**Data Science R Programming Task 1 Answer**

1. Vector Recycling

When applying an operation to two vectors that requires them to be the same length, R automatically recycles, or repeats, elements of the shorter one, until it is long enough to match the longer Vector.

Example:

|  |
| --- |
| Die  # 1 2 3 4 5 6  Die + 1:2  # 2 4 4 6 6 8  // (1:2) will be repeated to match the length of vector Die => (1,2,1,2,1,2) |

1. 3. Inner Multiplication and Outer Multiplication

A matrix in R can be created using matrix() function and this function takes input vector, nrow, ncol, byrow, and dimnames as arguments.

Inner Multiplication Operator: %\*%

Outer Multiplication Operator: %o%

Example: Inner Multiplication

|  |
| --- |
| Die # 1 2 3 4 5 6 Die %\*% Die  # 91  // 1\*1 + 2\*2 + 3\*3 + 4\*4 + 5\*5 + 6\*6 = 91 |

Example: Outer Multiplication

|  |
| --- |
| Die  # 1 2 3 4 5 6  Die %o% Die ## [,1] [,2] [,3] [,4] [,5] [,6] ## [1,] 1 2 3 4 5 6 ## [2,] 2 4 6 8 10 12 ## [3,] 3 6 9 12 15 18 ## [4,] 4 8 12 16 20 24 ## [5,] 5 10 15 20 25 30 ## [6,] 6 12 18 24 30 36  // Row 1: Die \* 1, Row 2: Die \* 2, .... , Row 6: Die \* 6 |

4. Functions in R

1. Sample()

The **sample**() **function in R** allows you to take a random **sample** of elements from a dataset or a vector, either with or without replacement.

The basic syntax for the sample() function is as follows:

**sample(x, size, replace = FALSE, prob = NULL)**

* x: a dataset or vector from which to choose the sample
* size: size of the sample
* replace: should sampling be with replacement? (this is FALSE by default)
* prob: a vector of probability weights for obtaining the elements of the

vector being sampled

Example: Rolling a dice (meaning select a random number within the 6 available numbers)

|  |
| --- |
| die # 1 2 3 4 5 6  sample(x = die, size = 1) ## 2 sample(x = die, size = 1) ## 1 sample(x = die, size = 1) ## 6  // Note: each time we generate a random sample, it’s likely that we will get a different set of elements each time |

1. seq()

seq() function generates a sequence of numbers.

The basic syntax for seq() function is as follows:

|  |
| --- |
| seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)), length.out = NULL, along.with = NULL, ...)  seq.int(from, to, by, length.out, along.with, …)  seq\_along(along.with)  seq\_len(length.out) |

* from, to: begin and end number of the sequence
* by: step, increment (Default is 1)
* length.out: length of the sequence
* along.with: take the length from the length of this argument

Example:

|  |
| --- |
| seq(0, 1, length.out = 11) # [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 seq(1, 9, by = 2) // matches end # [1] 1 3 5 7 9 |

1. rep()

rep() function replicates numeric values, or text, or the values of a vector for a specific number of times.

The basic syntax for rep() is as follows:

|  |
| --- |
| rep(x, ...) rep.int(x, times) rep\_len(x, length.out) |

* x: numeric vector, factor, list etc.
* . . . : arguments including times (default = 1), length.out: length of vector,

each (each elements how many times)

Example:

|  |
| --- |
| rep(8,4) # [1] 8 8 8 8 rep(NA,6) # [1] NA NA NA NA NA NA rep(1:5) # [1] 1 2 3 4 5 |

1. round()

**round(x, digits=0)** function rounds to the specified number of decimals.

**signif(x, digits=6)** function rounds to the specified number of significant digits.

**trunc(x, ...)** takes a single numeric argument x and returns a numeric vector containing the integers formed by truncating the values in x toward 0.

**ceiling(x)** function returns the smallest integers larger than the parameter.

**floor(x)** function returns the largest integer not greater than the giving number.

Example:

|  |
| --- |
| x <- rnorm(1) // generate a random float number x # [1] 0.7222297 floor(x) # [1] 0 ceiling(x) # [1] 1 round(x) # [1] 1 round(x,3) # [1] 0.722 signif(x) # [1] 0.72223 signif(x,3) # [1] 0.722 y <- trunc(x) y # [1] 0 |

1. factorial()

factorial(x) function computes the factorial of x, x!. factorial(x) = gamma(x+1)

lfactorial(x) function computes the natural logarithm of factorial of x, ln(x!).

* x: numeric vector

Example:

|  |
| --- |
| factorial(2) // = 2 × 1 # [1] 2 factorial(1) // = 1 × 1 # [1] 1 factorial(3) // = 3 × 2 × 1 # [1] 6 factorial(4) // = 4 × 3 × 2 × 1 # [1] 24 factorial(c(4,3,2)) # [1] 24 6 2 |

1. is()

is() is a function to test inheritance relationships between an object and a class or

between two classes

The basic syntax for is() is as follows:

|  |
| --- |
| is(object, class2) extends(class1, class2, maybe = TRUE, fullInfo = FALSE) |

* object: any R object.
* class1, class2: the names of the classes between which are relations are to be examined, defined, or (more efficiently) the class definition objects for the classes.
* fullInfo: In a call to extends, with class2 missing, fullInfo is a flag
* maybe: What to return for conditional inheritance. But such relationships are rarely used and not recommended, so this argument should not be needed.

is() is a function to check whether its argument is a primitive or a function.

The basic syntax for is() is as follows:

|  |
| --- |
| is.function(x) is.primitive(x) |

* x: an R object

Example:

|  |
| --- |
| is.function(1)  # [1] FALSE is.function (is.primitive)  # [1] TRUE  is.primitive(is.primitive)  # [1] FALSE  is.primitive(is.function)  # [1] TRUE |

1. mean()

mean() function calculates the arithmetic mean (average value).

The basic syntax for mean() is as follows:

|  |
| --- |
| mean(x, trim = 0, na.rm = FALSE, ...) |

* x: numeric vector
* trim: trim off a fraction at each end of the vector, default is 5%
* na.rm: whether NA should be removed, if not, NA will be returned

Example:

|  |
| --- |
| Val # 1 2 3 4 5  mean(Val) # 3 |

1. set.seed()

set.seed() function is the recommended way to specify seeds.

R has many functions for generating random data, such as rnorm(), rexp(), rpois() etc. However, the random data generated will be different each time. This will cause analysis results based on these random datasets not repeatable.

Example:

|  |
| --- |
| set.seed(20) x <- rnorm(20) x # [1] 1.16268529 -0.58592447 1.78546500 -1.33259371 -0.44656677 0.56960612 # [7] -2.88971761 -0.86901834 -0.46170268 -0.55554091 -0.02013537 -0.15038222 # [13] -0.62812676 1.32322085 -1.52135057 -0.43742787 0.97057758 0.02822264 # [19] -0.08578219 0.38921440 set.seed(20) y <- rnorm(20) y # [1] 1.16268529 -0.58592447 1.78546500 -1.33259371 -0.44656677 0.56960612 # [7] -2.88971761 -0.86901834 -0.46170268 -0.55554091 -0.02013537 -0.15038222 # [13] -0.62812676 1.32322085 -1.52135057 -0.43742787 0.97057758 0.02822264 # [19] -0.08578219 0.38921440 x == y # [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE # [16] TRUE TRUE TRUE TRUE TRUE mean(x) # [1] -0.1877639 mean(y) # [1] -0.1877639 |

5. Subset

subset() function returns subsets of vectors, matrices or data frames which meet conditions.

The basic syntax for subset() is as follows:

|  |
| --- |
| subset(x, ...) ## Default S3 method: subset(x, subset, ...) ## S3 method for class 'matrix' subset(x, subset, select, drop = FALSE, ...) ## S3 method for class 'data.frame' subset(x, subset, select, drop = FALSE, ...) |

* x: object to be subsetted
* subset: logical expression indicating elements or rows to keep: missing values are taken as false
* select: expression, indicating columns to select from a data frame
* drop: passed on to [ indexing operator

Example:

|  |
| --- |
| BOD # Time demand ## 1 1 8.3 ## 2 2 10.3 ## 3 3 19.0 ## 4 4 16.0 ## 5 5 15.6 ## 6 7 19.8  subset(BOD,select="Time") # Time ## 1 1 ## 2 2 ## 3 3 ## 4 4 ## 5 5 ## 6 7  subset(BOD,demand<16, select="demand") # demand ## 1 8.3 ## 2 10.3 ## 5 15.6 |

6. Program to calculate BMI (Body Mass Index) get user input and result should be integer

Code:

|  |
| --- |
| user.height <- readline(prompt="Enter height in meter: ")  user.weight <- readline(prompt="Enter weight in kilogram: ")  user.BMI <- user.weight/(user.height^2)  cat("Your BMI rate is: ", as.integer(user.BMI)) |

Output:

|  |
| --- |
| Enter height in meter: 1.7  Enter weight in kilogram: 60  Your BMI rate is: 21 |

7. Create a function to calculate the BMI Rating

Function:

|  |
| --- |
| calculate\_BMI <- function(height, weight) {  bmi = weight/(height^2)  return as.integer(bmi) } |