СТАТИСТИЧЕСКИЙ ЯЗЫК R

Краткая справка

Григорьева И.С.
Каф. матем. статистики ИМиМ

Оглавление

Полезные команды для управления объектами

Команда	Смысл
help( ) или ?	Справка по команде или объекту
apropos( )	Поиск названия команды по его части
ls( )	*list*, вывод списка объектов
rm( )	*remove*, удаление объектов
str( )	Структура объекта
head( )	Вывод первых нескольких значений объекта
summary( )	Вывод сводки по объекту
is.numeric( )	Проверка (является ли числовым значением)
as.numeric( )	Преобразование в тип «числовой»
vector( )	Создание объекта класса «вектор»
as.vector( )	Преобразование в объект класса «вектор»
is.vector( )	Проверка (является ли объект вектором)

Последние три функции являются образцами для аналогичных.

Основные объекты

Векторы, матрицы, массивы – объекты, содержащие данные одного типа. Различаются размерностью. Матрица двумерна, массив может иметь любую размерность, у вектора размерности нет (*NULL*).

Если действие с векторами производятся поэлементно, то более короткий вектор повторяется циклически.

Фактор – гибрид числового и символьного вектора. Имеет уровни (используются как имена категорий), в составе фактора указаны их номера. Фактор можно преобразовать как в числовой, так и в символьный векторы.

Список – объект, элементами которого могут быть объекты любого класса, в том числе другие списки.

Таблица (*data.frame*) – список столбцов одинаковой длины (векторов или факторов). Имеет дополнительно структуру строк.

Извлечение элементов из объектов

Для **векторов и векторо-подобных объектов** (в том числе факторов) используются индексы [ ], по числу размерностей. Также к любому из них можно обратиться с одним индексом. Отрицательный индекс означает, что этот элемент исключается из объекта.

Можно использовать логические индексы (логический вектор). В этом случае извлекаются те элементы, для которых индекс равен TRUE.

Для **списков** также можно использовать индексы (в том числе логические). Результатом будет снова список. Вместо номера элемента в квадратных скобках можно использовать его название в структуре.

Чтобы извлечь содержимое элемента, можно использовать двойные квадратные скобки [[ ]] или обратиться как к полю через знак $.

Для **таблиц** работают методы как для списка, если нужно извлечь столбцы или содержимое столбца. Кроме того, к ним можно обращаться как к матрицам, с помощью двух индексов. При этом столбец будет преобразован в вектор/фактор, а строка останется таблицей.

Применение команд к полям многомерных объектов

Существует группа команд типа *apply*( ). Собственно *apply*( ) применяется к массивам/матрицам. Объекты класса таблица преобразуются в матрицы. При этом может поменяться тип их элементов.

```
> apply(trees,2,mean)
   Girth   Height   Volume
13.24839 76.00000 30.17097
```

Для списков подходит команда *lapply*( ), она применяется к каждому элементу списка и на выходе дает список той же структуры. Другим вариантом является *sapply*( ), команда пытается преобразовать результат *lapply*( ) в вектор.

Команда *tapply*( ) применяет функцию к данным, сгруппированным по категориям.

```
> tapply(iris[,1],iris[5],mean)
Species
    setosa versicolor  virginica
     5.006      5.936      6.588
```

Обработка пропущенных и неверных значений

Обнаружение NA: *is.na*(*x*); *anyNA*(*x*); *is.nan*(*x*).

Удаление NA:

na.omit(*x*) — удаляет строки с *NA* из вектора, матрицы или *data.frame* (сохраняет индексы удалённых строк в атрибуте *na.action*).

complete.cases(*x*) — возвращает логический вектор: *TRUE* для **строк** без *NA*, *FALSE* — с *NA* (используется для фильтрации).

Вычисления с учётом *NA*: в командах *mean*( ); *sum*( ); *sd*( ); *var*( ); *median*( ) есть аргумент *na.rm*; для игнорирования *NA* надо положить *na.rm* = *TRUE*.

Замещение пропущенных значений средними (для вектора)

```
xx[is.na(xx)] <- mean(xx, na.rm = TRUE)
```

Пользовательские функции

Функция – это объект R, она создается командой *function*( ). В скобках перечисляются аргументы, возможно, с умолчаниями. После скобок выписывается тело функции в операторных скобках { }.

Для вызова функции ее записывают с фактическими параметрами:

Функция выдает в качестве значения последнее вычисленное выражение. Если такого нет (например, использован цикл), в конце тела функции пишется оператор *return*( ).

Замечание. Если внутри функции надо вывести что-то на экран, используйте команду *print*( ), простое указание переменной не даст никакого эффекта. То же верно для циклов.

Аргумент функции сам может быть функцией.

Применение функций в операторах apply( )

В операторах типа *apply*( ) можно использовать пользовательские функции. Например, если у какой-то функции нужно зафиксировать часть аргументов. Или выбрать из результата какой-то элемент.

Функцию можно создать отдельно или ввести непосредственно в *apply*( ).

```
> sapply(iris[,-5],function(x)t.test(x, mu=3)$p.value)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
1.478183e-84 1.092929e-01 4.945680e-07 5.069450e-63
```

При использовании стандартных функций некоторые аргументы можно указать после указания функции:

```
apply(dann,2,mean,na.rm=TRUE)
```

Циклы

Основной вид цикла (*for*-цикл) организуется следующим образом:

```
for(var in seq) expression
```

Здесь *seq* – объект или выражение, значением которого является вектор (или подобный объект).

Счетчик не обязан быть числовым, главное, чтобы его элементы можно было расположить по порядку.

```
for(nn in names(iris)){
  print(is.numeric(iris[[nn]]))
}
```

То же можно переписать так:

```
for(nn in iris){  print(is.numeric(nn)) }
```

Заметим, однако, что команда

```
for(nn in iris){  print(names(nn)) }
```

не сработает, так как *nn* будет принимать значения векторов/факторов, а не таблиц.

Условные операторы

Условные операторы имеют вид

```
if(condition) expression
if(condition) cons.expression  else  alt.expression
```

Замечание. Не всегда можно переносить на следующую строчку часть, начинающуюся с *else*, так как предыдущая строка воспринимается как законченная команда.

```
if(mean(mm)<0.5) {print("мало")}
  else {print("много")}
```

Лучше оставить *else* на предыдущей строке.

Например, оформим вывод информации о проверке гипотезы

```
cat("Отклонение",
     if (t.test(x,y)$p.value >0.05) "не","значимо\n")
```

Кроме *if…else*, существуют условные операторы *ifelse* и *switch*.

Графические команды бывают высокого и низкого уровня. Команды высокого уровня создают окно вывода (*device*) и, возможно, размечают его. Команды низкого уровня добавляют к размеченному окну новые элементы.

Команды высокого уровня: *plot*( ), *boxplot*( ), *hist*( ), *pie*( ), *barplot*( ) , *qqplot*( ), *qqnorm*( ), *image*( ), *contour*( ), *persp*( ) и другие.

Общие параметры: *main* – заголовок графика; *sub* – подзаголовок; *xlab*, *ylab* – подписи осей X и Y; *xlim*, *ylim* – пределы осей; *col* – цвет элементов; *pch* — тип символа для точек; *lty* — тип линии; *lwd* — толщина линии; *cex* — размер символов и текста (масштабный коэффициент) и т.д.

Команды работают по-разному для разных классов объектов, особенно это касается *plot*( ).

Команды низкого уровня: *points*( ) – точки; *lines*( ) – линии; *text*( ) – текст в области графика; *abline*( ) – прямые линии; *legend*( ) – легенда; *title*( ) – заголовки; *mtext*( ) – текст во внешних полях; и многие другие.

Для этих команд доступны в основном те же графические параметры, которые описаны выше.

Кроме того, некоторые **графические параметры** можно устанавливать командой *par*( ). В частности, те, которые касаются не конкретного графика, а всей системы изображений. Это, например, параметры *mfrow* и *mfcol*.

Интерактивная команда *locator*( ) возвращает координаты на графике, в которые кликнул пользователь. Можно предусмотреть несколько «кликов». Полученные координаты можно использовать в командах низшего уровня:

```
text(locator(1),label="Вот тут напишем")
```

Ввод данных из таблиц

Для ввода из Excel-подобных таблиц (с расширением .csv) используется сокращенный вариант: *read.csv*( ) – для региональных стандартов США/Великобритании, *read.csv2*() – для европейских региональных стандартов представления чисел.

Общая команда *read.table*( ) требует самостоятельной настройки некоторых параметров. Ее можно использовать для чтения из текстовых файлов.

Результатом является объект класса таблица. Столбцы в нем являются векторами. Некоторое время назад (до 2020 года) по умолчанию символьные векторы превращались в факторы. Если нужны именно факторы, можно установить параметр

stringsAsFactors = *TRUE*.

Вывод данных в Excel-подобные файлы

Основная команда вывода – *write.table*( ) с большим числом параметров. Частными случаями этой команды являются *write.csv*( ) и *write.csv2*( ) (в зависимости от региональных настроек). В этих командах предусмотрены некоторые разумные умолчания. Функция дозаписи отключена, параметр *append* недействителен.

Замечание о пути. Если читаемые или записываемые файлы не лежат в текущей папке, нужно указать путь к ней. При этом бакслэш меняется на прямой слэш или на двойной бакслэш.

Кроме того, при большом объеме ввода и вывода удобно применять вычисляемые имена файлов. В них можно включить и путь.

Вывод данных на экран

Для вывода на экран содержимого объекта используются команды *print*( ), *write*( ), *cat*( ) и другие.

Команда *print*( ) удобна для того, чтобы выводить хорошо структурированный объект, например, матрицу или таблицу.

Если мы хотим вывести вперемежку текст и числовые данные, удобно использовать команду *cat*( ) = «Concatenate And prinT».

Она самостоятельно объединяет разные объекты в строку, так что не нужно использовать команду *paste*( ). Конец строки надо указывать самостоятельно. Либо знаком "*n*", либо просто клавишей Enter внутри символьной строки.

Кроме *print*( ) и *cat*( ) в ***R*** есть ещё команда *write*(). Однако она слишком «портит» структуру выводимых данных.

Вывод данных в текстовый файл

Для вывода символьных данных в текстовый файл можно поступить так: сначала меняем «направление» вывода командой *sink*( ) с указанием файла для вывода. После неё команды *cat*( ), *print*( ) и т.п. выводят информацию в этот файл.

Когда вывод закончен, файл отсоединяют командой *sink*( ) без аргументов. Если надо добавить информацию к существующей, придайте параметру *append* команды *sink*( ) значение *TRUE*.

Замечание. Иногда после выполнения команды *sink*(<имя файла>) происходит ошибка и последующие команды не выполняются. В этом случае надо применить *sink*( ) несколько раз, пока он не укажет, что произошла ошибка.

Желательно, чтобы файл для вывода находился в текущей папке. В противном случае нужно указать путь к нему. Некоторые операторы сами управляют направлением вывода. Например, в оператор *cat*() можно включить название файла (и путь).

Вывод графики

Команда вывода совпадает с названием типа файла. Каждая такая команда перенаправляет графический вывод в соответствующий файл. Далее можно запустить команду высокого уровня и несколько команд низкого уровня.

Команды *bmp*( ), *jpeg*( ), *png*( ) и *tiff*( ) выводят в файл одну картинку.

Если нужно поместить в графический файл несколько страниц картинок, используйте команду (и формат) *pdf*( ).

После того, как вывод закончен, нужно отключить файл командой *dev.off*( ).

Команды, связанные с распределениями

С каждым распределением связаны четыре функции. Например, для нормального распределения *dnorm*( ), *pnorm*( ), *qnorm*() *rnorm*( ). Первые буквы обозначают *d* = density, плотность распределения, *p* = probability, функция распределения, *q* = quantile, квантили, *r* = random, случайные числа.

Выборки из набора значений образуются с помощью функции *sample*( ). По умолчанию выборка происходит без возвращения.

Для моделирования большого числа выборок можно применить такой подход:

```
n<-10; nvyb<-1000
x<-runif(n*nvyb)
dim(x)<-c(n,nvyb)
```

Проверка гипотез

Команды, выполняющие проверку гипотез, имеют названия вида *.*test*( ). Их можно найти с помощью команды *apropos*(“*test*”).

Команды для разных критериев могут отличаться способом описания данных. Одна или две выборки могут вводиться как отдельные аргументы. Другой вариант – данные помещают в один вектор, при этом в другом векторе или факторе указываются имена выборок.

В последнем случае вместо объекта с данными в команду вводится формула модели, *значения ~ категории*.

Название	Объект гипотезы	Для выборок	Для категорий
binom.test	Вероятность	+	
cor.test	Корреляция	+	+
fisher.test	Независимость	+	
kruskal.test	Сдвиги		+
ks.test	Распределения	+	
oneway.test	Средние		+

prop.test	Вероятности	+	
shapiro.test	Нормальность	+	
t.test	Средние	+	+
var.test	Дисперсии	+	+
wilcox.test	Сдвиги	+	+

Для критериев согласия есть возможность выбрать альтернативную гипотезу, парность (для двухвыборочных вариантов), а также построить доверительный интервал.

Ещё некоторые тесты

pairwise.prop.test	Попарные варианты тестов
pairwise.t.test	
pairwise.wilcox.test	
power.anova.test	Исследование мощности тестов
power.prop.test	
power.t.test	

Критерий хи-квадрат

Критерий хи-квадрат (χ^2, `chisq.test`( )) работает не со значениями с.в., а с размерами выборок. Поэтому данные, передаваемые ему, он округляет до целых и считает размерами групп.

Если на входе задан один числовой вектор, критерий применяется как критерий согласия (соответствие некоторому распределению, по умолчанию – равномерному).

Если задана таблица – она воспринимается как таблица сопряженности признаков и проверяется критерий независимости.

Для построения размеров групп или таблицы сопряженности можно использовать функцию `table`( ).

Если исходные данные символьные, можно не писать команду `table`( ), команда `chisq.test`( ) проведет группировку самостоятельно.

Если исходные данные числовые, их можно разбить на группы командой `cut`( ).

Уравнение регрессии

Команда для построения линейной модели – это *lm*( ). Основной ее аргумент – формула вида *отклик ~ воздействие*. Воздействие может состоять из нескольких переменных, их соединяют знаки + - . : * ^ *I*() (см. Приложение). Смысл этих знаков не арифметический.

Если воздействие задается одним показателем, полученную линию регрессии можно нарисовать с помощью команды *abline*( ).

```
plot(trees[,c(1,3)])
abline(lm(trees[,3]~trees[,1]), col=2)
```

Объясняющие переменные могут быть категориальными. Для этого они должны иметь класс «фактор».

Результат работы *lm*( ) выводится на экран. Кроме того, более подробную информацию о полученной модели можно получить с помощью команды *summary*( ).

Способы визуализации многомерных данных

Метод главных компонент

В *R* есть две функции, вычисляющие направления главных компонент, *prcomp*( ) и *princomp*( ).

```
numIris <- iris[,1:4]
prcomp(numIris)->pr
str(pr)

princomp(numIris)->prin
str(prin)
```

Все компоненты в масштабе:

```
par(mfrow=c(1,3))
plot(predict(prin),xlim=c(-3,4), ylim=c(-3,4),
              col=iris[,5],asp=1)
plot(predict(prin)[,3:4],xlim=c(-3,4), ylim=c(-3,4),
              col=iris[,5],asp=1)
plot(prin)
```

Команды для трех переменных

contour(*x*,*y*,*z*) – выполняет интерполяцию данных и создает контурный график.

filled.contour(*x*,*y*,*z*) – то же, что *contour*(), но заполняет области между контурами определёнными цветами.

image(*x*,*y*,*z*) – изображает исходные данные в виде квадратов, цвет которых определяется значениями *z*.

persp(*x*,*y*,*z*) – трехмерный график.

В этих командах *z* должно быть задано как матрица, каждый элемент которой соответствует паре элементов из *x* и *y*. Например, с помощью команды *outer*( ).

```
x<-y<-seq(-3,3,0.1)
z<-outer(x,y,function(u,v){(u+v)*exp(-(u^2+v^2))})
```

При этом команда *persp*(*x*,*y*,*z*) строит трехмерный (сетчатый) график, а *contour*(*x*,*y*,*z*) – его линии уровня.

Кластеризация методом k-means, с фиксированным числом кластеров реализуется командой *kmeans*( ). Исходные данные – числовая матрица или объект R, который можно преобразовать в неё.

Результатом является список класса *kmeans*, в частности, его элемент $*cluster* задает номера кластеров для кластеризуемых объектов (строк матрицы).

Иерархическая кластеризация методом слияния кластеров задается командой *hclust*( ). Входные данные – матрица класса *dist*; строится функцией *dist*( ), в которой реализованы разные виды расстояний.

Методы объединения кластеров указывается в аргументе *method*. Результатом является дендрограмма, которую можно визуализировать функцией *plot*( ).

Для разделения на кластеры используется функция *cutree*( ).

Иллюстрация к ЦПТ (локальный вариант)

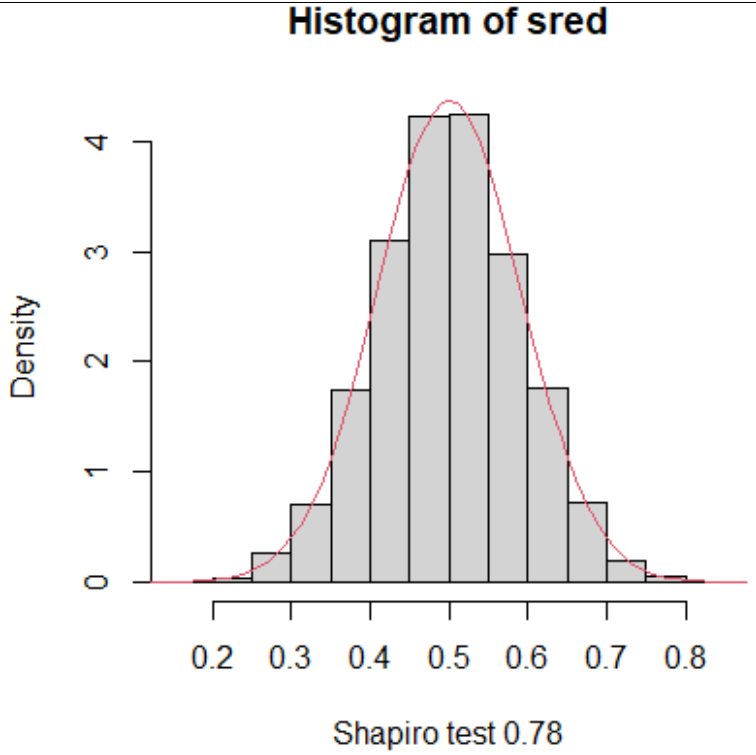
Найдем среднее значение n случайных величин, каждая из которых распределена равномерно на [0; 1].

```
CPT<-function(n=10,nvyb=5000){
  x<-runif(n*nvyb)
  dim(x)<-c(n,nvyb)
  apply(x,2,mean)->sred
  hist(sred, freq=FALSE,
     xlab=paste("Shapiro test",
          format(shapiro.test(sred)$p.value,dig=2)))
  t<-(0:100)/100
  lines(t,dnorm(t,0.5,1/sqrt(12*n)),col=2)
}
```

Больше 5000 выборок обработать нельзя, так как тест Шапиро не рассчитан на большее количество выборок.

Запустим функцию с аргументами по умолчанию:

```
CPT()
```



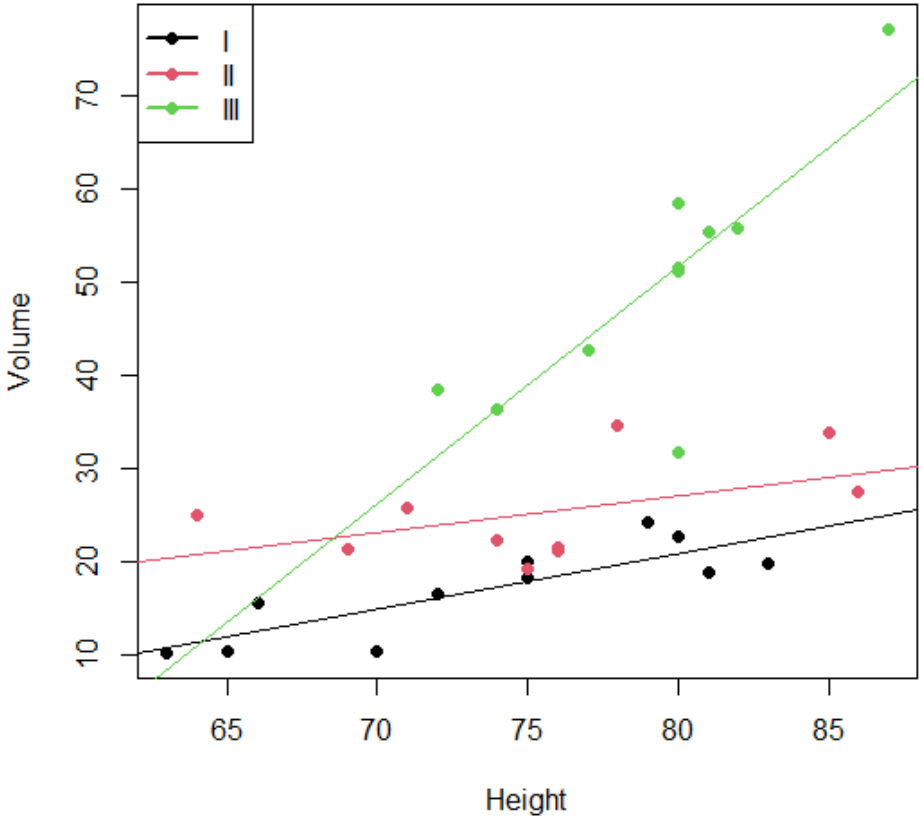
Нормальность подтверждается на уровне значимости 0,78.

Построение линейной модели

```
cut(trees[,1],quantile(trees[,1],(0:3)/3),
           c("I","II","III"),inc = TRUE)->diam

model<-lm(trees[,3]~diam*trees[,2])
coef(model)->coef
dim(coef)<-c(3,2)

coef[2:3,]<- sweep(coef[2:3,],2,coef[1,],"+")

plot(trees[,c(2,3)],col=diam,pch=16)
legend("topleft", legend = levels(diam), col = 1:3,
       lwd = 2, pch = 16)
for(i in 1:3) abline(coef[i,],col=i, lwd=2)
```

Программа строит три линии регрессии, в зависимости от значений первого показателя. Он разбит на три уровня, в них примерно поровну значений.

На графике каждому уровню соответствует свой цвет.

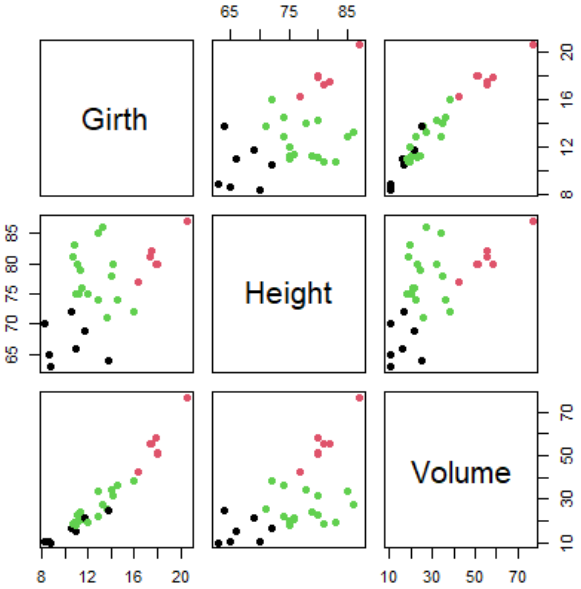

I
II
III
Volume
10
20
30
40
50
60
70
65
70
75
80
85
Height

Кластеризации с k-means

Чтобы исключить влияние разных масштабов компонент, шкалируем их так, чтобы у каждого столбца было среднее 0 и дисперсия 1.

```
kmeans(scale(trees),3,nstart = 5)->kmt
plot(trees, col=kmt$cluster,asp=1, pch=16)
```

Параметр *nstart* задает число прогонов метода, что позволяет уменьшить влияние случайного выбора начальных центров.

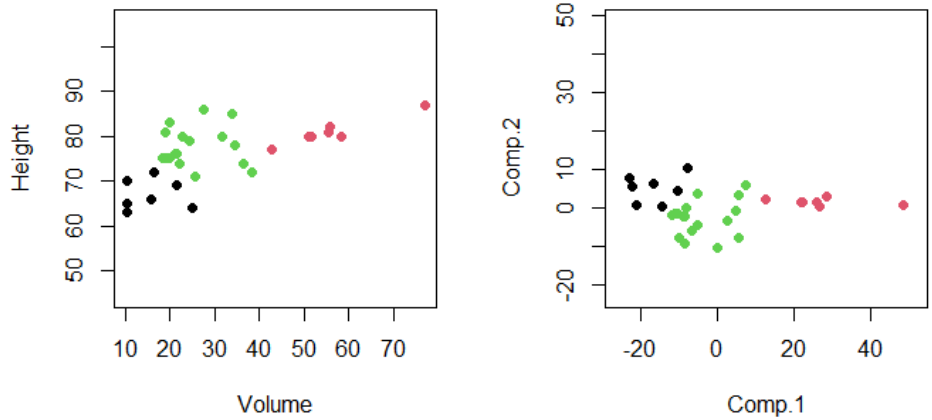


Визуализируем эти же данные с выделением главных компонент:

```
princomp(trees)->prin
```

Сравним исходный объект и «повернутый».

```
c(min(prin$scores),max(prin$scores))->lims
par(mfrow=c(1,2))
plot(trees[,2:3], col=kmt$cluster,asp=1, pch=16)
plot(predict(prin),xlim=lims, ylim=lims,
       col=kmt$cluster,asp=1, pch=16)
```

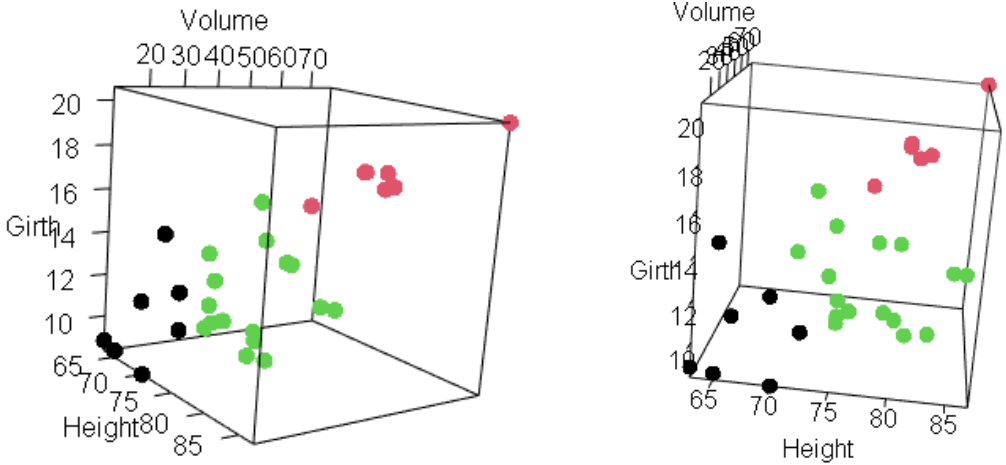


Видно, что первая компонента в основном определяется показателем *Volume*, а вторая – *Height*. И действительно:

```
> loadings(prin)
```

```
Loadings:
       Comp.1 Comp.2 Comp.3
Girth   0.176          0.980
Height  0.242 -0.969
Volume  0.954  0.229 -0.192
```

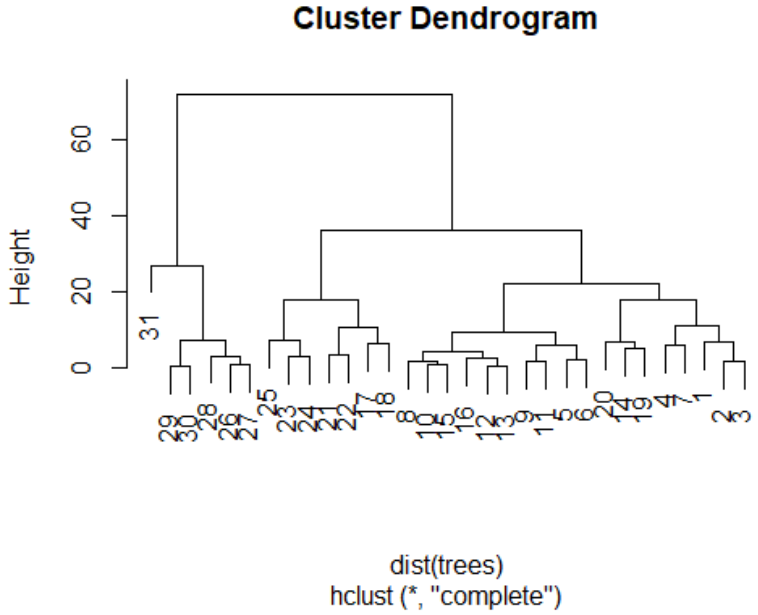
Трехмерное изображение (с помощью пакета rgl):

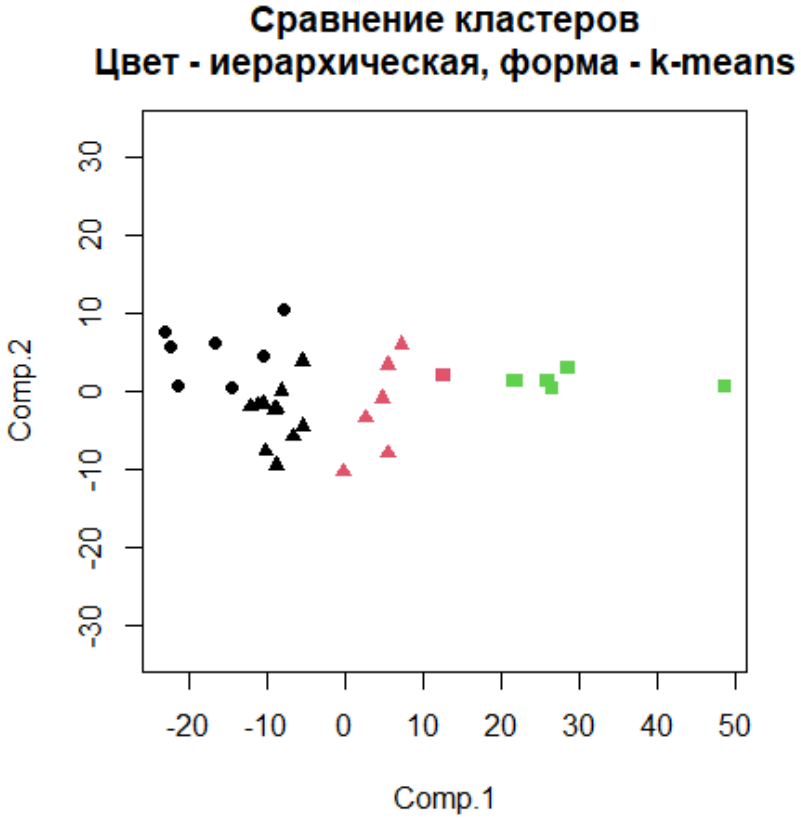


Иерархическая кластеризация

```
hclust(dist(trees))->htrees
plot(htrees)
```

На экран выводится дендрограмма




Сравнение кластеров
Цвет - иерархическая, форма - k-means
Comp.2
30
20
10
0
-10
-20
-30
-20
-10
0
10
20
30
40
50
Comp.1

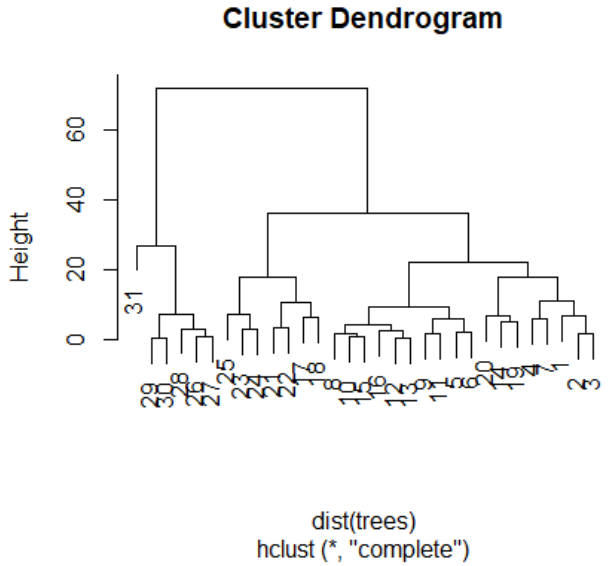
Вот скрипт:

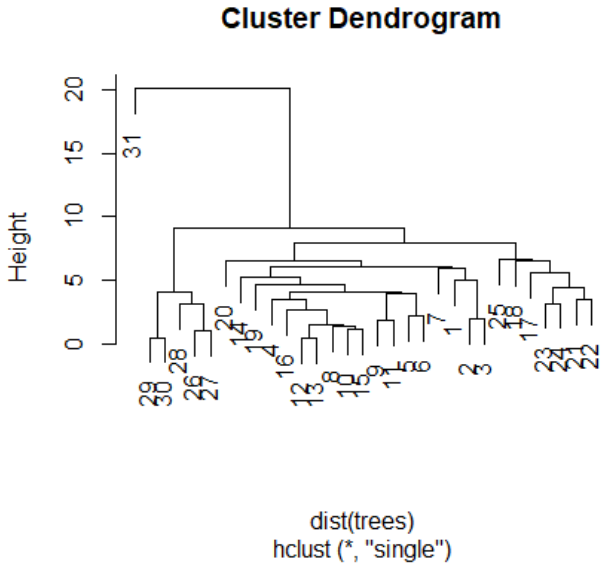
```
hclust(dist(trees))->htrees
kmeans(scale(trees),3,nstart = 5)$cluster->kmc

princomp(trees)->prin
plot(predict(prin),col=cutree(htrees, 3),
pch=(15:17)[kmc], asp=1)
title(main="Сравнение кластеров
Цвет - иерархическая, форма - k-means")
```

Использован метод главных компонент. Подумайте, как реализован вывод заголовка в две строки.

При иерархической кластеризации можно менять метод объединения кластеров.

```
par(mfrow=c(1,2))
plot(hclust(dist(trees)))
plot(hclust(dist(trees), method="single"))
```

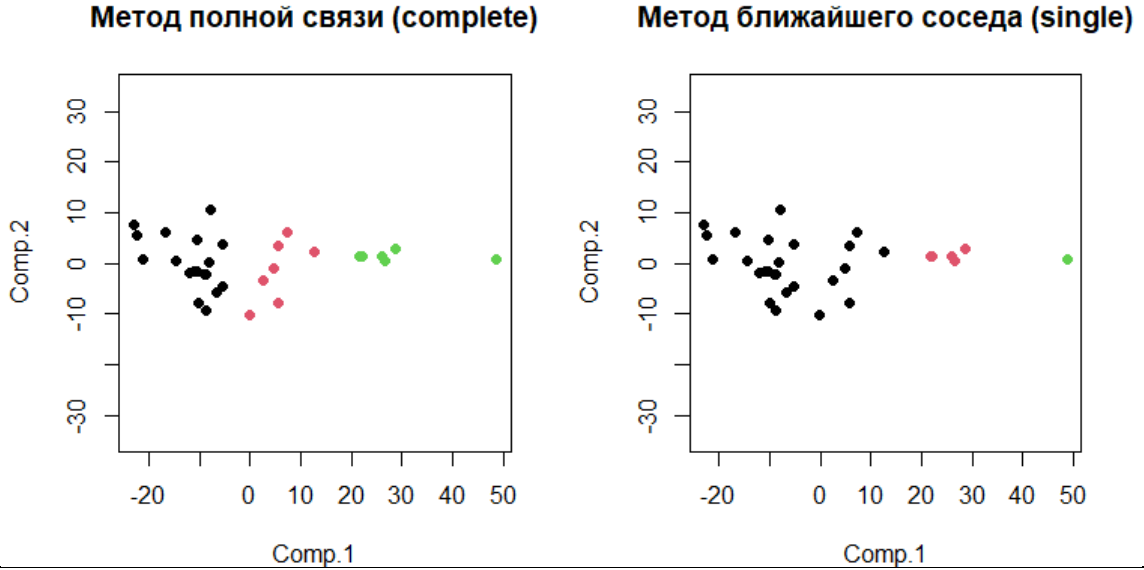




Для `trees` метод `complete` дает более сбалансированные кластеры.

Замечание. Слово `Height` в данном случае относится не к объекту `trees`, а к описанию дендрограммы.

Сравним два вида кластеризации в проекции на главные компоненты.




```
par(mfrow=c(1,2))
plot(predict(prin),col=cutree(hclust(dist(trees)),3),
   asp=1,pch=16, main="Метод полной связи (complete)")
plot(predict(prin),col=cutree(hclust(dist(trees),
                                 method="single"),3),
   asp=1,pch=16, main="Метод ближайшего соседа (single)")
```

Приложение. Заметки и трюки

Как по внешнему виду различить матрицу и таблицу? Например, по форме нумерации:

```
> xd
  V1 V2 V3 V4
1  1  3  5 NA
2  2  4 NA  3
> is.na(xd)
        V1    V2    V3    V4
[1,] FALSE FALSE FALSE  TRUE
[2,] FALSE FALSE  TRUE FALSE
```

Фактор выглядит так:

```
> aa<-c("first","second","third")
> as.factor(aa[sample(1:3,7,rep=TRUE)])
[1] second first  first  second third  third  third
Levels: first second third
```

Особенности работы с матрицами

При выделении из матрицы двух столбцов получаем матрицу; одного – вектор:

```
> dim(mm[,1:2])
[1] 3 2
> dim(mm[,1])
NULL
```

Попытка прибавить к строкам матрицы первую строку:

```
> x<-1:8
> dim(x)<-c(4,2)
> x
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> x[2:4,]+x[1,]
     [,1] [,2]
```

```
[1,]    3   11
[2,]    8    8
[3,]    5   13
```

Числа 1 и 5 прибавляются не к элементам столбцов, а змейкой сверху вниз, так как матрицы превращаются в векторы, причем по столбцам.

Для правильного прибавления можно придумать искусственные приемы, или использовать функцию *sweep*( )

```
> sweep(x[2:4, ], 2, x[1, ], "+")
     [,1] [,2]
[1,]    3   11
[2,]    4   12
[3,]    5   13
```

Трюки

Для подсчета количества элементов с определенным свойством используется *sum*( ), для подсчета доли – *mean*( )

```
> xd
  V1 V2 V3 V4
1  1  3  5 NA
2  2  4 NA  3
> sum(is.na(xd))
[1] 2
> apply(is.na(xd),2,mean)
 V1  V2  V3  V4
0.0 0.0 0.5 0.5
```

Можно применять *apply*( ), так как *is.na*( ) возвращает матрицу.

Для важных программ желательно удалять все ранее созданные объекты, чтобы они не конкурировали с новыми:

```
> rm(list=ls())
> ls()  # проверяем список объектов
character(0)
```

Операторы в формуле модели:

Оператор	Значение	Пример	Интерпретация
~	Зависимость	y ~ x	y зависит от x
+	Добавление переменной	y ~ x1 + x2	y зависит от x1 и x2
-	Исключение переменной	y ~ . - x3	Все переменные, кроме x3
.	Все остальные переменные в датасете	y ~ .	y зависит от всех остальных переменных в данных
:	Взаимодействие (перемножение)	y ~ x1:x2	Эффект взаимодействия x1 и x2
*	Взаимодействие + главные эффекты	y ~ x1 * x2	Эквивалентно x1 + x2 + x1:x2
^	Взаимодействие до указанного порядка	y ~ (x1 + x2)^2	Все главные эффекты и взаимодействия 2-го порядка
I()	«Изоляция» арифметических операций	y ~ I(x^2)	Квадратичный эффект x