Bioestadística Práctica

Santiago Benitez-Vieyra

Contents

Introducción **2**

Práctico 1. Introducción al Lenguaje R **5**

Práctico 2. Modelos Lineales Simples **12**

Práctico 3. Modelos Lineales Múltiples I **14**

Práctico 4. Modelos lineales múltiples II **15**

Práctico 5. Manejo de gráficos y representación gráfica de modelos lineales. **18**

Práctico 6. Otros paquetes gráficos: *lattice* y *ggplot2*. **24**

Práctico 7. Programación y construcción de funciones **27**

Práctico 8: Modelos Randomizados y Modelos Nulos **31**

Práctico 9: Bootstrap. **34**

Introducción

R

R es un lenguaje utilizado principalmente para realizar análisis estadísticos, gráficos y para programación. Puede ser copiado y distribuido de forma gratuita y permite que los usuarios estudien como funciona, lo modifiquen de acuerdo a sus necesidades y publiquen sus mejoras y extensiones para ser compartidas con otros usuarios. Presenta además numerosas ventajas para su uso en la investigación científica.

1. Es actualmente el software que ofrece mayor número de funciones estadísticas y aplicaciones para la creación de gráficos. Colaboradores de todo el mundo producen paquetes destinados a resolver problemas particulares o desarrollan determinadas técnicas estadísticas. Cada usuario puede, además, crear sus propias funciones y rutinas combinando las funciones existentes.

2. Existe una amplia comunidad de usuarios y por lo tanto, una gran oferta de ayudas para aquellos que se inician en el uso del programa, incluyendo foros de discusión en internet y una creciente bibliografía.

3. R es una *lingua franca* para el intercambio de ideas en el ámbito científico. De la misma manera en que se escribe un artículo para comunicar ideas, pueden escribirse rutinas para comunicar un análisis. Un número creciente de trabajos citan este software o presentan rutinas escritas en este lenguaje publicadas.

¿Qué es una rutina, qué es un código?

Las rutinas utilizadas por los científicos pueden ser separadas en dos grandes categorías (Mislan *et al.* 2016, doi 10.1016/j.tree.2015.11.006).

1. Las rutinas de análisis (analysis code) se utilizan para corregir errores en los datos, simular el resultado de modelos, realizar análisis estadísticos y crear figuras. Poner a disposición de otros investigadores las rutinas de análisis es necesario para que los resultados de un estudio puedan ser reproducidos.

2. El software científico (como puede ser un paquete de R o Python) es diseñado para poder ser utilizado en muchos proyectos diferentes. Puede ser el producto mismo de una investigación.

Sobre las rutinas de este manual.

En este manual las rutinas aparecen incrustadas en el pdf y pueden reconocerse por su distinta tipografía y por estar en recuadros. Sin embargo, por comodidad es recomendable utilizar block de notas, gedit, R scripts, RStudio o TinnR para realizarlas (evitar los procesadores de texto como Word o Writer). Todo el material de este curso también está disponibles como documentos de R Markdown en el repo https://github.com/santiagombv/Bioestadistica
El objetivo de este curso es capacitar al alumno para realizar una rutina de análisis por sí mismo, aplicando conocimientos de modelos lineales, manejos de gráficos y programación.

Instalación de R y de sus paquetes en Windows.

Instalación de R.

Entrar a http://cran.r-project.org/
En download R for Windows - base, seleccionar la última versión disponible de R (3.3.2, "Sincere Pumpkin Patch" actualmente). Guardar el archivo R-3.3.2-win.exe en cualquier parte de la computadora, ejecutarlo y seguir las instrucciones de instalación.

Instalación de paquetes.

Seleccionar Paquetes - Instalar Paquetes (la computadora debe estar conectada a internet).
Seleccionar el espejo CRAN desde donde se bajará el paquete (utilizar uno cercano).
Seleccionar un paquete de la lista despegable. Opciones similares pueden encontrarse en RStudio. Alternativamente utilizar:

```
install.packages("nombre_del_paquete")
```

Carga de paquetes.

Para que un paquete esté disponible en una sesión de trabajo seleccionar Paquetes – Cargar paquetes y elegirlo en la lista desplegable que se abre. Alternativamente puede cargarse escribiendo en la consola principal de R.

```
library(nombre_del_paquete)
```

Actualización de R.

En Windows se debe desinstalar y volver a instalar el programa en cada actualización. Alternativamente utilizar el paquete *installr*

```
install.packages("installr")
library(installr)
updateR() #seguir las instrucciones que aparezcan
```

Instalación de R y de sus paquetes en Ubuntu.

Instalación de R.

Puede instalarse R desde el centro de software de Ubuntu, pero habitualmente es una versión antigua, ya que tarda en actualizarse. Para evitar esto usted debe realizar 3 pasos.

1. Añadir una fuentes de software desde "Configuración del Sistema" - "Software y actualizaciones" - "Otro Software". Seleccionar "Añadir". En la ventana que solicita la línea de APT completa del repositorio colocar:
   deb http://cran.rstudio.com/bin/linux/ubuntu xenial
   En esta línea puede cambiar el espejo CRAN elegido (http://...), por alguno de los listados en http://cran.r-project.org/ Por ejemplo http://mirror.fcaglp.unlp.edu.ar/CRAN/bin/linux/ubuntu corresponde a la Universidad Nacional de la Plata. Puede cambiar también la distribución de Ubuntu empleada (xenial, trusty, saucy, quantal, etc. indicada en la última palabra).
2. Agregar una clave de seguridad al sistema (en caso de que esta opción "no funcione", ver otras posibilidades en http://cran.r-project.org/). Tipear en la terminal:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E084DAB9
```

3. Instalar R desde terminal.

```
sudo apt-get update
sudo apt-get install r-base
```

Instalación de paquetes.

La manera más práctica de hacerlo es instalar RStudio y utilizar su gestor de paquetes. Alternativamente se puede hacer desde terminal.

```
sudo R
install.packages("nombre_del_paquete")
```

Carga de paquetes.

Debe cargarse desde la consola principal de R o Rstudio.

```
library(nombre_del_paquete)
```

Actualización de R.

Al agregar una fuente de software a la computadora, R se actualizará junto a todos los otros programas, automáticamente. Para mantener al día la última versión de R basta con colocar en la terminal.

```
sudo apt-get update
sudo apt-get upgrade
```

Nota 1: en algunas versiones de Windows, el usuario no puede modificar las carpetas contenidas en "Archivos de programas". En ese caso, los paquetes no se guardarán correctamente y se borrarán cada vez que reinicie la computadora. Para evitarlo, localice la carpeta "R" dentro de los Archivos de programa, selecciónela con el botón derecho del mouse, abra la opción Propiedades - Seguridad, y asegúrese de permitir "control total" a todos los usuarios.
Nota 2: para este curso, se requieren los siguientes paquetes extra: *ggplot2* (y todas sus dependencias), *lattice*, *rgl*, *sciplot*, *car*, *arm*, *MuMIn* y *devtools*.
Nota 3. Se recomienda además utilizar la última versión de R por compatibilidad con algunas rutinas.
Nota 4: Para este curso utilizaremos como interfaz RStudio (descargar desde http://www.rstudio.com/).

Práctico 1. Introducción al Lenguaje R

Creación de objetos

R es un lenguaje orientado a objetos: cada comando crea un objeto que debe ser "nombrado" para que permanezca en la memoria utilizando una `<-` y su nombre debe ser "invocado" para que aparezca en pantalla. A pesar de ser un lenguaje de programación, R es comparativamente simple e intuitivo.

Objetos comunes en R.

data.frame	Objeto que contiene nuestros datos. Habitualmente se crea al importar datos externos, aunque pueden ser creados dentro del mismo R. Consiste en una serie de variables (*vectors*) de igual longitud y que pueden ser del mismo tipo o no.
vectors	Colección de datos del mismo tipo (números, letras, etc.) que puede ser creado dentro de R o importado como una columna de un *data.frame*. Existen muchos tipos de vectores, entre ellos: *numeric* consiste de números reales; *integer* consiste de números enteros; *character* contiene letras, nombres, etc. (notar que cada elemento se encuentra entre comillas, por ej. "A1"); *factor* sirve para representar variables categóricas, porta información sobre los niveles del factor (*levels*) y sobre si estos niveles siguen un orden o no.
matrix	Matriz formada por la unión de vectores de un mismo tipo y largo, por un solo vector que es partido en columnas y filas o (más habitualmente) producto de ciertas funciones, por ejemplo *cor*, que construye matrices de correlación.
list	Objeto que compuesto de objetos de distinto tipo y largo.

Cuatro símbolos básicos: # <- ? c

```
# El símbolo # desactiva el espacio a su derecha.
# Es útil para poner aclaraciones en nuestras rutinas.

# El comando más básico: Asignar crea objetos.
n <- 15
n   #escribir el nombre de un objeto es "invocarlo"

# al usar de nuevo el mismo nombre el objeto anterior se pierde ("pisar")
n <- 46 + 12
n

# los caracteres categóricos necesitan ser ingresados con comillas
di <- "A"
di

# Obtener ayudas, el símbolo ?
#si tenemos dudas sobre el funcionamiento de una función
? mean
? lm

# Si desconocemos el nombre de una función para realizar determinada
# tarea, puede realizarse una búsqueda con ?? (en los paquetes activos)
?? "linear models"

# crear vectores. La función concatenar c
vector <- c(1, 2, 3, 4)
vector <- c(1:4)
vector <- c("a", "b", "c", "d")
vector <- c("1", "a", "2", "b")
```

Funciones básicas: creación de vectores, secuencias y matrices

Las funciones de esta sección son las más sencillas de R. Como regla general las funciones siempre deben escribirse acompañadas de paréntesis, que contienen los elementos sobre los cuales la función actuará. Si no se colocan los paréntesis el programa mostrará el código mismo de la función.

```
### vectores ###
# Ya vimos c concatenar
vec <- c(1, 4, 6, 3, 7)
vec

# vectores de repetición. rep(lo_que_queremos_repetir, cuántas veces)
# esta función tiene dos argumentos, separados por una coma
xx <- rep("A", 50)
xx

# secuencias seq()
seq1 <- c(1:50)  # el símbolo : indica desde:hasta
seq1

seq2 <- seq(50) # estructura: desde uno hasta (...)
```

```r
seq2

seq3 <- seq(8,50) # estructura: (desde..., hasta...)
seq3

seq4 <- seq(from = 0, to = 0.99, by = 0.01) # argumentos explícitos
seq4

seq5 <- seq(0, 0.99, 0.01) # argumentos implícitos
seq5

seq6 <- seq(5, 10, length.out=20)
seq6

# combinando vectores de a pares: cbind y rbind
az <- cbind(seq1, xx)
az

za <- rbind(c(1:25), rnorm(25))
za

# construyendo una matriz: matrix()
m <- c(1:20)
matriz1 <- matrix(m, 4, 5) #(vector a usar, filas, columnas)
matriz1

# notar que es exactamente igual a
matriz2 <- matrix(c(1:20), 4, 5)
matriz2
```

Preparación e ingreso de datos: las funciones `read.table` y `read.csv`.

Los datos pueden prepararse con cualquier software para planillas de datos como Excel o LibreOffice Calc. Se recomienda:

Que los nombres de las columnas no tengan espacios ni comiencen con números.
Evitar los símbolos extraños como %, &, ^, ~, ñ, etc.
Utilizar nombres cortos.
Los datos faltantes no deben dejarse en blanco, sino señalarse con NA.

Una vez lista la planilla se recomienda guardarla en formato .txt o .csv (si bien R admite otros formatos). Pueden guardarse en cualquier carpeta del equipo. En el caso de que se escriba cada vez la ruta para encontrar ese archivo, es recomendable ubicar la carpeta cerca de la raíz del equipo (normalmente C:/ en Windows o /home en Ubuntu). En este manual se supondrá que los archivos de datos se han guardado en C:/RD. Puede evitarse escribir la ruta si al abrir R seleccionamos la carpeta que donde los archivos están guardados con la opción Archivo - cambiar dir o usando la función *setwd* (para examinar cual es la carpeta en uso, utilizar *getwd*). Esta opción es muy útil si vamos a abrir varios archivos de datos guardados en el mismo directorio. Finalmente, puede abrirse una ventana de búsqueda para seleccionar el archivo con la opción *file.choose()*.

```r
# la función read.table
# opción 1 con ruta completa
datos <- read.table("C:/RD/peces1.txt", header=TRUE)

# opción 2 seleccionando el directorio en uso previamente desde "Archivo"
# o con setwd
setwd("C:/RD/")
datos <- read.table("peces1.txt", header=TRUE)

# opción 3 Abre una ventana de búsqueda
# desventaja: requiere que el humano piense!
datos <- read.table(file.choose(), header=TRUE)

# los datos ya están guardados, pero si queremos verlos...
datos    #no siempre es buena idea, sobre todo si son muchos

# una forma práctica de ver solo las primeras filas es con head
head(datos)

# Información básica del set de datos
nrow(datos)     #Número de filas
ncol(datos)     #Número de columnas
names(datos)    #Nombre de las columnas
```

Indexación

Estas operaciones se realizan para extraer de un objeto la parte que nos interesa, por ejemplo una columna con una variable de un marco de datos.

> **IMPORTANTE** No usar attach(). Esta función aumenta las probabilidades de confundirse e introducir errores.

```r
# Para seleccionar una columna del marco de datos utilizamos $
datos$grupo

# los corchetes indican el contenido de un conjunto de datos,
# matriz o vector. La coma separa filas de columnas

#columnas
datos[, 1]
datos[, "grupo"]

# grupos de columnas
datos[, 1:3]
datos[, c("grupo", "largo.a")]
names <- c("grupo", "largo.b")
datos[, names]

# filas y datos individuales
datos[2, ]
datos[2, 3]
datos[-2, 2]

# indexando por una condición
```

```r
datos[datos$largo.a > 15, ]
datos[datos$largo.a > 15 & datos$grupo == "A", ]

# Indexación de vectores
vec <- datos$largo.a
vec[1]
vec[-1]
vec[vec > 15]

# omitir todos los NA de una base de datos
datos2 <- na.omit(datos)
datos2

# La mayoría de las funciones cuentan con dos modos alternativos de
# escribirse. En el modo "fórmula" existe un argumento "data" que evita
# indexar los nombres de las columnas usadas.

plot(datos$largo.a, datos$largo.b)      #modo por default: plot(x,y)

plot(largo.b ~ largo.a, data = datos)   #modo fórmula: plot(y~x, data)
```

Modificación y creación de columnas.

```r
# reemplazo (al usar el mismo nombre) una variable numérica por un factor
datos$trat
datos$trat <- as.factor(datos$trat)
datos$trat    # notar la lista de niveles del factor

# crear una nueva columna en una base de datos ya existentes
datos$pob <- c(rep("pob1", 15), rep("pob2", 15))
datos$pob <- as.factor(datos$pob)
datos$log.largo.a <- log(datos$largo.a)
head(datos)

# crear una nueva columna uniendo clasificadores
datos$clave <- paste(datos$pob, datos$trat, sep = ".")
head(datos)

# Usando factor para arreglos más complicados
datos$pais <- factor(datos$pob, levels = c("pob2", "pob1"),
                     labels = c("Bolivia", "Chile"))
head(datos)
```

Subdivisión de conjuntos de datos

```
# Subdivisión de un conjunto de datos
# Símbolos lógicos:
# == (igual)
# != (distinto),
# > (mayor),
# < (menor)
# >= (mayor o igual),
# <= (menor o igual).

dat1 <- subset(datos, datos$grupo == "A")
dat1

dat2 <- subset(datos, datos$grupo == "A" & datos$trat == 2)
dat2

# Crear una lista de bases de datos
ldat <- split(datos, f = datos$grupo)
ldat
ldat[["B"]] # indexacion especial
```

Funciones para extraer información de una columna o de un vector

```
mode(seq1)
class(xx)
length(seq1)
dim(matriz1)
max(datos$largo.a)
which.max(datos$largo.a)
min(datos$largo.a)
which.min(datos$largo.a)
```

Funciones estadísticas básicas

Se detallan abajo algunas como la media, varianza, desvío estándar, etc. En general si cualquiera de ellas se aplica sobre datos que contienen NA el resultado será NA también, por lo cual hay indicarle a la función que los elimine. Una alternativa es eliminar previamente todas las filas con datos faltantes usando la función na.omit.

```
# media
M <- mean(datos$largo.a, na.rm = TRUE)
M

# desvío estándar
S <- sd(datos$largo.a, na.rm = TRUE)
S

# varianza
V <- var(datos$largo.a, na.rm = TRUE)
V
```

```r
# sobre un conjunto de datos, se obtiene una matriz de varianza-covarianza
VC <- var(datos[,c(3,4)], na.rm = TRUE)
VC

# de forma parecida, puede obtenerse una matriz de correlación
CO <- cor(datos[, c(3, 4)], use="complete.obs")  #complete.obs elimina NA
CO

# en cambio, para realizar un test de correlación
cor.test(datos$largo.a, datos$largo.b, method = "pearson")

# resumen de datos
summary(datos)

# una función útil para aplicar una función a un subconjunto de los datos
# es aggregate, con una estructura ligeramente más complicada

MG<-aggregate(datos$largo.a, by=list(datos$grupo), FUN=mean, na.rm=TRUE)
MG
```

Otras funciones útiles y miscelánea

- Editar - Limpiar Consola (Ctrl+L): Borra la consola de R, pero no remueve los objetos de la memoria.
- Detener el cálculo actual. STOP o presionar Esc.
- Misc - Listar Objetos: Muestra los objetos guardados en la memoria.

```r
ls()
```

- Misc - Remover todos los objetos: Elimina todos los objetos guardados en la memoria.

```r
rm(list = ls(all = TRUE))
```

QUÉ SIGUE: Manipular bases de datos es un campo en desarrollo. El paquete *dplyr* de Hadley Wickham es altamente recomendable (https://cran.r-project.org/web/packages/dplyr/). Para una excelente introducción a *dplyr* consultar http://www.dataschool.io/dplyr-tutorial-part-2/ .

Práctico 2. Modelos Lineales Simples

Caso 1.

Se examina la relación entre la altura y la capacidad pulmonar. Los datos se encuentran en el archivo fumadores.txt.

```
fum <- read.table("C:/RD/fumadores.txt", header = TRUE)
#  fum <- read.table(file=file.choose(), header = TRUE)
fum

#inspección de datos 1: gráfico tipo Cleveland
plot(fum$ca.pulm)
plot(fum$alt)

#inspección de datos 2: gráfico bivariado
plot(fum$alt, fum$ca.pulm)

#modelo lineal (regresión)
fit<-lm(ca.pulm ~ alt, data=fum)

##componentes del objeto fit (algunos)
fit
fit$coefficients
fit$residuals[1:20] # sólo los primeros 20
fit$fitted[1:20] # sólo los primeros 20

##resúmenes de información
summary(fit)
anova(fit)

#diagnósticos gráficos. Layout dividirá la ventana gráfica en lo que #indique la matriz (4 en este caso
layout(matrix(1:4,2,2))
plot(fit)
layout(1)

#diagnósticos numéricos
shapiro.test(fit$residuals)
```

Caso 2.

Muertes de mosquitos en respuesta a distintos insecticidas. Datos de ejemplo cargado en R: InsectSprays.

```
data(InsectSprays)
head(InsectSprays)

plot(sqrt(count) ~ spray, data = InsectSprays)

#utilizando la function lm
fit.spray1 <- lm(sqrt(count) ~ spray, data = InsectSprays)
anova(fit.spray1)
summary(fit.spray1)
#utilizando la function aov
```

```r
fit.spray2 <- aov(sqrt(count)~spray, data = InsectSprays)
summary(fit.spray2)

#diagnósticos gráficos
layout(matrix(1:4,2,2))
plot(fit.spray2)
layout(1)

#diagnósticos numéricos
shapiro.test(resid(fit.spray2))
bartlett.test(resid(fit.spray2)~InsectSprays$spray)

#Test de Tukey
#notar que trabaja sobre un objeto aov, no sobre un objeto lm
tuk<-TukeyHSD (fit.spray2)
tuk
```

Ejercicios

1. Se intenta probar si el éxito reproductivo (medido como polinarios exportados) de la orquídea Cyclopogon elatus aumenta en flores con nectarios más profundos. El archivo cyclop.txt contiene las variables de interés nectario y pol.exp. Realizar gráficos exploratorios, ajustar un modelo de regresión lineal, extraer la tabla de parámetros y la tabla ANOVA y realizar gráficos de diagnóstico. ¿Qué opina de transformar los datos para satisfacer los requisitos del modelo lineal?
2. Determinar si existen diferencias en el peso de pecaríes según el mes de su captura: febrero, mayo, agosto y noviembre, con los datos del archivo pecaries.txt. Explorar los datos gráficamente, realizar un análisis de la varianza, obtener la tabla de parámetros y la tabla ANOVA. Realizar diagnósticos gráficos y numéricos (pruebas de Shapiro-Wilks y Bartlett), y realizar un test de Tukey.

Práctico 3. Modelos Lineales Múltiples I

Caso 1.

Se estudió el efecto del número de parejas sexuales sobre la longevidad de moscas de la fruta macho. Los tratamientos fueron: A = control (sin hembras), B = control (una hembra no receptiva al día), C = control (8 hembras no receptivas al día), D = una hembra virgen al día, E = 8 hembras vírgenes al día. Ya que la longevidad puede estar relacionada con el tamaño del insecto se tomó el diámetro del tórax como covariable. Los datos se encuentran en el archivo moscas.txt.

```
dat <- read.table("C:/RD/moscas.txt", header = TRUE)
#  dat <- read.table(file = file.choose(), header = TRUE)
plot(dat$torax)
plot(dat$vida)
plot(dat$trat,dat$vida)
plot(dat$torax,dat$vida)

#diferentes sumas de cuadrados
fit1.a <- lm(vida ~ trat*torax, data = dat)
fit1.b <- lm(vida ~ torax*trat, data = dat)
anova(fit1.a) #tipo I
anova(fit1.b)

library(car)
Anova(fit1.a, type = "II")
Anova(fit1.b, type = "II")
Anova(fit1.a, type = "III")
Anova(fit1.b, type = "III")

summary(fit1.a)

#selección del modelo
fit2 <- lm(vida ~ trat + torax, data = dat)
Anova(fit2, type = "II")
fit3 <- lm(vida ~ trat, data =dat)
Anova(fit3, type = "II")

layout(matrix(1:4, 2, 2))
plot(fit3)
layout(1)
```

Ejercicios

1. Se busca un modelo que prediga la biomasa (peso.seco) de las almejas en función de su largo máximo (largo). Debido al ciclo de vida de las almejas, la relación biomasa-largo puede variar a lo largo del año, por lo que se ha tomado el mes de colecta (mes) como covariable. Los datos para el ejercicio se encuentran en el archivo almejas.txt.
2. Se intenta determinar el efecto de la estación (IP= invierno/primavera vs. VO= verano/otoño) y la densidad de adultos (A=8, B=15, C=30, D=45 en 225 cm2) sobre la producción de huevos de bivalvos. Los datos para el ejercicio se encuentran en el archivo bivalvos.txt.

Práctico 4. Modelos lineales múltiples II

Caso 1.

Se intenta realizar un modelo que prediga el contenido de azúcar en el néctar en flores de Salvia polystachia, a partir de 7 variables morfológicas medidas. Los datos se encuentran en el archivo s_poly.txt.

```
dat <- read.table("C:/RD/s_poly.txt", header = TRUE)
#  dat <- read.table(file = file.choose(), header = TRUE)

# exploración de los datos
layout(matrix(1:8,2,4))
plot(dat$azucar); plot(dat$largo.l.tot); plot(dat$largo.tubo)
plot(dat$caliz.sup); plot(dat$caliz.med); plot(dat$caliz.inf)
plot(dat$labio.sup); plot(dat$labio.inf)
layout(1)

which.max(dat$azucar)
dat <- dat[-101, ]

# exploración bivariada de los datos
pairs(dat)

###DETECCIÓN DE LA COLINEALIDAD
library(car)

#A) Gráficos SPLOM
scatterplotMatrix(~ largo.l.tot + largo.tubo + caliz.sup + caliz.med +
   caliz.inf + labio.sup + labio.inf, #variables a graficar
   reg.line=lm,            #añadir rectas de regresión a los gráficos
   smooth=TRUE,            #añadir curvas suavizadas a los gráficos
   diagonal = 'density',   #gráficos de la diagonal
   data = dat)             #conjunto de datos

#B) Matrices de correlación
CORR <- cor(dat[, c("largo.l.tot", "largo.tubo", "caliz.sup",
   "caliz.med", "caliz.inf", "labio.sup", "labio.inf")],
   use = "complete.obs")
CORR

#C)Factores de inflación de la varianza
# primero construimos el modelo completo, sin interacciones

fit <- lm(azucar ~ largo.l.tot + largo.tubo + caliz.sup + caliz.med +
           caliz.inf + labio.sup + labio.inf, data = dat)
vif(fit)

#Decisión: usar el criterio de r < 0.7 y parcialmente los vif,
#descartamos el cáliz inferior y construimos un nuevo modelo

fit2 <- lm(azucar ~ largo.l.tot + largo.tubo + caliz.sup + caliz.med + labio.sup + labio.inf, data = da
summary(fit2)
anova (fit2)
layout(matrix(1:4, 2, 2))
```

```r
plot(fit2)
layout(1)

#revisando su colinealidad nuevamente
vif(fit2)

###SELECCIÓN DE MODELOS
##anova puede usarse para comparar dos modelos anidados
fit2<-lm(azucar ~ largo.l.tot + largo.tubo + caliz.sup + caliz.med + labio.sup + labio.inf, data=dat)
fit3<-lm(azucar ~ largo.tubo + caliz.sup + caliz.med + labio.sup + labio.inf, data=dat) #sin largo.l.to

anova(fit3, fit2)

##drop1 borra una variable del modelo a la vez y compara con el completo
drop1(fit2, test="F")

## stepAIC (MASS) realiza una búsqueda automática del mejor modelo
library(MASS)
stepAIC(fit2, direction ="both") #para usar AIC

#para usar BIC cambio los grados de libertad de la penalización
n<-nrow(dat)
stepAIC(fit2, direction="both", k=log(n))#ajustar el modelo final

### uso básico de MuMIn
library(arm)
library(MuMIn)

global <- lm(azucar ~ largo.l.tot + largo.tubo + caliz.sup + caliz.med + labio.sup + labio.inf, data = d

std.model <- standardize(global, standardize.y = FALSE)

set <- dredge(std.model)
set

top.mod <-get.models(set, subset = delta<2)
top.mod

AVG<-model.avg(top.mod)
summary(AVG)
```

Ejercicios.

1. Los datos del archivo Loyn.txt corresponden a un estudio donde la densidad de aves (ABUND) se midió en 56 parches del sur de Victoria, Australia. El objetivo del estudio es determinar cuáles características del hábitat explican esa abundancia. Para ello se midió: tamaño del parche (AREA), distancia al parche más cercano (DIST), distancia al parche grande más cercano (LDIST), altura s.n.m. (ALT), años de aislamiento (YR.ISOL) y un índice de pastoreo (GRAZE). Explorar gráficamente los datos y examinar si las variables AREA, DIST Y LDIST requieren una transformación. Examinar la colinealidad de las variables y seleccionar un modelo adecuado.
2. Los datos del archivo fumadores.txt corresponden a un estudio donde se intenta explicar la capacidad pulmonar en función de dos variables continuas (edad y altura) y dos variables discretas (fumar = si/no;

sexo = m/f). Observar cuidadosamente la colinealidad entre las variables (incluyendo las variables discretas).

Práctico 5. Manejo de gráficos y representación gráfica de modelos lineales.

Comandos gráficos básicos

- Comandos gráficos de alto nivel: abren una ventana gráfica y crean un gráfico nuevo completo. Modificando sus opciones es posible cambiar colores, tamaños, tipo de fuente y otras características.
- Comandos gráficos de bajo nivel: agregan más información a un gráfico ya existente, como puntos extra, líneas, funciones y leyendas.
- Comandos gráficos interactivos: permiten agregar o extraer información de un gráfico existente, usando el mouse.

Uso de la función plot.

```
datos <- read.table("C:/RD/cyclop.txt", header = TRUE)
#  datos <- read.table(file=file.choose(), header = TRUE)

# UNA VARIABLE
plot(datos$nectario) #gráfico tipo Cleveland
plot(~datos$nectario)#Stripchart

#con los datos ordenados de menor a mayor
plot(sort(datos$nectario))

# cambiando el tipo de gráfico
# probar los diferentes "type": p, l, b, o, h, s, S
plot(sort(datos$nectario), type = "h")

#agregamos una línea horizontal señalando la posición de la media
#usando la función de bajo nivel abline
m.nec <- mean(datos$nectario, na.rm = TRUE)
abline(h = m.nec)

# DOS VARIABLES
plot(datos$nectario, datos$pol.exp)       #forma plot(x,y)
plot(pol.exp ~ nectario, data=datos)      #forma plot(y~x)

#manipulación de los PARAMETROS GRÁFICOS
plot(datos$nectario, datos$pol.exp,
     pch= 16,                             #tipo de símbolo
     col= "blue2",                        #color del símbolo
     xlab= "profundidad del nectario",    #nombre eje x
     ylab= "polinios exportados")         #nombre eje y

#agregando una línea de regresión
fit<-lm(datos$pol.exp ~ datos$nectario)
summary(fit)
abline(fit, lty = 3, lwd = 2)
```

Devices

Colocando ?Devices obtenemos la lista de formatos gráficos disponibles en R. No todos los formatos son accesibles en todos los sistemas.
Los gráficos pueden guardarse (en Windows) con el botón derecho del mouse sobre el gráfico o utilizando Archivo - guardar como (la ventana gráfica debe estar en primer plano). Lo mismo puede hacerse utilizando comandos (veremos los principales) con la ventaja de un control más fino sobre la forma final del gráfico.

```
?Devices

#revisar donde se guardaran los gráficos
getwd()
#puede cambiarse con setwd()

# JPG
jpeg(file = "mi_grafico.jpg",
     width = 12, height = 10,  #ancho y alto en lo que indique units
     units = "cm",             #unidades (pixeles por defecto)
     quality = 95,             #grado de no-compresión
     res = 300)                #resolución en puntos por pulgada
plot(pol.exp ~ nectario, data=datos)  #comandos gráficos
dev.off()                      #cerrar el gráfico (importante!!)

# PDF
pdf(file = "mi_grafico.pdf",
    width = 7, height = 6,#ancho y alto en pulgadas (=2.54cm)
    paper = "a4")         #tamaño del papel, por defecto igual al gráfico
plot(pol.exp ~ nectario, data=datos)  #comandos gráficos
dev.off()                             #cerrar el gráfico

# EPS
postscript(file = "mi_grafico.eps",
    width = 7, height = 6,#ancho y alto en pulgadas (=2.54cm)
    paper = "a4",         #tamaño del papel, por defecto igual al gráfico
    horiz = TRUE)         #horizontal (FALSE) o vertical (TRUE)
plot(pol.exp ~ nectario, data=datos)  #comandos gráficos
dev.off()                             #cerrar el gráfico

# SVG
svg(file = "mi_grafico.svg",
    width = 7, height = 6)   #ancho y alto en pulgadas (=2.54cm)
plot(pol.exp ~ nectario, data=datos)  #comandos gráficos
dev.off()                             #cerrar el gráfico
```

Otros gráficos sencillos con variables continuas.

```
# histogramas
hist(datos$nectario, col = "orange1", main= "histograma",
     xlab = "nectario", ylab = "frecuencia")

#QQplots
qqnorm(datos$nectario)
```

División de la ventana gráfica.

```
#Básica con la función layout
layout(matrix(1:4, 2, 2))
hist(datos$nectario, col= "#CAFF70")
hist(datos$flores, col= "#BCEE68")
hist(datos$frutos, col= "#A2CD5A")
hist(datos$pol.exp, col= "#6E8B3D")
layout (1)

#Por defecto, layout() divide el dispositivo en dimensiones regulares:
#esto se puede modificar con las opciones widths y heights.
m <- matrix(1:4, 2, 2)
layout(m, widths=c(1, 2),
heights = c(2, 1))
layout.show(4)
hist(datos$nectario, col = "#CAFF70")
hist(datos$flores, col = "#BCEE68")
hist(datos$frutos, col = "#A2CD5A")
hist(datos$pol.exp, col = "#6E8B3D")
layout(1)

#división de la ventana usando las opciones de par
par(mfrow = c(2, 2))
hist(datos$nectario, col = "#CAFF70")
hist(datos$flores, col = "#BCEE68")
hist(datos$frutos, col = "#A2CD5A")
hist(datos$pol.exp, col = "#6E8B3D")

par(mfcol = c(1, 4))
hist(datos$nectario, col = "#CAFF70")
hist(datos$flores, col = "#BCEE68")
hist(datos$frutos, col = "#A2CD5A")
hist(datos$pol.exp, col = "#6E8B3D")

#cerrar la ventana gráfica para desactivar par o
par(mfcol=c(1,1)) #deshacer lo anterior

#revisar las opciones gráficas de par
?par
```

Gráficos de cajas.

```
peces <- read.table("C:/RD/peces.txt", header = TRUE)
#  peces <- read.table(file = file.choose(), header = TRUE)
peces$Species <- as.factor(peces$Species)

#dos formas alternativas de hacer box-plots
plot(peces$Species, peces$Weight)
boxplot(Weight ~ Species, data = peces, col = "light blue")
```

Interacciones entre variables categóricas.

```
bival <- read.table("C:/RD/bivalvos.txt", header = TRUE)
#  bival <- read.table(file = file.choose(), header = TRUE)

# modelo
fit <- lm(huevos ~ densidad*estacion, data = bival)
anova(fit)

library(sciplot)

#GRÁFICOS DE BARRAS
bargraph.CI(x.factor = densidad, #factor (eje x)
            response = huevos,   #respuesta (eje y)
            group = estacion,    #factor optativo (distintos colores)
            legend = TRUE,
            data = bival)

#GRAFICOS DE INTERACCIÓN
lineplot.CI(x.factor = densidad, response = huevos, group = estacion,
            data = bival, legend = TRUE,
            col = c("red", "black"),  #modificación de varios parámetros
            pch = c(1,20),
            xlab = "densidad",
            ylab = "número de huevos",
            trace.label = "estación")
```

Gráficos a partir de predichos.

```
datos <- read.table("C:/RD/cyclop.txt", header = TRUE)
#  datos <- read.table(file = file.choose(), header = TRUE)

fit.c <- lm(pol.exp ~ flores + I(flores^2), data = datos)
summary(fit.c)

#A partir de los coeficientes
B <- coef(fit.c)
o <- order(datos$flores) #obtenemos el orden de x
Y1 <- B[1] + B[2]*datos$flores[o] + B[3]*datos$flores[o]^2
plot(datos$flores, datos$pol.exp)
points(datos$flores[o], Y1, type = "l")
#A partir de los predichos.
Y2 <- predict(fit.c, datos[o, ])#los datos originales pueden reemplazarse
plot(datos$flores, datos$pol.exp)
points(datos$flores[o], Y2, type = "l")

#A partir de los predichos, con un set de datos nuevo y "suave"
FLO2<-seq(min(datos$flores, na.rm = TRUE),  #desde el mínimo
          max(datos$flores, na.rm = TRUE),  #hasta el máximo
          length.out = 200)                 #nueva longitud
Y3 <- predict(fit.c, newdata = data.frame(flores=FLO2))
```

```r
plot(datos$flores, datos$pol.exp)
points(FLO2, Y3, type = "l")
```

Interacciones de variables continuas: superficies.

Se necesitan datos uniformemente espaciados para construir superficies. Existen alternativas no paramétricas que "suavizan" los datos para darnos una idea mejor de la forma "real" de una superficie, en este caso lo haremos "a mano" usando un modelo lineal.

```r
datos <- read.table("C:/RD/cyclop.txt", header = TRUE)
#  datos <- read.table(file = file.choose(), header = TRUE)

fit <- lm(formula = pol.exp ~ flores * nectario + I(flores^2) + I(nectario^2), data = datos)

r.nec <- range(datos$nectario, na.rm = TRUE)
NEC <- seq(r.nec[1], r.nec[2], length.out = 100)
r.flo <- range(datos$flores, na.rm=T)
FLO <- seq(r.flo[1], r.flo[2], length.out = 100)

#Construcción de una grilla
grilla <- expand.grid(flores=FLO, nectario=NEC)

#predicción del eje Z
Z1<-predict(fit, newdata = grilla)
Z2<-matrix(Z1, 100, 100)
#GRAFICOS 3 D: DISTINTAS OPCIONES

#2D, con curvas de nivel (tipo "mapa topografico")
contour(x=FLO, y=NEC, z=Z2, col = "Black")

#2D, con un gradiente de color y curvas de nivel
filled.contour(x=FLO, y=NEC, z=Z2, color = rainbow)

#crear paletas de color
grises <- grey(seq(0, 1, 0.1)) #10 tonos de gris
varios <- rainbow(10, start=0.1, end=0.5)

#2D, con un gradiente de color y  sin curvas de nivel
image(x = FLO, y = NEC, z = Z2, col = grises)

#Superposición de image, contour y points
image(x=FLO, y=NEC, z=Z2, col = varios, xlab = "número de flores", ylab = "nectario")
contour(x = FLO, y = NEC, z = Z2, col = "red", add = TRUE)
points(x = datos$flores, y = datos$nectario, col = "red", pch = 19)

#3D, con rejilla y en perspectiva
persp(x = FLO, y = NEC, z = Z2)

#manejo de los parámetros gráficos (sólo algunos)
persp(x = FLO, y = NEC, z = Z2,
      border = "slateblue4",       #border: color de la rejilla
      col = "aquamarine2",         #col: color de la superficie
      xlab = "nectario",           #xlab, ylab, zlab: etiquetas ejes
```

```
      ylab = "número de flores",
      zlab = "polinarios",
      theta = -45,                  #theta: ángulo de giro horizontal
      phi = 20,                     #phi: ángulo de rotación vertical
      ticktype = "detailed",        #para que ponga el valor de los ejes
) -> res                          #guardo la "forma" del gráfico

#agregar de los puntos observados con la funcion trans3d
pru<-trans3d(x = datos$flores, y = datos$nectario, z = datos$pol.exp, pmat = res)
points(pru, col = "royalblue4", pch = 20)
```

Ejercicios.

1. Usar los datos de s_poly.txt para hacer gráficos bivariados, utilizar comandos gráficos de bajo nivel para darle colores y cambiar los símbolos y agregar una recta de regresión. Guarde el gráfico obtenido como pdf.
2. Represente gráficamente el resultado del ejercicio 2 del práctico 2:Determinar si existen diferencias en el peso de pecaríes según el mes de su captura: febrero, mayo, agosto y noviembre, con los datos del archivo pecaries.txt.

Práctico 6. Otros paquetes gráficos: *lattice* y *ggplot2*.

Gráficos del paquete *lattice*.

El paquete lattice ofrece flexibilidades especiales para representar interacciones y particularmente datos multivariados. Sus funciones son independientes de las funciones gráficas principales y se caracterizan por tener una gramática ligeramente distinta.

```
library (lattice)

fum <- read.table("C:/RD/fumadores.txt", header=T)
#  fum <- read.table(file=file.choose(), header=TRUE)

#gráficos bivariados
# Notar que ingresar los datos como una fórmula es más eficiente
xyplot(ca.pulm ~ alt, data = fum)
xyplot(ca.pulm ~ alt | edad, data = fum)
xyplot(ca.pulm ~ alt | fuma, data = fum)
xyplot(ca.pulm ~ alt | fuma*sexo, data = fum)
xyplot(ca.pulm ~ alt, groups = fuma, data = fum)

#ayuda de la funcion básica xyplot
?xyplot

#manipular alguna de sus opciones...
xyplot(ca.pulm ~ alt, groups = fuma, data = fum, xlab = "altura",
       ylab = "capacidad\npulmonar", col = c("red", "black"), pch = 19,
       lwd = 2, key = list(text = list(c("no fumadores", "fumadores")),
                           space = "bottom",
                           points = list(pch = 19, col = c("red", "black"))))

#GRÁFICOS DE CAJAS
bwplot(ca.pulm ~ fuma | sexo, data = fum)

#HISTOGRAMAS Y DIAGRAMAS DE DENSIDAD
histogram(~ ca.pulm, data = fum)
densityplot(~ ca.pulm | fuma, data = fum)
```

Gráficos del paquete *ggplot2*

Este paquete ofrece gráficos elegantes e intuitivos. Sin embargo, su escritura posee una gramática propia (una especie de "dialecto" dentro de R), por lo que requiere cierto tiempo hasta lograr su aprendizaje. La ayuda básica para *ggplot2* pude hallarse en la página http://docs.ggplot2.org/current/
Un gráfico de *ggplot2* debe contar de al menos estos tres elementos:

- ggplot: función principal donde se especifica el set de datos y se crea el gráfico.
- aes: Especifica las variables y además colores, forma, transparencia, tipo de línea, etiquetas de los ejes, etc.
- geoms: objetos geométricos como puntos, barras, líneas, etc. En la práctica indica el tipo de gráfico a realizar.

```
fum <- read.table("C:/RD/fumadores.txt", header=T)
#  fum <- read.table(file=file.choose(), header=TRUE)
```

```r
library(ggplot2)

#utilizando ggplot
ggplot(data = fum, aes(x = edad, y = ca.pulm)) + geom_point()

#notar como los gráficos se van acumulando
g1 <- ggplot(data = fum, aes(x = edad, y = ca.pulm))
g2 <- g1 + geom_point()
g2

#modificando algunos detalles comunes
g1 <- ggplot(data = fum, aes(x = edad, y = ca.pulm, color = fuma))
g2 <- g1 + geom_point(size=3, aes(shape=sexo))
g3 <- g2 + xlab("Edad")
g4 <- g3 + ylab("Capacidad Pulmonar")
g5 <- g4 + theme_bw()
g5

#modificando los colores
g1 <- ggplot(data = fum, aes(x = edad, y = ca.pulm, color = fuma))
g2 <- g1 + scale_colour_manual(values=c("red", "black"))
g3 <- g2 + geom_point(size=3)
g3

#modificando los colores con una escala continua
g1 <- ggplot(data = fum, aes(x = edad, y = ca.pulm, color = alt))
g2 <- g1 + geom_point(size=3)
g2

#modificando tamaños con una escala continua
g1 <- ggplot(data = fum, aes(x = edad, y = ca.pulm, size = alt, color = fuma))
g2 <- g1 + geom_point()
g2

#Faceting (división del la ventana gráfica como en lattice)
g1 <- ggplot(data = fum, aes(x = edad, y = ca.pulm, color = fuma))
g2 <- g1 + geom_point(size=3) + theme_bw()
g3 <- g2 + facet_grid(. ~ sexo)
g3

g1 <- ggplot(data = fum, aes(x = edad, y = ca.pulm, color = fuma))

g2 <- g1 + geom_point(size = 3) + theme_bw()
g3 <- g2 + facet_grid(fuma ~ .) + scale_color_manual(values = c("red", "blue"))
g3

#agregar líneas de regresión
g1 <- ggplot(data = fum, aes(x = edad, y = ca.pulm, color = fuma))
g2 <- g1 + geom_point(size = 3) + theme_bw()
g3 <- g2 + geom_smooth(method = "lm")
g3

#gráficos de cajas
```

```r
g1 <- ggplot(data = fum, aes(x = fuma, y = ca.pulm))
g2 <- g1 + geom_boxplot()
g2

#histogramas
g1 <- ggplot(data = fum, aes(x = ca.pulm))
g2 <- g1 + geom_histogram(fill="red")
g2
```

Ejercicios

1. Utilizar los datos de turnera.txt para hacer una regresión con dummy para el polen.tot en función del morfo y del área del pétalo. Representar los resultados utilizando lattice (sin líneas de regresión).
2. Utilizando los mismos datos realice gráficos de densidad para la altura del estambre (alt.estambre) según el morfo, utilizando lattice y ggplot2.
3. Utilizar los datos de Bahamas2.txt para construir una superficie en la cual el logaritmo de la densidad del pez loro (Parrot) se encuentre en función de la riqueza de especies de coral (CoralRichness) y de la cobertura de coral (CoralTotal).

Práctico 7. Programación y construcción de funciones

Introducción.

Las herramientas de programación en R son usadas con dos fines principales:
* Construir funciones que realicen tareas no contempladas en un paquete de R, o combinar funciones existentes para dar origen a otras nuevas.
* Reducir el número de operaciones que debe realizar el operador humano, por ejemplo al hacer simulaciones o al aplicar una misma función a numerosos sets de datos.

Construcción de funciones.

Se realiza mediante la función *function*. Esta función requiere de dos grupos de argumentos: aquellos sobre los cuales esa función actuará (los datos, parámetros a usar, etc.) que se introducen entre paréntesis, y las operaciones a realizar con esos argumentos, que se introducen entre llaves.

Caso 1.

El índice de diversidad de Shannon para un sitio se define por la fórmula

$$H' = -\sum p_i ln(p_i)$$

Donde S es el número de especies, n_i es el número de individuos de la especie i, N es el número total de individuos en el sitio y p_i es, por tanto, la abundancia relativa de cada especie. Primero mostraremos el desarrollo para calcular H' en un conjunto de datos consistentes en un vector donde cada valor corresponde al número de individuos de una especie diferente. Luego, construiremos una función para que la fórmula utilice cualquier set de datos.

```
DAT <- c(13, 2, 4, 0, 36, 3, 2, 7, 1, 11)  # datos originales (inventados)
dat <- (DAT[DAT>0])                        # elimino los 0 (por los log)
N <- sum(dat)                              # total de individuos
p <- dat/N                                 # abundancias relativas
H <- -sum(p*log(p))
H                                          # resultado

# construcción de la función
# notar las similitudes y diferencias con los pasos de arriba.
shann <- function(x) {
  z <- (x[x>0])
  N <- sum(z)
  p <- z/N
  H <- -sum(p*log(p))
  H                       # la última línea es lo que mostrará la función
}

# probamos la función con el set de datos original
shann(DAT)

# probamos la función con el otro set de datos
Y <- c(67, 8, 1, 6, 8, 5, 9, 7, 10, 1, 0, 1, 2,0, 2, 1)
shann(Y)
```

Funciones de simulación y muestreo.

En muchos casos en estas rutinas usamos datos simulados utilizando, por ejemplo, *rnorm*. Estas funciones forman una familia donde la primera letra informa si son una distribución de probabilidad (p), de densidad (d) o una muestra al azar (r) y la segunda parte informa sobre la distribución (norm = normal, unif = uniforme, poiss = poisson, etc.).

```
# simulación de datos
X <- rnorm(100, mean = 4, sd = 0.5)
hist(X)

# probabilidad de observar el valor z < 5.4 bajo la distribución simulada arriba
pnorm(5.4, mean = 4, sd = 0.5)

# valor del percentil 75 bajo la misma distribución
qnorm(0.75, mean = 4, sd = 0.5)
```

Por otra parte, la función básica de muestreo en R es *sample*. En el caso de que queramos que otro usuario obtenga exactamente los mismos resultados que nosostros en una simulación debemos fijar la "semilla" con *set.seed()*.

```
# Exploración de las utilidades de sample
AA <- c(1:10)
AA

#sólo cambio el orden: permutación
sample(AA, size = 10, replace = FALSE)

#muestreo aleatorio sin remplazo
sample(AA, size = 5, replace = FALSE)

#muestreo aleatorio con remplazo
sample(AA, size = 5, replace = TRUE)

#idem, pero de igual n que la muestra original, base del bootstrap
sample(AA, size = 10, replace = TRUE)

?sample
```

Programación con R

Una ventaja de R es la posibilidad de programar una serie de análisis para que se ejecuten de manera sucesiva, sin necesidad de muchos conocimientos técnicos sobre programación, ya que el lenguaje de R es comparativamente sencillo. Una manera de realizar una serie de funciones repetitivas es mediante bucles ("loops"). Las principales funciones que realizan bucles son *for* e *if*. Para evitar realizar bucles pueden utilizarse las funciones de la serie *apply*.

for posee la estructura general: para (cada elemento en un vector) {realizar x acción}. Si necesitamos que los resultados de la acción sean guardados, es necesario crear un vector vacío antes de ejecutar la función y luego rellenarlo con el resultado del bucle.

```
Dat <- runif(1000)
vec <- numeric(length=50)
vec                          # vector vacío antes del loop
for(i in 1:50) {vec[i] <- mean(sample(Dat, 20))}
```

```
vec                          # vector luego de aplicar el loop
plot(density(vec))               #teorema central del límite
```

if agrega una condición: si (condición) entonces hacer X, en caso contrario hacer Y.

```
# construir una distribución normal truncada
X1 <- rnorm(1000, mean= 3, sd=5)
X2 <- numeric(1000)
for (i in 1:1000) if (X1[i] < 0) X2[i] <- NA else X2[i] <- X1[i]
hist(X2)
```

apply aplica una función a las columnas o filas de una matriz. Por ejemplo, para obtener la media de cada columna de una matriz de 5 columnas y 40 filas, con números generados al azar desde una distribución normal. Existen funciones similares como *lapply*, *tapply*, *sapply* (ver ayuda de la función *apply* para conocer la utilizadad de cada una).

```
m <- rnorm(200)
X <- matrix(m, 40, 5)

apply(X, 2, mean)
apply(X, 1, mean)
```

Siempre que sea posible, es preferible aplicar una función *apply* (o álgebra sencilla) en vez de un bucle hecho con *for*.

```
# ejemplo 1
A <- c(8, 23, 11, 10)
B <- c(15, 3, 2, 1)

#con loop (MAL)
C <- numeric(4)
for(i in 1:4) C[i] <- A[i] + B[i]
C

# vectorizado (BIEN)
C <- A + B
C

# ejemplo 2
m <- rnorm(200)
X <- matrix(m, 40, 5)

# con loop (MAL)
med <- numeric(5)
for(i in 1:5) med[i] <- mean(X[, i])
med

#vectorizado (BIEN)
med<-apply(X, 2, mean)
med
```

Ejercicios

1. Construir una función que calcule el error estándar de la media y calcular el CV de todas las variables en el archivo s_poly.txt (guardándolos en un solo vector).

$$S.E. = \frac{s}{\sqrt{n}}$$

2. Utilizando la siguiente tabla de presencia/ausencia de especies en dos sitios.

Especie	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
Sitio 1	0	0	0	1	1	0	1	1	1	1	0	0	0	1	1
Sitio 2	1	0	0	0	1	0	1	1	1	0	0	0	0	0	1

Construir una función para calcular el índice de Sørensen para datos de presencia/ausencia, según la fórmula

$$Q_s = \frac{2C}{A + B}$$

Donde A y B es el número de especies de los sitios 1 y 2, respectivamente, y C es el número de especies compartidas por los sitios 1 y 2. Probar la eficacia de la función con otro set de datos.

Práctico 8: Modelos Randomizados y Modelos Nulos

Caso 1. Modelos lineales randomizados.

Se intenta conocer si la cantidad de grasa en la leche de vaca es influida por su raza (Ayrshire, Canadian, Guernsey, Holstein-Fresian o Jersey). Se plantea en primer lugar un modelo lineal general. Ante la violación de los supuestos del modelo se realizará un modelo randomizado construyendo una tabla ad-hoc de pseudo-F.

```
vacas <- read.table("C:/RD/vacas.txt", header = TRUE)
#  vacas <- read.table(file = file.choose(), header = TRUE)

fit <- lm(fat ~ breed, data = vacas)
summary(fit)
anova(fit)
layout(matrix(1:4, 2, 2))
plot(fit)  #examinar violación de supuestos
layout(1)

#Guardar las F observadas
Fobs <- anova(fit)[1, 4]
Fobs

#construir una function que repita los pasos del modelo
rd.aov <- function(Y, X){
  s.Y <- sample(x = Y, size = length(Y), replace = F)   #"reshuffling"
  fit <- lm(s.Y ~ X)                    #modelo lineal
  res <- anova(fit)[1, 4]               #extracción de las F
  c(res)                                #mostrar: un vector de F
}

#Probar la función varias veces (debe dar distinto resultado)
rd.aov(Y = vacas$fat, X = vacas$breed)
rd.aov(Y = vacas$fat, X = vacas$breed)
#Realizar réplicas para obtener la distribución de pseudo F
pseudoF <- replicate(1000, rd.aov(Y = vacas$fat, X = vacas$breed))
PF <- t(pseudoF)    #transposición para que cada F quede en una columna

#histograma para ver los resultados simulados y los observados
hist(PF)
abline(v = Fobs, col = "red")

hist(PF, xlim=c(0, 51))
abline(v = Fobs, col = "red")

#valores P
P.breed <- length(PF[PF >= Fobs]) / length(PF)
P.breed
#como nuestra simulación llegó hasta mil
#no podemos asegurar que sea 0, sino P<0.001
```

Caso 2: Un Monte Carlo sencillo.

Se intenta probar si la distribución espacial de árboles en un área de 4 km2 se aparta de una distribución aleatoria uniforme.

```
esp <- read.table("C:/RD/espMC.txt", header = TRUE)
#  esp <- read.table(file = file.choose(), header = TRUE)
plot(y ~ x, data=esp)

#calcular las distancias euclídeas entre las filas de la base de datos
#como parámetro observado se elige la media  de todas las distancias

Dobs<-mean(dist(esp))       #revisar la ayuda de la función dist
Dobs

#Realizamos una simulación de la distribución de los árboles según una #distribución uniforme
X.azar <- runif(n = nrow(esp), min = 0, max = 2)
Y.azar <- runif(n = nrow(esp), min = 0, max = 2)
plot(X.azar, Y.azar)
Dazar <- mean(dist(cbind(X.azar, Y.azar)))
Dazar

#Repetimos la simulación 1000 veces y guardamos los resultados
Ds <- numeric(length(1000))
for (i in 1:1000) {
  X.azar <- runif(n=24, min=0, max=2)
  Y.azar <- runif(n=24, min=0, max=2)
  Ds[i] <- mean(dist(cbind(X.azar, Y.azar)))
}

#visualizamos los resultados con un histograma
hist(Ds)
abline(v = Dobs, col = "red")
#calcular el valor P
P <- length(Ds[Ds < Dobs]) / length(Ds)
P
```

Ejercicios

1. Los datos del archivo nidos.txt contienen la cantidad de nidos de cierta especie de ave (nidos) presentes en árboles de diferentes diámetros (diametros) en 3 sitios diferentes (sitio).

- Realizar el modelo lineal correspondiente para probar la existencia de un efecto del sitio y del diámetro del árbol sobre la cantidad de nidos (añadir interacción).
- Realizar diagnósticos y discutir si se cumplen los supuestos de un modelo lineal.
- Aplicar un modelo lineal randomizado.

2. El test de Mantel es utilizado para evaluar la asociación entre dos matrices de distancia (por ejemplo distancia genética y distancia geográfica) calculando la asociación entre dos matrices y comparando esta medida observada con la esperada por azar. La asociación entre matrices se calcula utilizando la correlación pareada entre los elementos de las matrices. Dado que las matrices X e Y son simétricas se utiliza solamente la mitad de ellas (arriba o abajo de la diagonal), esta mitad puede obtenerse por indexación.

```r
# la mitad superior de la matriz X es
X[col(X) > row(X)]
```

Obtener la asociación observada robs entre las dos matrices de distancia de los archivos yanomamaGEN y yanomamaGEO, correspondientes a las matrices de distancia geográfica y genética entre 19 aldeas Yanomama en Brasil. Realizar una randomización para obtener pseudo-valores de r y obtener su distribución. Probar si r_{obs} es significativa. Notar que para ingresar estas dos matrices de datos se debe utilizar `header = FALSE` en la función `read.table`.
Nota 1. Existen funciones para realizar el test de Mantel en el paquete *vegan*, puede confirmar su resultado consultándolas).
Nota 2: evite usar el test de Mantel en la vida real, hay alternativas más adecuadas, por ejemplo protest en el mismo paquete vegan.

3. Hall et al. (Evolution (2008) 62-9: 2305–2315) proponen un test mediante permutaciones para conocer si existen diferencias significativas entre la oportunidad para la selección (I) masculina y femenina que actúan en una población. Hacer este test utilizando los datos de cyclopogon sabiendo que:

- La oportunidad de la selección (I) se define como la varianza del éxito reproductivo.
- Las medidas de éxito femenino y masculino son frutos y pol.exp.

Para realizar este análisis la base de datos debe ser reordenada para agregar una variable de clasificación en masculino y femenino. Luego calcular la diferencia observada entre I_{masc} e I_{fem}. Realizar una randomización para obtener 1000 pseudo-valores de esta diferencia y probar la significancia de la diferencia observada respecto a la distribución obtenida.

Práctico 9: Bootstrap.

Caso1

Se aplicará un bootstrap para establecer los intervalos de confianza de los coeficientes de regresión de un modelo lineal entre el éxito reproductivo de Turnera ulmifolia L. (medido como número de semillas) y características de sus flores (número de flores, producción de néctar y tamaño medio de la corola).

```
ulm <- read.table("C:/RD/ulmifolia.txt", header = TRUE)
#  ulm <- read.table(file = file.choose(), header = TRUE)

reg <- lm(semillas ~ flores + nectar + tamano, data = ulm)
summary(reg)

layout(matrix(1:4, 2, 2))
plot(reg)
layout(1)

# Guardamos los coeficientes observados
Bobs <- reg$coeff
Bobs

# BOOTSTRAP
library(boot)

# PASO 1: crear una función para "bootstrapear"
REG <- function(datos, indices) {
  newdata <- datos[indices, ]   #sobre esta línea actuará la función boot
  fit <- lm(semillas ~ flores + nectar + tamano, data = newdata)
  fit$coeff
}

# PASO 2: el bootstrap en sí
?boot

boot.reg <- boot(ulm, REG, 9999)
boot.reg
summary(boot.reg)

#histograma de los resultados
layout(matrix(1:4, 2, 2))
hist(boot.reg$t[, 1])
abline(v = Bobs[1], col="red")
hist(boot.reg$t[, 2])
abline(v = Bobs[2], col="red")
hist(boot.reg$t[, 3])
abline(v = Bobs[3], col = "red")
hist(boot.reg$t[,4])
abline(v = Bobs[4], col = "red")
layout(1)

## PASO 3: intervalos de confianza
?boot.ci #examinar la ayuda
```

```r
# prueba, los 5 intervalos distintos que calcula R
int3.95<-boot.ci(boot.reg, conf = 0.90, type = "all", index = 3)
int3.95

# Examinamos que el intervalo de confianza NO incluya al 0
#(nos salteamos la significancia de la ordenada al origen)
# intervalos para la pendiente de "flores"
int2.90 <- boot.ci(boot.reg, conf = 0.90, type = "bca", index = 2)
int2.95 <- boot.ci(boot.reg, conf = 0.95, type = "bca", index = 2)
int2.99 <- boot.ci(boot.reg, conf = 0.99, type = "bca", index = 2)
int2.999 <- boot.ci(boot.reg, conf = 0.999, type = "bca", index = 2)

int2.90
int2.95
int2.99
int2.999

# intervalos para la pendiente de "nectar"
int3.90 <- boot.ci(boot.reg, conf = 0.90, type = "bca", index = 3)
int3.95 <- boot.ci(boot.reg, conf = 0.95, type = "bca", index = 3)
int3.99 <- boot.ci(boot.reg, conf = 0.99, type = "bca", index = 3)
int3.999 <- boot.ci(boot.reg, conf = 0.999, type = "bca", index = 3)

int3.90
int3.95
int3.99
int3.999

# intervalos para la pendiente de "tamaño"
int4.90 <- boot.ci(boot.reg, conf = 0.90, type = "bca", index = 4)
int4.95 <- boot.ci(boot.reg, conf = 0.95, type = "bca", index = 4)
int4.99 <- boot.ci(boot.reg, conf = 0.99, type = "bca", index = 4)
int4.999 <- boot.ci(boot.reg, conf = 0.999, type = "bca", index = 4)

int4.90
int4.95
int4.99
int4.999

# PASO 4: Un plot de diagnóstico: el jack.after.boot
# influencia de las observaciones individuales sobre la pendiente de
# "flores"
jack.after.boot(boot.reg, index = 2)

# ídem sobre la pendiente de "néctar"
jack.after.boot(boot.reg, index = 3)


# ídem sobre la pendiente de "tamaño"
jack.after.boot(boot.reg, index = 4)

# la línea horizontal es una medida de la influencia de cada observación
# la línea vertical presenta los percentiles 5, 10, 16, 50, 84, 90 y 95 de
```

```r
# la distribución del coeficiente

# Otro plot de diagnóstico alternativo puede realizarse con plot
# esto nos puede ayudar a decidir el tipo de intervalo

plot(boot.reg, index = 2, jack = T)
plot(boot.reg, index = 3, jack = T)
plot(boot.reg, index = 4, jack = T)
```

Ejercicios

1. El número de ramas y el número de semillas en ulmifolia.txt no son variables de distribución normal. Obtenga intervalos de confianza del 95% utilizando bootstraps para el índice de correlación (r de Pearson) entre estas 2 variables. Comparar los resultados con el intervalo de confianza obtenido de aplicar la función cor.test, que asume normalidad de las variables.
2. Utilice un bootstrap para calcular un intervalo de confianza para el índice de diversidad de Shannon, obtenido en el práctico 8.