Pràctica 2: Neteja

Roger Bosch Mateo

12/19/2019

Contents

1 Descripció del dataset

Per a la resolució d'aquesta segona i última pràctica de l'assignatura de *Tipología i cicle de vida de les dades* s'ha decidit escollir un dataset present en el repositori de CU Irvine Machine Learning: Wine Dataset.

El dataset conté els resultats d'haver realitzat un anàlisi químic de vins cultivats a la mateixa regió d'Itàlia però provinents de tres conreus diferents. Aquest està compost per 178 observacions amb un total de 14 atributs cadascun; un atribut que ens especifica quin vi és i tretze més provinents de l'anàlisi químic.

1.1 Atributs

Els atributs són:

- **type**: El tipus de vi que es tracta. Hi ha un total de 3 classes diferents.
- **alcohol**: Graduació d'alcohol.
- **malic**: Concentració d'àcid màlic que és el responsable de l'aciedesa del raïm.
- **ash**: Concentració de cendra (g/L).
- **alcalinity**: Alcalinitat de les cendres
- **magnesium**: Concentració de magnesi.
- **phenols**: Valor total dels fenols del vi que afecten al sabor, color i textura com àcids fenòlics, flavanols o antocianines entre d'altres.
- **flavanoids**: Subgrup dels phenols. Normalment sobre el 90% dels fenols del vi son flavanoids. Afecten doncs també al saber, color i textura del vi.
- **nonflavanoids**: Fenols que no són flavanoids.

- **proanthocyanins**: Proantocianidina, en part dona el color dels vins.
- **color**: Intensitat del color del vi.
- **hue**: Tonalitat del vi.
- **dilution**: Dilució del vi.
- **proline**: Prolina, un aminoàcid present en el vi.

1.2 Importància i objectius de l'anàlisi

A partir d'aquest conjunt de dades es planteja la problemàtica de determinar quines són les variables més important que ens permeten diferenciar entre diversos tipus de vins. A partir d'aquestes variables podrem crear models que ens classifiquin cada observació del vi en un determinat grup.

En el sector aquestes dades són d'interés pre qualsevol empresa que es dediqui al cultiu i fabriacació de vi. A partir d'anàlisi de vins existents en el mercat podrem veure quins vins són més similars als nostres i quins són més dispars, fet que pot ajudar a determinar estratègies de marqueting en la venda d'aquests segons els atributs diferenciadors. A més, si aquests vins els identifiquèssim amb el seu preu podriem inferir en el preu que tindria el nostre.

2 Integració i selecció de les dades d'interès a analitzar.

EL primer às per a realitzar la integració de les dades serà carregar aquestes en memòria. Per a fer-ho, utilitzarem el fitxer CSV de l'enllaç proporcionat i la funció read.csv() de R. Com el fitxer CSV no conté el nom de les variables a l'inici d'aquest, ho indicarem utilitzant el valor Fals a header.

```
# Carreguem les dades
wine <- read.csv("../data/wine.csv", header = FALSE)
```

Seguidament, hem d'afegir els noms corresponents a cada variable. Així ens serà més fàcil tractar-les.

```
# Afegim els HEADERS segons queda especificat en el fitxer que descriu el dataset.
header <- c("type", "alcohol", "malic", "ash", "alcalinity", "magnesium",
            "phenols", "flavanoids", "nonflavanoid", "proanthocyanins",
            "color", "hue", "dilution", "proline")
names(wine) <- header
```

Pel que fa la selecció de les dades d'interés inicialment seran totes les presents en el dataset, perquè totes aquestes són el resultat d'un anàlisi químic i inicialment no podem dir quines són les més irrellevant i obviar-les ja que per fer-ho necessitem analitzar-les.

Cal dir però que realment el que ens interessa es veure si els atributs ens permet classificar els diversos tipus de vins. Evidentment podrem aplicar tant algorismes no supervisats com supervisats per a fer-ho. En els no supervisats el valor del tipus de vi no serà necessari per a la classificació, tot i que pot resultar igualment interessant per veure la proporció d'observacions classificads correctament.

En resum, utilitzarem totes les dades per a realitzar l'anàlisi d'aquestes.

3 Neteja de les dades

Per començar amb la neteja de les dades, visualitzem que les dades s'hagin carregat correctament.

```
str(wine)
```

```
## 'data.frame':    178 obs. of  14 variables:
##  $ type           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ alcohol        : num  14.2 13.2 13.2 14.4 13.2 ...
##  $ malic          : num  1.71 1.78 2.36 1.95 2.59 1.76 1.87 2.15 1.64 1.35 ...
##  $ ash            : num  2.43 2.14 2.67 2.5 2.87 2.45 2.45 2.61 2.17 2.27 ...
```

```
##  $ alcalinity     : num  15.6 11.2 18.6 16.8 21 15.2 14.6 17.6 14 16 ...
##  $ magnesium      : int  127 100 101 113 118 112 96 121 97 98 ...
##  $ phenols        : num  2.8 2.65 2.8 3.85 2.8 3.27 2.5 2.6 2.8 2.98 ...
##  $ flavanoids     : num  3.06 2.76 3.24 3.49 2.69 3.39 2.52 2.51 2.98 3.15 ...
##  $ nonflavanoid   : num  0.28 0.26 0.3 0.24 0.39 0.34 0.3 0.31 0.29 0.22 ...
##  $ proanthocyanins: num  2.29 1.28 2.81 2.18 1.82 1.97 1.98 1.25 1.98 1.85 ...
##  $ color          : num  5.64 4.38 5.68 7.8 4.32 6.75 5.25 5.05 5.2 7.22 ...
##  $ hue            : num  1.04 1.05 1.03 0.86 1.04 1.05 1.02 1.06 1.08 1.01 ...
##  $ dilution       : num  3.92 3.4 3.17 3.45 2.93 2.85 3.58 3.58 2.85 3.55 ...
##  $ proline        : int  1065 1050 1185 1480 735 1450 1290 1295 1045 1045 ...
```

Amb una primera ullada no veiem res estranyat durant la càrrega d'aquestes. Totes la variables que s'han carregat són de tipus numèric, tal i com esperavem per la descripció del dataset.

3.1 Tractament de zeros o elements buits

Usualment, en els conjunts de dades s'utilitzen diferents valors per marcar l'ausència d'aquests com poden ser els "0", els "NA" o fins i tot alguns caràcters com podria ser "?". Segudiament, comprovarem si les dades carregades presenten alguns d'aquests valors i en cas afirmatiu decidirem com procedir.

- Elements buits:

```
# Valors buits en les variables
colSums(is.na(wine))
```

```
##            type         alcohol           malic             ash      alcalinity
##               0               0               0               0               0
##       magnesium         phenols      flavanoids    nonflavanoid proanthocyanins
##               0               0               0               0               0
##           color             hue        dilution         proline
##               0               0               0               0
```

- Zeros:

```
# Valors buits en les variables
colSums(wine == 0)
```

```
##            type         alcohol           malic             ash      alcalinity
##               0               0               0               0               0
##       magnesium         phenols      flavanoids    nonflavanoid proanthocyanins
##               0               0               0               0               0
##           color             hue        dilution         proline
##               0               0               0               0
```

- Caràcters: Com totes les dades són numèriques tampoc trobem cap cas.

Per tant podem dir que les dades carregades d'aquest conjunt de dades no presenten zeros ni elements buits, i per tant, no caldrà tractar-los.

Evidentment és mol usual trobar casos en els que aquests valors són presents, i segons cada cas podrem decidir per exemple si eliminar aquestes observacions, inferir un valor (ja sigui la mitjana o amb algorismes més complexos com knn) o decidir si mantenir el valor (un cero pot ser perfectament un valor vàlid).

3.2 Identificació i tractament dels valors extrems

Els valors extrems (outliers) són aquells valors de les observacions que es troben tres desviacions estàndard per sobre o per sota. Una manera senzilla de detecar-los és amb la funció **boxplot.stats** que ens crea un llistat amb aquests valors.

Crearem doncs un bucle senzill en el que si les dades presenten outliers ens mostri els valors d'aquests, i a més un petit resum d'aquella columna per poder determinar més correctament que fer en cada cas.

```
print("***** Outliers identification *****")
```

```
## [1] "***** Outliers identification *****"
```

```
for (column in names(wine)){
  out_values <- boxplot.stats(wine[[column]])$out
  # If no outliers
  if(length(out_values) == 0){
    print(paste0("- No outliers found for attribute ", column))
  }
  # if there are outliers
  else{
    print(paste0("- ", column, ":"))
    print(out_values)
    print(summary(wine[[column]]))
  }
}
```

```
## [1] "- No outliers found for attribute type"
## [1] "- No outliers found for attribute alcohol"
## [1] "- malic:"
## [1] 5.80 5.51 5.65
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.740   1.603   1.865   2.336   3.083   5.800
## [1] "- ash:"
## [1] 3.22 1.36 3.23
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.360   2.210   2.360   2.367   2.558   3.230
## [1] "- alcalinity:"
## [1] 10.6 30.0 28.5 28.5
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   10.60   17.20   19.50   19.49   21.50   30.00
## [1] "- magnesium:"
## [1] 151 139 136 162
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   70.00   88.00   98.00   99.74  107.00  162.00
## [1] "- No outliers found for attribute phenols"
## [1] "- No outliers found for attribute flavanoids"
## [1] "- No outliers found for attribute nonflavanoid"
## [1] "- proanthocyanins:"
## [1] 3.28 3.58
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.410   1.250   1.555   1.591   1.950   3.580
## [1] "- color:"
## [1] 10.80 13.00 11.75
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.280   3.220   4.690   5.058   6.200  13.000
## [1] "- hue:"
## [1] 1.71
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4800  0.7825  0.9650  0.9574  1.1200  1.7100
## [1] "- No outliers found for attribute dilution"
```

```
## [1] "- No outliers found for attribute proline"
```

La meitat dels atributs del conjunt de dades no presenten cap tipus d'outlier, mentre que la meitat que si que en presenta no són moltes les observacions en que aquest cas es dona. Addicionalment, si ens fixem en els valors dels outliers, tot i que estadísticament siguin valors considerats com a tal aquests es podrien donar perfectament durant l'anàlisi químic i per tant el que farem serà mantenir aquests valors dintre de les observacions i procedir com si res.

4 Anàlisi de les dades

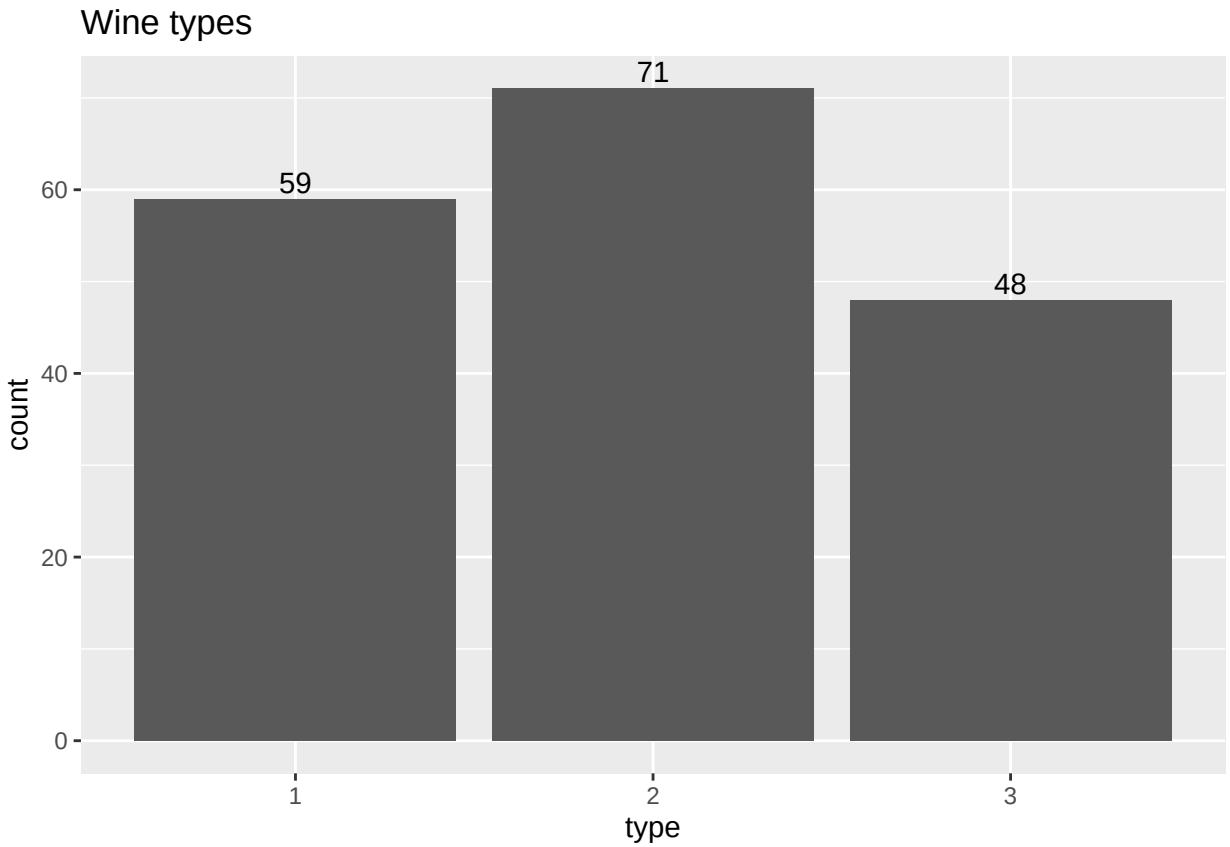
4.1 Selecció dels grups de dades que es volen analitzar

Bàsciament, el que volem realitzar en aquesta segona pràctica és la classificació dels **tres tipus de vins** gràcies als seus atributs. Evidentment en el dataset no contem amb més informació sobre el tipus de vi, i només sabem que pertanyen al grup 1, 2 o 3. Per tant, el que farem serà definir aquests únics tres nivells al conjunt de dades.

```
wine$type <- as.factor(wine$type)
```

Anem doncs a veure els tipus de vins.

```
library(ggplot2)
ggplot(wine, aes(x=type)) + geom_bar() + labs(title="Wine types") + geom_text(stat='count', aes(label=.
```



Com ja haviem dit tenim tres tipus de vins. A més, la quantitat de cadascun no és homogènia i trobem 59 vins del tipus 1, 71 vins del tipus 2 i 48 vins del tipus 3.

4.2 Comprovació de la normalitat i homogeneïtat de la variància

La comprovació de la normalitat i homogeneïtat de la variància de les dades és un punt molt important en la fase de l'anàlisi de les dades. Segons els resultats que obtinguem, per exemple, si podem corroborrar que les dades segueixen una distribució normal i homoscedasticitat podrem aplicar proves per contrast d'hipòtesi de tipus paramètric com *t Student* mentre que si no és el cas ens haurem de decantar per probes no paramètriques com *Wilcoxon*.

Per a la comprovació de la normalitat utilitzarem els tests de *Kolmogorov-Smirnov* i *Shapiro-Wilk*. Aquest segon és considerat un mètode més conservador que el primer ja que asumeix que les dades no segueixen una distribució normal.

Per a la comproació de la homoscedasticitat utilitzarem o bé el test de *Levene* si les dades sequeixen una distribució normal o el test de *Fligner-Killeen* en cas que no.

```
library(lawstat)
norm_homo_test <- data.frame(variable=header[-1], shapiro=NA, kolmogorov=NA, levene=NA, fligner=NA)
print("***** Outliers identification *****")
```

```
## [1] "***** Outliers identification *****"
```

```
index <- 1
for (col_name in norm_homo_test$variable){
  values <- wine[[as.symbol(col_name)]]

  shapiro_test <- shapiro.test(values)
  norm_homo_test[index, 2] <- ifelse (shapiro_test$p.value <= 0.05, "NOT NORM", "NORM")
  kolmogorov <- ks.test(values, pnorm, mean(values), sd(values))
  norm_homo_test[index, 3] <- ifelse (kolmogorov$p.value > 0.05, "NORM", "NOT NORM")

  if (norm_homo_test[index, "shapiro"] == "NORM" | norm_homo_test[index , "kolmogorov"] == "NORM"){
    levene_test <- levene.test(wine[["malic"]], wine$type)
    norm_homo_test[index, 4] <- ifelse (levene_test$p.value > 0.05, "HOMO", "NOT HOMO")
  }
  else{
   fligner_test <- fligner.test(wine[[column]], wine$type)
   norm_homo_test[index, 5] <- ifelse (fligner_test$p.value > 0.05, "HOMO", "NOT HOMO")
  }

  index <- index + 1
}
```

Anem ara a veure'n els resultats.

```
norm_homo_test
```

```
##           variable  shapiro kolmogorov   levene  fligner
## 1          alcohol NOT NORM       NORM NOT HOMO     <NA>
## 2            malic NOT NORM   NOT NORM     <NA> NOT HOMO
## 3              ash NOT NORM       NORM NOT HOMO     <NA>
## 4       alcalinity     NORM       NORM NOT HOMO     <NA>
## 5        magnesium NOT NORM       NORM NOT HOMO     <NA>
## 6          phenols NOT NORM       NORM NOT HOMO     <NA>
## 7       flavanoids NOT NORM       NORM NOT HOMO     <NA>
## 8     nonflavanoid NOT NORM   NOT NORM     <NA> NOT HOMO
## 9  proanthocyanins NOT NORM       NORM NOT HOMO     <NA>
## 10           color NOT NORM       NORM NOT HOMO     <NA>
## 11             hue NOT NORM       NORM NOT HOMO     <NA>
```

```
## 12        dilution NOT NORM   NOT NORM     <NA> NOT HOMO
## 13         proline NOT NORM   NOT NORM     <NA> NOT HOMO
```

Pel que fa a la normalitat de les dades, en 5 de les variables els dos tests estan completament d'acord, mentre que en la resta estan en desacord. L'única variable que de ben segur segueix una distribució normal és "alcalinity", mentre que "malic", "nonflavanoid", "dilution" i "proline" no segueixen una distribució normal.

En la resta de variables els tests tenen resultats diferents, encara que serem positius i agafarem els resultats del test de Kolmogorov i suposarem que la resta també segueixen una distribució normal.

Finalment, podem veure com totes la variables no tenen una igualtat entre la variància dels grups. Aquest punt és profitós per a nosaltres perquè podem asumir que els diferents tipus de vins tindran característiques diferents.

4.3 Aplicació de proves estadístiques

4.3.1 Estudi de la correlació de les variables

L'estudi de la correlació de les variables és un punt clau en l'anàlisi de les dades. Aquest ens permetrà veure quines dades estàn més o menys correlacionades entre sí. Veurem doncs quines són les variables que més correlacionades estan entre si, i visualitzarem com queden els grups de dades dintre d'aquestes. Cal dir des d'un inici, que tot i que dues variables tinguin una alta correlació no vol dir en cap moment que ens ajudi a poder-les classificar millor, sinó únicament que hi ha una dependència lineal entre elles.

En aquest apartat cal tenir en compte que com la majoria de les dades seguien una distribució normal, utilitzarem el mètode **pearson**. Addicionalment, per a cada parella d'atributs farem un test de correlació per determinar si aquesta és significativa o no i eliminar les que no ho siguin.

Cal dir que aquest tros de codi de calcula una matriu amb les parelles d'atributs ha sigut extret d' aquí.

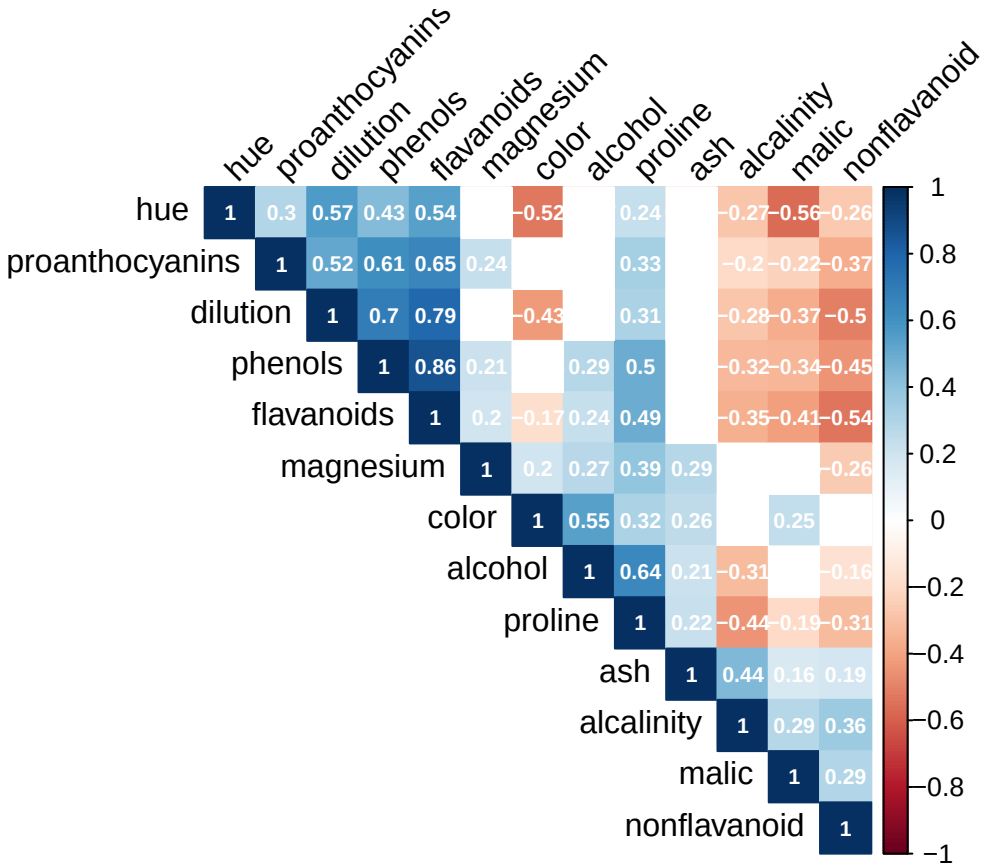
El primer pas és crear una matriu amb el test de correlació per parelles.

```
# El codi d'aquesta funció ha sigut extret del següent enllaç i adaptat per aquesta pràctica.
# http://www.sthda.com/english/wiki/visualize-correlation-matrix-using-correlogram
calculate_cor_matrix <- function(mat, ...) {
    mat <- as.matrix(mat)
    n <- ncol(mat)
    p.mat<- matrix(NA, n, n)
    diag(p.mat) <- 0
    for (i in 1:(n - 1)) {
        for (j in (i + 1):n) {
            tmp <- cor.test(mat[, i], mat[, j], method="pearson", ...)
            p.mat[i, j] <- p.mat[j, i] <- tmp$p.value
        }
    }
  colnames(p.mat) <- rownames(p.mat) <- colnames(mat)
  p.mat
}
```

Seguidament seleccionem les dades indicades eliminant la primera columna de classificació per tipus, calculem la matriu de correlació i mostrem en una gràfica els valors de les correlacions.

```
library(corrplot)
library(dplyr)

cor_dataset <- wine[, -1]
# Fem el test de correlació per cada parella d'atributs
cor_matrix <- calculate_cor_matrix(cor_dataset)
# Creem la matriu de correlació per parelles d'atributs
```

```
correlation_matrix <- cor(cor_dataset,
                          method = "pearson", # Mètode Spearam
                          use="complete.obs"   # Ignorem els NAs
                          )

# Plasmem en un gràfic els resultats
corrplot(correlation_matrix, type="upper", method="color",
         addCoef.col = "white", sig.level = 0.05, p.mat=cor_matrix,
         order="hclust",
         tl.col="black", tl.srt = 45,
         number.cex = 0.7,
         insig = "blank"
         )
```



En la gràfica podem veure la correlació entre les variables representades entre -1 i 1. Els valors que s'apropen a -1 inidquen una correlació negativa i els que s'apropen a 1 una correlació positiva. Cal mencionar que una correlació de -0.8 i 0.8 són igual de potents entre elles i l'únic que canvia és si la correlaicó és positiva o negativa.

Primer de tot podem veure com la majoria de les variables estan correlacionades entre si (amb més o menys intensitat) de manera positiva. Podem destacar per exemple la correlació positiva entre "flavanoids" i "phenols" o "dilution" i "flavanoids", o la correlació negrativa entre "hue" i "malic".

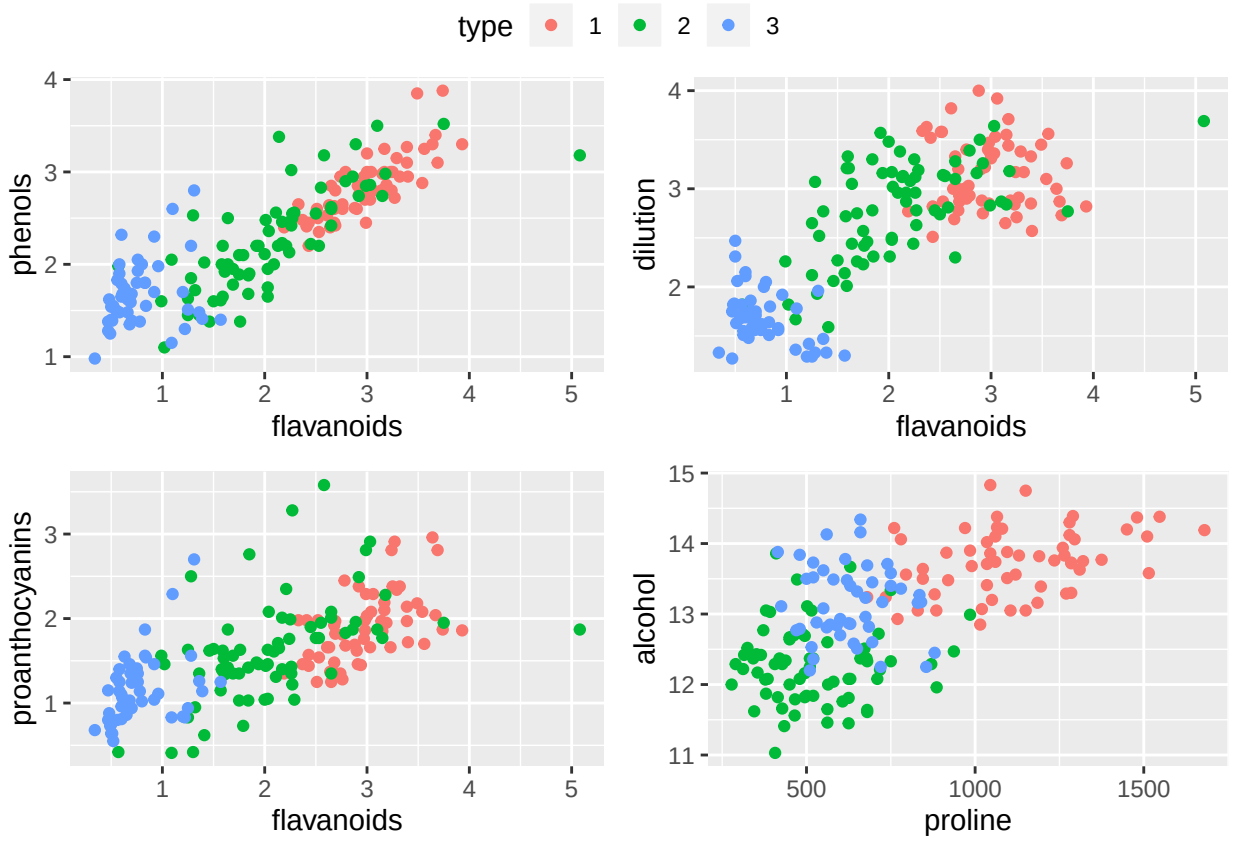
Podem veure també com la variable que té una correlació més forta amb les altres és "flavanoids" que té una correlació superior a 0.5 amb 6 de les 13 variables (gairebé 7 perquè en tenim una amb 0.49). La variable "ash" en canvi en 5 de les 13 variables la correlació entre aquestes és insignificant.

Anem ara a veure com estan repartits els grups en aquestes variables.

```
library(ggpubr)
```

```
## Loading required package: magrittr
```

```
plot_relationship <- function(x, y){
  ggplot(wine, aes(x = (!!as.symbol(x)) , y = (!!as.symbol(y)), color=type)) + geom_point()
}

ggarrange(
  plot_relationship("flavanoids", "phenols"),
  plot_relationship("flavanoids", "dilution"),
  plot_relationship("flavanoids", "proanthocyanins"),
  plot_relationship("proline", "alcohol"),
  common.legend = TRUE
)
```

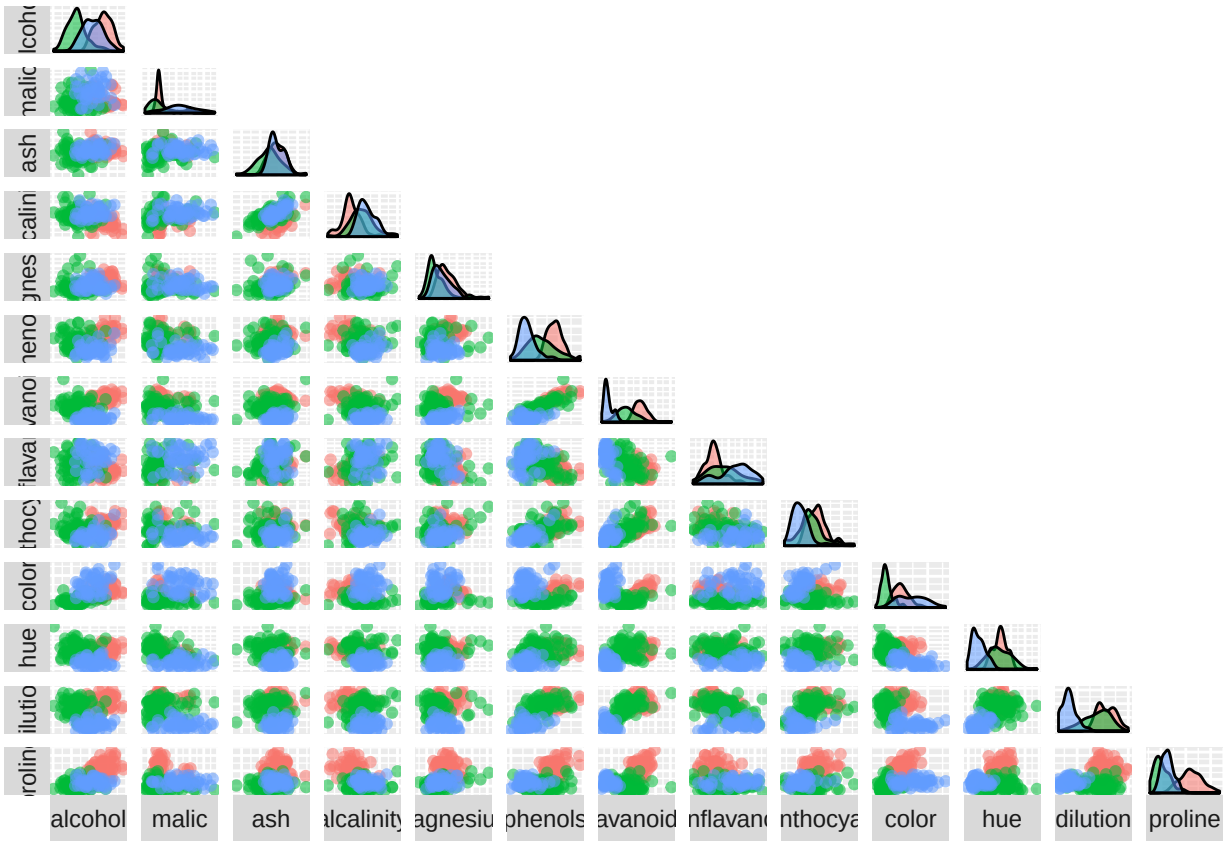


En els gràfics podem veure com tot i que els grups que es visualitzen no estan perfectament diferenciats, aquests presenten diferències entre si i cada grup es posiciona en una part concreta del gràfic.

Deixant de banda la correlació entre les variables, alternativament podem fer aquestes representacions que hem fet més amunt per a cada parella d'atributs.

```
library(GGally)

ggpairs(wine,
        columns=2:14, aes(colour=type, alpha=0.5),
        lower=list(continuous="points"),
        upper=list(continuous="blank"),
        axisLabels="none", switch="both"
```

```
)
```



Tot i que al haver tantes variables les gràfiques queden molt petites i costa diferenciar podem veure com clarament la variable "plorine" combinada amb alguna de les altres com pot ser "hue", "color" o "dilution" ens permeten diferenciar d'una manera clara els diversos grups de vins.

4.3.2 Mètodes de classificació no supervisats

Els mètodes de classificació no supervisats són aquells que sense indicar-li a l'algrosime a quin grup pertany cada observació aquests busca agrupar-les segons les similituds entre observacions.

És evident que en aquest conjunt de dades disposem d'aquesta informació i que podriem utilitzar directament un model supervisat. Utilitzarem igualment el model no supervisat per veure si realment hi ha diferències significatives entre cada cluster mitjançant dos mètodes diferents: k-means i hierarchical agglomerative clustering.

4.3.2.1 K-means

K-means és el primer algorisme que utilitzarem no supervisat per a la classificació dels vins. El primer pas serà doncs "eliminar" del conjunt de dades la variable que ens diu originalment a quin grup pertanyen.

```
# Separem les columnes
wine_types <- wine[, 1]
wine <- wine[,2:14]
```

Seguidament haurem de normalitzar les dades per poder aplicar de manera més eficient l'algorsime. En cas de no normalitzar les dades degut a que les escales de les variables són molt diferents, per aquelles que tinguin un rang més gran l'algorisme li acabaria donant més importància (per exemple la variable "proline").

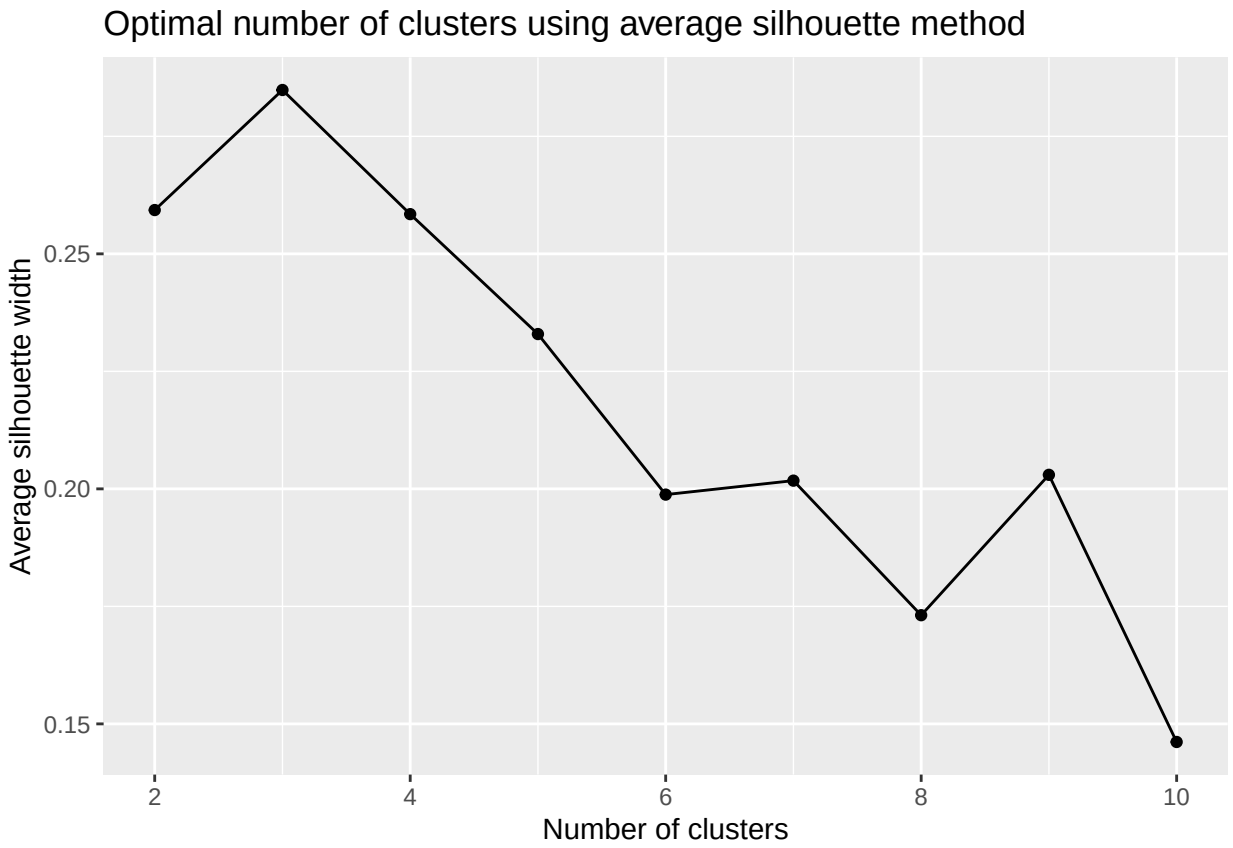
Utilitzarem una normalització perquè les dades tinguin una mitjana 0 i una desviació estàndard 1.

```r
# Method that normalizes a given data
normalize_col <- function(data){
  (data - mean(data)) / sd(data)
}

# Method that normalizes the specified columns of a dataset
normalize <- function(dataset, columns){
  as.data.frame(lapply(dataset[,columns], normalize_col))
}

zWine <- normalize(wine, names(wine))
```

Tot i que ja sabem el nombre de clústers que originalment tenen les dades (3), al aplicar un algorsime no supervisat no es solen tenir aquest valor. És per això que existeixen diversos mètodes que ens diuen quin és el valor de k més indicat per al conjunt de dades.

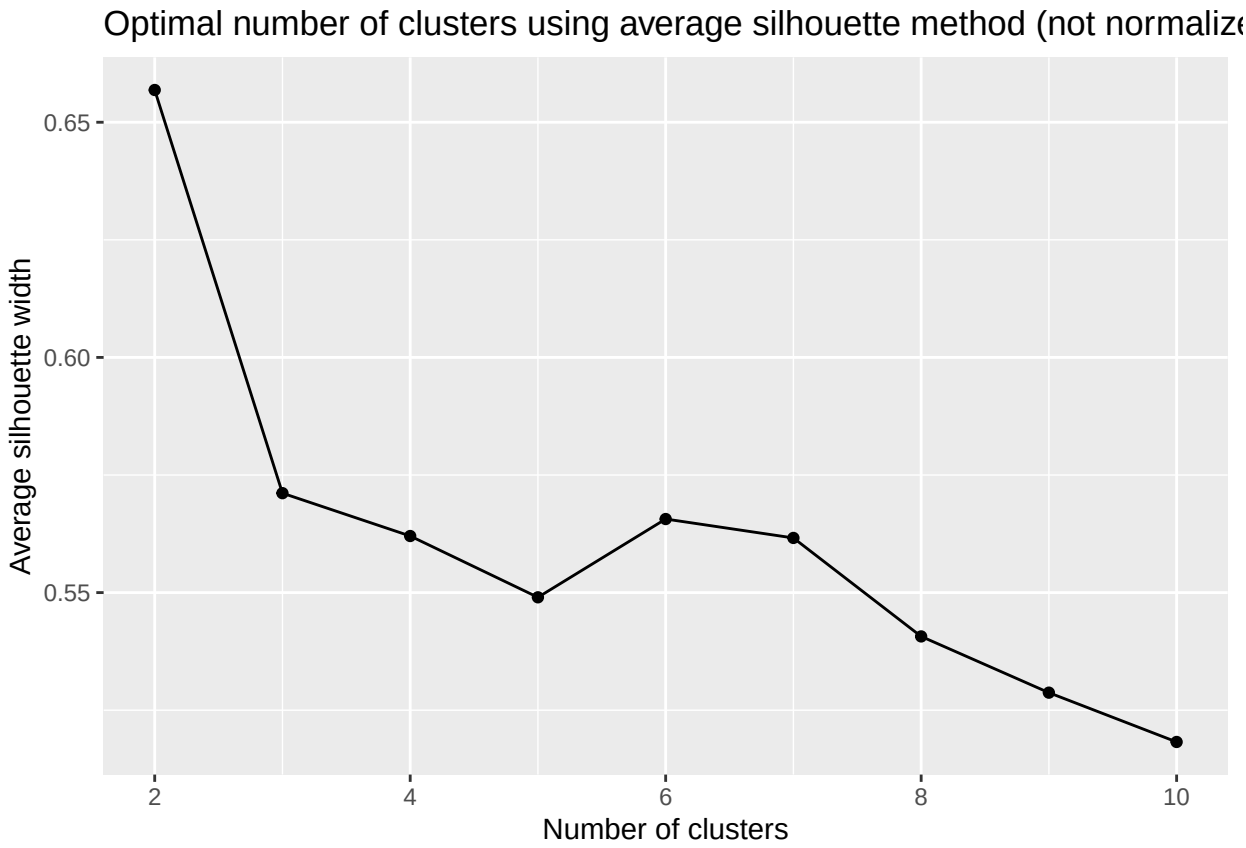
El primer de tots és el **average silhouette** que mesura la qualitat del clustering determinant per cada observació com d'aprop es troba del seu cluster i com de lluny dels altres. Com més elevat sigui el valor de la silueta, millor serà el nombre de clusters escollits.

```r
library(cluster)
# Funció que ens permet calcular la silueta donat un data_frame i un nombre de clusters.
single_silhouette <- function(data_frame, k_cluster){
  # Apliquem l'algorisme k-means al dataframe amb n_clusters
  # nstart indiquem que com a mínim 50 valors inicials aleatoris són agafats
  # iter.max per assegurar-nos que convergeix
  km <- kmeans(data_frame, k_cluster, nstart = 50,iter.max = 15)
  # Càlcul de la silueta
  sk <- silhouette(km$cluster, dist(data_frame))
  # Retornem la mitjana
  return(mean(sk[, 3]))
}

# Funció que retorna un gràfic amb les diverses siluetes pel rang de valors de cluster especificat.
calculate_silhouette <- function(data_frame, n_clusters, title){
  # Creem un vector de 10 posicions (encara que utilizarem de la 2 a la 10).
  silhouette_cluster <- vector(length = 10)
  # Calculem la silueta pels clusters de 2 a 10
  for (k_cluster in 2:10){
    silhouette_cluster[k_cluster] <- single_silhouette(data_frame, k_cluster)
  }

  # Creem el gràfic inserint només els valors del vector de 2 a 10 i el retornem
  plot <-  ggplot(data = data.frame(x = 2:10, y = silhouette_cluster[2:10]), aes(x = x, y = y)) + geom_
    labs(title=title, x="Number of clusters", y="Average silhouette width")

  return(plot)
}

calculate_silhouette(zWine, 10, "Optimal number of clusters using average silhouette method")
```



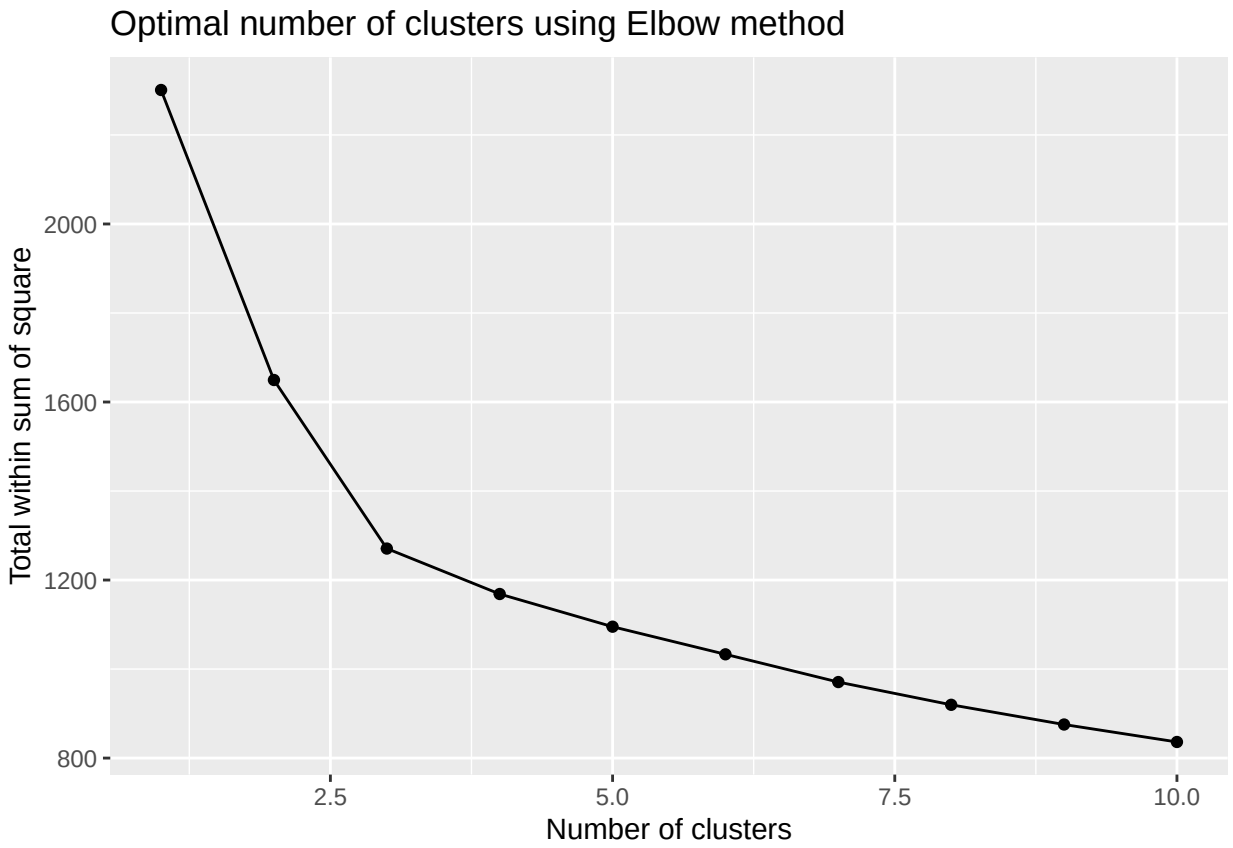
El nombre de cluster que presenta una millor silueta és en aquest cas **3**. Cal esmentar que la normalització ha servit ja que en el cas de no haver-la utilitzat el valor òptim seria de 2 clusters com es veu en la següent imatge.

```
# Creem el gràfic inserint només els valors del vector de 2 a 10
calculate_silhouette(wine, 10, "Optimal number of clusters using average silhouette method (not normali
```



El segon mètode és **l'Elbow** que busca la menor suma del quadrat de les distàncies dels punts de cada grup respecte al seu centre. Per obtenir aquest valor no hem de fer molta cosa ja que és calculat automàticament pel *kmeans*.

```r
# Mètode que retorna la menor suma del quadrat de les distàncies dels punts de cada grup
# respecte al seu centre donat un data_frame i un nombre de clusters.
single_elbow <- function(data_frame, n_clusters){
  return(kmeans(data_frame, n_clusters, nstart = 50,iter.max = 15)$tot.withinss)
}

# Funció que retorna un gràfic representant l'elbow
calculate_elbow <- function(data_frame, n_clusters, title){
  # Creem un vector de 10 posicions (encara que utilizarem de la 2 a la 10).
  elbow_cluster <- vector(length = 10)
  # Calculem l'elbow pels clusters de 2 a 10
  for (n_clusters in 1:10){
    elbow_cluster[n_clusters] <- single_elbow(data_frame, n_clusters)
  }

  # Creem el gràfic inserint només els valors del vector de 2 a 10
  plot <- ggplot(data = data.frame(x = 1:10, y = elbow_cluster[1:10]), aes(x = x, y = y)) + geom_line()
    labs(title=title, x="Number of clusters", y="Total within sum of square")

  return(plot)
}

calculate_elbow(zWine, 10, "Optimal number of clusters using Elbow method")
```

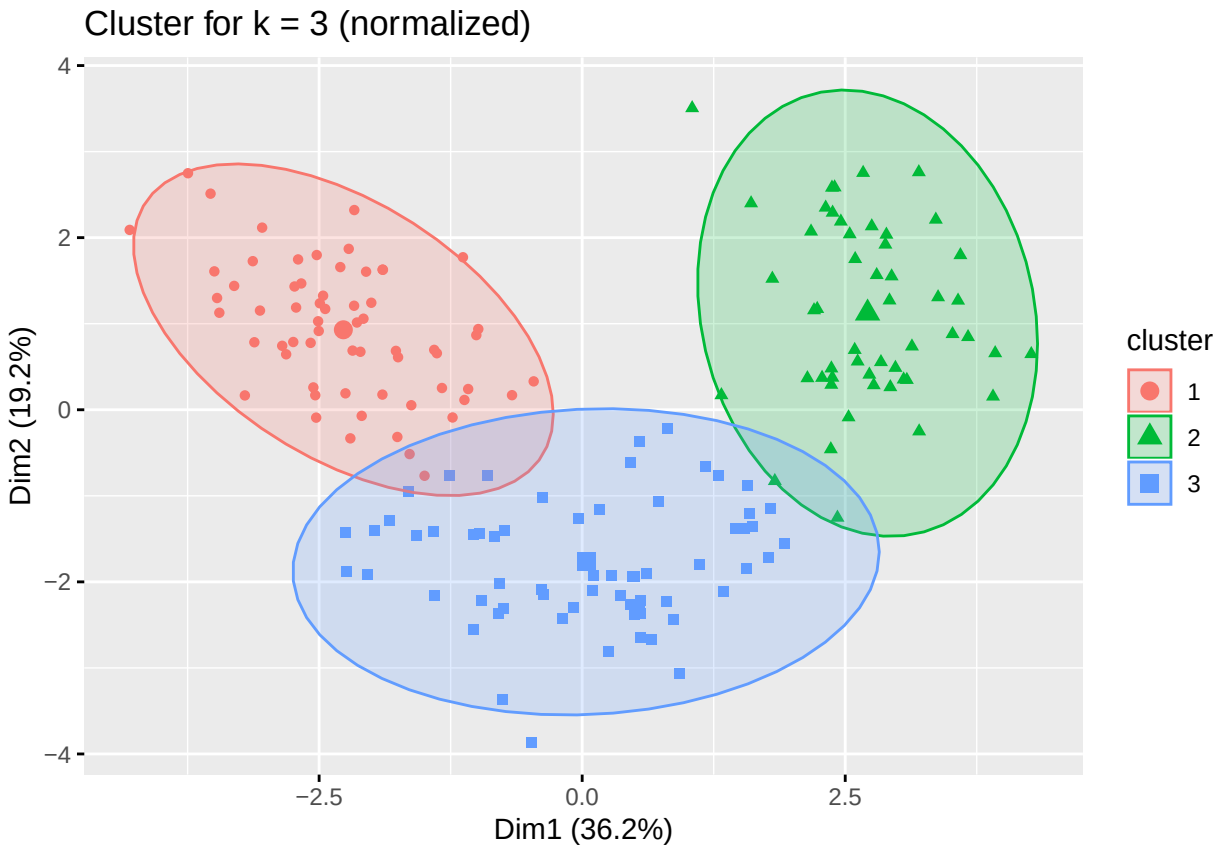


Per trobar el valor desitjat del cluster, hem de trobar dintre del gràfic el colze (elbow). Com podem observar del cluster 3 al 10 es forma una línia gairebé constant, i del 3 al 2 presenta una diferència. Per tant és fàcil veure com el colze es troba una altre vegada en el nombre **3**.

Per tant podem veure com la millor manera d'agrupar les dades és a partir de **3** clusters diferents, fet ideal per nosaltres que sabem que originalment les dades tenen 3 clusters.

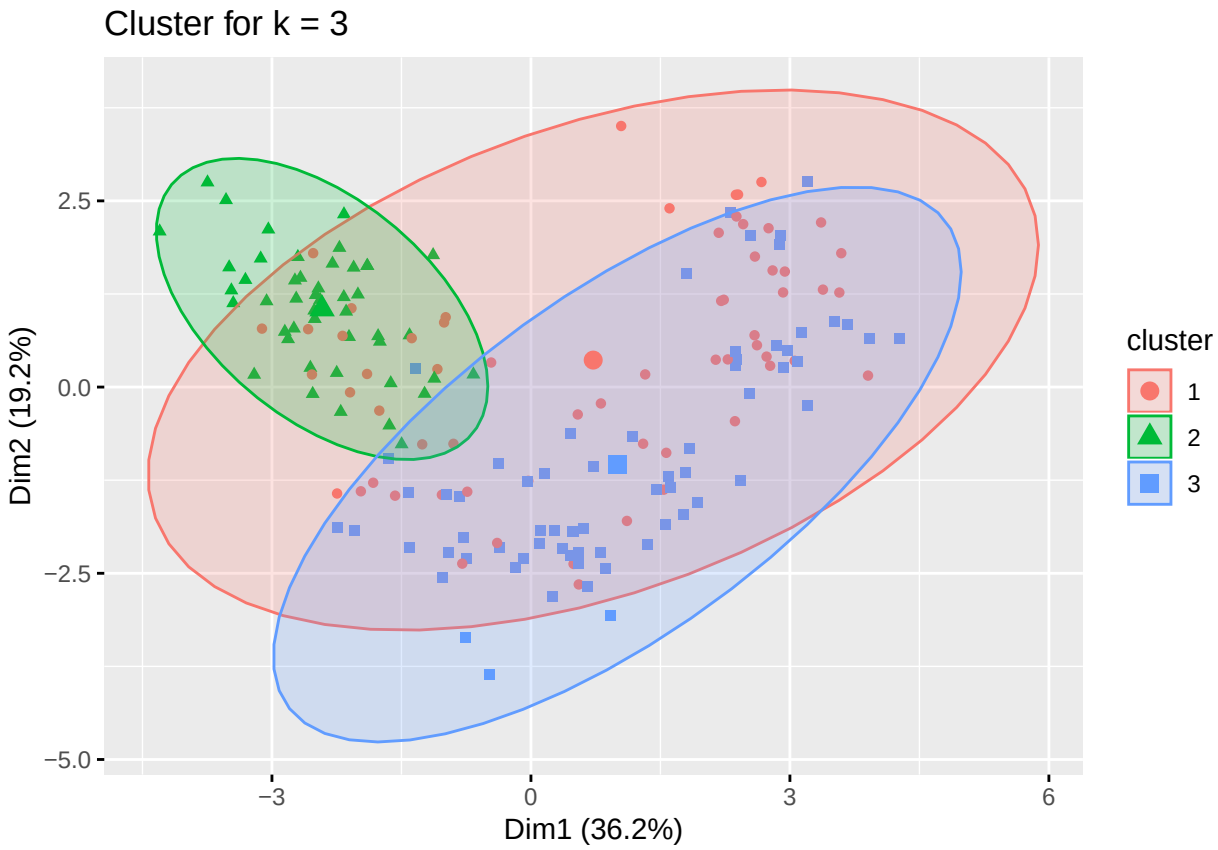
Seguidament anem a utilitzar l'algorimse k-means.

```
library(factoextra)

get_kmeans_plot <- function(dataset, k, title){
  # Posem una llavor per fer que les representacions sempre quedin iguals i les
  # explicacions donades siguin coherents amb els colors assignats al gràfic.
  set.seed(1)
  # Apliquem l'algorisme k-means al dataframe amb k clusters,
  # nstart indiquem que com a mínim 50 valors inicials aleatoris són agafats
  # iter.max per assegurar-nos que convergeix
  result <- kmeans(dataset, centers = k, nstart = 50,iter.max = 15)

  # Graphical representation of the clusters
  fviz_cluster(result, data = dataset, show.clust.cent = TRUE, geom = "point", ellipse.type = "norm", ma
}

get_kmeans_plot(zWine, 3, "Cluster for k = 3 (normalized)")
```



Els diferents vins queden clarament diferenciats en 3 tipus de vins diferents. És fàcil veure com la normalització de les dades ha jugat un paper clau en la definició d'aquests clusters.

```
get_kmeans_plot(wine, 3, "Cluster for k = 3")
```



Com es pot veure els grups creats en cas de no normalitzar les dades són molt caòtics.

Finalment, tot i que a simple vista podem veure com en el cas de normalitzar les dades els grups estan ben definits la millor manera de comprovar-ho es contrasant-ho amb la classificació original de les dades.

```
# Function that classifies all of the objects of a given dataset by adding
# the column "cluster"
get_classification <- function(seed, dataset, k){
  set.seed(seed)
  result <- kmeans(dataset, centers = k, nstart = 50,iter.max = 15)

  dataset$cluster <- as.factor(result$cluster)

  return(dataset)
}

# Funció que donat un dataset classificat i una classificació retorna la probabilitat d'acert
calculate_probability <- function(clusters, types){
  return((sum(clusters == types)/length(clusters))*100)
}


kmeans3_wine <- get_classification(seed = 6, dataset = zWine, k=3)

# Calculem la probabilitat
calculate_probability(kmeans3_wine$cluster, wine_types)
```

```
## [1] 96.62921
```

```r
# Mostrem com s'ha classificat
table(clusters= kmeans3_wine$cluster, original=wine_types)
```

```
##         original
## clusters  1  2  3
##        1 59  3  0
##        2  0 65  0
##        3  0  3 48
```

En el 96.63% dels casos, l'algorisme kmeans ha aconseguit classificar correctament els tipus de vins. En aquest cas només 6 valors s'han classificat incorrectament: 3 del grup 1 que eren del grup 3 i 3 del grup 2 que eren del grup 3.

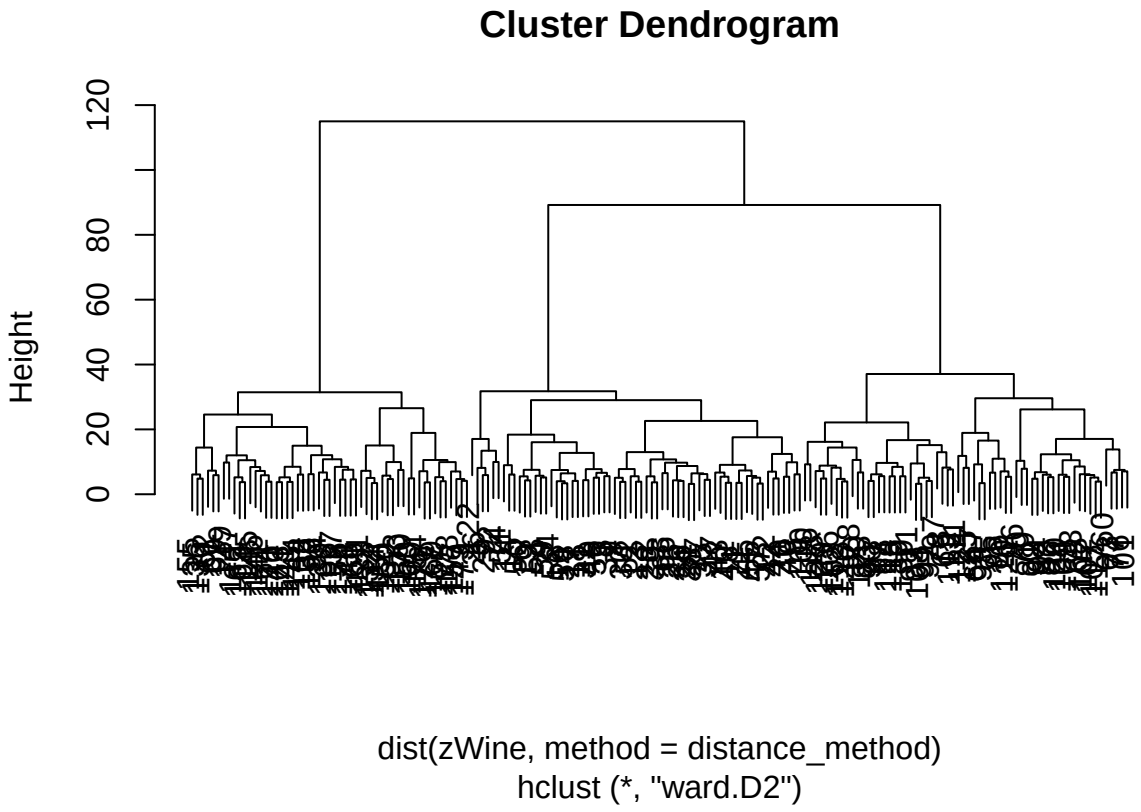
4.3.2.2 Hierarchical agglomerative clustering

Aquest mètode no supervisat busca crear una jerarquia entre els clusters. Hi ha diverses estratègies per enfocar-ho, i en aquest cas ens centrarem en utilitzar la agglomerativa que té un enfocament *bottom_up* on cada observació pertany inicialment al seu propi cluster, i per parelles es van ajuntant en el mateix cluster i formant una jerarquia cap amunt. La gràcia està en el mètode és la selecció de la mètrica per calcular la distància entre observacions i el criteri escollit per juntar aquestes parelles de clusters (*linkage criteria*).

Aquest mètode no supervisat, a diferència de k-means no necessita doncs en cap moment el nombre de clusters perquè comença amb tants clusters com observacions i acaben totes en un sol cluster.

Anem doncs a aplicar l'algorisme.

```r
hierarchical_agglomerative_clustering <- function(data, distance_method, linkage_criteria){
  hclust(dist(zWine, method=distance_method), method=linkage_criteria)
}

hac <- hierarchical_agglomerative_clustering(zWine, "manhattan", "ward.D2")
plot(hac)
```



Com es pot veure en el dendograma no tenim cap dubte que les dades s'agrupen en 3 clusters ben diferenciats entre si, perquè a l'altura 30 ja tenim els 3 clusters diferenciats i no és fins a l'altura 90 que dos d'aquests s'ajunten.

Igual que en el cas anterior anem a veure com de "bé" ho ha fet l'algorisme.

```
calculate_probability(cutree(hac, 3), wine_types)
```

```
## [1] 94.94382
```

En aquest cas l'algorisme aconsegueix una classificació correcta del 94.94% de les observacions.

4.3.3 Mètodes de classificació supervisats

En els mètodes de classificació supervisats és molt important dividir els conjunts de dades en dos grups: **train i test**. Aquest fet ens permetrà posar a prova el nostre model creat i podrem jugar per evitar per exemple fer *overfitting* en aquest.

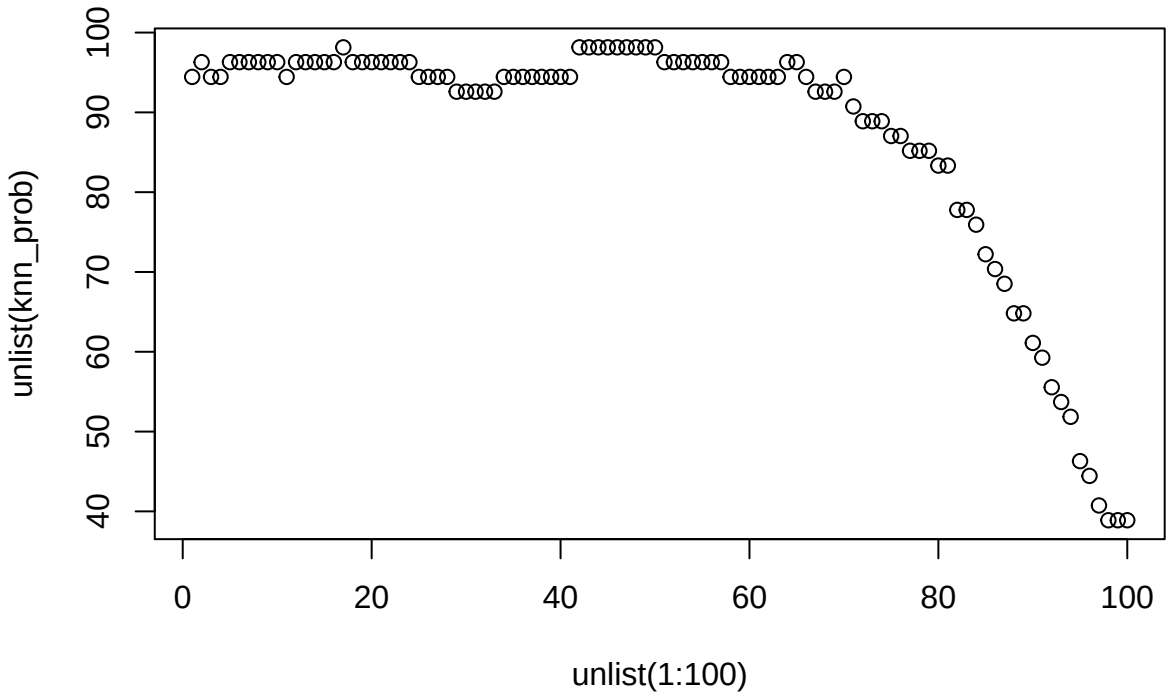
Separarem el dataset 70/30 entre train i test i aplicarem l'algorisme. Primer de tot definim la funció que ens facili la feina.

```
library(class)

# Creem una classe que ens emmagatzemi la informació
setClass("Sample",
  slots = c(
    train_sample = "data.frame",
    train_type = "factor",
    test_sample = "data.frame",
    test_type = "factor"
  )
```

```
)

setClass("Knn",
   slots = c(
     sample = "Sample",
     knn = "factor",
     accuracy = "numeric"
   )
)

# Funció que ens divideix el dataset
create_sample <- function(dataset, types, train_prop, seed){
  set.seed(seed)
  # We create a sample with the specified proportions
  my_sample <- sample(1:nrow(dataset), train_prop * nrow(dataset))


  # We obtain the train and test samples
  train_sample <- dataset[my_sample,]
  train_type <- types[my_sample]
  test_sample <- dataset[-my_sample,]
  test_type <- types[-my_sample]

  return(new("Sample", train_sample=train_sample, train_type=train_type, test_sample=test_sample, test_
}
```

Seguidament aplicarem l'algorisme, però hem de decidir el valor de k. Per a decidir-lo, probarem quin s'adapta millor.

```
get_knn <- function(dataset, types, train_prop, k){
  # Cridem a l'algorisme
  set.seed(6)
  # Dividim el dataset entre train i test
  samples <- create_sample(dataset, types, train_prop, 6)

  # Apliquem l'algorsime
  knn_wine <- knn(samples@train_sample, samples@test_sample, cl=samples@train_type, k=k)

  accuracy <- calculate_probability(knn_wine, samples@test_type)

  return(new("Knn", sample=samples, knn=knn_wine, accuracy=accuracy))
}

knn_prob <- list()
for(i in 1:100){
  knn_prob[i] <- get_knn(zWine, wine_types, 0.7, i)@accuracy
}

# Mostrem un gràfic per la probabilitat
plot(unlist(1:100), unlist(knn_prob))
```



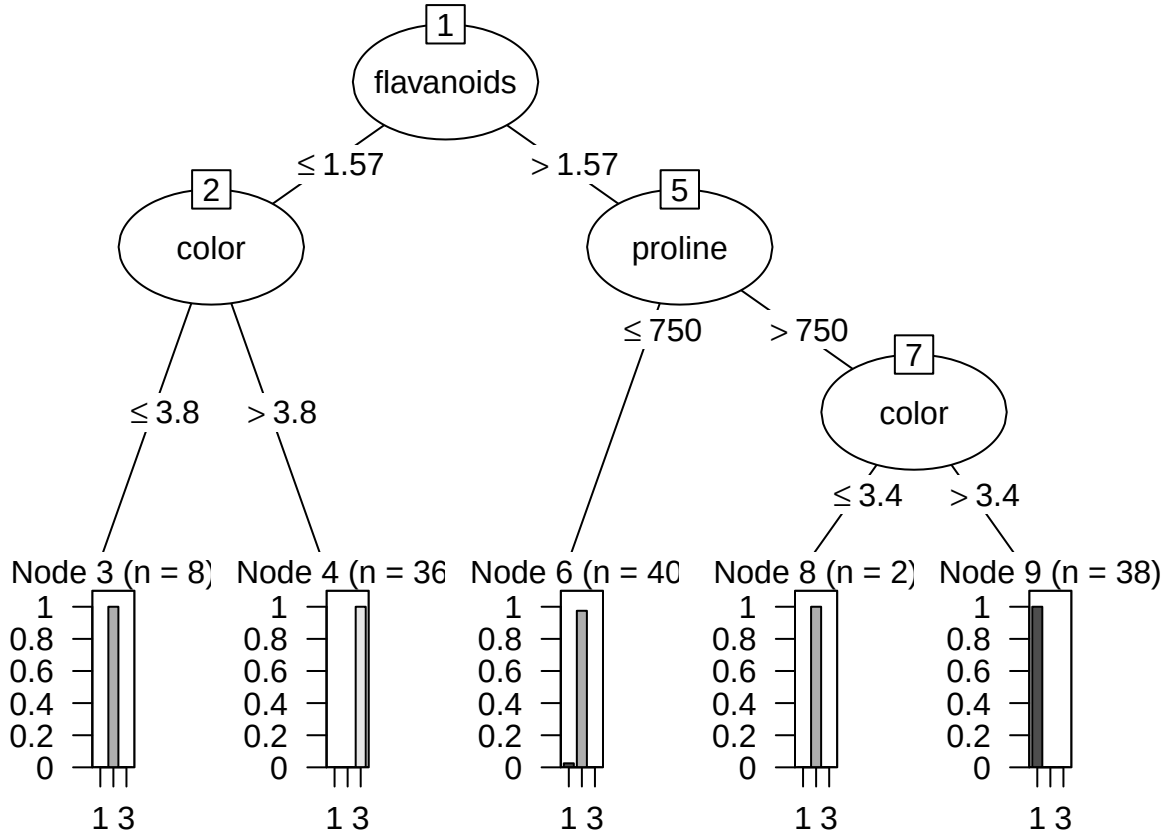

```
# Juntament amb el valor màxim assolit
max(unlist(knn_prob))
```

```
## [1] 98.14815
```

Com es pot observar en el gràfic a partir de k>70 l'algorisme comença a mostrar un comportament molt dolent, mentre que per a k<70 els porcentatges d'acerts solen estar entre el 90% i el 98.15%. Per tant el valor de k que s'ajusta més és 17. Podriem dir que és un valor de k ni molt petit (que podria produir overfitting) ni molt gran (que podria produir underfitting), i que crea un model de gran qualitat.

4.3.4 Arbres de classificació

```
library(C50)
wine.sample <- create_sample(wine, wine_types, 0.7, 1)
wine.modelC50 <- C5.0(wine.sample@train_sample, wine.sample@train_type)
plot(wine.modelC50)
```




```
predicted_model <- predict(wine.modelC50, wine.sample@test_sample, type="class")
print(sprintf("Accuracy: %.2f %%",100*sum(predicted_model == wine.sample@test_type) / length(predicted_m
```

```
## [1] "Accuracy: 96.30 %"
```

5 Conclusions