Apuntes de Métodos Estadísticos con R

Prof: María Ignacia Vicuña
Pontificia Universidad Católica de Chile

1. Introducción

R es un software estadístico de libre acceso el cual puede ser utilizado en diferentes sistemas operativos como Windows, MacOS y Linux. R es un lenguaje de programación en el que se introducen códigos para posteriormente ser ejecutados. Una de las grandes ventajas de R es que es un programa de código abierto en el que miles de personas de todo el mundo colaboran en el desarrollo de nuevas metodologías, de manera que se pueden acceder a los paquetes descargándolos como también compartir los propios paquetes con otros.

2. Preliminares

2.1. Instalación de R

La descarga del archivo de instalación de R se realiza desde uno de los links de abajo dependiento del sistema operativo:

Microsoft Windows:	http://cran.r-project.org/bin/windows/base/
MacOS:	http://cran.r-project.org/bin/macosx/
Linux:	http://cran.r-project.org/bin/linux/

2.2. Consola y Editor de R

En la **consola de R**, se puede escribir directamente el código de R, y pulsar enter para ver el valor. Sin embargo, esta no es la manera más eficiente de trabajar en R. Si se quiere guardar el trabajo, corregirlo, repetirlo, etc, es más conveniente usar el **editor de R**. Se debe seleccionar archivo, nuevo Script (documento en blanco del editor) en el cual se puede escribir los programas y guardar. En sistema Windows, para mayor comodidad, se puede seleccionar la opción "Dividir verticalmente"del menú Ventana de la consola. La ejecución del código desde el script se hace desde cualquier posición de la línea o seleccionando

Windows:	tecla F5 o Control+R
MacOS:	comand + enter

Se puede incluir comentarios que R no leerá si utilizamos el símbolo # al comienzo de la línea.

2.3. Instalación de Paquetes

R contiene dos tipos de paquetes, los del tipo **Base** los cuales están incorporados automáticamente en la instalación de R, y los paquetes de **contribución** los cuales se deben descargar para su instalación. Ejecutando el comando getOpcion() en la consola se obtiene las aplicaciones que contiene el paquete base

```
getOption("defaultPackages")
 "datasets"  "utils"     "grDevices" "graphics"  "stats"     "methods"
library(help = "datasets")
```

Existe una gran variedad de paquetes de contribución que son aporte de personas a lo largo del mundo (los cuales son gratuitos). Se requiere conexión a internet para descargarlo e instalarlo y se debe ejecutar

```
install.packages("nombre")
```

Una vez instaslado el paquete se carga con el comando `library(nombre)`

2.4. Operaciones Básicas en R

2.4.1. Funciones Comunes

suma:	+	raíz cuadrada:	sqrt()	tangente:	tan()	productoria:	prod()
resta:	-	elevado:	$\wedge$	logaritmo:	log()	exponencial:	exp()
multiplicación:	*	coseno:	cos()	π:	pi()	valor absoluto:	abs()
división:	/	seno:	sin()	sumatoria:	sum()	ordenar:	sort()

2.4.2. Tipos de Objetos

variable:	`x = 3+4`
vector:	`x = c(1,2,3,4,5)`
secuencia:	`x=seq(-1,1,0.05)`
repetición:	`x = rep(1,100)`
vector con caracteres:	`x = c("Probabilidades", .Estadística")`
matriz:	`A<-matrix(c(1,2,3,4,5,6,7,8,9),ncol =3,nrow = 3)`
multiplicación matrices:	`A%*%A`

2.4.3. Operaciones Lógicas

$<, >, <=$ y $>=$ son los operadores menor, mayor, menor igual y mayor igual respectivamente.
== es el operador de igualdad.
& , | son los operadores y,o respectivamente.

2.4.4. Ayuda de R

Signo pregunta + el nombre del comando: Por ejemplo: `?matrix`

3. Manejo de Datos

3.1. Importar Datos

Importar datos de archivo texto: `read.table()`, la cual tiene muchas opciones, pero las más importantes son `read.table(‘‘archivo.txt",header=FALSE,dec=".")` donde

archivo.txt	archivo que se quiere importar
header	TRUE si la primera línea del archivo contiene nombre de las variables
dec	carácter que separa los números decimales

También se puede usar comando `read.table(file.choose())` donde da la opción de buscar el archivo.

Importar datos de archivo excel: desde la hoja de datos seleccionando el área deseada y se copia en el portapapeles usando `ctrl+c` en Windows o `cmd+c` en MacOS y luego se ejecuta en R:

Windows:	`datos = read.table(çlipboard")`
MacOS	`datos = read.table(pipe("pbpaste"))`

3.2. Exportar Datos

Se exportan datos de tipo texto con el comando `write.table()`.Las opciones más habituales son:

`write.table(Datos,file= "archivo.txt",sep="\t",na="NA",dec=".")}.`

Los nuevos datos se guardaran en la ubicación del directorio que se está usando. Para ver cual es el directorio que está usando R, se usa el comando `getwd()` y para cambiar la ubicación del directorio se usa el comando `setwd()`. Por ejemplo, si queremos guardar la base de datos en el escritorio, hay que ejecutar el comando `setwd( ‘‘/Users/Desktop")`

3.3. Ciclos y Ejecución Condicional

El comando `if()` es para realizar construcciones condicionales a partir de un argumento lógico. Por ejemplo,

```
if(x>1){2*x}
else{-x}
```

De esta manera, si el valor de x es mayor que uno la función lógica es verdadera y debe ejecutar $2 * x$, por el contrario ejecuta $-x$.

Para realizar ciclos se utiliza el comando `for()`. Por ejemplo,

```
x=c()
for(i in 1:100){
x[i] = i^2
}
```

Así el ciclo va de la iteración 1 hasta 100, y rellena cada componente del vector x por el valor de i elevado a 2.

3.4. Crear Funciones

Para crear una función en R, se utiliza el comando `function()`. Por ejemplo, el siguiente código crea una función que calcula la varianza de los datos:

```
varianza = function(x){
 sum((x - mean(x))^2)/n
}
```

Luego a partir de un conjunto de datos, para calcular la varianza utilizando la función debemos ejecutar `varianza(datos)`.

3.5. Filtros en una variable

Por ejemplo, si `x = c(1,1,2,2,0,0,0,0,0,1)` en donde 0 representa si es "estudiante", 1 si es "ayudante´´ y 2 si es "profesor". Así, para seleccionar sólo los estudiantes debemos ejecutar `x[x==0]`, y si se quiere seleccionar a los son profesores o ayudantes se debe ejecutar `x[x==1|x==2]`

4. Estadística Descriptiva

Las funciones `mean(), var(),sd(), median(),min(), max()` y `quantile(), IQR()`
calculan la media, varianza, desviación estándar, mediana, mínimo, máximo, cuantiles y rango intercuantil de una variable numérica. La función `summary()` entrega un resumen de las medidas descriptivas principales. Resúmenes por grupo de variables: mediante función `apply()`
Las funciones `table()` y `prop.table(table())` obtienen las distribuciones de frecuencias absolutas y relativas respectivamente.

4.1. Gráficos

Gráfico de puntos	`plot()`
Diagrama de barras Cualitativas:	`barplot(table())`
Diagrama de barras Discretas:	`plot(table())`
Diagrama de sectores:	`pie(table())`
Histograma:	`hist()`
Diagrama de cajas:	`boxplot()`
superponer líneas	`lines()`
superponer puntos	`points()`

5. Distribuciones de Probabilidad

Distribución	densidad o masa	Función de Distribución	Función cuantil	Muestras aleatorias
Binomial(n,p)	dbinom(x,n,p)	pbinom(x,n,p)	qbinom(prob,n,p)	rbinom(N,n,p)
Poisson(λ)	dpois(x,λ)	ppois(x,λ)	qpois(prob,λ)	rpois(N,λ)
Geométrica(p)	dgeom(x,p)	pgeom(x,p)	qgeom(prob,p)	rgeom(N,p)
BinNeg(a,p)	dnbinom(x,a,p)	pnbinom(x,a,p)	qnbinom(prob,a,p)	rnbinom(N,a,p)
Uniforme(a,b)	dunif(x,a,b)	punif(x,a,b)	qunif(prob,a,b)	runif(N,a,b)
Exponencial(λ)	dexp(x,λ)	pexp(x,λ)	qexp(prob,λ)	rexp(N,λ)
Normal(μ,σ^2)	dnorm(x,μ,σ)	pnorm(x,μ,σ)	qnorm(prob,μ,σ)	rnorm(N,μ,σ)
Gamma(α,λ)	dgamma(x,a,λ)	pgamma(x,a,λ)	qgamma(prob,a,λ)	rgamma(N,a,λ)
tstudent(ν)	dt(x,ν)	pt(x,ν)	qt(prob,ν)	rt(N,ν)

6. Estimación Puntual

6.1. Estimación Máximo Verosímil

El paquete MASS contiene la función `fitdistr()` que calcula los estimadores máximos verosímiles y los errores estándar asociados a una función de distribución a partir de un conjunto de datos. Por ejemplo `ajuste =fitdistr(datos, "densidad")` donde densidad es el modelo a ajustar: "geometric","Poisson", "negative binomial", "normal","exponential","gamma","beta",etc...

También se puede calcular usando la función `optimize()` que calcula el máximo de la función de verosimilitud. Por ejemplo, para el caso Binomial(n,p),

```
L = function(p, x) prod(dbinom(x, size = 1, prob = p))
 optimize(L, interval = c(0, 1), x = datos, maximum = TRUE)
```

Si se quiere graficar funciones de verosimilitud que dependen de dos parámetros (como el caso Normal), el comando `persp()` realiza gráficos en 3d. Además con el comando `contour()` se pueden graficar las curvas de nivel de la función.

7. lntervalos de Confianza

Suponga que una variable aleatoria proviene de una población con media μ y varianza σ^2 y se extrae una muestra aleatoria de tamaño n. Denotemos por $X_1, ..., X_n$ la muestra.

7.1. Intervalo de Confianza para μ:

Población	n	σ^2	Intervalo de Confianza
Normal	-	conocido	`library(TeachingDemos)`
			`z.test(x, sigma, conf.level =1-alpha)$conf.int`
Normal	pequeño	desconocido	`t.test(x,conf.level =1-alpha)$conf.int`
Cualquiera	grande	conocido	`library(TeachingDemos)`
			`z.test(x, sd(x), conf.level =1-alpha)$conf.int`

7.2. Intervalo de Confianza para σ^2:

Población	Intervalo de Confianza
Normal	`library(TeachingDemos)`
	`sigma.test(x, sigma = sd(x),conf.level =1-alpha)$conf.int`

7.3. Intervalo de Confianza para p:

Suponga que $X_1, ..., X_n$ es una muestra aleatoria desde una población Bernoulli(p) y n grande

Intervalo de Confianza
`prop.test(exitos,n,conf.level =1-alpha)$conf.int`

Muestras Independientes:

Suponga que $X_1, ..., X_n$ es una muestra aleatoria desde una población $N(\mu_x, \sigma_x^2)$ e $Y_1, ..., Y_m$ es una muestra aleatoria desde una población $N(\mu_y, \sigma_y^2)$.

7.4. Intervalo de Confianza para $\mu_x - \mu_y$:

σ_x^2 y σ_y^2	Intervalo de Confianza
conocidas	`mean(x) - mean(y)+c(-1,1)*qnorm(1-alpha/2)*sqrt(sigma2x/n + sigma2y/m)`
desconocidas e iguales	`t.test(x,y,conf.level=1-alpha,var.equal=T)$conf.int`
desconocidas y distintas	`t.test(x,y,conf.level=1-alpha)$conf.int`

7.5. Intervalo de Confianza para σ_x^2/σ_y^2 :

Intervalo de Confianza
`var.test(x,y,conf.level=1-alpha)$conf.int`

Muestras Dependientes o Pareadas:

Suponga que $X_1, ..., X_n$ es una muestra aleatoria desde una población $N(\mu_x, \sigma_x^2)$ e $Y_1, ..., Y_m$ es una muestra aleatoria desde una población $N(\mu_y, \sigma_y^2)$.

7.6. Intervalo de Confianza para $\mu_x - \mu_y$:

Intervalo de Confianza
`t.test(x,y,paired = TRUE, conf.level = 1-alpha)$conf.int`

8. Test de Hipótesis

Suponga que se dispone una población de una variable aleatoria con media μ y varianza σ^2 y se extrae una muestra aleatoria de tamaño n. Denotemos por $X_1, ..., X_n$ la muestra.

8.1. Test de Hipótesis para μ :

$$H_0 : \mu = \mu_0$$

Población	H_1	Test de Hipótesis
Normal σ^2 conocido	$H_1 : \mu \neq \mu_0$	`library(TeachingDemos)` `z.test(x, mu = mu0, sigma, alternative = "two.sided", conf.level = 1-alpha)`
	$H_1 : \mu > \mu_0$	`library(TeachingDemos)` `z.test(x, mu = mu0, sigma, alternative = "greater", conf.level = 1-alpha)`
	$H_1 : \mu < \mu_0$	`library(TeachingDemos)` `z.test(x, mu = mu0, sigma, alternative = "less", conf.level = 1-alpha)`

Población	H_1	Test de Hipótesis
Normal σ^2 desconocido n pequeño	$H_1 : \mu \neq \mu_0$	`t.test(x, mu = mu0, alternative = "two.sided", conf.level = 1-alpha)`
	$H_1 : \mu > \mu_0$	`t.test(x, mu = mu0, alternative = "greater", conf.level = 1-alpha)`
	$H_1 : \mu < \mu_0$	`t.test(x, mu = mu0, alternative = "less", conf.level = 1-alpha)`

Población	H_1	Test de Hipótesis
Cualquiera σ^2 conocido n grande	$H_1 : \mu \neq \mu_0$	`library(TeachingDemos)` `z.test(x, mu = mu0, sd(x), alternative = "two.sided", conf.level = 1-alpha)`
	$H_1 : \mu > \mu_0$	`library(TeachingDemos)` `z.test(x, mu = mu0, sd(x), alternative = "greater", conf.level = 1-alpha)`
	$H_1 : \mu < \mu_0$	`library(TeachingDemos)` `z.test(x, mu = mu0, sd(x), alternative = "less", conf.level = 1-alpha)`

8.2. Test de Hipótesis para σ^2:

$$H_0 : \sigma^2 = \sigma_0^2$$

Población	H_1	Test de Hipótesis
Normal	$H_1 : \sigma^2 \neq \sigma_0^2$	`library(TeachingDemos)`
		`sigma.test(x, sigma = sigma0, alternative ="two.sided", conf.level =1-alpha)`
	$H_1 : \sigma^2 > \sigma_0^2$	`library(TeachingDemos)`
		`sigma.test(x, sigma = sigma0, alternative ="greater", conf.level =1-alpha)`
	$H_1 : \sigma^2 < \sigma_0^2$	`library(TeachingDemos)`
		`sigma.test(x, sigma = sigma0, alternative ="less", conf.level =1-alpha)`

8.3. Test de Hipótesis para p:

Suponga que $X_1, ..., X_n$ es una muestra aleatoria desde una población Bernoulli(p) y n grande

$$H_0 : p = p_0$$

H_1	Test de Hipótesis
$H_1 : p \neq p_0$	`prop.test(x, p = p0, alternative="two.sided",correct = TRUE, conf.level =1-alpha)`
$H_1 : p > p_0$	`prop.test(x, p = p0, alternative ="greater", correct = TRUE, conf.level =1-alpha)`
$H_1 : p < p_0$	`prop.test(x,p = p0, alternative ="less", correct = TRUE, conf.level =1-alpha)`

Muestras Independientes:

Suponga que $X_1, ..., X_n$ es una muestra aleatoria desde una población $N(\mu_x, \sigma_x^2)$ e $Y_1, ..., Y_m$ es una muestra aleatoria desde una población $N(\mu_y, \sigma_y^2)$.

8.4. Test de Hipótesis para $\mu_x - \mu_y$:

$$H_0 : \mu_x - \mu_y = \delta$$

σ_x^2 y σ_y^2 conocidas

H_1	Test de Hipótesis
$H_1 : \mu_x - \mu_y \neq \delta$	`Región rechazo: abs((mean(x)-mean(y)-delta)/(sqrt(sigma2x/n + sigma2y/m)))>qnorm(1-alpha/2)`
$H_1 : \mu_x - \mu_y > \delta$	`Región rechazo: (mean(x)-mean(y)-delta)/(sqrt(sigma2x/n + sigma2y/m))>qnorm(1-alpha)`
$H_1 : \mu_x - \mu_y < \delta$	`Región rechazo: (mean(x)-mean(y)-delta)/(sqrt(sigma2x/n + sigma2y/m))<-qnorm(1-alpha)`

$\sigma_x^2 = \sigma_y^2 = \sigma^2$ desconocidas

H_1	Test de Hipótesis
$H_1 : \mu_x - \mu_y \neq \delta$	`t.test(x,y,mu=delta, alternative="two.sided",conf.level=1-alpha,var.equal=T)`
$H_1 : \mu_x - \mu_y > \delta$	`t.test(x,y,mu=delta, alternative="greater",conf.level=1-alpha,var.equal=T)`
$H_1 : \mu_x - \mu_y < \delta$	`t.test(x,y,mu=delta, alternative="less",conf.level=1-alpha,var.equal=T)`

σ_x^2 y σ_y^2 distintas y desconocidas

H_1	Test de Hipótesis
$H_1 : \mu_x - \mu_y \neq \delta$	`t.test(x,y,mu=delta, alternative="two.sided",conf.level=1-alpha,var.equal=F)`
$H_1 : \mu_x - \mu_y > \delta$	`t.test(x,y,mu=delta, alternative="greater",conf.level=1-alpha,var.equal=F)`
$H_1 : \mu_x - \mu_y < \delta$	`t.test(x,y,mu=delta, alternative="less",conf.level=1-alpha,var.equal=F)`

8.5. Test de Hipótesis para σ_x^2/σ_y^2 :

$$H_0 : \sigma_x^2/\sigma_y^2 = 1$$

H_1	Test de Hipótesis
$H_1 : \sigma_x^2/\sigma_y^2 \neq 1$	`var.test(x,y,alternative="two.sided",conf.level=1-alpha)`
$H_1 : \sigma_x^2/\sigma_y^2 > 1$	`var.test(x,y,alternative="greater",conf.level=1-alpha)`
$H_1 : \sigma_x^2/\sigma_y^2 < 1$	`var.test(x,y,alternative="less",conf.level=1-alpha)`

8.6. Test de Hipótesis para $p_x - p_y$:

Suponga que $X_1, ..., X_n$ es una muestra aleatoria desde una población Bernoulli(p_x) e $Y_1, ..., Y_m$ es una muestra aleatoria desde una población Bernoulli(p_y), con n y m grandes.
Para utilizar el comando `prop.test()` debemos previamente crear una matriz con los datos de la siguiente manera: `tabla = matrix(c(n11,n12,n21,n22),2,2,byrow=TRUE)`, donde `n11` es el número de éxitos de la muestra X, `n12` es el número de fracasos de la muestra X, `n21` es el número de éxitos de la muestra Y, `n22` es el número de fracasos de la muestra Y.

$$H_0 : p_x - p_y = 0$$

H_1	Test de Hipótesis
$H_1 : p_x - p_y \neq 0$	`prop.test(tabla, alternative = "two.sided", correct = FALSE))`
$H_1 : p_x - p_y > 0$	`prop.test(tabla, alternative = "greater", correct = FALSE))`
$H_1 : p_x - p_y < 0$	`prop.test(tabla, alternative = "less", correct = FALSE))`

Muestras Dependientes o Pareadas:

Suponga que $X_1, ..., X_n$ es una muestra aleatoria desde una población $N(\mu_x, \sigma_x^2)$ e $Y_1, ..., Y_m$ es una muestra aleatoria desde una población $N(\mu_y, \sigma_y^2)$.

8.7. Test de Hipótesis para $\mu_x - \mu_y$:

$$H_0 : \mu_x - \mu_y = \delta$$

H_1	Test de Hipótesis
$H_1 : \mu_x - \mu_y \neq \delta$	`t.test(x,y, alternative="two.sided",mu=delta, paired = TRUE, conf.level = 1-alpha)`
$H_1 : \mu_x - \mu_y > \delta$	`t.test(x,y, alternative="greater",mu=delta, paired = TRUE, conf.level = 1-alpha)`
$H_1 : \mu_x - \mu_y < \delta$	`t.test(x,y, alternative="less",mu=delta, paired = TRUE, conf.level = 1-alpha)`

8.8. Potencia del Test

Para cada uno de los test vistos anteriormente se puede calcular la función potencia. Por ejemplo, para el caso en que la muestra aleatoria de tamaño n provenga de una población Nomal(μ, σ^2), donde σ^2 es conocido, y se desea sea testear $H_0 : \mu = \mu_0$ se tiene que la función potencia (que depende del valor de μ_1 de la hipótesis alternativa) está dada por

H_1	Potencia del Test
$H_1 : \mu \neq \mu_0$	`1-pnorm(qnorm(1-alpha/2,mean=0,sd = 1) + (mu0-mu1)*sqrt(n)/sigma, mean=0,sd=1) + pnorm(qnorm(alpha/2,mean=0,sd = 1) + (mu0-mu1)*sqrt(n)/sigma, mean=0,sd=1)`
$H_1 : \mu > \mu_0$	`1-pnorm(qnorm(1-alpha,mean=0,sd = 1) + (mu0-mu1)*sqrt(n)/sigma, mean=0,sd=1)`
$H_1 : \mu < \mu_0$	`pnorm(qnorm(alpha,mean=0,sd = 1) + (mu0-mu1)*sqrt(n)/sigma, mean=0,sd=1)`

En el caso en que σ^2 es desconocido y n pequeño, se utiliza la distribución t-student del estadístico.

H_1	Potencia del Test
$H_1 : \mu \neq \mu_0$	`1-pt(qt(1-alpha/2, df = n-1) + (mu0-mu1)*sqrt(n)/sd(x) ,df =n-1) + pt(qt(alpha/2,df =n-1) + (mu0-mu1)*sqrt(n)/sd(x), df = n-1)`
$H_1 : \mu > \mu_0$	`1-pt(qt(1-alpha, df = n-1) + (mu0-mu1)*sqrt(n)/sd(x) ,df =n-1)`
$H_1 : \mu < \mu_0$	`pt(qt(alpha,df =n-1) + (mu0-mu1)*sqrt(n)/sd(x), df = n-1)`

Para las pruebas t, existe un comando que calcula la función potencia:

```
power.t.test(n, delta = mu0-mu1,alternative = "one.sided", type= "one.sample")
```

8.9. Test de Bondad de Ajuste: χ^2 de Pearson

Sea $X_1, ..., X_n$ una muestra aleatoria simple de tamaño n proveniente de una población. Las pruebas de bondad de ajuste tienen por objetivo determinar si los datos se ajustan a una determinada distribución. De esta manera, se quiere testear

$$\begin{aligned} H_0 : &\quad X \text{ tiene distribución } f(y, \theta_0) \\ H_1 : &\quad X \text{ no tiene distribución } f(y, \theta_0) \end{aligned}$$

Se debe agrupar los valores de la variable aleatoria en un número finito de clases. Suponga que $X_1, ..., X_n$ toma valores en las clases: $C_1, C_2, C_3, .., C_k$
Sea O_i=Número de individuos de la muestra en la clase C_i y sea $P(X \in C_i) = \pi_i, i = 1, ..., k$ se tiene que $\sum_{i=1}^{k} O_i = n$ y $\sum_{i=1}^{k} \pi_i = 1$. Así las hipótesis a testear es equivalente a:

$$\begin{aligned} H_0 : &\quad \pi_1 = \pi_{01}, \pi_2 = \pi_{02}, ..., \pi_k = \pi_{0k} \\ H_1 : &\quad \text{al menos un } \pi_i \neq \pi_{01} \text{ con } i = 1, ..., k \end{aligned}$$

El comando que realiza el test χ^2 de bondad de ajuste es `chisq.test(obs, p=p0))` donde `obs` es un vector de largo k y corresponde al número de invididuos observados en la muestra que pertenece a la categoría C_i con $i = 1, .., k$ y `p0` corresponde a las probabildades bajo H_0, es decir, $(\pi_{01}, \pi_{02}, ..., \pi_{0k})$. Hay que tener cuidado con el valor-p que entrega el test, ya que cuando se estiman parámetros de la población hay que corregir los grados de libertad. Es conveniente utilizar el estadístico y calcular el valor-p a parte como `pchisq(X2,df = k-g-1, lower.tail = FALSE)` donde `X2` es el valor del estadístico del text, `g` es el número de parámetros a estimar.

9. Análisis de Regresión

La función `cor()` calcula la correlación entre dos variables. Si la función de entrada es una matriz de datos calcula las correlaciones de la variables entre sí. El comando `cor.test()` realiza el test de correlación, cuya sintaxis básica es la siguiente:

```
cor.test(x,y,alternative= c("two.sided", "less","greater"), conf.level = 1 - alpha)
```

Para ajustar una regresión lineal se utiliza la función `lm()`. La función `summary()` entrega un resumen del ajuste, reportando los estimadores de los coeficientes del modelo, su error estándar, prueba de significancia, R^2 entre otros. Por ejemplo, para ajustar una regresión lineal simple, donde y es la variable dependiente y x es la variable independiente,

```
modelo  = lm(y~x)
summary(modelo)
```

La tabla Anova del modelo se realiza con el comando `anova(modelo)`

Ahora, si se tienen más predictores, denotados por $x_1, x_2, ..., x_p$ el ajuste de la regresión múltiple se hace mediante,

```
modelo  = lm(y~x1+x2 + ....+ xp)
summary(modelo)
```

Además, para ajustar un modelo sin intercepto basta poner un -1 al comienzo:

```
modelo  = lm(y~ -1 + x1+x2 + ....+ xp)
summary(modelo)
```

Ahora, si se tiene un predictor que es una variable categórica, por ejemplo: sexo de un individuo, donde toma el valor 0 si es hombre y 1 si es mujer. Debemos especificar que dicha variable es categórica (de lo contrario lo tomará como una variable numérica) utilizando el comando `as.factor()` y en el ajuste la función `lm` automáticamente crea una variable dummie. De esta manera, si se ajusta una regresión de y con la variable `sexo`, se debe ejecutar

```
sexo = as.factor(sexo)
modelo  = lm(y~ sexo)
summary(modelo)
```

Para realizar predicciones para un valor dado de la variable independiente se utiliza la función `predict(modelo,newdata, inverval,level)` donde,

`modelo`	modelo de regresión lineal
`newdata`	valores de la variable independiente para los que se haran predicciones
`interval`	"confidence´´ o "prediction"
`level`	nivel de significancia de los intervalos