Misc R commands and tips

Matthew Castelo, [matthew.castelo@medportal.ca](mailto:matthew.castelo@medportal.ca)

### - Steps for cleaning data

1. Load data and check type of each variable

dataset <- read\_csv('C:/Users/matth/Documents/data.csv') #Tibble format  
str(dataset)

1. Convert variables as needed, including factors with factor labels and levels
2. Check for duplicates

sum(duplicated(dataset))

1. Use summary(dataset) to look at each variable and examine for outliers, nonsensical values. Consider setting to missing
2. Look for missing with:

colSums(is.na(dataset))  
md.pattern(dataset, rotate.names = TRUE)

1. Remove or impute missing data. For remove use:

na.omit(dataset)  
#Or  
dataset\_1 <- dataset %>%   
 na.omit()

1. Consider using ggpairs(dataset) to look for preliminary relationships between variables

### - In RStudio, the shortcut for a ‘pipe’ in Tidyverse AKA ‘%>%’ is CTR-SHIFT-M

### - Greater than or equal to (>=), less than or equal to (<=)

### - To see a dataset simply type name in and hit enter, or try the following:

View(dataset) #Table format  
head(dataset, 10) #First 10 rows  
tail(dataset, 10) #Last 10 rows

### - To see underlying dataset structure use:

glimpse(dataset) #Tidyverse format  
str(dataset)

### - Changing multiple column names very efficiently

#First, to get the existing column names  
colnames(dataset)  
  
#Then, create a vector of the new column names that are wanted  
colnames(dataset) <- c('new\_var\_1', 'new\_var\_2', 'new\_var\_3')

### - Fast descriptive statistics. For all variables:

summary(dataset)

For a single variable:

summary(dataset$variable)

### - To select multiple different values of a variable use %in% c(value1, value2, etc):

dataset %>%   
 filter(variable %in% c(1, 2, 3))

### - if/else. Logical statement following by action if true, then action if false.

dataset %>%   
 mutate(new\_variable = ifelse (old\_variable == 'value', 1, 0))

To use a straight if or else (this will only work on elements, not vectors):

if (logical statement) { code }  
 else { code }

### - Tips for missing data

To find number of missing in each column:

colSums(is.na(dataset))  
#For patterns in all the data  
md.pattern(dataset, rotate.names = TRUE)

To replace values with missing:

dataset$variable[dataset$variable==99] <- NA  
#Or, using dplyr  
replace(variable, condition, replacement)  
#Over multiple columns, use sapply  
variables <- c('var\_1', 'var\_2', 'var\_3')  
dataset[variables] <- sapply(dataset[variables], function(x) ifelse(x==0, NA, x))

To remove missing values:

dataset\_1 <- dataset %>%   
 na.omit()

Good piece of code to find patterns in missing data (for one variable):

dataset %>%  
 mutate(miss\_variable = is.na(variable)) %>%  
 group\_by(miss\_variable) %>%  
 summarize\_all(median, na.rm=TRUE)

### - Converting to factors

For a single variable:

dataset$variable <- factor(dataset$variable, levels = c(), labels = c())

To convert multiple variables to factors at the same time:

categorical\_vars <- c("variable\_1","variable\_2","variable\_3")  
dataset[categorical\_vars] <- lapply(dataset[categorical\_vars], factor)

To drop empty factor levels:

droplevels()

### - Converting from character to date

dataset\_1 <- dataset %>%   
 mutate(new\_date = as.Date(old\_date, format = '%Y-%m-%d'))

The following symbols are used to represent various dates:

%d - day as number %a - abbreviated weekday (Mon) %A - unabbreviated weekday (Monday) %m - month as number %b - abbreviated month (Jan) %B - unabbreviated month (January) %y - two-digit year (07) %Y - four-digit year (2007)

### - Converting all character variables to factors, and in general performing the same function on multiple variables in dplyr - across()

dataset %>%   
 mutate(across(where(is.character), factor))  
  
#Can create new columns with pre-defined names  
iris %>%  
 group\_by(Species) %>%  
 summarise(across(starts\_with("Sepal"), mean, .names = "mean\_{.col}"))

### - Perform a simple left join

merged\_dataset <- dataset\_1 %>%   
 left\_joint(dataset\_2, by='study\_id')

### - Writing a function

my\_function <- function(x) {  
 x <- x^2  
 return(x)  
 }  
  
my\_function(dataset)

### - Running simple loops. The code below will run the loop 6 times:

x <- c() #Define empty elements before using them in the loop  
  
for (i in 1:6) {   
 x[i] <- i^2  
 }  
print(x)

### - Splitting one dataset into multiple based on the levels of a factor

#The output will be a list of datasets - can access them using [[1]], [[2]], etc  
split\_dataset <- dataset %>%   
 mutate(splitting\_var = factor(splitting\_var)) %>%   
 group\_split(splitting\_var)  
  
#Re-combining them:  
combined\_dataset <- rbind(split\_dataset[[1]], split\_dataset[[2]], split\_dataset[[3]])

### - Robust (sandwich-type) variance estimator when using propensity score weights

* <https://www.coursera.org/lecture/crash-course-in-causality/data-example-in-r-Ie48W>
* a regular GLM model is used with the weight statement
* the sandwich package is used to estimate the covariance structure of the beta coefficients (that’s the vcovHC function)
* the diag function gives only the variances for each beta (diagonals in the covariance matrix)
* the square root of the variances gives the standard error
* the confidence intervals are then manually calculated from the standard errors

x <- glm(highpain ~ treatment,weights=weight\_ate, data=dataset3, family=binomial(link='log'))  
beta <- coef(x)  
SE <- sqrt(diag(vcovHC(x,type='HC0')))  
lci <- (beta[2]-1.96\*SE[2])  
uci <- (beta[2]+1.96\*SE[2])  
estimate <- c(beta,lci,uci)  
exp(estimate)  
  
#p-value  
z\_stat <- beta/SE  
p\_values <- pchisq(z\_stat^2, 1, lower.tail=FALSE)  
p\_values

For robust variance estimator with the cluster defined use slightly different vcov function:

SE <- sqrt(diag(vcovCL(model\_output, cluster=dataset$cluster\_variable)))

### - Frequency tables

table(dataset$variable)  
#Or, for 2x2 tables  
table(dataset$var\_1, dataset$var\_2)  
#To make a properly formatted confusion matrix (2x2 table) may need to change the order:  
table(dataset$var\_1, dataset$var\_2)[2:1, 2:1]

To get nicely formatted contingency 2x2 tables using the gmodels package:

CrossTable(x = dataset$var\_1, y = dataset$var\_2)  
#Can add option chisq=TRUE

### - Function to normalize a variable

#Min-max normalization  
normalize <- function(x) {  
 return((x - min(x)) / (max(x) - min(x)))  
 }

an alternative to min-max normalization is z-score standardization:

z\_normalize <- function(x) {  
 return((x - mean(x)) / (sd(x)))  
 }

### - Splitting dataset into train and test sets

train <- dataset %>%   
 slice\_sample(prop=0.25) #splitting into 25% for train set and rest for test  
  
train\_index <- as.numeric(rownames(train))  
  
test <- dino\_normalized[-train\_index,]

### - Estimating relative risks and risk differences instead of odds ratios:

#RR  
model <- glm(outcome ~ exposure, data=dataset, family=binomial(link='log'))  
summary(model)  
  
#RD  
model <- glm(outcome ~ exposure, data=dataset, family=binomial(link='identity'))  
summary(model)

### - Indexing [row,column]

dataset[2:5] will select columns 2 to 5 in the dataset of interest. These are equivalent:

dataset$variable1   
dataset[,1]

Can also use indexing for logical statements and changing values/variables:

dataset$age[dataset$age > 99] <- NA #This sets age>99 equal to missing

Can use more complex indexing to remove rows only for certain commands. For example, if you want to run a ttest for age only on treatment 1 and treatment 2, removing treatment 3:

t.test(age ~ treatment, data=dataset[!dataset$treatment==3,])

Can do the same but more complex - say now there are 5 treatments and only want to compare the first 2:

t.test(age ~ treatment, data=dataset[!dataset$treatment %in% c(3,4,5),])

### - Using TableOne package. CreateTableOne gives the package all the information and options are selected in the print function. Make sure categorical variables are factors.

all\_vars <- c('var\_1','var\_2','var\_3')  
continuous\_vars <- c('var\_1','var\_2')  
  
table\_1 <- CreateTableOne(vars=all\_vars, strata="exposure\_variable", data=dataset, test=TRUE) #this will give p-values  
table\_1\_done <- print(table\_1, nonnormal=continuous\_vars, #this will give median/IQR for variables you choose  
 smd=TRUE, #Standardized mean differences  
 minMax=TRUE, #Will give range instead of IQR)  
 write.csv(table\_1\_done, file="C:/Users/matth/Documents/R/file\_name.csv")

### - Simple univariate tests

wilcox.test(outcome ~ predictor, data=dataset, na.rm=TRUE, #if you want to remove NAs  
 paired=TRUE) #for paired you have to have all of one pair then the next in order  
t.test(same as above)  
fisher.test(dataset$var\_1, dataset$var\_2)  
chisq.test(same as above)

### - Saving a publication quality ggplot

#Run plot you want to save, then:  
ggsave('C:/Users/matth/Documents/R/sample\_plot.png',  
 dpi=300, dev='png', height=10, width=10, units="cm")

### - Adding horizontal/vertical lines and labels to ggplot2

+geom\_segment(aes(x=0, xend=20, y=5, yend=5))  
+annotate("text",x=1,y=1,hjust=0,#This aligns to the left of the text  
 label="What you want it to say")  
#Adding axis and titles  
+labs(x="X-axis title",y="Y-axis title",title="Title of plot")

### - Adding manual rectangle in ggplot

+ geom\_rect(aes(xmin = 0, xmax = 10, ymin = 2, ymax = 4),  
 fill = "transparent", color = "red", size = 1)

### - Manually setting x and y axis breaks in ggplot

+ scale\_x\_continuous(breaks=seq(0,100,5)) #Will go from 0 to 100 in steps of 5

### - Log transformation for axis when negative values are present

#Since the log of a negative number does not exist, need GGALLIN package  
+ scale\_y\_continuous(trans=pseudolog10\_trans)  
  
#Alternatively, if negative values don't make sense can impute all negatives as zero. Then add a constant of 1 to all values. That works because log(1) = 0 so the zeros appropriately remain as zero  
add\_one\_log <- function(x) {  
 x <- log(x+1)  
 x <- round(x, digits=2)  
 return(x)  
}  
  
dataset %>%   
 mutate(across(everything(), add\_one\_log))

### - Adding strings on the end of variable values using paste function

paste(variable\_name, "text or number you want on the end", sep="\_")   
#This will put and underscore and then whatever you have in the middle on the end of the variable value

### - Manual code for confidence intervals (regular variance estimator, not robust)

x <- glm(response ~ predictor, data=dataset, family=binomial(link=logit))  
beta <- coef(x)  
SE <- sqrt(diag(vcov(x)))  
lci <- (beta[2]-1.96\*SE[2])  
uci <- (beta[2]+1.96\*SE[2])  
estimate <- c(beta,lci,uci)  
exp(estimate) #If you need to exponentiate

### - Removing strings from values and manipulating text. Use the ‘stringr’ package and works well within dplyr.

For example, removing the $ sign from a variable:

dataset$new\_variable <- str\_remove(dataset$old\_variable, "$")

To remove white spaces before and after:

str\_trim()

To remove all of a group of characters:

str\_remove\_all("-") #This will remove dashes

To convert upper case to lower case:

str\_to\_lower()

To replace characters in a string, for example dashes with spaces:

str\_replace\_all(variable, "-", " ")

To remove characters at the beginning or end of a string:

str\_sub(variable, start=1, end=4) #Will remove everything after position 4

Removing non-ASCII chars (e.g. emojis)

gsub("[^\x01-\x7F]", "", variable)

Good functions are also contained in the ‘tm’ package

stripWhitespace(variable) #Stripping extra spaces between words  
removeWords(variable, stopwords('english')) #Removing English stopwords

Concatenate two character variables into one

new\_variable = str\_c(var\_1, var\_2, sep = "-") #Separating with a dash

### - Text-based machine learning methods

A corpus contains all the words contained in all documents/observations, after cleaning. Cleaning usually includes removing puncuation, digits, stop words, converting to lower case, and removing extra white spaces. The ‘tm’ package is used.

library(tm)  
corpus <- Corpus(VectorSource(dataset$text\_variable)) #Creating corpus

Now the frequency of each word in each document is summed to create a document-term sparse matrix

matrix <- DocumentTermMatrix(corpus)

Sparse terms can be removed to reduce high dimensional data

new\_matrix <- removeSparseTerms(DocumentTermMatrix(corpus), 0.99)

However, the format is in a sparse matrix for memory purposes. For calculations, an actual matrix needs to be created

as.matrix(new\_matrix)

Sometimes, zero counts can cause problems and those documents/observations need to be removed. However it can be difficult because the text labels are usually in a separate vector. This code will solve that problem

library(slam)  
#Identifying documents that have zero occurrences and will throw errors later  
matrix\_cleaned <- docu\_term\_matrix[row\_sums(docu\_term\_matrix) > 0, ]  
#Extracting the row names of the empty documents  
index <- matrix\_cleaned$dimnames$Docs  
  
#Filtering where the variable ID above equals to non-empty documents and therefore the labels will still match the document term matrix  
dataset\_cleaned <- original\_dataset %>%   
 mutate(ID = as.character(row\_number())) %>%   
 filter(ID %in% index)

A better method is to go one step further and create a weighted Tf-Itf (term-frequency/inverse-term-frequency) matrix. This takes into account how often a word is used in the entire corpus, so common words among all observations are weighted less, and rare but important words weighted more.

weighted <- weightTfIdf(docu\_term\_matrix, normalize = TRUE)  
matrix\_weighted <- as.matrix(weighted)

Creating a word cloud

library(wordcloud)  
freq <- colSums(matrix\_weighted)  
  
set.seed(2020)  
wordcloud(names(freq), freq)

Now the matrix weighted by Tf-Itf can be passed to k-means clustering, classification, etc.

### - Computing intra-class or intra-cluster correlation coefficients froma mixed effects linear model

Using the nlme package. ICC can only be reliably calculated for random-intercepts models. It is defined as:

Variance(random intercept) / Variance(random intercepts) + Variance(residuals)

In a fixed effect only linear model, the residuals are the only source of variance estimated in the model. In a random intercepts model, there are two sources, the random intercept and residuals. The ICC is the proportion of total variance explained by the random intercept.

library (nlme)  
  
r\_intercept\_model <- lme(response ~ predictor\_1 + predictor\_2,  
 random = ~ 1 | patient\_id, #Random intercepts  
 data = dataset)  
  
summary(r\_intercept\_model)  
  
#Look at the standard deviation of the random intercept and the residuals in the random effects section of the model output. Square them to get the variances then follow the standard equation for ICC

### - Displaying variance - covariance matrices in mixed models using nlme

Using lme(), you can choose to display 3 types of VarCov matrices using the getVarCov() function. The default is to display the random effects. Say you had random intercepts and random slopes for 2 covariates, the VarCov matrix would be a 3x3 matrix. This is called the G matrix for random effects.

If you specify type = “conditional”, the VarCov matrix for residuals for individuals is displayed. This is the VarCov matrix for measurement occasion 1, 2, …, n for individuals. This is the R matrix. Unless repeated measures are specified, this is just sigma\*(Identity matrix) because normally measurements are assumed to be independent.

If you specify type = “marginal”, this is the VarCov matrix for the response at measurment occasions for the population. This basically takes into account all the random and fixed effects and tells you how they are related across measurement occasions for the entire population. If you built a random intercept and random slopes mixed model to approximate an unstructured covariance matrix for repeated measures, your marginal VarCov matrix would be analogous to the conditional (default) VarCov matrix in the repeated measures model (gls() function).

#Compound symmetry repeated measures  
cs\_model <- gls(response ~ predictor,  
 corr = corSymm(, form = ~ measure\_occasion | id),  
 weights = varIdent(form = ~ 1 | measure\_occasion),  
 data = dataset)  
  
getVarCov(cs\_model)  
  
#Note for repeated measures we used the gls() function not lme() and the default behaviour of getVarCov() is different for the two model objects. For gls it prints the R matrix. For lme it prints the G matrix (random effects VarCov matrix).  
  
#This should be analogous to the following:

### - Computing VIFs for multicollinearity using ‘car’ package

model <- lm(response ~ predictor, data=dataset)  
vif(model)

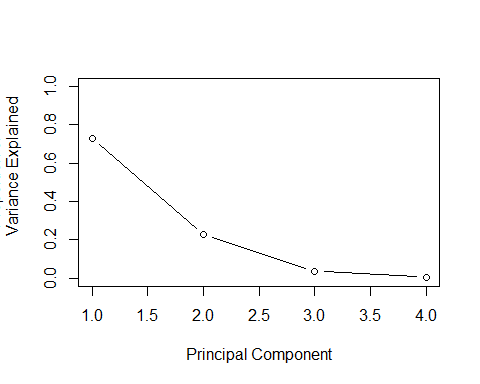
### - Linear discriminant analysis and PCA with biplots

Linear discriminant analysis (LDA)

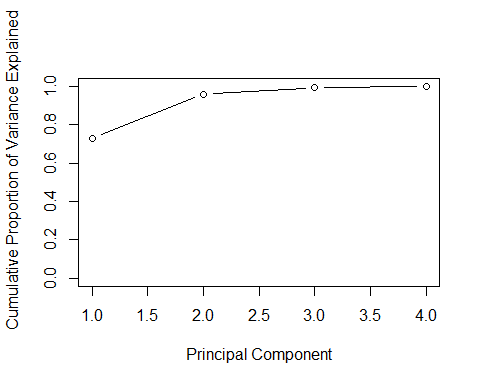
library(ggord)  
library(MASS)  
  
#First can consider scaling the data to help with normality assumption  
iris\_1 <- iris %>%   
 mutate(across(-Species, scale))  
  
lda\_model <- lda(Species ~ ., data=iris\_1) #Using all predictors (must be continuous)  
#Each linear discriminant axis tried to maximize separateness and are independent from each other - the number of linear discriminant axes = 1 - (# of classes). LD1 and LD2 in the model output gives the beta coefficients to create the first and second linear discriminant axes, respectively  
  
lda\_preds <- predict(lda\_model)  
#Gives the predicted class, posterior probabilities, and points on the linear discriminant axes for each observation  
  
#To create a simple biplot with each observation plotted on the first 2 linear discriminant axes using the ggord package (more detailed code in PCA below)  
ggord(lda\_model, iris$Species)+  
 theme\_classic()  
  
#The LDA function also can perform leave-one-out cross validation. The following code performs this and gives an example where the macro-averaged precision, recall, and F1 score can be calculated for a 3-class example (more complicated than simple sensitivity/specificity)  
  
lda\_cv <- lda(class ~ ., data=dataset, CV=TRUE) #leave one out cross validation  
table <- table(lda\_cv$class, dataset$class) #Make sure confusion matrix is formatted correctly  
  
precision\_1 = table[1,1] / (table[1,1]+table[1,2]+table[1,3])   
precision\_2 = table[2,2] / (table[2,1]+table[2,2]+table[2,3])  
precision\_3 = table[3,3] / (table[3,1]+table[3,2]+table[3,3])  
precision\_avg = (precision\_1 + precision\_2 + precision\_3)/3  
  
recall\_1 = table[1,1] / (table[1,1]+table[2,1]+table[3,1])  
recall\_2 = table[2,2] / (table[1,2]+table[2,2]+table[3,2])  
recall\_3 = table[3,3] / (table[1,3]+table[2,3]+table[3,3])  
recall\_avg = (recall\_1 + recall\_2 + recall\_3)/3  
  
f1\_avg = (2\*precision\_avg\*recall\_avg)/(precision\_avg+recall\_avg)

PCA

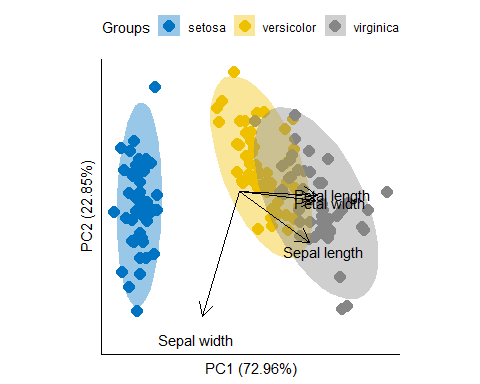
pca\_out <- prcomp(iris[!names(iris) == 'Species'],   
 scale = TRUE,   
 retx=TRUE) #This gives the loadings (coefficients)  
  
# These are the loadings, the coefficients connecting the features and the principal components  
pca\_out$rotation  
  
# The rotated data  
pca\_out$x  
  
#Looking at the proportion of variance explained by each PC  
pve <- ((pca\_out$sdev)^2)/sum(((pca\_out$sdev)^2))  
plot(pve, xlab="Principal Component", ylab="Proportion of  
 Variance Explained ", ylim=c(0,1) ,type="b")



plot(cumsum(pve), xlab="Principal Component", ylab="  
 Cumulative Proportion of Variance Explained ", ylim=c(0,1) ,  
 type="b")



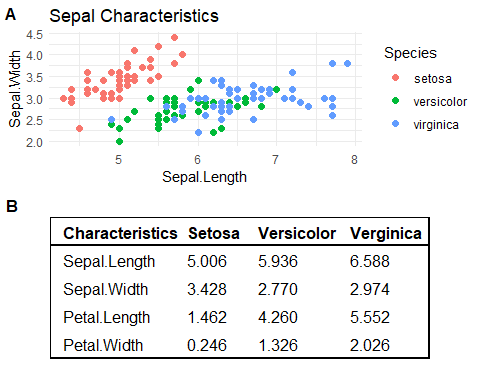
cumsum(pve) #Cumulative sum of variance explained for increasing # of principal components  
  
#Can use ggord for PCA as well to make biplots  
library(ggord)  
library(ggsci) #For nice colors  
  
labels <- list(Sepal.Length=' `Sepal length` ', #Changing arrow names  
 Sepal.Width=' `Sepal width` ',  
 Petal.Length=' `Petal length` ',  
 Petal.Width=' `Petal width` ')  
  
ggord(pca\_out, iris$Species,   
 vec\_lab=labels, #To have nice names on arrows  
 vec\_ext=3, #To control how long the arrows are (they remain proportional)  
 alpha\_el=0.4, #Transparency of the cluster bubbles   
 size=4) + #Size of the data points  
 theme\_classic()+  
 theme(axis.text.x = element\_blank(),  
 axis.text.y = element\_blank(),  
 axis.ticks=element\_blank())+  
 theme(legend.position = 'top')+  
 scale\_fill\_jco()+  
 scale\_color\_jco()

 ### - Converting data from long to wide format (and back) using dplyr

#The pivot\_longer() and pivot\_wider() functions  
dataset %>%   
 pivot\_wider(names\_from = label\_variable,  
 values\_from = values\_variable)  
  
dataset %>%   
 pivot\_longer(cols = c(var\_1, var\_2, var\_3),  
 names\_to = 'overall\_var',  
 values\_to = 'value')

### - Creating combined ggplots with inset tables using cowplot function

library(cowplot)  
library(grid)  
library(gridExtra)  
library(gtable)   
  
#In this example, 1 ggplot is combined with a simple table and simple formatting  
ggplot\_1 <- iris %>%   
 ggplot(aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +  
 geom\_point(size=2)+  
 theme\_minimal()+  
 labs(title = "Sepal Characteristics")  
  
mean\_characteristics <- iris %>% #This summarizes mean characteristic and reshapes table  
 group\_by(Species) %>%   
 summarize(across(everything(), mean)) %>%   
 pivot\_longer(cols = c(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width),  
 names\_to = 'characteristic',  
 values\_to = 'value') %>%   
 pivot\_wider(names\_from = Species,  
 values\_from = value)  
  
#Now converting to a table that can be inserted into the figure  
iris\_table <- tableGrob(mean\_characteristics,  
 cols=c('Characteristics', 'Setosa', 'Versicolor', 'Verginica'),  
 theme = ttheme\_minimal(core=list(fg\_params=list(hjust=0, x=0.1)),  
 rowhead=list(fg\_params=list(hjust=0, x=0)),  
 colhead=list(fg\_params=list(hjust=0,   
 x=0.1))),   
 rows = NULL)  
  
#This adds a box around the table  
iris\_table <- gtable\_add\_grob(iris\_table, grobs = rectGrob(gp = gpar(fill = NA, lwd = 2)),  
 t = 1, #We want the box to start at the first row   
 b = nrow(iris\_table), #Want the box to end at the end of table  
 l = 1, #Start box at first column  
 r = ncol(iris\_table)) #Same as above  
  
#Adding another box for the column headings  
iris\_table <- gtable\_add\_grob(iris\_table, grobs = rectGrob(gp = gpar(fill = NA, lwd = 2)),  
 t = 1, b = 1, l = 1, r = ncol(iris\_table))  
  
  
#Now creating the overall figure by combining the two ggplots and formatted table. The way this works is the plot area is a grid 1x1 and the bottom left corner of each figure is given an x/y coordinate and a height/width  
ggdraw() +  
 draw\_plot(ggplot\_1, x = 0, y = 0.5, width = 1, height = 0.5)+  
 draw\_plot(iris\_table, x = 0, y = 0, width = 1, height = 0.5)+  
 draw\_plot\_label(label = c('A', 'B'), size = 12,  
 x = c(0, 0), y = c(1, 0.5))



ggsave('C:/Users/matth/Documents/R/folder/plot\_1.png',  
 dpi=300, dev='png', height=20, width=20, units="cm")

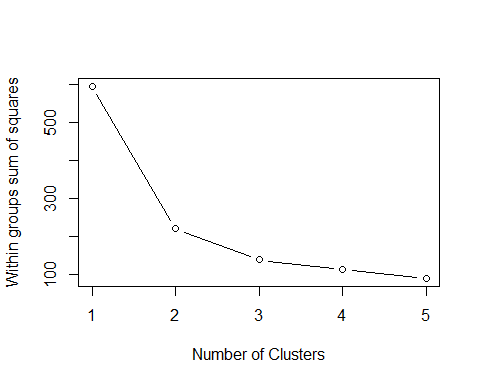
A simpler method for just plots that don’t include tables is the ggarrange() function in the ggpubr package

library(ggpubr)  
  
ggarrange(plot\_1, plot\_2, plot\_3, plot\_4,  
 nrow=2, ncol=2)  
  
ggsave('C:/Users/matth/Documents/R/folder/plot\_1.png',  
 dpi=300, dev='png', height=20, width=20, units="cm")

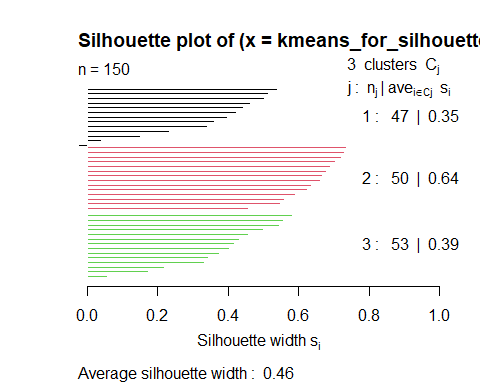
### - K-means clustering, hierarchical clustering and choosing optimal cluster size

K-means

library(cluster)  
  
#Scaling data  
iris\_cluster <- iris %>%   
 mutate(across(-Species, scale))  
  
#Finding optimal number of clusters using a skree plot  
wss <- 0  
  
set.seed(2020)  
# For 1 to 3 cluster centers  
for (i in 1:5) {  
 km\_model <- kmeans(iris\_cluster[1:4], i, nstart=20)  
 # Save total within sum of squares to wss variable  
 wss[i] <- km\_model$tot.withinss  
}  
  
# Plot total within sum of squares vs. number of clusters  
plot(1:5, wss, type = "b",   
 xlab = "Number of Clusters",   
 ylab = "Within groups sum of squares")



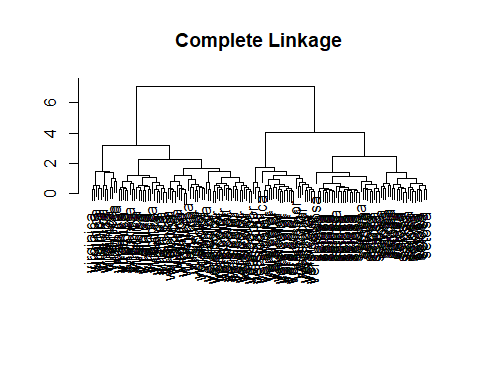
#Look for the 'elbow' on the skree plot for a possible optimal cluster size  
  
  
#Trying to identify number of clusters using silhouette plot (alternative to skree plot)  
#Running a loop to find optimal clusters with silhouette plot  
avg\_sil\_width <- c()  
cluster\_size <- c()  
  
set.seed(2020)  
for (q in 2:5) {  
 kmeans\_for\_silhouette <- kmeans(iris\_cluster[1:4], q, nstart=20)  
 silhouette\_model <- silhouette(kmeans\_for\_silhouette$cluster, dist(iris\_cluster[1:4]))  
 sil\_sum <- summary(silhouette\_model)  
 avg\_sil\_width[q] <- sil\_sum$avg.width  
 cluster\_size[q] <- q  
}  
  
optimal\_cluster <- data.frame(cluster\_size = na.omit(cluster\_size),  
 width = na.omit(avg\_sil\_width))  
  
optimal\_cluster %>%   
 slice\_max(width, n=1) %>%   
 View()  
  
# Plotting 2 clusters on silhouette plot. The width refers to how 'alike' observations are within each cluster. Negative values indicate that observation is more similar to a different cluster from the one to which it was assigned.  
set.seed(2020)  
kmeans\_for\_silhouette <- kmeans(iris\_cluster[1:4], 3, nstart=20)  
silhouette\_model <- silhouette(kmeans\_for\_silhouette$cluster,dist(iris\_cluster[1:4]))  
summary(silhouette\_model)  
  
plot(silhouette\_model, col = c(1:3))



#Evaluating performance of 3 clusters using k means  
set.seed(2020)  
km\_model <- kmeans(iris\_cluster[1:4], 3, nstart=20)  
  
table(km\_model$cluster, iris\_cluster$Species)

Hierarchical clustering

iris\_dist <- dist(iris[1:4])  
h\_clust\_model <- hclust(iris\_dist, method='complete')  
  
plot(h\_clust\_model, labels=iris$Species, main=" Complete Linkage ",   
 xlab ="", sub ="", ylab ="")



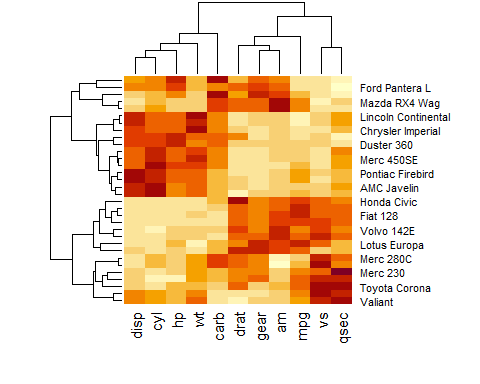
#If we want to cut the tree at three clusters  
cut\_tree <- cutree(h\_clust\_model, k=3)  
  
table(cut\_tree, iris$Species )

### - Heatmaps

The heatmap() function has a strange format by default. It wants each observation to have a unique name/ID and these must be the actual row names rather than another column. This can be done by using the rownames() function. In this example, the mtcars dataset already has each rowname as a unique make/model of a car.

It also requires the input to be a matrix rather than a dataset

mtcars\_scaled <- mtcars %>%   
 mutate(across(everything(), scale))  
  
rownames(mtcars\_scaled) <- rownames(mtcars)  
  
heatmap(as.matrix(mtcars\_scaled))



### - Linear algebra in R

Creating a matrix

my\_matrix <- matrix(c(1,2,3,4,5,6,7,8,9), nrow=3, byrow=TRUE)  
my\_matrix

## [,1] [,2] [,3]  
## [1,] 1 2 3  
## [2,] 4 5 6  
## [3,] 7 8 9

Transpose

t(my\_matrix)

## [,1] [,2] [,3]  
## [1,] 1 4 7  
## [2,] 2 5 8  
## [3,] 3 6 9

Matrix multiplication

matrix\_1 %\*% matrix\_2

Trace

sum(diag(my\_matrix))

Determinant

det(my\_matrix)

End