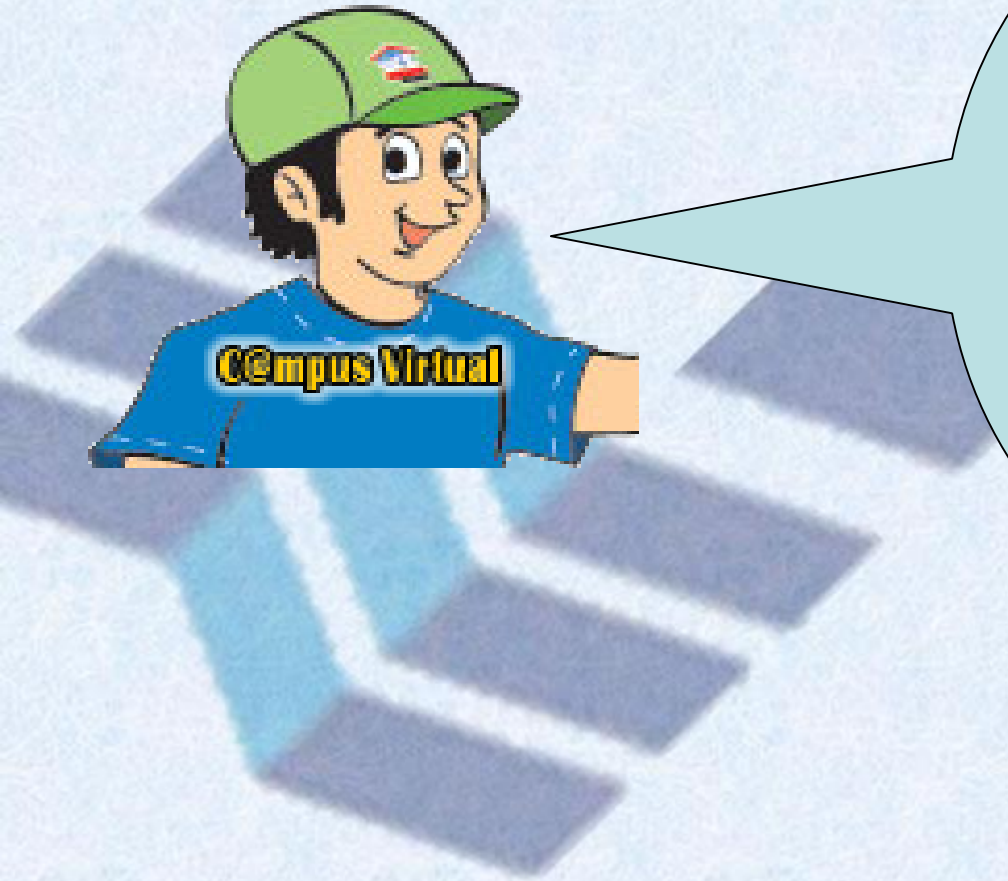
CURSO TALLER SOFTWARE R


ENEI
Escuela Nacional de Estadística e Informática
INEI
INSTITUTO NACIONAL DE ESTADÍSTICA E INFORMÁTICA
contáctenos: enei@inei.gob.pe / 433-3127
Pasaje Hernán Velarde 285 Lima.
Entre la cuadra 01 y 02 de la Av. Arequipa.
Correo: enei@inei.gob.pe / campusvirtual@inei.gob.pe
Teléfonos: 433-3127 - 332-4650
Centro Andino de Formación y Capacitación en Estadística
Cursos Especializados en Estadística e Informática

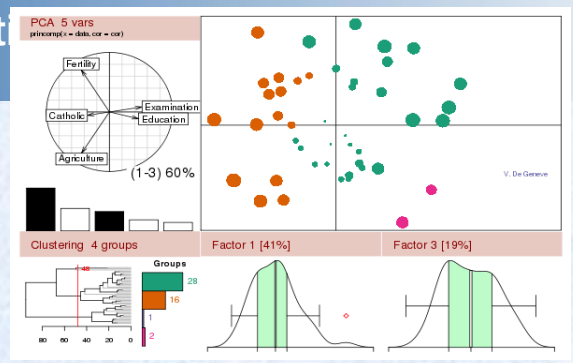


Estimado alumno, buen día. Cualquier consulta no dudes en comentarme o avisarme.

Este curso es netamente práctico y se que lograremos objetivos importantes.

Recuerda siempre nuestro correo para cualquier consulta

campusvirtual@inei.gob.pe



MANIPULACIONES SIMPLES: NÚMEROS Y VECTORES

- Vectores y asignaciones
- Aritmética de vectores.
- Generación de secuencias regulares.
- Valores missing.
- Vectores lógicos.
- Vectores de cadenas de caracteres (strin
- Indexación de vectores.
- Coerción de tipos.
- Otros tipos de objetos en R.
- Ejercicios propuestos.

Vectores y asignaciones

R trabaja sobre estructuras de datos. La estructura más simple es un vector numérico, que consiste en un conjunto ordenado de números.

```
#Creamos un vector de reales mediante la función c y lo guardamos en la variable x.
> x <- c(1.3, 2.5, 4.2, 9.7, 8.1)

#Un número por sí mismo es un vector de longitud 1
> v <- 5

#Otras maneras de asignar menos utilizadas
> assign("x", c(1.3, 2.5, 4.2, 9.7, 8.1))  #Instrucción assign
> c(1.3, 2.5, 4.2, 9.7, 8.1) -> x  #Asignación en la otra dirección
```

Si no se utiliza ninguna de las tres maneras de asignación ("<-", "->", "assign") el resultado de nuestra expresión se muestra por pantalla pero no quedará guardado.

```
#Expresión: el resultado no se guarda
> c(x,0,x)
 [1] 1.3 2.5 4.2 9.7 8.1 0.0 1.3 2.5 4.2 9.7 8.1

#Objeto especial de R que guarda el resultado del último comando ejecutado
> .Last.value
 [1] 1.3 2.5 4.2 9.7 8.1 0.0 1.3 2.5 4.2 9.7 8.1
```

Aritmética de vectores

Al contrario que la mayoría de lenguajes de programación, R tiene aritmética vectorial, por lo que los vectores pueden aparecer en las expresiones que generamos.

En caso que los vectores que aparecen en una expresión no sean de la misma longitud, el más corto se "recicla" hasta que alcanza la longitud del más largo.

```
#Generamos dos vectores.
> x <- c(1.3, 2.5, 4.2, 9.7, 8.1)
> y <- c(x,0,x)

#Utilizamos x e y en una nueva expresión. Como x es más corto
#que y, se reciclará para adquirir su misma longitud. R nos avisa
#de este hecho porqué los dos vectores no tienen una longitud
#múltiple. El 1 en este caso también se recicla y pasa a ser un
#vector de once unos.
> v <- 2*x + y + 1
Warning message:
longer object length
        is not a multiple of shorter object length in: 2 * x + y
> v
 [1]  4.9  8.5 13.6 30.1 25.3  3.6  7.3 11.9 24.6 26.9 11.7
```

Operador/función	Símbolo/instrucción
suma	+
resta	-
multiplicación	*
división	/
módulo	%%
división entera	%/%
raíz cuadrada	sqrt
logaritmo nep.	log
log gen	logb
exponencial	exp
seno	sin
coseno	cos
tangente	tan
máximo	max
mínimo	min
rango	range
longitud	lengt
sumatorio	h sum
producto	prod
media	mean
desv. estándar	sd
varianza	var

Generación de secuencias regulares

R dispone de instrucciones para generar secuencias de números. Una de las más utilizadas es el operador ":"

```
#Generamos un vector con los números 1, 2, 3, 4, …, 29, 30.
> 1:30 #Esto es equivalente al vector c(1, 2, …, 29, 30)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

#El operador ":" tiene la máxima preferencia
> n <- 10
> 1:n-1    #Aquí prevalece ":" sobre "-"
 [1] 0 1 2 3 4 5 6 7 8 9
> 1:(n-1) #Forzamos la prioridad del "-"
 [1] 1 2 3 4 5 6 7 8 9
```

Con la función *seq* también se pueden generar secuencias de números

```
#Generamos una secuencia de 1 a 30 saltando dos números cada vez
> seq(1,30,by=2)
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29

#La función seq admite también la longitud de la secuencia que queremos generar, de manera que ella misma
#decide el intervalo automáticamente
> seq(1,30,length=15)
 [1]  1.000000  3.071429  5.142857  7.214286  9.285714 11.357143 13.428571 15.500000 17.571429
19.642857 21.714286 23.785714 25.857143 27.928571 30.000000
```

La función *rep* sirve para generar repeticiones de objetos (escalares o vectores)

Valores missing

En R los valores "desconocidos" o "no disponibles" (*missings*) se simbolizan con el valor especial NA (*Not Available*). Cualquier operación que incluya un NA en general devolverá NA como resultado.

La función *is.na* nos permite saber si un elemento es *missing* o no.

```
#Generamos un vector con los números 1, 2, 3 y un missing al final
> z <- c(1:3, NA)
> z
[1]  1  2  3 NA
> is.na(z) #Miramos que valores del vector son missing.
[1] FALSE FALSE FALSE  TRUE

#La expresión z==NA no funciona!!!
> z==NA
Error: object „z' not found
```

Hay un segundo tipo de *missings* que se producen por computación numérica, lo que se llama *Not a Number*. En R se simbolizan con el valor NaN.

```
> 0/0  #Provocamos un error numérico
[1] NaN
```

La función *is.na* retorna TRUE tanto para los NA como para los NaN. En cambio, la función *is.nan* sólo retorna TRUE para los NaN.

Vectores lógicos

R permite la manipulación de cantidades lógicas. Los valores de un vector lógico pueden ser TRUE o T (cierto), FALSE o F (falso) y NA/NaN.

Los vectores lógicos se generan mediante condiciones:

```
#Generamos un vector de 1 a 10
> x <- 1:10
#cond1 vector lógico, de la misma longitud que x, donde cada casilla
#nos dice si la correspondiente casilla de x cumple la condición x>7
> cond1 <- x > 7
> cond1
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

En R los vectores lógicos se pueden utilizar en aritmética ordinaria, siendo substituído (coercionado) el FALSE por 0 y el TRUE por 1.

```
> cond2 <- x >= 9   #Generamos otra condición
> cond1 & cond2   #Hacemos una and lógica de las dos condiciones
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
> !cond1   #Negación lógica del vector cond1
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
```

Operador	Símbolo
igualdad	==
desigualdad	!=
menor	<
menor igual	<=
mayor	>
mayor igual	>=
an d lógica	&
or lógica	|
negación lógica	!

Vectores de cadenas de caracteres (strings)

Los vectores de cadenas de caracteres se usan a menudo en R, principalmente para guardar etiquetas.

Los caracteres pueden ser entrados utilizando comillas simples ('') o dobles (""), aunque después R los muestra normalmente rodeados de comillas dobles. Como en C, utilizan el carácter "\" como carácter de escape. Algunos caracteres útiles son el salto de linea (\n) y tabulador (\t).

La función *paste* toma un número arbitrario de argumentos y los concatena uno a uno para transformarlos después en caracteres

```
#Concatenamos un vector de dos letras y otro de dos números posición a posición
> paste(c("X","Y"),1:2,sep="")
[1] "X1" "Y2"

#Ahora concatenamos vectores de diferente longitud y hacemos que el separador de la concatenación sea un
#punto ".". Nótese el reciclaje del vector corto.
> paste(c("X","Y"),1:4,sep=".")
[1] "X.1" "Y.2" "X.3" "Y.4"

#La función paste tiene otro argumento, llamado collapse, que fusiona todos los strings que genera para dar
#una única cadena de caracteres de salida. El valor de collapse es lo que se coloca en medio de las
#diferentes cadenas de caracteres que se unen.
> paste(c("X","Y"),1:4,sep=".",collapse="-")
[1] "X.1-Y.2-X.3-Y.4"
```

Indexación de vectores (1)

Se puede seleccionar un subconjunto de los elementos de un vector añadiéndo al lado de su nombre un *vector de índices* entre corchetes ([]). Los vectores de índices pueden ser de cuatro tipos:

- Vectores lógicos: han de tener la misma longitud que el vector del cual se quieren seleccionar elementos. Los valores que corresponden a TRUE en su correspondiente posición del vector lógico serán seleccionados, y los que corresponden a FALSE seran omitidos.

- Vectores de números enteros positivos: los valores que contienen deben estar entre 1 y *length(x)*. Los correspondientes elementos del vector *x* son seleccionados y concatenados en ese orden.

- Vectores de números enteros negativos: especifican los valores que serán excluídos en vez de incluídos.

- Vectores de *strings*: sólo se pueden utilizar cuando el objecto tiene un atributo llamado *names* (nombres) que identifica sus componentes. En este caso, un subvector del vector de nombres se utilizaría de la misma manera como se utilizaría un vector de enteros positivos. La ventaja es que estos nombres suelen ser más fáciles de recordar que los índices de los elementos a seleccionar.

Indexación de vectores (2)

```
> x <- c(1:5,NA,6:8,NA,9,10) #Generamos un vector con enteros de 1 a 10 y algunos missings por medio.

> x[!is.na(x)]  #Indexación por vector lógico: escogemos aquellos elementos que no son NA.
[1]  1  2  3  4  5  6  7  8  9 10

> x[!is.na(x) & x%%2==0]  #Vector lógico más complejo.
[1]  2  4  6  8 10

> x[1:5]  #Indexación mediante vector de enteros positivos: nos quedamos con los 5 primeros elementos de x.
[1] 1 2 3 4 5

> x[c(1,3,5,7,9,11)]  #Nos quedamos con las posiciones impares
[1] 1 3 5 6 8 9

> x[-(1:5)]  #Indexación por enteros negativos: eliminamos los 5 primeros elementos de x
[1] NA  6  7  8 NA  9 10

> y <- c(5,18,7,13)  #Indexación por vectores de strings.
> names(y) <- c("uno","dos","tres","cuatro")  #Añadimos los names al vector.
> y[c("dos","tres")]  #Seleccionamos mediante los names.
 dos tres
  18    7
```

Una indexación también puede aparecer en la parte izquierda de una asignación.

```
> x[is.na(x)] <- 0  #Substituimos los NA por 0.
> x[x>5] <- -x[x>5]  #A los valores del vector superiores a 5 les cambiamos el signo.
```

Coerción de tipos

R, como la mayoría de lenguajes de programación, dispone de funciones para realizar la coerción (transformación) de tipos.

Para que la coerción tenga lugar no se ha de dar ninguna incompatibilidad entre el tipo origen y el tipo destino.

Así como las funciones de coerción, que suelen comenzar con la palabra *as* seguida de un punto (*as.integer*) y el tipo de destino, también existen las funciones de verificación de tipos, que se componen de *is* seguido de un punto y el tipo a verificar (*is.integer*).

```
# Coerción correcta
> a <- c("1","2","3")
> b <- as.numeric(a)
> b
[1] 1 2 3

# Coerción incorrecta. Se introducen NA's.
> a <- c("1","2","x")
> b <- as.numeric(a)
> b
[1]  1  2 NA
```

Funciones más habituales

as.character
as.integer
as.double
as.complex
as.numeric
as.logical
as.factor

Otros tipos de objetos en R

- *Arrays* y matrices (*matrix*): generación multidimensional de los vectores. Todos los elementos de la matriz han de ser del mismo tipo.

- Factores (*factor*): útiles para el uso de datos categóricos.

- Listas (*list*): generalización de los vectores donde los elementos pueden ser de diferentes tipos (incluso vectores o nuevas listas).

- *Data frames*: matrices donde las diferentes columnas pueden tener valores de diferentes tipos.

- Funciones (*function*): conjunto de código de R ejecutable y parametrizable.

- Cualquier objeto en R tiene las propiedades *mode* y *length.*

    - *Mode*: tipo de datos de los elementos que forman un objeto (*numeric*, *complex* ,*logical* y *character*).

    - *Length*: número de elementos que contiene el objeto.

Ejercicios propuestos

1. Con la ayuda de la función *rnorm*, generar un vector de 100 valores que sigan una distribución normal (0,1) y guardarlo en una variable llamada *x* (tienes que hacer *x<-rnorm(100)*). Crear un vector con los valores de las posiciones pares (*p*) de *x* y otro con los valores de las posiciones impares (*i*). Verificar que los dos vectores tienen la misma longitud. Calcular la media y desviación estándar tanto de *p* como de *i*. Calcular también la media y desviación estándar de *p* e *i* posición a posición (este último punto sin utilizar las funciones *mean* ni *sd*).

2. Generar mediante la instrucción *seq* el vector (0, 0.1, 0.2, 0.3, …, 1). Sumar a este vector ahora el vector (1,2,3): ¿qué hace R, da algun mensaje? Hay que vigilar con el reciclaje!!

3. Generar un vector de 1 a 200. Dividir los 100 primeros elementos del vector por el módulo de la división de su correspondiente posición entre 5. Con la ayuda de la función *is.finite*, que tiene un funcionamiento idéntico a *is.na*, canviar los valores infinitos resultantes por NA. Hacer una subselección de este vector para quedarse sólo con los elementos que no son NA. Ahora hacer una subselección para descartar los diez primeros números de este vector resultante (tienes que decir explícitamente que quieres quitar estos diez elementos, no que queres quedarte con todos los restantes).

4. La instrucción *rep(x, times)* sirve para hacer repeticiones *times* veces del objeto *x*. Crear un vector que sea la repetición del vector (1, 2, 3, 4, 5) tres veces. Ahora crear el vector (1, 1, 1, 2, 2, 2, …, 5, 5, 5).
   ¿Cómo tiene que ser ahora el parámetro *t im es*? (Pista: le tienes que decir cuántas veces seguidas quieres repetir cada elemento del vector). ¿Puedes crear el vector (1,2,2,3,3,3,4,4,4,4,5,5,5,5,5)? Concatena estos tres vectores que has obtenido uno detrás del otro.
5. Volvamos a los vectores *i* y *p* del ejercicio 1. Crea un vector lógico que indique cuando alguno de los dos valores que se aparean posición a posición son mayores que zero.

Ejercicios propuestos

6. R dispone de dos objetos predeterminados, *letters* y *LETTERS*, que corresponden al conjunto de letras minúsculas y mayúsculas, respectivamente. Con la ayuda de la función *paste* y estos dos objetos, conseguir obtener el vector ("aA", "bB", "cC", …, "zZ"). ¿Puedes obtener también el *string* "aA-bB-cC-…-zZ"? (Pista: utilizar el argumento *collapse*).

7. Crea la siguiente secuencia de textos: "Dato1", "Dato2", …., "Dato10"

Comunicación constante con la Escuela del INEI

Correo de la Escuela del INEI
enei@inei.gob.pe

Área de Educación Virtual
(campusvirtual@inei.gob.pe)