist(iris$Petal.Length[iris$Species == "versicolor"],

main = "Petal Length: Versicolor")

*# solo con estos petalos*

hist(iris$Petal.Length[iris$Petal.Length < 2],

main = "Petal Length < 2")

*#con ambas condiciones*

hist(iris$Petal.Length[iris$Species == "virginica" &

iris$Petal.Length < 5.5],

main = "Petal Length: Short Virginica")

A graph of a number of different sizes

Description automatically generated with medium confidence

Subconjunto:

*#subconjunto*

i.setosa <- iris[iris$Species == "setosa", ]

*#ver sus datos*

head(i.setosa)

summary(i.setosa$Petal.Length)

hist(i.setosa$Petal.Length)

principal components:

Los Principal Components (Componentes Principales) en R se refieren a una técnica de reducción de dimensionalidad que transforma un conjunto de variables correlacionadas en un conjunto nuevo de variables no correlacionadas, llamadas componentes principales.

library(datasets)

?mtcars

head(mtcars)

*#seleccionar variables*

cars <- mtcars[, c(1:4, 6:7, 9:11)]

head(cars)

*#media a 0 y varianza 1*

pc <- prcomp(cars,

center = TRUE,

scale = TRUE)

*#lo mismo pero con las variables*

pc <- prcomp(~ mpg + cyl + disp + hp + wt + qsec + am +

gear + carb,

data = mtcars,

center = TRUE,

scale = TRUE)

summary(pc)*#resumen de los componentes*

plot(pc)*# el primero es muy importante, tiene un valor alto*

pc*#muestra la rotacion, similar a la correlacion entre las variables*

library(dplyr)

predict(pc) %>% round(2)*#para cada fila del conjunto, calcula el componente*

biplot(pc)*#muestra la direccion de cada variable*

*#caballos y cilindros van dirigidos a coches mas pesados*

Pacman:

pacman es un paquete en R que facilita la gestión de paquetes. Permite instalar, cargar y gestionar múltiples paquetes en una sola línea de código.

**if** (!require("pacman")) install.packages("pacman")

*# cargar paquete*

library(pacman)

*# Instalar paquetes*

pacman::p\_install("ggplot2", "dplyr", "readr")

*# Cargar paquetes*

pacman::p\_load(ggplot2, dplyr, readr)

Recorrer dataframe:

*#cargamos datos*

data(mtcars)

*#tamaño*

n\_row <- nrow(mtcars)

n\_col <- ncol(mtcars)

*# inicializar lista*

valores <- list()

*# recorrer filas*

**for** (i **in** 1:n\_row) {

*# recorrer columna*

**for** (j **in** 1:n\_col) {

*# obtener valor fila y columna*

valor <- mtcars[i, j]

*# guardar valor en la lista*

valores[[paste0("fila", i, "\_columna", j)]] <- valor

}

}

*# mostrar valores*

valores

*#resultado como: $fila32\_columna6 [1] 2.78*

Tidyverse

tidyverse es un conjunto de paquetes en R que proporciona un conjunto coherente y poderoso de herramientas para manipulación de datos, análisis y visualización. Está diseñado para hacer que el trabajo con datos sea más fácil y eficiente al promover un enfoque coherente y basado en la gramática de datos.

Algunos de los paquetes más importantes incluidos en tidyverse son:

* ggplot2: Para visualización de datos, creación de gráficos y visualización de resultados.
* dplyr: Para manipulación de datos, filtrado, selección, agrupación y unión de datos.
* tidyr: Para reorganizar datos, transformar entre formatos anchos y largos, y manejar valores perdidos.
* readr: Para importar datos de archivos planos de manera eficiente y rápida.
* purrr: Para programación funcional y operaciones en listas y vectores.
* tibble: Para crear y trabajar con data frames modernos y eficientes.

*# tidyverse*

**if** (!require("tidyverse")) install.packages("tidyverse")

library(tidyverse)

*# cargamos conjunto de datos*

data(mtcars)

mtcars

**print**(dim(mtcars))*#dimension*

**print**(length(mtcars))*#da las columnas*

**print**(nrow(mtcars))*#da las filas*

**print**(ncol(mtcars))*#da las columnas*

*# filtramos solo los que tienen mas de 150 caballos*

mtcars\_filtrado <- mtcars %>%

filter(hp > 150)

**print**(dim(mtcars\_filtrado))*#dimension*

*# resumen de los datos, agrupandolos*

resumen\_estadistico <- mtcars\_filtrado %>%

summarise(media\_hp = mean(hp),

mediana\_mpg = median(mpg),

maximo\_disp = max(disp))

*# ver relacion entre hp y mpg*

grafico <- ggplot(mtcars\_filtrado, aes(x = hp, y = mpg)) +

geom\_point() +

geom\_smooth(method = "lm") +

labs(x = "Caballos de Fuerza", y = "Millas por Galón", title = "Relación entre HP y MPG")

*# ver resumen y grafico*

**print**(resumen\_estadistico)

**print**(grafico)

A graph of a graph with black dots and a blue line

Description automatically generated

Summarise:

library(dplyr)

*# datos*

ventas <- data.frame(

region = c("A", "A", "B", "B", "A", "B"),

mes = c("enero", "febrero", "enero", "febrero", "enero", "febrero"),

ventas = c(1000, 1500, 2000, 2500, 1200, 1800)

)

*# sumar ventas por region, utiliza sumarise*

resumen\_ventas <- ventas %>%

group\_by(region) %>%

summarise(ventas\_totales = sum(ventas))

**print**(resumen\_ventas)

**print**(class(resumen\_ventas))

Tamano dataframe:

**print**(dim(mtcars))*#dimension*

**print**(length(mtcars))*#da las columnas*

**print**(nrow(mtcars))*#da las filas*

**print**(ncol(mtcars))*#da las columnas*

Plotly:

Caret:

Tibble y operaciones sobre el:

Un tibble es similar a un data frame, pero con algunas diferencias importantes en la forma en que se imprime y se manejan los datos.

library(tibble)

*# Crear tibble*

datos <- tibble(

ID = c(1, 2, 3, 4),

nombre = c("Juan", "María", "Pedro", "Ana"),

edad = c(30, 25, 35, 28),

ciudad = c("Madrid", "Barcelona", "Sevilla", "Valencia")

)

*# imprimirlo*

**print**("Tibble original:")

**print**(datos)

*# transformar celdas que cumplen condicion*

datos <- datos %>%

mutate(edad = ifelse(edad > 30, edad + 1, edad))

*# mostrarlo*

**print**("Tibble con transformación aplicada:")

**print**(datos)

*# añadir fila*

nueva\_fila <- tibble(

ID = 5,

nombre = "Luis",

edad = 29,

ciudad = "Málaga"

)

*#insertar fila*

datos <- bind\_rows(datos, nueva\_fila)

*# mostrar*

**print**("Tibble con nueva fila añadida:")

**print**(datos)

Insertar columna sobre tibble:

*# Insertar columna con condicion*

datos <- datos %>%

mutate(tipo\_cliente = if\_else(ciudad == "Madrid" & edad > 27, "cliente Madrid", "cliente externo"))

*# Mostrarlo*

**print**("Tibble con nueva columna añadida:")

**print**(datos)

Tidyr para pivotar dataframe:

library(tidyr)

*# datos*

temperaturas <- data.frame(

dia\_semana = c("lunes", "martes", "miércoles"),

hora\_0 = c(20, 22, 19),

hora\_1 = c(18, 21, 20),

hora\_2 = c(17, 20, 19)

)

temperaturas

*# una fila por dia, hora y temperatura*

temperaturas\_reorganizadas <- temperaturas %>%

pivot\_longer(cols = starts\_with("hora"),

names\_to = "hora",

values\_to = "temperatura")

**print**(temperaturas\_reorganizadas)*#lunes hora\_0 20*

Añadir filas a un dataframe:

*# dataframe*

datos <- data.frame(

ID = c(1, 2, 3, 4),

nombre = c("Juan", "María", "Pedro", "Ana"),

edad = c(30, 25, 35, 28),

ciudad = c("Madrid", "Barcelona", "Sevilla", "Valencia"),

stringsAsFactors = FALSE

)

*# Mostrarlo*

**print**("Dataframe original:")

**print**(datos)

*# Crear fila*

nueva\_fila <- c(ID = 5, nombre = "Luis", edad = 40, ciudad = "Málaga")

*# añadir la fila*

datos <- rbind(datos, nueva\_fila)

*# reiniciar ids*

rownames(datos) <- NULL

*# Mostrarlo*

**print**("Dataframe con nueva fila añadida:")

**print**(datos)

Insertar columna en dataframe con condicion:

*# Insertar columna*

datos$tipo\_cliente <- ifelse(datos$ciudad == "Madrid" & datos$edad > 27, "cliente Madrid", "cliente externo")

*# Mostrarlo*

**print**("Dataframe con nueva columna añadida:")

**print**(datos)

Guardar grafico en html:

install.packages("gapminder")

library(gapminder)

*#cargar datos*

data("gapminder")

library(ggplot2)

library(plotly)

*#ggplot ejemplo con puntos*

pp <-ggplot(gapminder, aes(x = gdpPercap, y = lifeExp, col = continent)) +

geom\_jitter() + scale\_x\_log10()

pp

*# grabar el gráfico*

pp1 <- ggplotly(pp)

htmlwidgets::saveWidget(as\_widget(pp1), "C:\\Users\\jmlozanoo\\Documents\\GitHub\\BIB-R\\yt\\yt-exer\\mi\_grafico.html")

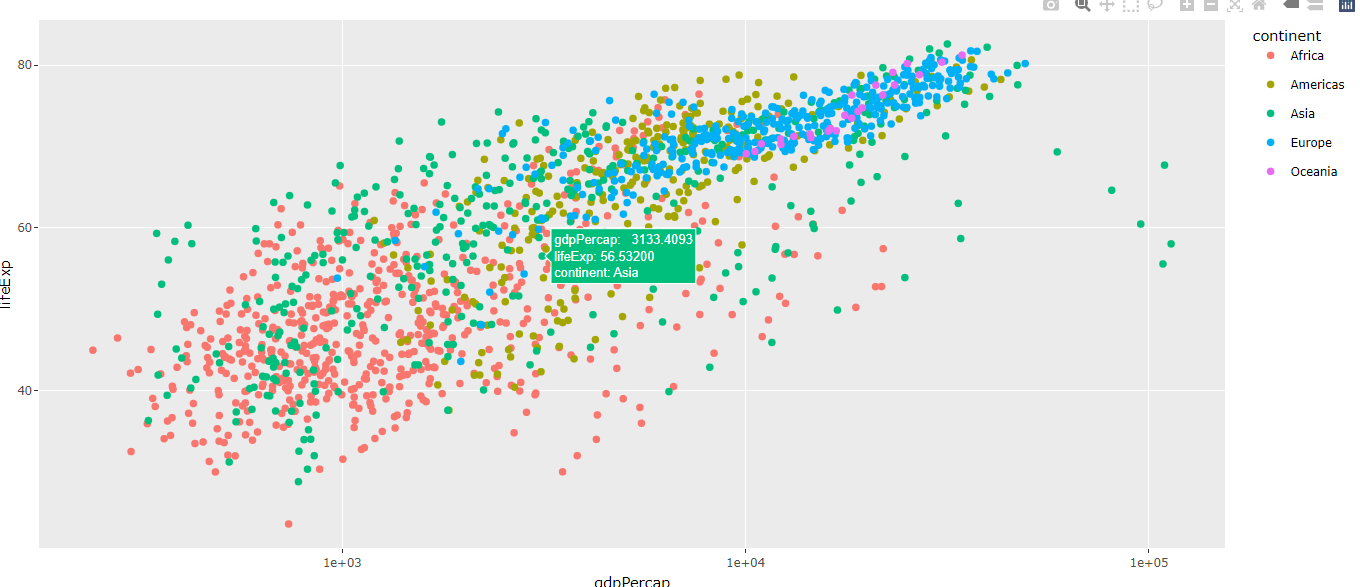


Grafico de puntos cambiando forma:

*#grafico de puntos cambiando forma*

pp2 <- ggplot(gapminder, aes(x = gdpPercap, y = lifeExp, shape = continent, col = **as**.factor(year))) +

geom\_jitter() + scale\_x\_log10()

pp2

A graph with many colored dots

Description automatically generated

Grafico con plotly:

*#crear grafico con plotly con color por continente*

g <- plot\_ly(gapminder, x = ~gdpPercap, y = ~lifeExp, color = ~continent)

g <- add\_markers(g)

*#cambiar escala eje x*

g <- layout(g, xaxis = list(type = "log"))

htmlwidgets::saveWidget(as\_widget(g), "C:\\Users\\jmlozanoo\\Documents\\GitHub\\BIB-R\\yt\\yt-exer\\plotly.html")

A graph showing a number of colored dots

Description automatically generated with medium confidence

Regresion lineal con caret:

*# Cargar paquete*

**if** (!require("caret")) install.packages("caret")

library(caret)

*# datos*

data(mtcars)

*# Dividir datos*

set.seed(123) *# semilla*

indices <- createDataPartition(mtcars$mpg, p = 0.8, list = FALSE)

datos\_entrenamiento <- mtcars[indices, ]

datos\_prueba <- mtcars[-indices, ]

*# modelo de regresion lineal*

modelo <- train(mpg ~ ., data = datos\_entrenamiento, method = "lm")

*# resumen*

**print**(modelo)

*# Predecir valores*

predicciones <- predict(modelo, newdata = datos\_prueba)

*# Calcular RMSE*

rmse <- sqrt(mean((predicciones - datos\_prueba$mpg)^2))

**print**(paste("RMSE:", rmse))