

Pràctica 2: Neteja i anàlisi de les dades

Roger Bosch Mateo

12/19/2019

Índex

1	Llibreries utilitzades	2
2	Descripció del dataset	2
2.1	Atributs	2
2.2	Importància i objectius de l'anàlisi	2
3	Integració i selecció de les dades d'interès a analitzar.	3
4	Neteja de les dades	3
4.1	Tractament de zeros o elements buits	3
4.2	Identificació i tractament dels valors extrems	4
5	Anàlisi de les dades	5
5.1	Selecció dels grups de dades que es volen analitzar	5
5.2	Comprovació de la normalitat i homogeneïtat de la variància	6
5.3	Aplicació de proves estadístiques	7
6	Exportació de les dades	25
7	Conclusions	25
	Referències	27

1 Llibreries utilitzades

Per a la resolució d'aquesta pràctica s'han utilitzat diverses llibreries per facilitar la feina d'aplicació d'algorismes o visualització d'aquests. Les llibreries utilitzades són:

- ggplot2 (Wickham 2016) per a la visualització de gràfiques, juntament amb ggpubr (Kassambara 2019) per juntar-les en una sola imatge i GGally (Schloerke et al. 2018) per estudiar les variables per parelles.
- factoextra (Kassambara and Mundt 2017), cluster (Maechler et al. 2019) per a tasques d'elecció del nombre de clusters i la seva visualització.
- corrrplot (Wei and Simko 2017) per a la visualització de la matriu de correlació.
- dplyr (Wickham et al. 2019) per al filtratge i selecció de dades.
- lawstat (Gastwirth et al. 2019) per aplicar el test de Levene.
- C50 (Kuhn and Quinlan 2018) per als arbres de classificació.

2 Descripció del dataset

Per a la resolució d'aquesta segona i última pràctica de l'assignatura de *Tipologia 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 una anàlisi química 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 química.

2.1 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'acidesa 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 el 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 són flavanoids. Afecten doncs també al sabor, 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.

2.2 Importància i objectius de l'anàlisi

A partir d'aquest conjunt de dades es planteja la problemàtica de determinar quines són les variables més importants 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'interès per qualsevol empresa que es dediqui al cultiu i fabricació 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 màrqueting en la venda d'aquests segons els atributs diferenciadors. A més, si aquests vins els identifiquéssim amb el seu preu podríem inferir en el preu que tindria el nostre.

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

El primer pas 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 a la selecció de les dades d'interès 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 irrelevantes i obviar-les perquè 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 classificades correctament.

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

4 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 les variables que s'han carregat són de tipus numèric, tal i com esperàvem per la descripció del dataset.

4.1 Tractament de zeros o elements buits

Usualment, en els conjunts de dades s'utilitzen diferents valors per marcar l'absència d'aquests com poden ser els "0", els "NA" o fins i tot alguns caràcters com podria ser "?". Seguidament, 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 molt usual trobar casos en els quals 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 zero pot ser perfectament un valor vàlid).

4.2 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 detectar-los és amb la funció **boxplot.stats** que ens crea un llistat amb aquests valors.

Crearem doncs un bucle senzill en el qual 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 mena d'outlier, mentre que la meitat que sí que en presenta no són moltes les observacions en què aquest cas es dona. Addicionalment, si ens fixem en els valors dels outliers, encara 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.

5 Anàlisi de les dades

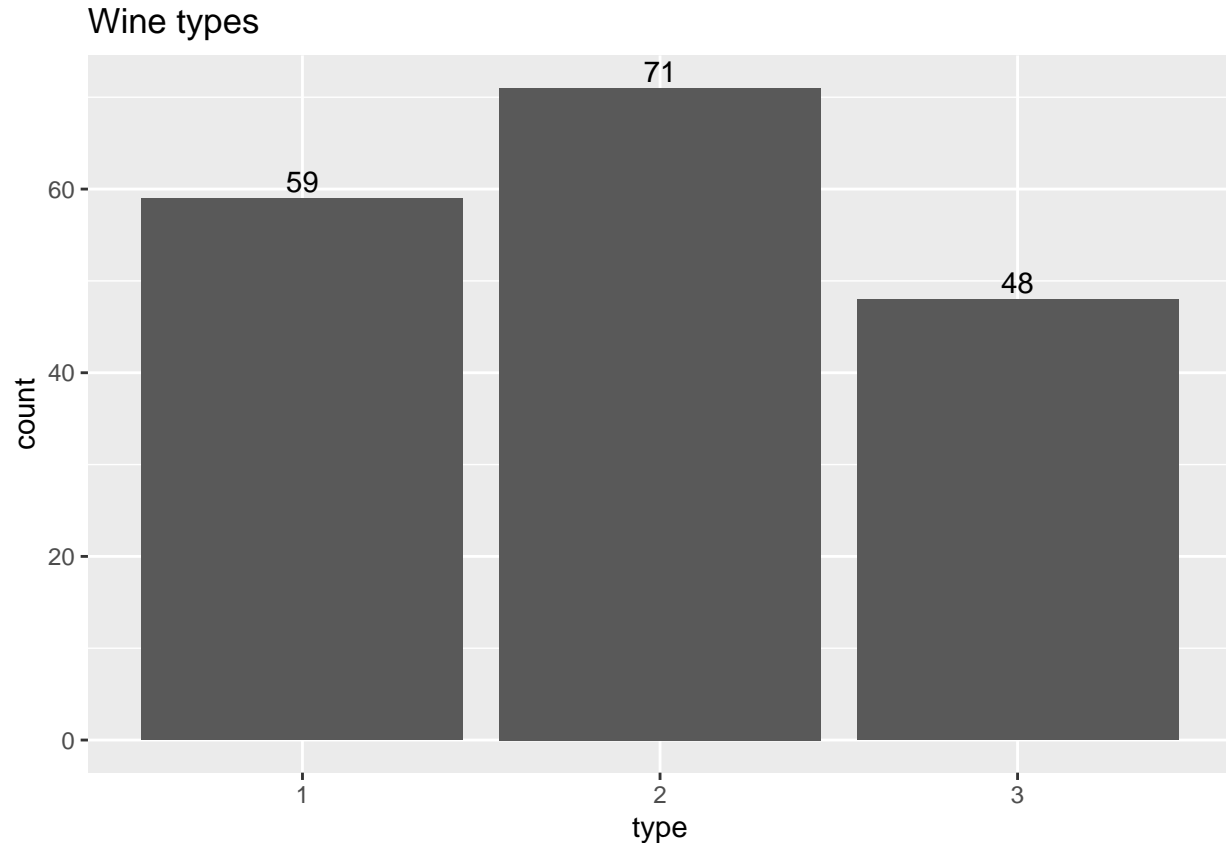
5.1 Selecció dels grups de dades que es volen analitzar

Bàsicament, 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 comptem 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=..count..), size=4, vjust = -0.3)
```



Com ja havíem 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.

5.2 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 corroborar que les dades segueixen una distribució normal i homoscedasticitat podem aplicar proves per contrast d'hipòtesi de tipus paramètric com *t Student* mentre que si no és el cas ens haurem de decantar per proves 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 assumeix que les dades no segueixen una distribució normal.

Per a la comprovació de l'homoscedasticitat utilitzarem o bé el test de *Levene* si les dades segueixen 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)
```

```

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[[as.symbol(col_name)]], 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    HOMO    <NA>
## 2       malic NOT NORM NOT NORM    <NA> NOT HOMO
## 3        ash NOT NORM      NORM NOT HOMO    <NA>
## 4    alkalinity  NORM      NORM    HOMO    <NA>
## 5    magnesium NOT NORM      NORM    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    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 "alkalinity", 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 4 variables (alcohol, alkalinity, magnesium i proanthocyanins) mantenen una homogeneïtat entre la variància del tipus de vi, mentre que les altres 9 variables no.

5.3 Aplicació de proves estadístiques

5.3.1 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 estan 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 que 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.

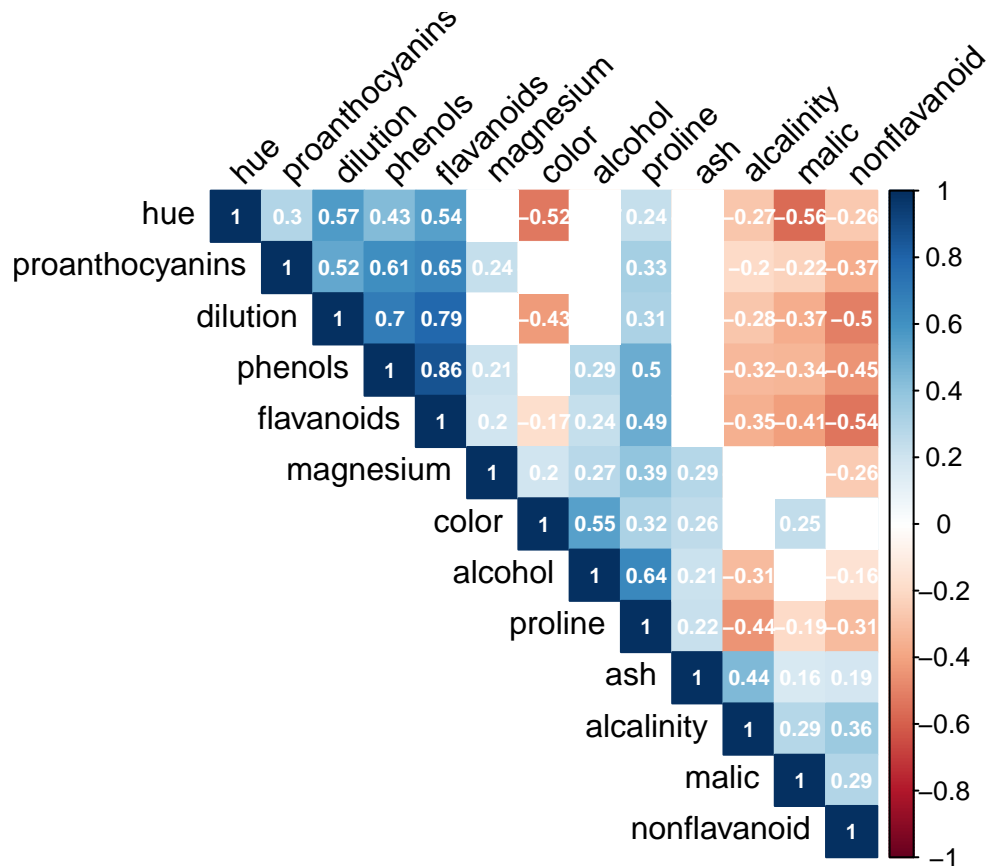
```
# El codi d'aquesta funció ha sigut extret del següent enllaç
# i adaptat per aquesta pràctica.
# http://www.sthda.com/english/wiki/visualize-correlation-matrix-using-correlogram
calculate_cor_matrix <- function(mat, ...) {
  mat <- as.matrix(mat)
  n <- ncol(mat)
  p.mat<- matrix(NA, n, n)
  diag(p.mat) <- 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"
                          )

# 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 indiquen 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 correlació 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 negativa entre “hue” i “malic”.

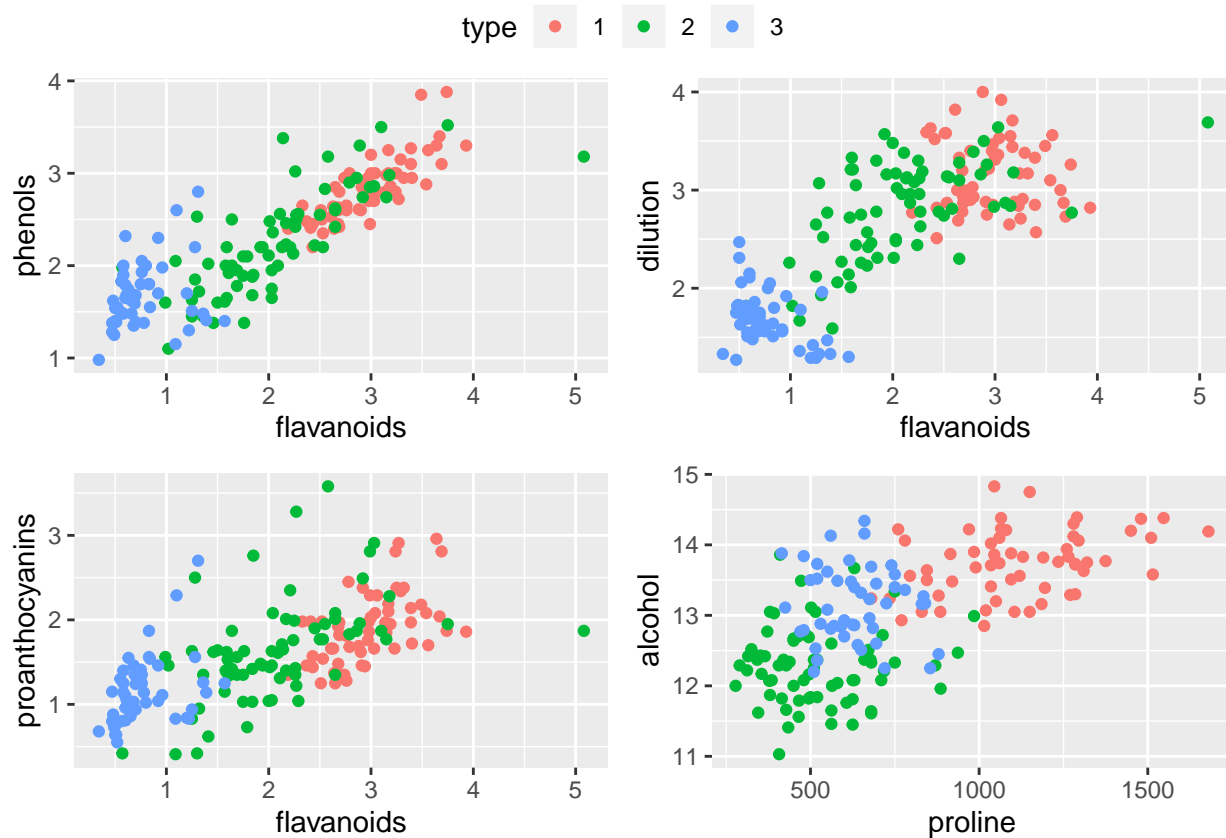
Podem veure també com la variable que té una correlació més forta amb les altres és “flavanoids” que té una correlació superior a 0.5 amb 6 de les 13 variables (gairebé 7 perquè en tenim una amb 0.49). La variable “ash” en canvi en 5 de les 13 variables la correlació entre aquestes és insignificant.

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

```
library(ggpubr)

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
)
```

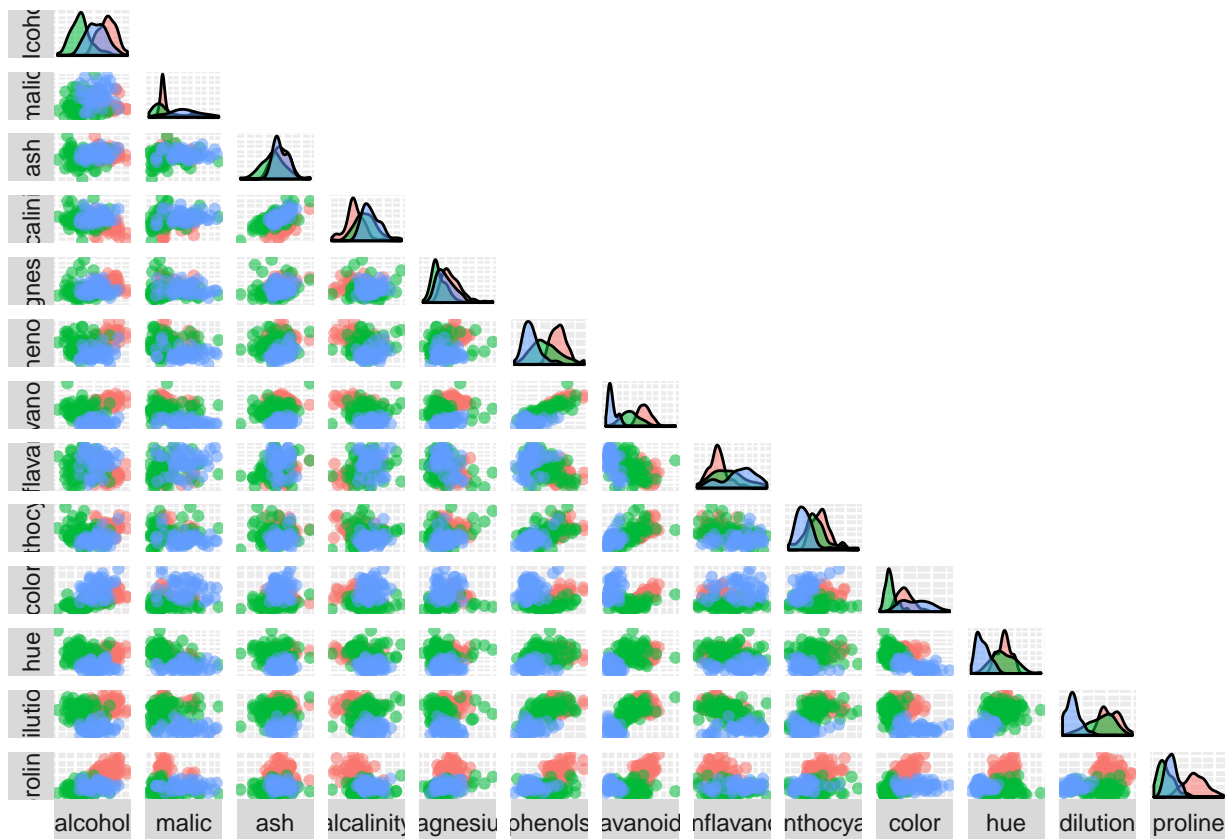


En els gràfics podem veure com tot i que els grups que es visualitzen no estan perfectament diferenciats, aquests presenten diferències entre si i cada grup es posiciona en una part concreta del gràfic.

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 a l'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.

5.3.2 Kruskal-Wallis

La següent prova que realitzarem és la de Kruskal-Wallis que ens permetrà identificar quines variables mostren diferències significatives per al tipus de vi.

A l’aplicar el test, aquest ens dirà si hi ha diferències significatives entre dos o més tipus de vins, però no ens dirà en quins. Per tant, primer de tot anem a veure quines variables presenten diferències significatives.

```
for(col in header[-1]){
  if (kruskal.test(x=wine[[col]], g=wine$type)$p.value < 0.5){
    print(paste0(col, " : has significant differences between the different types of wines"))
  }
}
```

```
## [1] "alcohol : has significant differences between the different types of wines"
## [1] "malic : has significant differences between the different types of wines"
## [1] "ash : has significant differences between the different types of wines"
## [1] "alkalinity : has significant differences between the different types of wines"
## [1] "magnesium : has significant differences between the different types of wines"
## [1] "phenols : has significant differences between the different types of wines"
## [1] "flavanoids : has significant differences between the different types of wines"
## [1] "nonflavonoid : has significant differences between the different types of wines"
## [1] "proanthocyanins : has significant differences between the different types of wines"
## [1] "color : has significant differences between the different types of wines"
## [1] "hue : has significant differences between the different types of wines"
```

```
## [1] "dilution : has significant differences between the different types of wines"
## [1] "proline : has significant differences between the different types of wines"
```

Com es pot observar totes 13 variables presenten una diferència significativa. Seguidament estudiarem com de significatives són. Per exemple, per una variable ho podríem calcular utilitzant el test de Wilcox.

```
pairwise.wilcox.test(wine[["color"]], wine$type, p.adjust.method = "holm")
```

```
##
## Pairwise comparisons using Wilcoxon rank sum test
##
## data: wine[["color"]] and wine$type
##
## 1 2
## 2 <2e-16 -
## 3 3e-05 <2e-16
##
## P value adjustment method: holm
```

En aquest cas com $p < 0.05$ en tots els casos de les parelles de vins per al color podem dir que totes presenten diferències significatives.

Per a fer-ho més genèric, agafarem totes les variables, aplicarem el test, i obtindrem la parella amb el “p valor” més elevat per a cada variable. Així podrem veure en el pitjor dels casos quines variables tenen totes les parelles significatives i quines almenys una no.

```
library(dplyr)
difference <- data.frame(variable=header[-1], max_diff=NA, max_val=NA)
index <- 1
for(col in header[-1]){
  print(col)
  test <- pairwise.wilcox.test(wine[[col]], wine$type, p.adjust.method = "holm")
  difference[index, 2] <- max(na.omit(as.numeric(unlist(test$p.value))))
  difference[index, 3] <- test[["p.value"]][[1]] + test[["p.value"]][[2]] +
    test[["p.value"]][[4]]
  index <- index + 1
}
```

```
## [1] "alcohol"
## [1] "malic"
## [1] "ash"
## [1] "alcalinity"
## [1] "magnesium"
## [1] "phenols"
## [1] "flavanoids"
## [1] "nonflavanoid"
## [1] "proanthocyanins"
## [1] "color"
## [1] "hue"
## [1] "dilution"
## [1] "proline"
```

```
difference %>% arrange(desc(max_diff))
```

```
##      variable      max_diff      max_val
## 1      ash 6.700496e-01 6.703250e-01
## 2      hue 5.618454e-01 5.618454e-01
## 3     malic 1.739680e-02 1.739680e-02
```

```
## 4      alkalinity 1.407722e-02 1.407724e-02
## 5      magnesium 1.167133e-03 2.311185e-03
## 6      proanthocyanins 5.762978e-04 5.767891e-04
## 7      nonflavanoid 5.656758e-04 1.131357e-03
## 8      dilution 3.708681e-05 3.708681e-05
## 9      color 2.979159e-05 2.979159e-05
## 10     proline 1.375055e-05 1.375055e-05
## 11     alcohol 1.633498e-07 1.633547e-07
## 12     phenols 4.867799e-09 6.599860e-09
## 13     flavanoids 4.344098e-14 4.344912e-14
```

2 de les 13 variables presenten almenys en una parella diferències no significatives. Totes les altres 11 presenten diferències significatives, essent flavanoids la que més seguida de phenols i alcohol. Per veure el global de les variables amb “p valor” més baix, és a dir, amb la diferència més significativa podem sumar el p-valor de cada variable per parella.

```
difference %>% arrange(desc(max_val))
```

```
##      variable      max_diff      max_val
## 1      ash 6.700496e-01 6.703250e-01
## 2      hue 5.618454e-01 5.618454e-01
## 3      malic 1.739680e-02 1.739680e-02
## 4      alkalinity 1.407722e-02 1.407724e-02
## 5      magnesium 1.167133e-03 2.311185e-03
## 6      nonflavanoid 5.656758e-04 1.131357e-03
## 7      proanthocyanins 5.762978e-04 5.767891e-04
## 8      dilution 3.708681e-05 3.708681e-05
## 9      color 2.979159e-05 2.979159e-05
## 10     proline 1.375055e-05 1.375055e-05
## 11     alcohol 1.633498e-07 1.633547e-07
## 12     phenols 4.867799e-09 6.599860e-09
## 13     flavanoids 4.344098e-14 4.344912e-14
```

Si comparem les dues taules, veurem com les úniques variables que s’han canviat la posició són proanthocyanins i nonflavanoid. Aquest nou ordre ens permetrà saber quines són les variables més importants a tenir en compte a l’hora de crear els que ens prediguin el tipus de vi. ### Mètodes de classificació no supervisats

Els mètodes de classificació no supervisats són aquells que sense indicar-li a l’algorisme 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 podríem 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.

5.3.2.1 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’algorisme. En cas de no normalitzar les dades a causa del fet 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.

```
# 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), a l'aplicar un algorisme 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.

```
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]))
}

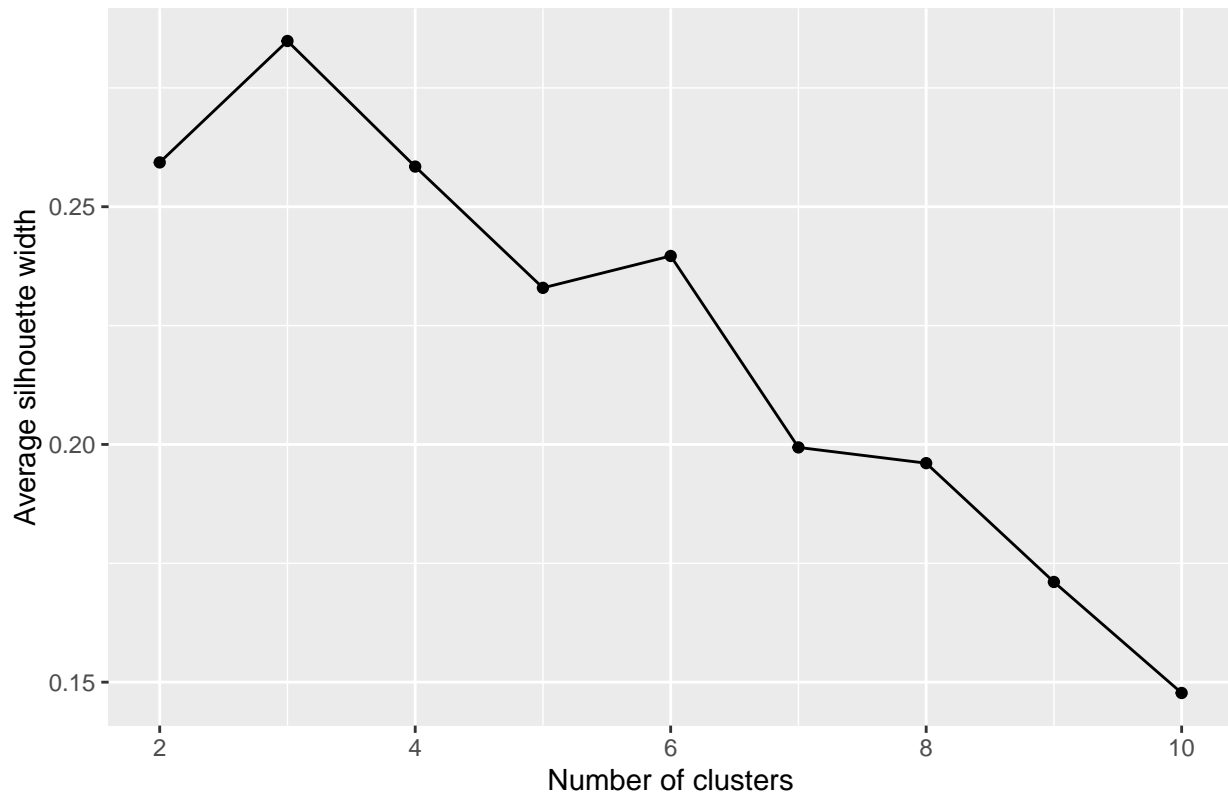
# Funció que retorna un gràfic amb les diverses siluetes pel rang de
#valors de cluster especificat.
calculate_silhouette <- function(data_frame, n_clusters, title){
  # Creem un vector de 10 posicions (encara que utilitzarem de la 2 a la 10).
  silhouette_cluster <- vector(length = 10)
  # Calculem 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_line() + geom_point() +
    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")
```

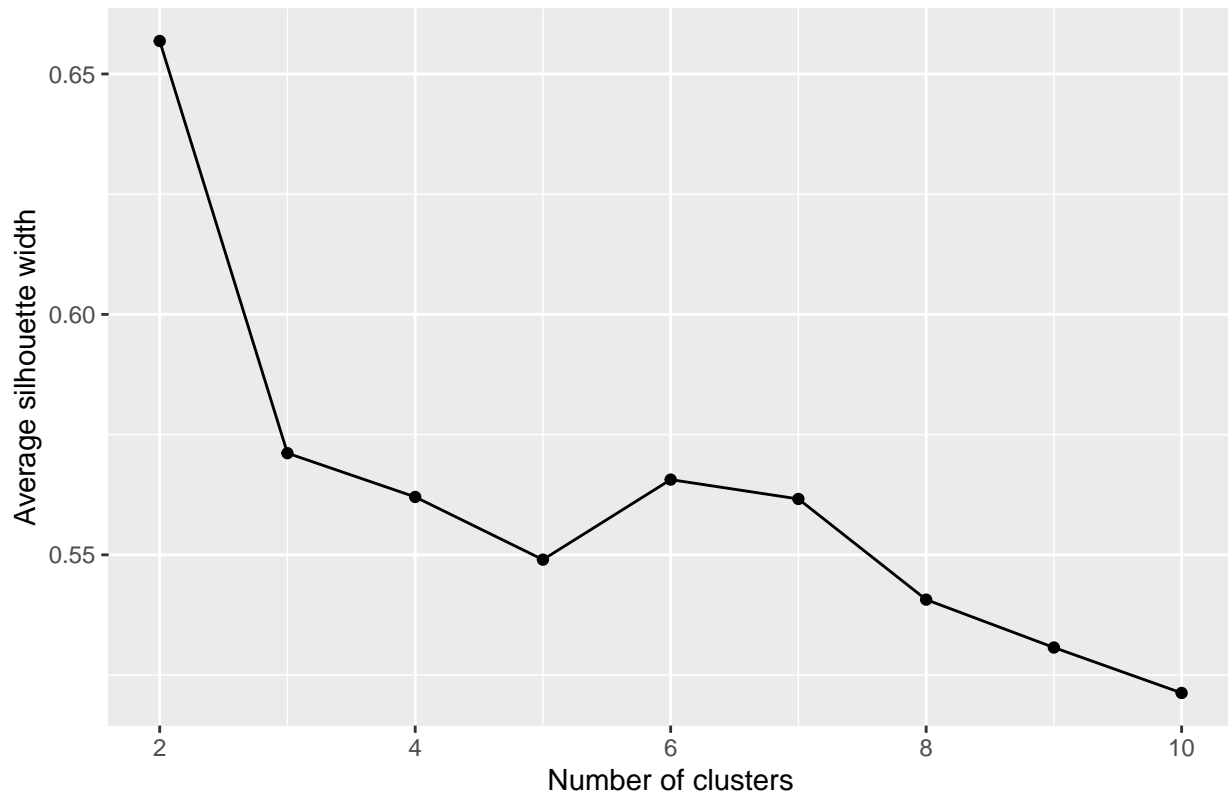
Optimal number of clusters using average silhouette method



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 normalized)")
```

Optimal number of clusters using average silhouette method (not normalized)



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*.

```
# 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)
}

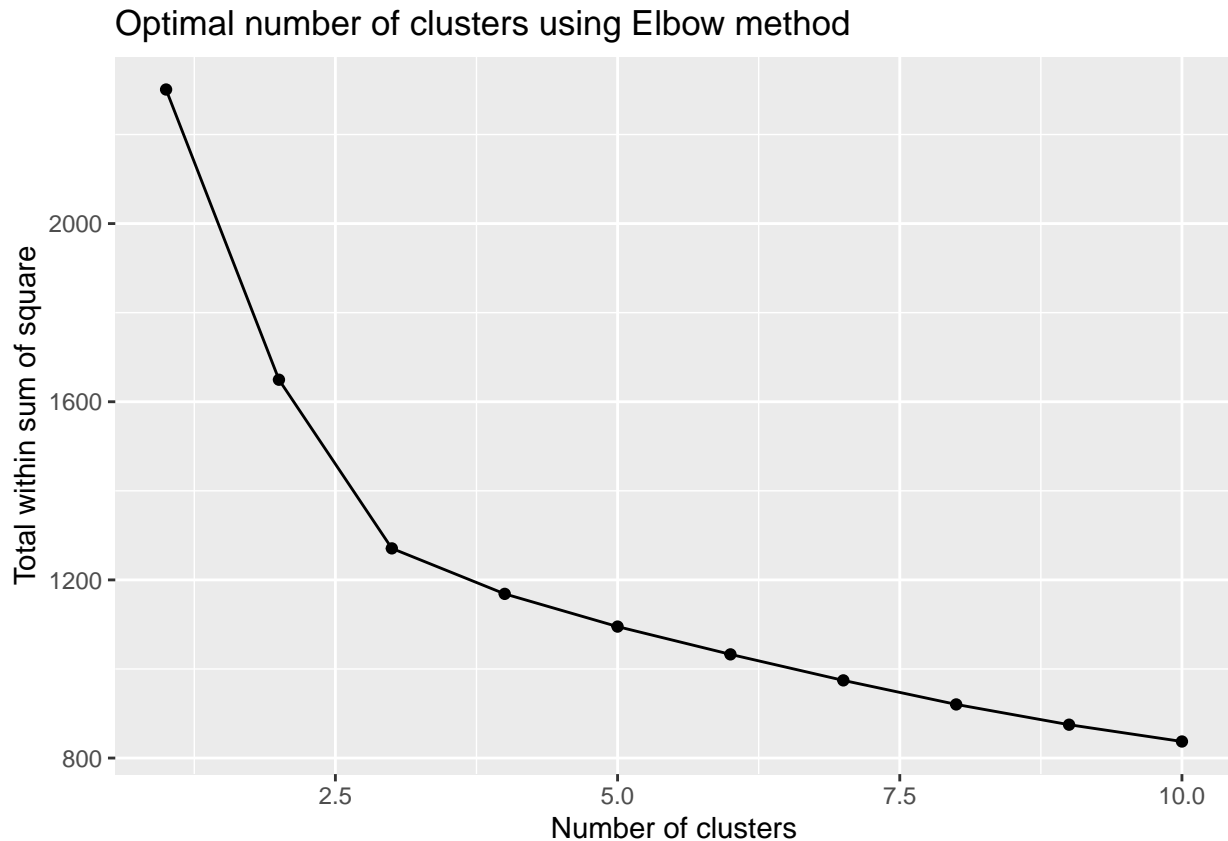
# Funció que retorna un gràfic representant l'elbow
calculate_elbow <- function(data_frame, n_clusters, title){
  # Creem un vector de 10 posicions (encara que utilitzarem de la 2 a la 10).
  elbow_cluster <- vector(length = 10)
  # Calculem l'elbow pels clusters de 2 a 10
  for (n_clusters in 1:10){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

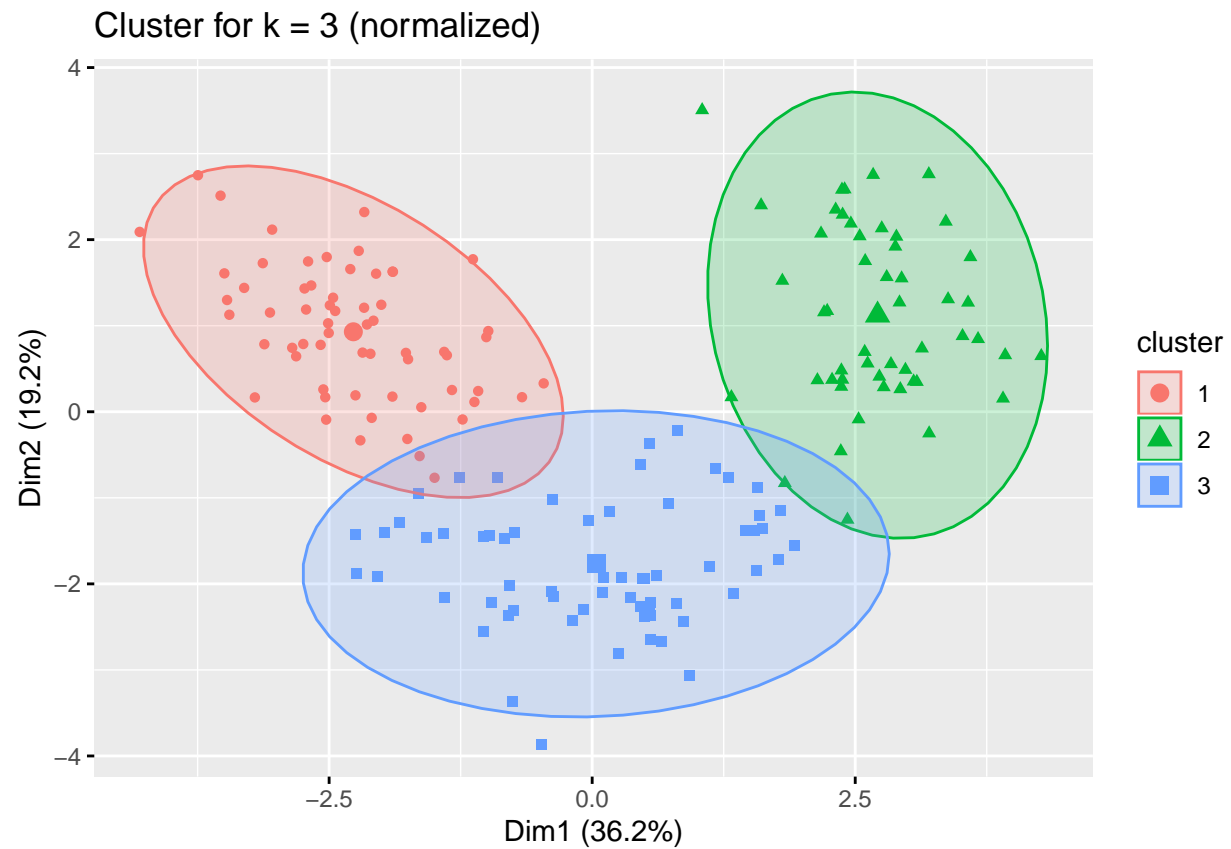
Seguidament anem a utilitzar l'algorisme k-means.

```
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)

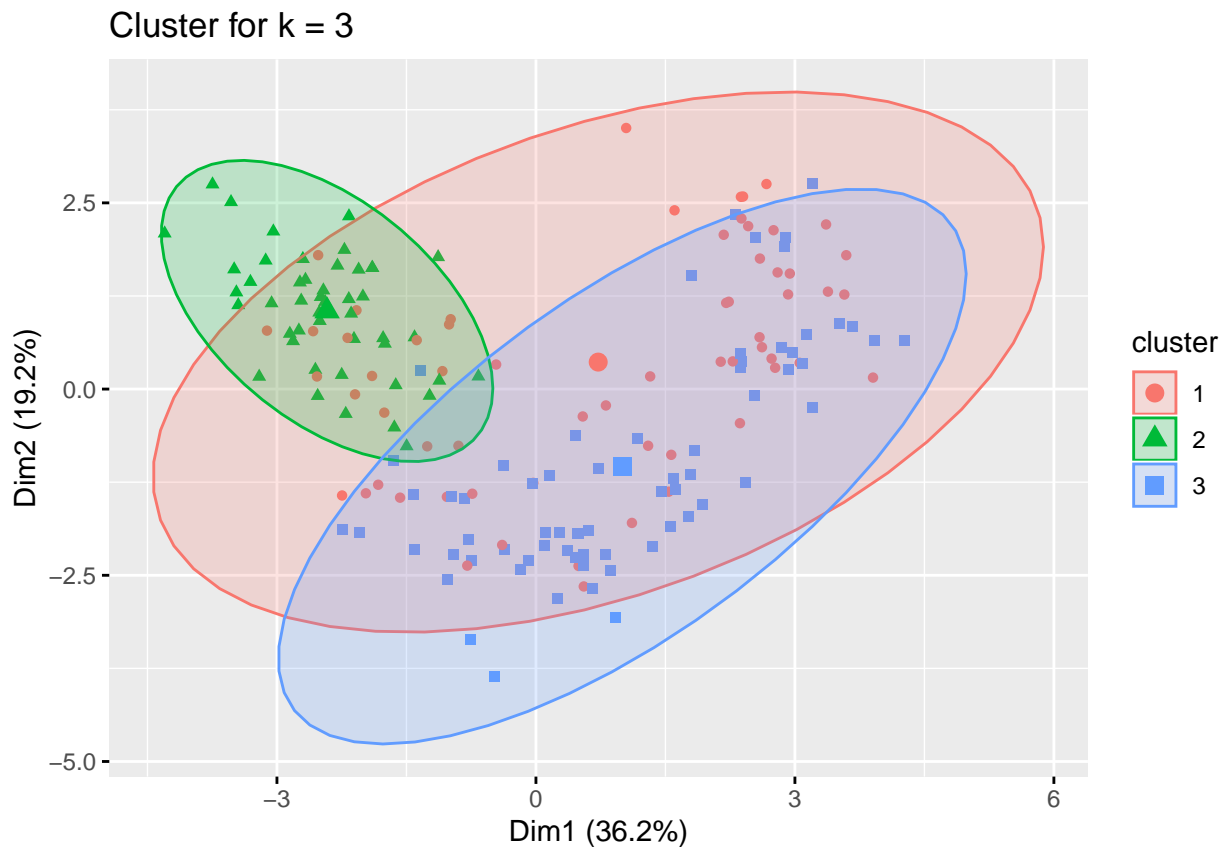
  # Graphical representation of the clusters
  fviz_cluster(result, data = dataset, show.clust.cent = TRUE, geom = "point",
               ellipse.type = "norm", main = title)
}

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 és contrastant-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
```

```
# 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.

Ara el que podem provar és a crear un model amb menys variables i veure com es comporta. Gràcies al test de Kruskal-Wallis podem triar les variables que estadísticament presenten una major diferència i que per tant ens haurien d'ajudar a crear millors models.

Primer crearem un model amb les 5 millors variables del conjunt segons el resultat obtingut.

```
kmeans3_wine_simple <- get_classification(seed = 6, dataset = zWine[, c(1, 6, 7, 10, 13)], k=3)
# Calculem la probabilitat
calculate_probability(kmeans3_wine_simple$cluster, wine_types)
```

```
## [1] 96.06742
```

```
# Mostrem com s'ha classificat
table(clusters= kmeans3_wine_simple$cluster, original=wine_types)
```

```
##          original
## clusters  1  2  3
##          1 58  4  0
##          2  1 65  0
##          3  0  2 48
```

Com podem veure amb tan sols 5 variables aconseguim una precisió d'un 96.07%, només un 0.56% inferior al creat amb totes les variables.

Ara podem fer el mateix, però per a les dues pitjors.

```
kmeans3_wine_simple <- get_classification(seed = 6, dataset = zWine[, c(2,3)], k=3)
# Calculem la probabilitat
calculate_probability(kmeans3_wine_simple$cluster, wine_types)
```

```
## [1] 57.86517
```

```
# Mostrem com s'ha classificat
table(clusters= kmeans3_wine_simple$cluster, original=wine_types)
```

```
##          original
## clusters  1  2  3
##          1 33 18 14
##          2 19 41  5
##          3  7 12 29
```

Com es pot observar la selecció de les variables és molt important en la creació del model, perquè amb aquestes dos variables que tenen una parella d'atributs que no són estadísticament diferents obtenim un model amb tan sols un 57.86% de precisió.

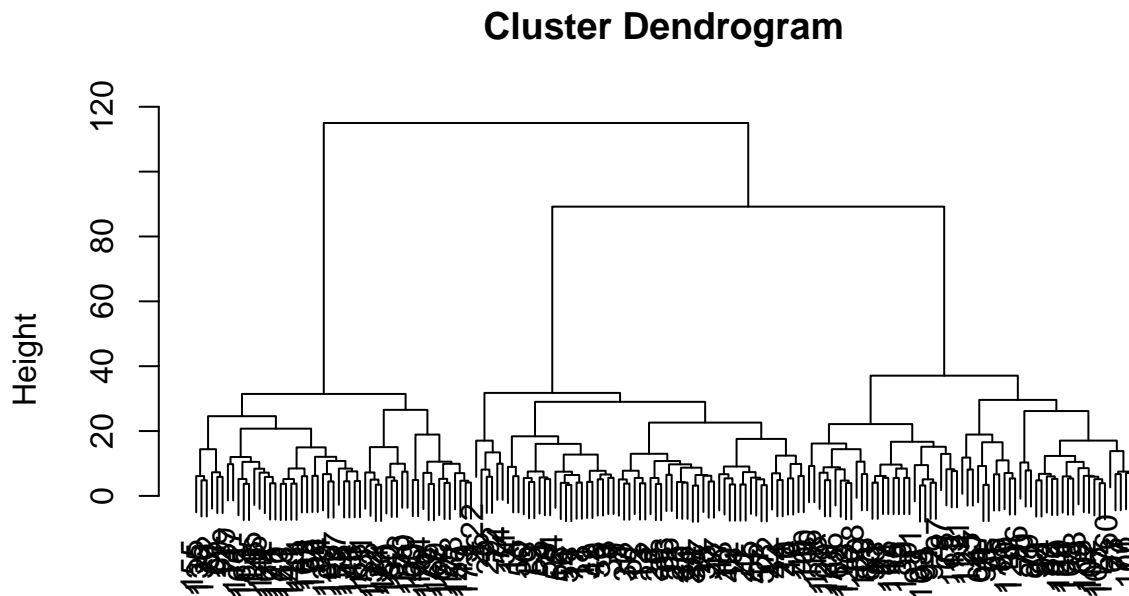
5.3.2.2 Hierarchical agglomerative clustering

Aquest mètode no supervisat busca crear una jerarquia entre els clusters. Hi ha diverses estratègies per enfocar-ho, i en aquest cas ens centrarem a utilitzar l'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.

```
hierarchical_agglomerative_clustering <- function(data, distance_method, linkage_criteria){  
  hclust(dist(data, method=distance_method), method=linkage_criteria)  
}  
  
hac <- hierarchical_agglomerative_clustering(zWine, "manhattan", "ward.D2")  
plot(hac)
```



```
dist(data, method = distance_method)  
hclust (*, "ward.D2")
```

Com es pot veure en el dendrograma 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.

5.3.3 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 faci 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_type=test_type))
}
```

Seguidament aplicarem l'algorisme, però hem de decidir el valor de k. Per a decidir-lo, provarem 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'algorisme
```

```

knn_wine <- knn(samples@train_sample, samples@test_sample, cl=samples@train_type, k=k)

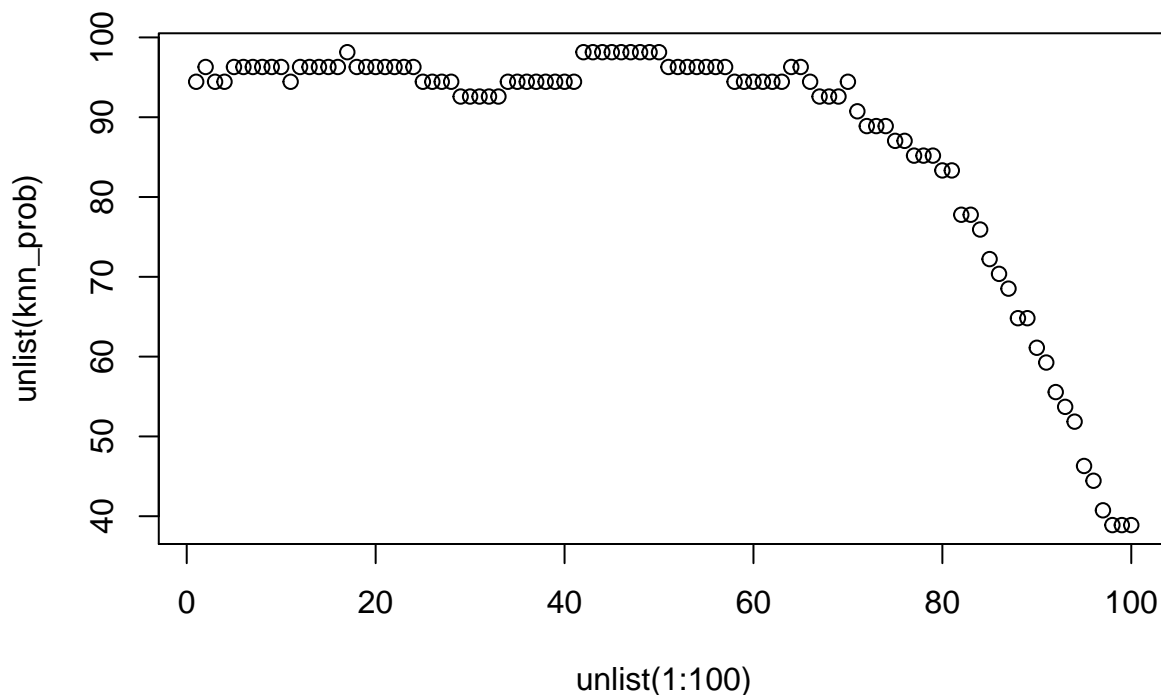
# Calculem la precisió de l'algorisme
accuracy <- calculate_probability(knn_wine, samples@test_type)

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 percentatges 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.

5.3.4 Arbres de classificació

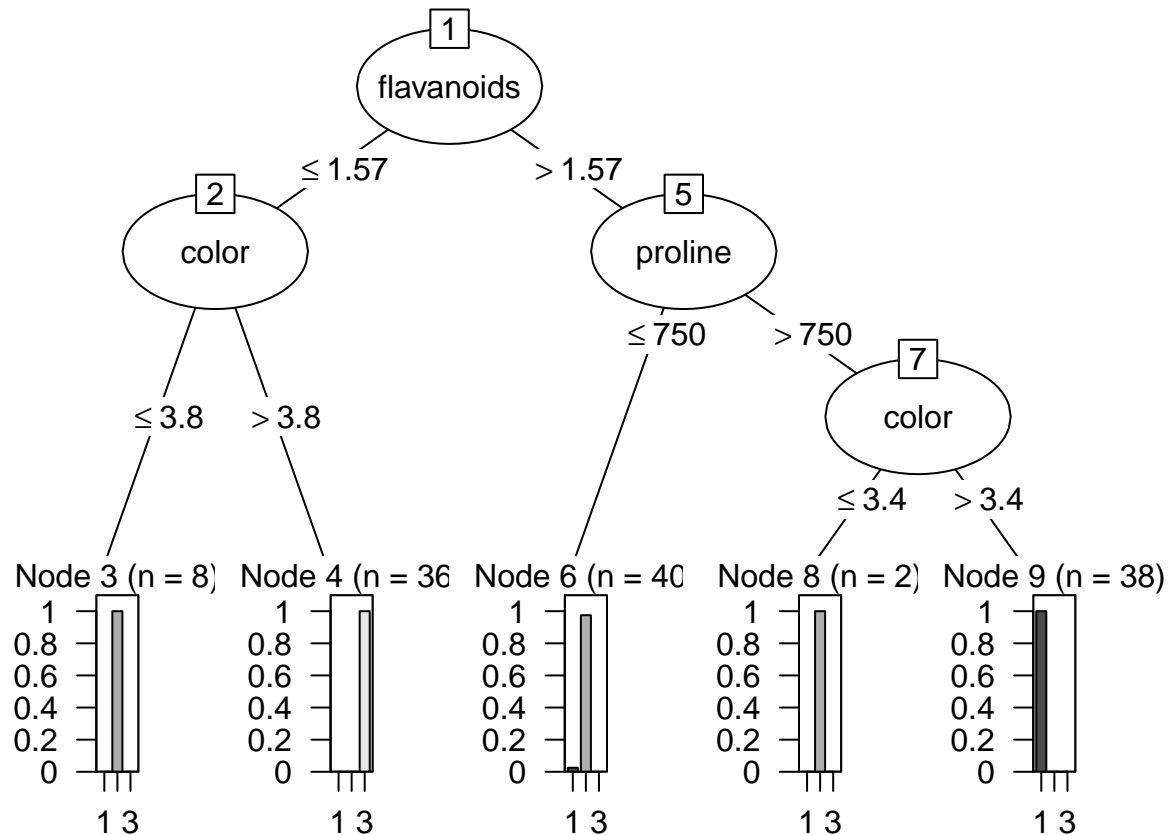
Els arbres de classificació formen parts dels algorismes supervisats, i són utilitzats per predir el valor d'una variable categòrica (en aquest cas el tipus de vi) segons els valors que prenen les altres variables. A més de permetre crear aquests models per predir els valors, són una eina molt potent des del punt de vista d'anàlisi exploratòria perquè ens permet entendre les variables que ajuden a classificar i per tant les característiques de cada vi.

Anem doncs a aplicar l'algorisme gràcies a la llibreria C50.

```
library(C50)
# Separem el dataset en train i test
wine.sample <- create_sample(wine, wine_types, 0.7, 1)

# Apliquem l'algorisme
wine.modelC50 <- C5.0(wine.sample@train_sample, wine.sample@train_type)

# El visualitzem
plot(wine.modelC50)
```



Com es pot veure en l'arbre de decisió obtingut els vins de tipus 1 són aquells en què el valor de “flavanoids” és superior a 1.57, el valor de “proline” és superior a “750” i el valor de “color” superior a 3.4. En canvi, els vins de tipus 3 són aquells amb el valor de “flavanoids” inferior o igual a 1.57 i amb “color” superior a 3.8. Pel que fa al grup 2, en aquest arbre està més repartit pels diferents nodes (en un total de 3) tot i que el grup principal és aquell amb “flavanoids” superior a 1.57 i “proline” inferior o igual a 750. Les altres dues fulles de l'arbre que contenen observacions del grup 2 una només en té 2 i l'altre 8.

En aquest cas, si ens fixem amb les variables que s'han utilitzat per a la creació de l'arbre veiem com apareixen 3 de les 5 variables que apareixen en aquest són les que hem vist que eren les que presentaven diferències més significatives entre els tipus de vins i amb la qual hem pogut crear un model molt més simple amb l'algorisme k-means. En cas d'utilitzar-lo aquí, veuríem com l'arbre de decisió creat és el mateix.

Com en els casos anteriors anem a veure com de “bo” és el model creat per a la predicció dels tipus de vins.

```
# Predim els nous valors utilitzant el model creat
predicted_model <- predict(wine.modelC50, wine.sample@test_sample, type="class")
# Calculem la precisió
print(100*sum(predicted_model == wine.sample@test_type) / length(predicted_model))
```



```
## [1] 96.2963
```

En aquest cas l'algorisme ha aconseguit un resultat molt bo amb una precisió del 96.30 %.

6 Exportació de les dades

Finalment escriurem les dades que tenim en el dataset en un fitxer CSV. Hi afegirem els valors obtinguts de cada algorisme de classificació juntament amb la classificació original en els casos dels algorismes no supervisats.

```
# Afegim els tipus
wine$type <- wine_types
wine$kmeans_clusters <- kmeans3_wine$cluster
wine$hac <- cutree(hac, 3)

# Exportació de les dades
write.csv(wine, "../data/wine_out.csv")
```

7 Conclusions

En aquesta segona pràctica hem pogut treballar la càrrega, neteja i anàlisi de les dades en aquest per al conjunt de dades **Wine Dataset**.

Hem vist com el conjunt de dades no presentava zeros o elements buits, però si alguns valors extrems (outliers) que hem decidit mantenir perquè en cap cas es tractaven de valors erronis, i es podien donar perfectament durant l'anàlisi químic dels components del vi.

Tanmateix, hem realitzat una anàlisi estadística de la normalitat i homoscedasticitat de les dades, on hem vist resultats controvertits depenent de la prova estadística escollida. Tot i això hem vist com la majoria de les dades seguien una distribució normal i en cap cas homoscedàstica, fet que ens ha permès intuir que les dades es podrien classificar fàcilment.

Pel que fa a l'anàlisi de les dades primer s'ha realitzat un estudi de la correlació de les variables i després s'han aplicat algorismes no supervisats i supervisats per a la classificació de les dades.

En l'estudi de la correlació hem vist com hi havien algunes variables molt positivament correlacionades i que a més al visualitzar-les permetien diferenciar molt clarament els diferents tipus de vins.

En el cas dels algorismes n'hem aplicat un total de quatre: dos de no supervisats (k-means i hierarchical agglomerative clustering) i dos de supervisats (arbres de classificació i Knn). Cal mencionar que tots han obtingut un molt bon resultat amb precisions no inferiors al 94.94%. L'algorisme que millor ha funcionat és el supervisat **Knn** que ha obtingut un 98.15% d'encerts, seguit del k-means amb un 96.29%, els arbres de decisió amb un 96.29% i el hierarchical agglomerative clustering amb un 94.94%.

Cal dir però que tot i que Knn aconsegueix el major *ratio* d'acerts, aquests és només un 3% superior al que ha obtingut pitjor resultat. La gràcia és que tots els altres algorismes ens permeten extreure informació valuosa i d'una manera senzilla com poden ser els arbres de decisió amb les variables clau que separen els diferents grups, o la qualitat de la separació d'aquests grups amb hierarchical agglomerative clustering o k-means.

Destacar la potència dels algorismes no supervisats que sense saber prèviament els grups aconsegueix una classificació excel·lent. A més tot i que hierarchical agglomerative clustering té el "pitjor" ratio dels quatre algorismes aquest l'aconsegueix sense ni tan sols haver d'especificar el nombre de clusters presents en el conjunt de dades.

Finalment destacar com amb només 5 variables, per exemple en el cas de l'algorisme k-means hem pogut crear un model igual de vàlid, i que aquestes variables s'han vist encara més reforçades al veure-les aparèixer a

l'arbre de classificació sense previament haver-ho especificat.

Referències

- Gastwirth, Joseph L., Yulia R. Gel, W. L. Wallace Hui, Vyacheslav Lyubchich, Weiwen Miao, and Kimihiro Noguchi. 2019. *Lawstat: Tools for Biostatistics, Public Policy, and Law*. <https://CRAN.R-project.org/package=lawstat>.
- Kassambara, Alboukadel. 2019. *Ggpubr: 'Ggplot2' Based Publication Ready Plots*. <https://CRAN.R-project.org/package=ggpubr>.
- Kassambara, Alboukadel, and Fabian Mundt. 2017. *Factoextra: Extract and Visualize the Results of Multivariate Data Analyses*. <https://CRAN.R-project.org/package=factoextra>.
- Kuhn, Max, and Ross Quinlan. 2018. *C50: C5.0 Decision Trees and Rule-Based Models*. <https://CRAN.R-project.org/package=C50>.
- Maechler, Martin, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. 2019. *Cluster: Cluster Analysis Basics and Extensions*.
- Schloerke, Barret, Jason Crowley, Di Cook, Francois Briatte, Moritz Marbach, Edwin Thoen, Amos Elberg, and Joseph Larmarange. 2018. *GGally: Extension to 'Ggplot2'*. <https://CRAN.R-project.org/package=GGally>.
- Wei, Taiyun, and Viliam Simko. 2017. *R Package Corrplot: Visualization of a Correlation Matrix*. <https://github.com/taiyun/corrplot>.
- Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>.
- Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2019. *Dplyr: A Grammar of Data Manipulation*. <https://CRAN.R-project.org/package=dplyr>.