basic-vector-ops

@MaxMLang (Github)

bin_count

This function counts the number of observations in a specific bin. The user can specify the cutoff points. The function returns a named vector.

Inputs

- `observations`: A numeric vector containing observations
- `cuts`: A numeric vector with cutoff points
- `binnames`: A character vector specifying the labels for the bins in the output

Output

- A named vector with the number of observations in each bin.

Code

```
bin_count <- function(observations, cuts, binnames) {
  cumobs <- vapply(c(cuts, Inf), function(x) sum(observations < x), 0)
  result <- diff(c(0, cumobs))
  names(result) <- binnames
  result
}
```

Worked example

```
observations <-  c(18, 22, 28)
cuts  <- c(15, 19.5, 24)
binnames <-  c("Severely underweight", "Underweight", "Healthy", "Overweight")
bin_count(observations, cuts, binnames)
```

```
## Severely underweight          Underweight              Healthy
##                    0                    1                    1
##           Overweight
##                    1
```

bin_data

This function bins each data point into a category / bin. The user can specify the cutoff points. The function returns a named vector.

Inputs

- `observations`: A numeric vector containing observations
- `cuts`: A numeric vector with cutoff points
- `binnames`: A character vector specifying the labels for the bins in the output

Output

- A named vector with length of the input data that is now binned.

```
bin_data <- function(observations, cuts, binnames) {
  cut(observations, breaks = c(-Inf, cuts, Inf), labels = binnames, right = FALSE, ordered_result = TRU
}
```

Worked Example

```
observations <- c(20, 23, 28)
cuts <- c(16, 18.5, 25)
binnames <-  c("Severely underweight", "Underweight", "Healthy", "Overweight")
bin_data(observations, cuts, binnames)
```

```
## [1] Healthy    Healthy    Overweight
## Levels: Severely underweight < Underweight < Healthy < Overweight
```

`mark_divisible`

This function marks every number that is divisible by the specified the divisor up to the specified length of the sequence.

Inputs

- `up.to`: A numeric vector specifying the end of the sequence
- `divisor`: A numeric vector specifying the divisor
- `marker`: A character vector specifying the marker

Output

- A named vector with length of the input data that is now binned.

```
mark_divisible <- function(up.to, divisor = 2, marker = "even") {
  # your code
  sequence <- seq_len(up.to)
  ret <- as.character(sequence)
  is.divisible <- sequence %% divisor == 0
  ret[is.divisible] <- marker
  ret
}
```

Worked Example

```
up.to <- 50
divisor <- 9
marker <-  "X"
mark_divisible(up.to, divisor, marker)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "X"  "10" "11" "12" "13" "14" "15"
## [16] "16" "17" "X"  "19" "20" "21" "22" "23" "24" "25" "26" "X"  "28" "29" "30"
## [31] "31" "32" "33" "34" "35" "X"  "37" "38" "39" "40" "41" "42" "43" "44" "X"
## [46] "46" "47" "48" "49" "50"
```

`find_fibonacci`

Finds the n th Fibonacci number. It is a recursive approach. ## Inputs * `n` a non-negative scalar integer value

Output

- scalar value: the n'th Fibonacci number.

```
find_fibonacci <- function(n) {
  assertCount(n, tol = 1e-100)
  if (n <= 1) {
    return(n)
  }
  find_fibonacci(n - 1) + find_fibonacci(n - 2)
}
```

Worked example

```
find_fibonacci(5)
```

```
## [1] 5
```

```
find_fibonacci(14)
```

```
## [1] 377
```

`vec_threshold`

Returns all elements of the vector that are at least onee number greater than the threshold.

Inputs

- `vectors`: a list of `numeric` vectors
- `threshold`: a scalar numeric

Outputs

Character vector containing all elements from `vectors` that contain at least one number greater than `threshold`. # # Examples: # ex03VectorThreshold(# list(numeric(0), 1:3, 8:11, c(-100, 100)), # threshold = 10) # –> list(8:11, c(-100, 100)) # ex03VectorThreshold(list(), 10) –> list() # ex03VectorThreshold(list(numeric(0), 0, -1), 10) –> list() # ex03VectorThreshold(10, 10) –> ERROR (`vectors` is not a list of numerics) # ex03VectorThreshold(list(10), "10") –> ERROR (threshold is not a numeric) # # You may want to use `Filter`, and the solution of ex02VectorCondition may be useful here.

```
vec_threshold <- function(vectors, threshold) {
  assertList(vectors, any.missing = FALSE)
  Filter(vec_condition(threshold), vectors)
}
```

```
# helper function
vec_condition <- function(threshold) {
  assertNumber(threshold)
  function(vect) {
    assertNumeric(vect, any.missing = FALSE)
    any(vect > threshold)
  }
}
```

Worked example

```
vec_threshold(list(numeric(0), 1:3, 8:11, c(-100, 100)), threshold = 10)
```

```
## [[1]]
## [1]  8  9 10 11
##
## [[2]]
## [1] -100  100
```

```
vec_threshold(list(numeric(0), 0, -1), 10)
```

```
## list()
```