<- es igual a =

|> para concatenar varias funciones

\\ es y , | es o

# **LIBRERÍAS BÁSICAS**

library(car)

library(rio)

library(dplyr)

library(readr)

library(haven)

library(tidyverse)

library(DescTools)

library(descr)

library(ggplot2)

library(gtsummary)

library(hmisc)

library(ggthemes)

library(gganimate)

# **CARGA DE DATOS**

*SCRAPPING*

library(htmltab)

LinkPage= " http…."

LinkPath=’ ’ (del xpath)

data = htmltab(doc = linkPage, which =linkPath) \*De dónde--> linkPage, qué tabla--> linkPath

*IMPORTAR (Propietary Software)*

library(rio)

link="link de github"

data=import(link)

*Guardar base como csv*

write.csv(DataLimpia, "DataLimpia.csv", row.names=F)

# **GENERAL**

head(matriz o tabla) 🡪 Si los datos son muy numerosos, previsualiza los primeros 6 datos de la tabla.

tail(matriz o tabla) 🡪 visualiza los últimos datos

\*\* eleva a potencia

== equivalencia , < menor, > mayor, & para varias condiciones/pedidos

\\ sirve para emergencias cuando no corren los splits porque no identifica caracteres

!= 🡪 significa “diferente a”, osea que NO esté

row.names(matriz)=matriz$variablequetienelosnombres 🡪 rownames

set.seed(xxx) 🡪 semilla aleatoria para fijar datos en un año específico

matriz[#,#]="nuevo nombre o número" 🡪 agregar información a una celda en específico, tu método

dim(matriz)🡪 longitud de filas y columnas del data frame

nrow(matriz) #filas

filter(#:#) 🡪 mostrar o filtrar ciertas filas

ncol(matriz) #columnas

select (#:#) 🡪 mostrar o filtrar ciertas columnas

el [x,] es para acceder a una fila y el [,x] es para acceder a una columnas

data <- data[,-#] 🡪 borrar una columna

data <- data[,-(#:#)] 🡪 borrar varias columnas

data <- data[-#,] 🡪 borrar una fila

data <- data[-(#:#),] 🡪 borrar varias filas

lenght(matriz$variable) 🡪 cantidad de elementos en una variable

str(matriz) 🡪 estructura de la matriz

str(matriz$variable) 🡪 estructura de una variable

class(matriz$variable9 🡪 tipo de variable

table(matriz$variable) 🡪 visualizar una variable

matriz |> rename(nuevavariable = variable) 🡪 cambia el nombre de una variable

matriz |> group\_by(variable) 🡪 agrupar datos en función de una variable

matriz |> count(variable) 🡪 contar el número de observaciones de una variable

matriz |> arrange(desc(variable)) 🡪 ordenar de forma descendente los valores de una variable

matriz |> arrange(variable) 🡪 ordenar de forma ascendente los valores de una variable

summary(matriz) o summary (matriz$variable, na.rm = T) 🡪 estadísticos descriptivos de una data o una variable numérica, evitando NAs

summarise(matriz$variable= 🡪 variable resumen dentro del paquete tidyverse, que permite crear variables resumen que afectan a toda la matriz

matriz |> drop.na(variable) 🡪 elimina los NAs de una variable

matriz = mutate(nueva variable = forma de cálculo) 🡪 crea una nueva columna o modifica una columna, permite hacer cálculos

matriz <- mutate(nuevavariable=case\_when(variable<# ~ “Atributo1”, variable<# ~ “Atributo2”, variable<=# ~ “Atributo3”)) 🡪 crea columnas nuevas o modifica columnas existentes en un data frame, basado en condiciones específicas. Se debe combinar con levels = c

prop.table(table(matriz$variable))\*100 🡪 tabla de proporciones

max(matriz$variable): valor máximo

min(matriz$variable): valor mínimo

mean(matriz$variable): media

median(matriz$variable): mediana

mode(matriz$variable): moda

sum(matriz$variable): suma

sd(matriz$variable): desviación estándar

str(matriz$variable): características o estructura de la variable

label(matriz$variable) = “Etiqueta que le das” 🡪 agregar una etiqueta a una variable

rgdal::readOGR 🡪 :: para especificar función de un paquete

# **FUNCIONES IMPORTANTES**

## **Conversiones básicas**

matriz$variable = as.numeric (matriz$variable) 🡪 convertir un factor a vector/numérica

matriz[,#columnas]<-lapply(matriz[,#columnas], función) 🡪 lapply para asignar una función a varias columnas

matriz$variable = factor (matriz$variable) 🡪 convertir un vector a factor

matriz$variable = factor (matriz$variable, levels = c(1,2,3,…), labels = c(“a”,”b”,”c”…) , ordered = TRUE) 🡪 convertir un vector a factor, darle niveles, darle etiquetas y ordenarlo

matriz |> arrange(desc(variable)) 🡪 ordenar de forma descendente los valores de una variable

matriz |> arrange(variable) 🡪 ordenar de forma ascendente los valores de una variable

matriz$variable <- cut (matriz$variable, breaks = c (limite1con2, limite2con3, limite3con4, limite4), labels = c(“a”,”b”,”c”…) , include.lowest = TRUE, ordered = TRUE) 🡪 para cortar la variable ordinal en grupos de datos

matriz$variablenueva = as.Date(matriz$variable, "%d/%m/%y) 🡪 transformar a fecha

## **Listas**

typeof(variable) 🡪 tipo de variable

variable = list(“valor 1”, “valor 2”…) 🡪 crear una lista

variable[1] 🡪 ver primer valor

tail[variable,1] 🡪 ver último valor

variable[2:4] 🡪 ver del 2do al 4to valor

variable[c(1,3,4)] 🡪 ver 1er, 3er y 4to valor

mutate(nueva variable = forma de cálculo) 🡪 crea una nueva columna o modifica una columna, permite hacer cálculos

variable[[2]] = “valor nuevo” 🡪 cambiar el valor 2 por un valor nuevo

variable[variable==’valor viejo’]=”valor nuevo” 🡪 cambiar valor viejo por uno nuevo

variable[1(variable)]=NULL 🡪 borrar 1er valor de la variable

variable=variable[-length(variable)] 🡪 borrar último valor de la variable

variable[1:3]=NULL 🡪 borrar valores del 1 al 3 de la variable

variable[c(1,3,5)]=NULL 🡪 borrar valores 1, 3 y 5 de la variable

variable=variable[!variable %in% list(´valor´)] 🡪 borrar todos los valores que dicen “valor” de la variable

variable=append(variable, “valor”) 🡪 agregar un valor al final de la variable

variable=append(variable, “valor”, after=2) 🡪 agregar un valor en la posición 3

append(variable1,variable2) 🡪 concatenar 2 variables

## **Vectores (usar c para concatenar)**

vector=c(“valor 1”, “valor 2”, “valor 3”) 🡪 crear un vector

vector=vector[vector!=3] 🡪 elimina el valor 3 de la vector

vector=vector[!vector%in%c(1,45)] 🡪 elimina los valores 1 y 45 de la vector

vector=c(vector,5) 🡪 agrega el valor 5 al final de la vector

vector=append(vector, 5, after=1) 🡪 agrega el valor 5 después del valor 1

vector3=c(vector1,vector2) 🡪 concatenar vectores

## **Data frame**

crear listas 1, 2 y 3 con valores🡪 consolidado=list(variable1=lista1,variable2=lista2, variable3=lista3) 🡪 matriz=as.data.frame(do.call(cbind,consolidado) 🡪 crear data frame

class(matriz) 🡪 estructura de la matriz, si es un tupple o un data frame

str(matriz) 🡪 detalles de la matriz

dim(matriz) 🡪 número de filas y columnas de la matriz

length(matriz) 🡪 número de filas de la matriz

head(matriz,2) 🡪 mostrar 2 primeras filas de la matriz

names(matriz) 🡪 nombres de las columnas o variables de la matriz

matriz[ ,1] 🡪 pide información de la primera columna de datos

matriz[ ,c(1,2)] 🡪 pide información de las dos primeras columnas de datos

matriz[ ,´variable1´] 🡪 pide información de la variable1

matriz[ ,c(´variable1´,´variable2´)] 🡪 pide información de las variables 1 y 2

matriz$variable 🡪 mostrar una variable

matriz[1,´variable1´] 🡪 mostrar fila 1 de la columna 1

matriz[c(1,2),c(´variable1´,´variable2´)] 🡪 mostrar filas 1 y 2 de las variables 1 y 2

matriz[1,´variable1´]=´nuevo valor´🡪 reemplaza valor de una celda en una variable

matriz[c(1,2),´variable1´]=c(´nuevo valor´, nuevo valor´) 🡪 reemplaza valor de varias celdas en una variable

byeRows=c(1,2) 🡪 matriz=matriz[-byeRows,] 🡪 eliminar filas 1 y 2 de la matriz

row.names(matriz)=NULL 🡪 reestablecer índices

byeColumns=c(´variable1´) 🡪 matriz[,-byeColumns] 🡪 eliminar columna

matriz[1,´variable1´]=NA 🡪 eliminar celda (intersección fila y columna)

variable=c(T,F,T,F] 🡪 matriznueva=cbind(matriz,variablenueva=variable) 🡪 agregar nueva columna con información a una copia de la matriz

matriz[which.max(matriz$variable),] 🡪 valor máximo de una variable

matriz[which.min(matriz$variable),] 🡪 valor mínimo de una variable

matriz[matriz$variable==´lo que buscas´,] 🡪 buscar información específica en una variable

matriz[matriz$variable %in% c(‘opción1´,´opción2´),] 🡪 buscar información específica en base a 2 opciones (quién es esto o el otro)

matriz[!matriz$variable %in% c(ópción1´, ópción2´),] 🡪 buscar información específica en base a 2 opciones (quién NO es esto o el otro)

# **OPERACIONES**

sum(matriz$variable) 🡪 suma de valores de una variable numérica

# **GRÁFICOS**

lm(matriz$variable) 🡪 modelo lineal

data |> ggplot() + aes(x=variableindependiente, y=variabledependiente) + geom\_point() + scale\_x\_log10() + labs(title="xxx", subtitle="xxx", caption="Fuente: xxx/Elaboración Propia", x="nombrevariableindependiente", y="nombrevariabledependiente") + theme\_classic() 🡪 ggplot de dispersión con títulos

* + facet\_wrap(~Terceravariable) + aes(color=Terceravariable) 🡪 ggplot para organizar la información en función de una tercera variable
* + geom\_hline(aes(yintercept = mean(variabledependiente)), linetype = "dotdash", color = "blue") 🡪 agregarle un estadístico descriptivo respecto a la variable dependiente

data |> count(variable) |> drop\_na(variable) |> ggplot() + aes(x = variable) + geom\_bar() + labs(x="Nombrevariable", y="Frecuencia") 🡪 gráfico de barras simple

* |> mutate(variable = fct\_reorder(variable, n, .desc = TRUE)) 🡪 ordenar de forma descendente la variable en el plot

data |> drop\_na(variableindependiente, variabledependiente) |> ggplot()+ aes(x = variableindependiente, fill = variabledependiente)+ geom\_bar(position = "fill") 🡪 gráfico de barras acumuladas

* + theme\_economist() + scale\_fill\_economist() + theme(legend.text = element\_text(size = 5)) 🡪 agregarle un tema

data |> ggplot() + aes(y = variable) + geom\_boxplot() + theme\_stata() + scale\_color\_stata() 🡪 boxplot para variable numérica con tema

* aes(y = PBIPC, colour=Continent) 🡪 también puedes organizarlo en base a una segunda variable

# **LIMPIEZA**

data |> mutate(Variablenueva = str\_replace\_all(`variable`, ",|\\.", ".")) 🡪 cambiar todos los “,.” y “,” y convertirlos en “.”. | es "o" , \\ es “y”.

data$variable=parse\_number(data$variable) 🡪 extrae el primer número que encuentra

matriz$variable = gsub("signo1|signo2", "nuevo cambio", matriz$variable) 🡪 | es "o"

data$variable=gsub(pattern = "Â",replacement = "", data$variable) 🡪 reemplazar un carácter no deseado

matriz[,#columnas]<-lapply(matriz[,#columnas], función) 🡪 lapply para asignar una función a varias columnas

matriz = matriz[complete.cases(matriz),] 🡪 elimina NA's (elimina filas, ojo)

matriz[-#,] 🡪 Para eliminar una fila

matriz[-c(#:#),] 🡪 Para eliminar un grupo de filas

matriz[,-#)] 🡪 Para eliminar una columna

matriz[,-c(#:#)] 🡪 Para eliminar un grupo de columnas

colnames(matriz)[colnames(matriz)=="Nombre a cambiar"] = "nuevo nombre" 🡪 Cambiar nombre de una columna

matriz[matriz$variable=="nombre del elemento",'Nombre de la columna']="nuevo nombre"🡪 Cambiar nombre de una fila en una columna

library(stringr)

m$v = str\_split(string = m$v,

pattern = 'lo que va a separar',

simplify = T)[,2] 🡪 con qué columna te quedas. Agregar n=# para determinar número de particiones.

str\_pad("739",width=5,side="left",pad="0") 🡪 para agregarle a algo, con un tamaño determinado, hacia la derecha o izquierda, un número. Sale 00739

str\_detect(vector, "texto a detectar")

matriz[grep("lo q debe buscar",matriz$variable),] 🡪 busca o identifica algo en las filas de esta columna

matriz=matriz[ - grep("lo q debe buscar",matriz$variable),] 🡪 eliminar datos de filas de una columna

str\_replace(vector, "texto a cambiar", "nuevo texto")

str\_to\_lower(vector) 🡪 cambia a minúsculas

str\_to\_upper(vector) 🡪 cambia a MAYÚSCULAS

matriz[,]=lapply(matriz[,], trimws,whitespace = "[\\h\\v]") 🡪 elimina espacios en blanco antes y después de texto

matriz[duplicated(matriz$variable),] 🡪 Chequear duplicados en filas de una columna

str\_split(string = matriz$variable,

pattern = 'xxxx',

simplify = T)[,#]%>%parse\_number() 🡪 combinar split con parse number

data <- data[, c(2,3,1)]🡪 reordenar columnas

## **Formateo de tablas**

matriz[,#columnas]=lapply(matriz[,#columnas], función) 🡪 lapply para asignar una función a varias columnas

m$v=parse\_number(m$v) 🡪 extrae el primer número que encuentra

m$v = str\_split(string = m$v,

pattern = 'lo que va a separar',

simplify = T)[,2] 🡪 con qué columna te quedas, 1 o 2

## **Aggregating**

aggregate(cbind(V1, V2) ~ v3 + v4, data = matriz, sum) 🡪 v3 y v4 son las categorías por las que v1 y v2 se van a ordenar. Puedes cambiar “sum” por otra operación que necesites. Quitar el + en caso sea necesario.

## **3 formas de Merge**

matrizfinal=merge(matriz1, matriz2, by="categoría común")

matrizfinal=merge(matriz1, matriz2, by=0) 🡪 ponerlo en cualquier orden

matrizfinal=merge(matriz1, matriz2, by.x="categoría común", by.y="categoría común")🡪 con esto se eliminan casos no comunes

matrizfinal=merge(data1,data2,all.x=T, all.y=T) 🡪 esto (all.x=T, all.y=T) para no borrar elementos que no concuerden

dataLimpia=data[data$variable=='lo q está mal',"Variable"]="Nuevo cambio" 🡪 cambiar elemento de una columna

names(matriz)[names(matriz)=='Variable']='Nuevo Cambio' 🡪 cambiar nombres de columnas

matriz[!complete.cases(matriz),] 🡪 muestrame todo lo que no sean datos completos de esta base

matriz = matriz[complete.cases(matriz),] 🡪 eliminar NA's (elimina filas, ojo)

matriz = na.omit(matriz) 🡪 borrar NA's

**APPENDING:** Appending es unir diversas tablas en una sola, pero todas tienen los mismos nombres de columna.

df1234=rbind(df1,df2,df3,df4)

**LAPPLY**: data[,c(#:#)]=lapply(data[,c(#:#)],as.numeric) 🡪 ojo, para una sola columna no se necesita lapply, es para varias

**INTERVALOS CON GRUPOS DE CORTE**

summary(m$v) 🡪 de acá sacar máximos y mínimos

CORTES=c(0, 2.5, 5, 7.5, 10) 🡪 ejemplo con 5 cortes, puedes cambiar al gusto

m$variableconcortes=cut(m$v, breaks = CORTES, ordered\_result = TRUE)

table(m$variableconcortes) 🡪 aquí observas resultado

**CLUSTERS NO JERARQUICOS (PARTICIONES CON PAM)**

library(cluster) / library(plyr)

dist=daisy(matriz[,c(#:#)], metric="gower", stand=T) 🡪 en c poner las variables de interés de donde quieres extraer distancias. Stand para estandarizar (siempre), también puedes usar scale=T

res.pam=pam(dist,#clusters,cluster.only = F) 🡪 TRUE sale sólo cluster, FALSE sale toda la info. Usa F

matriz$clusterpam=res.pam$clustering 🡪 en la base de datos se crea una columna que el asigna a cada elemento un código de cluster

\*Creas un objeto en donde veas las características de cada cluster (numéricos): AGGREGATING

objeto=aggregate(as.matrix(cbind(matriz[,c(#variablesdeinterés)]))~ clusterpam,

data=matriz, FUN=plyr::each(MD = median, Media = mean))

Nuevodf=t(as.data.frame(objeto))🡪 t es para voltear la tabla y crear df, filas se vuelven columnas

Nuevodf

**PREPARACIÓN/RECODE CLUSTERS DE CATEGÓRICAS(NOMINALES)/ORDINALES** 🡪 as.factor 🡪 as.numeric 🡪 ver en una table(m$v) lo que tienes 🡪 (m$v)=recode(m$v,"#=# ; #=# ; #=NA", as.factor=T) \*\*\*poner FALSE para cuando hagas el aggregate categorico (agregar as.ordered si es necesario) 🡪 levels(m$v)=c("....")🡪 m$v=ordered(m$v) 🡪 table(m$v) 🡪 str(m$v) (que salga como factor)

**OTRO RECODE🡪** base$variable=recode(base$variable,"1"='Mal',"2"='Mal',"3"='Bien',"4"='Bien') ; data$variable=as.ordered(data$variable)

**OPCIÓN EXTRA AGGREGATE (DDPLY) para ordinales (primero recodificarlos)**tabla = ddply(matriz[,c(#:#)], "clusterpam", colwise(Median))nuevatabla = t(as.data.frame(tabla))

\*Opción extra para que una matriz se vea más bonita para la web en formato html

library(knitr) / library(kableExtra)

kable(matriz, format = "html", digits = 2)%>%

kable\_styling(bootstrap\_options = c("striped", "hover", "condensed", "responsive"), font\_size = 10)

**Visualización gráfica (SÓLO PARA PAM):**

matriz$clusterpam = as.factor(matriz$clusterpam) 🡪 **NO OLVIDAR ESTE PASO**

levels(matriz$clusterpam)=c(" "," ") 🡪 nombres de los clusters

fviz\_cluster(object = list(data=dist, cluster = matriz$clusterpam),

geom = c("text"), main="Título a cambiar",

ellipse = FALSE, labelsize = 8,

repel = T)

CLUSTERS JERÁRQUICOS (Y COMANDOS PARA AMBOS)

**CLUSTERS SÓLO JERÁRQUICOS / COMANDOS PARA JERÁRQUICOS Y NO JERÁRQUICOS**

**\*Aglomerativo jerárquico (AGNES)**

res.agnes = hcut(dist, k = # , hc\_func='agnes',hc\_method = "ward.D")

matriz$clusteragnes=res.agnes$cluster

**\*Divisivo jerárquico (DIANA)**

res.diana <- hcut(dist, k = # , hc\_func='diana')

matriz$clusterdiana = res.diana$cluster

\***Gráficos (DENDOGRAMAS SÓLO PARA JERÁRQUICOS)**

fviz\_dend(res.agnes, cex = 0.7, horiz = T)

fviz\_dend(res.diana, cex = 0.7, horiz = T)

**DECRIPCIÓN DE VARIABLES PARA CLUSTERS JERÁRQUICOS Y NO JERÁRQUICOS**

describeBy(data$v1, group=data$clusterPAM, digits = 2) 🡪 para análisis descriptivo que relacione UNA variable con los clusters de UN método

\*\*\*describeBy(as.matrix(cbind(data[,c(#:#)])), group=data$clusterPAM, digits = 2) 🡪 análisis descriptivo de VARIAS variables con UN método

table(m$v1 , m$clusterPAM ,dnn = c('v1','cluster')) 🡪 creación de tablas de contingencia, dnn=c(" ") es para títulos

**PREPARACIÓN CLUSTERS DE CATEGÓRICAS(NOMINALES)/ORDINALES** 🡪 as.factor 🡪 as.numeric 🡪 ver en una table(m$v) lo que tienes 🡪 (m$v)=recode(m$v,"#=# ; #=# ; #=NA", as.factor=T) \*\*\*poner FALSE para cuando hagas el aggregate categorico (agregar as.ordered si es necesario) 🡪 levels(m$v)=c("....")🡪 m$v=ordered(m$v) 🡪 table(m$v) 🡪 str(m$v) (que salga como factor)

**DETERMINAR NÚMERO CLUSTERS** (si salen diferentes en particionante y jerárquico escoger menor, **escoger número anterior al mínimo):**

dist = daisy(matriz[,#:#], metric="gower", stand = T)

fviz\_nbclust(matriz[,#:#], pam,diss=dist,method = "gap\_stat",k.max = 10,verbose = F) 🡪 para PAM

fviz\_nbclust(matriz[,#:$], hcut,diss=dist,method = "gap\_stat",k.max = 10,verbose = F) 🡪 para AGNES y DIANA

**\*Creamos los clusters según el número sugerido y hacemos EVALUACIÓN GRÁFICA** (El gráfico que tiene menos siluetas negativas es el preferible a los demás. Escoger gráfico con mayor peso (average silhouette width)) Funciones y métodos mejoran varianzas.

res.pam = pam(dist, # ,cluster.only = F)

res.agnes = hcut(dist, k = #,hc\_func='agnes',hc\_method = "ward.D")

res.diana = hcut(dist, k = #,hc\_func='diana')

fviz\_silhouette(res.pam)

fviz\_silhouette(res.agnes)

fviz\_silhouette(res.diana)

**\*EVALUACIÓN NUMÉRICA DE CASOS NO ASIGNADOS CON SILUETA NEGATIVA** (Cambiar pam por diana o agnes)

poorPAM=data.frame(res.pam$silinfo$widths) 🡪 creación de un dataframe con valores "mal asignados".

poorPAM$nombres=row.names(poorPAM)

poorPAMcases=poorPAM[poorPAM$sil\_width<0,'nombres'] 🡪 explorar casos negativos. Cambiar pam por diana o agnes según corresponda

poorPAMcases 🡪 ver cuáles son los negativos

length(poorPAMcases) 🡪 cantidad de negativos

**\*TEORÍA DE CONJUNTOS** para comparar resultados de 2 métodos de evaluación numérica: intersect(poorAGNEScases,poorPAMcases), setdiff(,), union(,). Recuerda que: intersección es "y", setdiff es "en uno sí pero en otro no" y unión es "o"

**DENSIDAD DE CLUSTERS:** La estrategia basada en densidad junta a los casos cuya cercanía entre sí los diferencia de otros. Genera las coordenadas de cada elemento (x=dim1,y=dim2):

**Se hace un mapa de posiciones de los elementos y se proyectan en un plano bidimensional:**

proyeccion = cmdscale(g.dist, k=2,add = T) 🡪 # k es número de dimensiones SIEMPRE VA A SER 2. Se crea una lista, de ahí vemos la variable Points que es de donde sacaremos las columnas para las dimensiones.

matriz$dim1 <- proyeccion$points[,1]

matriz$dim2 <- proyeccion$points[,2]

\***Gráfico** de posición de elementos respecto a ambas dimensiones

ggplot(matriz,aes(x=dim1, y=dim2,label=row.names(matriz))) +

geom\_text(size=2)

Creas **columnas con información de clusters y los vuelves factores** 🡪 **NO OLVIDAR ESTE PASO**

matriz$pam=as.factor(res.pam$clustering)

matriz$agnes=as.factor(res.agnes$cluster)

matriz$diana=as.factor(res.diana$cluster)

\*Determinar máximos y mínimos para el gráfico **(límites):**

min(matriz[,c('dim1','dim2')])

max(matriz[,c('dim1','dim2')])

limites=c(minimo,maximo) 🡪 creas un objeto con los límites. Intenta que haga un cuadrado jalando los valores.

Nuevo **gráfico a colores de acuerdo con clusters de posición de elementos respecto a dimensiones**. Cambiar color=pam y title="PAM" por agnes o diana según corresponda:

base= ggplot(matriz,aes(x=dim1, y=dim2)) + ylim(limites) + xlim(limites) + coord\_fixed() \*\*\*esto no cambia

base+geom\_point(size=2, aes(color=pam)) + labs(title = "PAM") \*

base + geom\_point(size=2, aes(color=agnes)) + labs(title = "AGNES") \*

base + geom\_point(size=2, aes(color=diana)) + labs(title = "DIANA") \*

(esto continúa en DBSCAN)

**MÉTODO DE DENSIDAD DBSCAN (SIEMPRE CONTINÚA DENSIDAD DE CLUSTERS porque usa límites de las dimensiones):** Te muestra nuevos clusters y valores atípicos.

g.dist.cmd = daisy(matriz[,c('dim1','dim2')], metric = 'euclidean') 🡪 **Distancia euclidiana**

library(dbscan) / library(fpc)

kNNdistplot(g.dist.cmd, k=#) 🡪 (k= número de variables que utilizaste en dist), esto es el Epsilon

abline(h=0.125, lty=2) 🡪 extra con epsilon

Para saber cuántos clusters hay, se crea una lista en db.cmd. Esto es generado por la máquina, te muestra atípicos. En MinPts pones el número de variables que usaste en dist y el epsilon te lo da el profesor:

db.cmd = fpc::dbscan(g.dist.cmd, eps=0.06, MinPts=#,method = 'dist')

db.cmd

\*En columna 0 salen outliers y los demás son los nuevos números de clusters agrupados

matriz$dbCMD=as.factor(db.cmd$cluster) 🡪 pones los datos en una columna de la matriz original

**Gráfico sin texto**

base= ggplot(matriz,aes(x=dim1, y=dim2)) + ylim(limites) + xlim(limites) + coord\_fixed()

dbplot= base + geom\_point(aes(color=dbCMD))

dbplot

**Gráfico con texto**

dbplot + geom\_text\_repel(size=2,aes(label=row.names(matriz)))

**Sólo atípicos**

LABEL=ifelse(matriz$dbCMD==0,row.names(matriz),"")

dbplot + geom\_text\_repel(aes(label=LABEL),

size=2,

direction = "y", ylim = 0.45,

angle=45,

segment.colour = "grey")

**TEORÍA**

* Direccionales o asimétricas- sí lógica de influencia o impacto de x e y
* No direccionales o simétricas- no lógica x e y
* Variable latente🡪 concepto con un atributo que engloba/acopla observables
* Variable observable🡪 las que se miden para entender el concepto
* Probar que variables observables no estén correlacionadas entre sí. Usar siempre la latente.
* Paramétrico🡪 que datos (n grande, muestra representativa) se comportan de forma normal (**normalidad es cuando una variable tiene moda=media, es una campana**). Tienden a distribución normal.
* No paramétrico🡪 variables no se comportan de forma normal, mundo real.

Variable de interés- dependiente.

**REGRESIÓN**

**ESTADÍSTICA 1**

Usar summary(data) o str(data) para visión general

**Analizar variables numéricas y categóricas (volver as.numeric o as.factor con lapply)**. Esto es importante, dejar las categóricas como factores para que no se trasgreda información durante la regresión.

Crear una hipótesis, hacer modelo y resumen del modelo. Ver asteriscos, R2Ajustado (explica en \_%), ver pvalue. Interpretar resultados para ver importancia de las variables en tu hipótesis. Aquí entran variables control.

Crear modelos con diferente número de variables independientes. Identificar parámetros para ecuación y=a+bx #a es intercepto/constante y b es pendiente de la variable

**Gráfico en caso de modelos de 2 independientes y 1 dependiente:**

library(scatterplot3d) ; scatterplot3d(hsb[,c('SCI','WRTG','MATH')])

**Gráfico en caso de modelos de 2 independientes, 1 dependiente y 1 dicotómica:**

colors <- paleta[as.numeric(data$dicotomica)]

scatterplot3d(data[,c('v1','v2','dependiente')],color=colors)

**Usar stargazer para comparar modelos**

stargazer(reg1,reg2,reg3,reg4,type="text",df=FALSE) #ver r2 ajustado para cuál explica más

Gráfico🡪 library(ggplot2) ; library(sjPlot)

plot\_models(reg1,reg2,reg3,reg4,vline.color = "grey",m.labels=c("Modelo 1","Modelo 2","Modelo 3", "Modelo 4"))

**EXTRAS MAGALLANES**

**Gráfica de correlación** **2 variables**🡪 library(ggplot2)

base=ggplot(data=hsb, aes(x=independiente, y=dependiente))

scatter = base + geom\_point()

scatter

**Gráfica tridimensional**🡪 library(scatterplot3d) ; scatterplot3d(data[,c('v1','v2','dependiente')])

**Gráfica bidimensional**🡪 base1=ggplot(data=matriz, aes(x=v1, y=dependiente)) ; base1 + geom\_point(aes(color = v3))

Gráfica de boxplot para dependiente con variables dicotómicas🡪 base2=ggplot(data=matriz, aes(x=dicotómica, y=dependiente)) ; base2 + geom\_boxplot(notch = T)

**Gráfica barras de error** para dependiente con variables dicotómicas 🡪 library(ggpubr) ; ggerrorplot(data=data, x = "dicotómica", y = "dependiente") **#se busca que no haya traslape, osea igualdad de valores medios**

**Gráfica para presencia/ausencia de normalidad** (normalidad si los puntos no se alejan de la diagonal)🡪 library(ggpubr) ; ggqqplot(data=matriz,x="dependiente") + facet\_grid(. ~ dicotómicaindep)

Gráfico de 2 variables independientes, 1 dependiente y una dicotómica🡪 base3=ggplot(data=data, aes(x=v1, y=dependiente)) ; base3 + geom\_point(aes(size = v2, color=dicotómica , shape=dicotómica)) + scale\_shape\_manual(values=c(3, 2))

Otro gráfico🡪 base3 + geom\_point(aes(color = v2)) + facet\_grid(~dicotómica)

**Shapiro Test para explicar normalidad**🡪 library(knitr) ; library(magrittr) ; library(kableExtra)

Modelo4=formula(dependiente~dicotómica)

tablag= aggregate(modelo4, data, FUN = function(x) {y <- shapiro.test(x); c(y$statistic, y$p.value)})

**Mann-Whitney es camino no paramétrico para ver la diferencia de valores medios**🡪

Modelo4=formula(dependiente~dicotómica)

tf4=t.test(modelo4,data=hsb)[c('estimate','p.value')] ; wilcoxf4=wilcox.test(modelo4,data=hsb)['p.value'] #igualdad de valores medios si pvalue>0.05

**CORRELACIÓN Y REGRESIÓN**

**CORRELACIÓN**

Se asume normalidad (Pearson) y en casos no paramétricos/no normales (Spearman). No controla variables, no variables fijas.

**REGRESIÓN LINEAL**

Técnica donde hay que definir una variable dependiente y una o más independientes. Las independientes pueden tener rol predictor, dependiendo del diseño de investigación, aunque por defecto tiene un rol explicativo.

La regresión sí quiere informar cuánto una variable (independiente) puede explicar la variación de otra (dependiente), de ahí que es una técnica para probar hipótesis direccionales o asimétricas (las correlaciones tiene hipótesis simétricas). La regresión busca proponer un modelo, es decir una ecuación, que recoja como una variable explicaría a otra. Eso luego se plasma en una ecuación donde se predice el cambio en la dependiente por el aumento/disminución en una de las independientes.

Sólo diagnósticos después de escoger un modelo.

**Analizar variables numéricas y categóricas (volver as.numeric o as.factor con lapply)**. Esto es importante, dejar las categóricas como factores para que no se trasgreda información durante la regresión.

**Visualizar la dispersión de variables de interés con cualquiera de estos 2 gráficos:**

ggplot(data=data, aes(x=independiente, y=dependiente)) + geom\_point() #2 variables

scatterplot3d(data[,#:#]) #3 variables

**Cálculo de coeficientes de correlación:**

1. Ver si se comporta de forma normal con evaluación visual🡪 ggqqplot(data=data,x="dependiente") / ggqqplot(data=matriz,x="dependiente") + facet\_grid(. ~ dicotómicaindep)
2. Shapiro test para confirmar NORMALIDAD en distribución🡪 shapiro.test(data$dependiente) #si pvalue MENOR A 0.05 entonces NO normal

Una vez identificada la normalidad se puede realizar el análisis de correlación: si las variables son normales: Pearson / **no** normales: spearman

f1 = formula(~independiente + dependiente)

Ejemplo Pearson🡪 cor.test(f1,data=data)

Ejemplo Spearman🡪 cor.test(f1,data=data,method='spearman')[c('estimate','p.value')]

\*Rho o cor: ver si es positivo o negativo, ver fuerza 0-3/3-6/6-1

\*Pvalue: significancia. Si es menor a 0.05 sí hay asociación.

**Análisis de regresión lineal:**

modelo1 = formula(dependiente ~ v1 + v2)

reg1=lm(modelo1,data=data)

summary(reg1)

stargazer(reg1,type = "text")

**Chequear summary. En stargazer**: ver asteriscos para significancia en cada variable. Aquí valores para ecuación (los que tienen asterisco, número de encima). PONER TODAS LAS VARIABLES EN LA ECUACIÓN A PESAR DE NO SER SIGNIFICATIVAS. Con **estadístico F (F statistic)** mientras más grande más significativo y ve si modelo explica (si o no, representa al pvalue). R2 ve cuánto explica el modelo con porcentaje (R2 ajustado es SÓLO cuando se comparan varios modelos). Decir si relación es directa o indirecta, en cuánto aumenta/disminuye la dependiente respecto a la independiente.

tanova=anova(reg1,reg2,reg3) ; stargazer(tanova,type = 'html',summary = F) 🡪 tabla de anova (propone que los modelos no difieren (aceptable pasar de uno a otro)). Busca rechazar igualdad de modelos (P<0.05 es significativo el paso).

**Gráfico para varios modelos**🡪 library(ggplot2) ; library(sjPlot)

plot\_models(reg1,reg2,reg3,reg4,vline.color = "grey",m.labels=c("Modelo 1","Modelo 2","Modelo 3", "Modelo 4"))

**DIAGNÓSTICOS DE REGRESIÓN (para lineal y para logística)**

Para que se considere que el modelo de regresión elegido es el adecuado, debemos verificar algunos requisitos a posteriori (todo sobre residuos y modelo):

1. Linealidad: Relación lineal entre x e y. Línea debe tender a ser horizontal🡪 plot(reg1, 1)
2. Homocedasticidad: Se asume que el error del modelo de regresión no afecta la dispersión de la estimación: Los residuos no deberían afectar. Homocedasticidad es que los atípicos no afecten (la distancia entre puntos se mantiene), poco homo es que hay lugar para heterocedasticidad (que sí afecten y la distancia entre los puntos no se mantiene, eso se ve en el gráfico tb como desorden o un cono)🡪 plot(reg1, 3) ; bptest(reg1) #pvalue menor a 0.05 NO hay homocedasticidad
3. Distribución normal de residuos🡪 plot(reg1, 2) ; shapiro.test(reg1$residuals) #pvalue mayor a 0.05 SÍ hay normalidad de residuos
4. No multicolinelidad: Si los predictores tienen una correlación muy alta entre sí, hay multicolinealidad, lo cual no es deseable🡪 VIF(reg1) # coeficientes deben estar entre 1-5, si es mayor a 5 no es aceptable y hay multicolinelidad. Acá puedes usar factorial (EFA) para ver lo de polígamas (print(resfa$loadings,cutoff = 0.5)
5. Valores influyentes: Hay casos particulares, que tienen la capacidad de trastocar lo que casi todos puntos representan. Si se logra identificar y eliminar esos valores se pueden corregir errores del modelo

plot(reg1, 5) #ver en data cuál es el valor del sesgo de la línea horizontal

checkreg1=as.data.frame(influence.measures(reg1)$is.inf)

#Buscar en la tabla checkreg1 los TRUE en el índice de cook y hat para saber qué dato sacar

checkreg1[checkreg1$cook.d | checkreg1$hat,]

**EFA (EXPLORATORIO)**

*Hacemos análisis factorial para reducir las variables en otras variables resumen. Mientras la clusterización agrupaba filas, la factorización agrupa columnas. REDUCE VARIABLES EN VARIABLES RESUMEN, AGRUPA COLUMNAS/VARIABLES. Estas nuevas variables tienen un nombre: "variable latente". Usa correlación.*

**Primero analiza matriz de correlación**

corMatrix=polycor::hetcor(data)$correlations

**Exploración con y sin significancia. Si hay bloques correlacionados, puede haber buen análisis factorial:**

ggcorrplot(corMatrix,

p.mat = cor\_pmat(corMatrix),

insig = "pch") #el pch ayuda a ver cuáles NO son significativos realmente

**Diagnóstico de la matriz de correlaciones**

psych::KMO(corMatrix) 🡪 Keyser Meier Olkin ayuda a verificar si es aplicable el factorial. Ver Overall MSA y que sea mayor que 0.6. Delimita calidad del análisis factorial, ver peso de cada variable observable. Las de peso ligero no aportan. (también usar KMO(corMatrix)$MSA ; KMO(corMatrix)$MSAi)

cortest.bartlett(corMatrix,n=nrow(theData))$p.value>0.05 🡪 prueba de identidad si es que las variables se correlacionan consigo mismas, es como multicolinealidad. Se espera que sea False.

is.singular.matrix(corMatrix) 🡪 prueba singular para ver si variables se correlacionan sólo con una variable. Se espera que sea False.

**Número de factores**

fa.parallel(data,fm = 'ML', fa = 'fa') 🡪 recomendación de número

**Redimensionar a número de factores que salieron en el parallel**

resfa <- fa(data,nfactors = #,cor = 'mixed',rotate = "varimax",fm="minres")

print(resfa$loadings)🡪 resultados iniciales, ve aportes de cada variable a cada factor. Ver Cumulative Var y analizar valores en los MR

print(resfa$loadings,cutoff = 0.5) 🡪 resultados más significativos para saber a qué factor va cada observable. Esta función borra valores menores a 0.5 para mayor precisión, sin embargo se sabe que hay variables observables que se encuentran en 2 factores (son polígamas, por así decirlo)

**Visualización en diagrama**🡪 fa.diagram(resfa)

**Pruebas**:

resfa$crms (raíz de error cerca cero) 🡪 si cerca a 0 buen ajuste

resfa$RMSEA (raíz del error menor 0.05) 🡪 menor a 0.05 buen ajuste

resfa$TLI (Tucker-lewis) 🡪 tiende a 1 para buen ajuste, debe ser mayor a 0.9

sort(resfa$communality) 🡪 qué variables aportaron más a los factores

sort(resfa$complexity) 🡪 qué variables contribuyen a más de un factor

**Volver valores nuevo data frame**

FAcasos= as.data.frame(resfa$scores)

summary(FAcasos)

dataFA=cbind(data ,as.data.frame(FAcasos))

**Valores proyectados (dar nombre a los factores resultantes). Relaciones y nuevas variables con normalize (usa la data original y la nueva):**

data$faVar1=normalize(dataFA$MR1,

method = "range",

margin=2, # by column

range = c(0, 10))

**CFA (CONFIRMATORIO)**

*Evalúa qué tan bien resultó el EFA desde un modelo de hipótesis y la conexión entre las observables y las latentes resultantes. Usa covarianza, usa SEM (structural equation model). SEM calcula las latentes usando CFA, y lleva a cabo una mejor regresión con las variables latentes (mejor que sólo usar lm, lm no sabe que usa latentes). El SEM nos informa que aun cuando por partes parece haber correspondencias, en su totalidad la teoría planteada tiene problemas para ser generalizada con la data disponible. Correr un SEM es mejor que usar regresion lineal ante la presencia de latentes y cuando queremos probar TODO en simultáneo, ve conexión con latentes.*

*Si la exploración apoyaba nuestro marco teórico, podemos proponer cómo construir los índices y analizamos resultados exitosos si cada indicador tiene una buena conexión con su latente (ver p valor). Se realizan tests (Chi2, Tucker Lewi, Raiz error cuadrático medio). Usa regresión para comparar. Se describe el modelo (teoria) a probar con las latentes y la regresión.*

resfa <- fa(data,nfactors = 3,cor = 'mixed',rotate = "varimax",fm="minres")

fa.diagram(resfa)

**Creación modelo** (V1,2,3 son variables resumen, letras son variables dentro) **y fit.**

model <- ' V1 =~ a + b + c + d + e

V2 =~ f+ g + h

V3 =~ i + j + k '

fit <- sem(model, data=Data)

**Tests**

allParamCFA=parameterEstimates(fit,standardized = T)

allFitCFA=as.list(fitMeasures(fit)) 🡪 evalúa SEM

**Ver p value para analizar si indicadores tienen buena relación con los latentes**

kable(allParamCFA[allParamCFA$op=="=~",])

**Análisis modelo**

allFitCFA[c("chisq", "df", "pvalue")] # pvalue>0.05 no significativo (chi cuadrado)

allFitCFA$tli # > 0.90 significativo (Tucker levis)

allFitCFA[c('rmsea.ci.lower','rmsea' ,'rmsea.ci.upper')] # debe ser menor a 0.05 para que sea significativo el error cuadrático

**Se añaden los índices a una nueva data**

Data2=as.data.frame(cbind(Data,lavPredict(cfa\_fit)))

summary(Data2)

**Podemos hacer una regresión con los índices**

summary(lm(V1~ V2 + V3 ,data = Data2))

**Gráficos**

old=apply(Data2[,c("a","b","c","d","e")],1,mean)

new=Data2$V1

plot(old,new)

**Correlación**

cor(old,new)

**REGRESIÓN LOGÍSTICA**

**Variable dependiente es categórica**. No es como una regresión lineal porque la logística busca hallar la probabilidad. Probabilidad de éxito o fracaso, aquí trabajamos con variables dicotómicas. En la lineal solo se predice el valor de la variable dependiente numérica. Buscamos ver que el aumento en una variable aumente o disminuya la probabilidad de éxito en una relación dicotómica.

El primer nivel de cada categórica es la categoría de referencia. Utiliza tabla de contingencia para relacionar fila y columna. Analiza probabilidades. Analiza ODDS (odds suele representarse además como la razón entre dos probabilidades: la probabilidad que ocurra un evento dividido por la probabilidad que NO ocurra ese evento). Odds ratio divide odds y ve la diferencia de categoría dentro del evento. Contingencia muestra porcentajes también.

La regresión logística modela el comportamiento de la probabilidad del evento de interés. Toma en cuenta la probabilidad en términos de odds también. Ve referencias y efectos que alterarán la probabilidad.

Al comparar varias regresiones logísticas vemos que estos presentan valores diferentes del criterio de información de Akaike (AIC), se considera que un modelo es mejor si tiene un AIC menor a los demás. Esto sugiere que el tercer modelo es el mejor y el segundo el peor. Como los valores del AIC están muy cercanos confirmemos usando el test de razón de verosimilitud (likelihood ratio test -LRT)

**Hacer la limpieza, volver ordinales, recode, merge, set.seed si es necesario.**

base[,-dependiente]=lapply(base[,-dependiente], as.ordered)

**Regresión (binomial porque recuerda que ves probabilidades):**

rlog2=glm(dependiente~., data=base, family = binomial) 🡪 . significa tomar todas las variables

library(stargazer)

stargazer(rlog2,type="text") 🡪 ver SÓLO asteriscos para significancia (no interpretes). Ver L

\* Akaike Inf. Crit. 🡪 sirve para comparar modelos, es como el R2 ajustado. Menor AKAIKE, mejor el modelo, usualmente el mejor es el 3ro.

**Cómo entender el modelo:**

library(margins)

(model = margins(rlog2)) 🡪 aquí sí se interpreta. Un punto en la v1 va a aumentar en ### la probabilidad de la dependiente. Igual se interpretan los que no tengan asterisco. No elimines variables. Aumenta/disminuye la probabilidad de éxito de la dependiente por cada aumento de ## en la variable. NO HAY ECUACIÓN, SÓLO INTERPRETACION. Te da resultados por niveles ordinales. VE PROBABILIDADES.

**Gráfico marginalidad:**

library(ggplot2)

base= ggplot(margins,aes(x=factor, y=AME)) + geom\_point()

base + geom\_errorbar(aes(ymin=lower, ymax=upper))

**Comparar modelos logísticos (como tabla de anova)** **test de razón de verosimilitud (likelihood ratio test -LRT):**

library(lmtest)

lrtest(rlog1,rlog2, rlog3)

library(pander)

pander(lrtest(rlog1,rlog2, rlog3),caption = "LRT para los tres modelos")

**Evaluación del modelo:**

predicted=plogis(predict(rlog2,base[,-dependiente])) 🡪 eliminas la dependiente pq solo trabajas con independientes

library(InformationValue)

confusionMatrix(base$dependiente,predicted) 🡪 la antidiagonal ∕ debería tener numeros muy bajos o 0 para saber qué tan bueno es el modelo. La interpretación es que el modelo no explica bien algunas variables.

sensitivity(actuals = as.numeric(base$dependiente),predictedScores = predicted) 🡪 qué tanto acertamos a predecir el evento. Qué tanto puede predecir el modelo

specificity(actuals = as.numeric(base$dependiente),predictedScores = predicted) 🡪 no ocurrencia, predicción del fallo

exp(cbind(OR = coef(rlog2), confint(rlog2))) 🡪 ODDS ratio ve asociación de las categoría con el modelo

\*Se pueden predecir valores concretos.

**Conclusión**🡪 sensitividad debería ser mayor que especificidad. Analizar si variables son significativas. Interpretar matriz de confusión en si es que se necesitan más variables.