Lecture 12 - Loops - Apply Family

Packages

```
# none
```

Apply family of functions

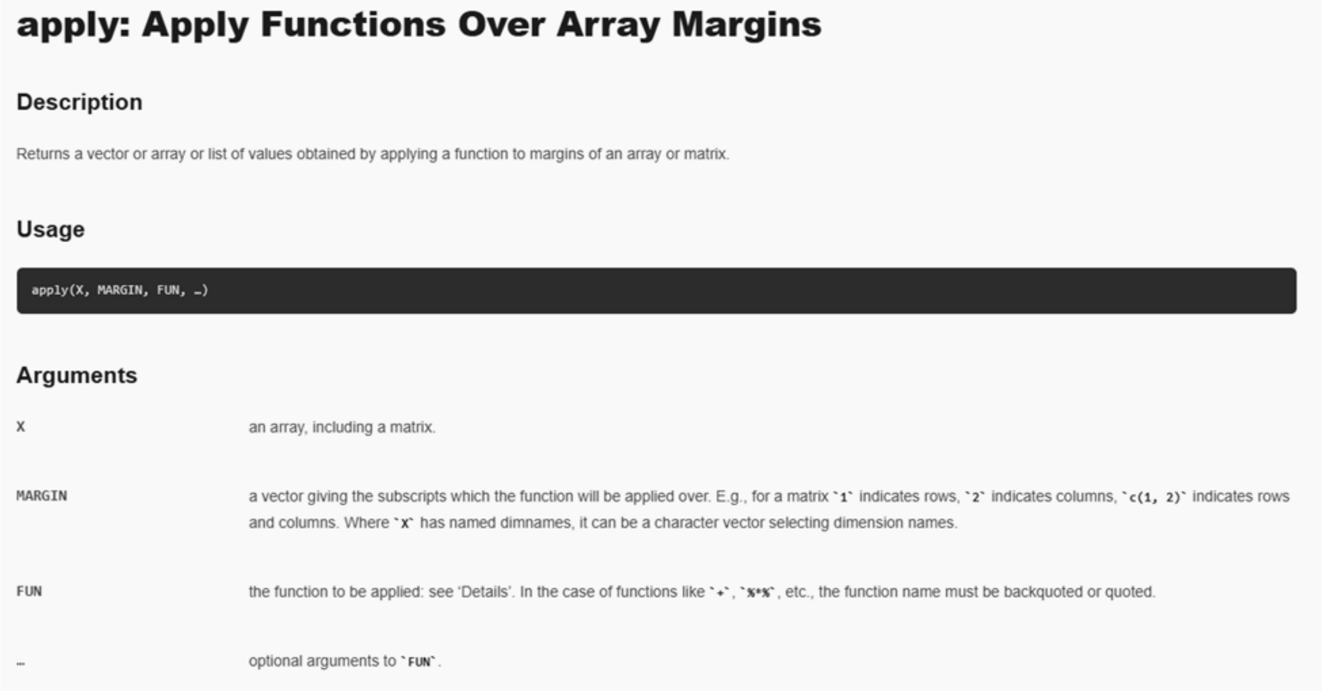
- The prior lecture covered loops, which is a mechanism used in many other languages
- R conveniently has a family of functions called the `apply` family, which performs implicit looping
- Most of the time, we can use these functions instead of creating our own for loops!

- Today, we will cover the following four apply family functions
  - `apply()`
    - It takes a data structure, a margin (specifying whether to operate along rows or columns), and a function to apply. It returns a vector or matrix depending on the input and the margin.
  - `lapply()` - List Apply
    - This function applies a function to each element of a list, returning a list of the results. It maintains the structure of the input list.
  - `sapply()` - Simple Apply
    - Similar to lapply, but it attempts to simplify the output if possible, returning a vector or matrix instead of a list. This is useful when the output of the function is consistent across all elements.

  - `tapply` - Table Apply
    - This function applies a function to elements grouped by a factor or categorical variable, returning a table of results.
- We will also cover the powerful `do.call()` function

`apply()`

- The `apply()` function applies a function to the rows or the columns of a matrix or dataframe and outputs a vector or list



apply: Apply Functions Over Array Margins

Description

Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

Usage

```
apply(X, MARGIN, FUN, …)
```

Arguments

X	an array, including a matrix.
MARGIN	a vector giving the subscripts which the function will be applied over. E.g., for a matrix `1` indicates rows, `2` indicates columns, `c(1, 2)` indicates rows and columns. Where `X` has named dimnames, it can be a character vector selecting dimension names.
FUN	the function to be applied: see 'Details'. In the case of functions like `+`, `%*%`, etc., the function name must be backquoted or quoted.
…	optional arguments to `FUN`.

- Find the mean of each row of the `mtcars` data frame

```
# view first few lines of data frame
head(mtcars)
```

A data.frame: 6 × 11

```
            mpg   cyl   disp   hp   drat   wt   qsec   vs   am   gear
```

	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<(
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	
Hornet											

```
# find mean of each row, 1 = row
rowmeans <- apply(mtcars, 1, mean)
print(rowmeans)
```

```
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive 
           29.90727            29.98136            23.59818            38.73955 
  Hornet Sportabout             Valiant          Duster 360           Merc 240D 
           53.66455            35.04909            59.72000            24.63455 
           Merc 230            Merc 280           Merc 280C          Merc 450SE 
           27.23364            31.86000            31.78727            46.43091 
         Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln Continental 
           46.50000            46.35000            66.23273            66.05855 
  Chrysler Imperial            Fiat 128         Honda Civic      Toyota Corolla 
           65.97227            19.44091            17.74227            18.81409 
      Toyota Corona    Dodge Challenger         AMC Javelin          Camaro Z28 
           24.88864            47.24091            46.00773            58.75273 
   Pontiac Firebird           Fiat X1-9       Porsche 914-2        Lotus Europa 
           57.37955            18.92864            24.77909            24.88027 
     Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E 
           60.97182            34.50818            63.15545            26.26273 
```

Use `data.frame()` to convert the output vector to a dataframe.

```
# find mean of each row
rowmeans <- apply(mtcars, 1, mean)
rowmeans <- data.frame(rowmeans)
head(rowmeans)
```

A data.frame: 6 × 1

	rowmeans
	<dbl>
Mazda RX4	29.90727
Mazda RX4 Wag	29.98136

Mazda RX4 Wag	29.98136
Datsun 710	23.59818
Hornet 4 Drive	38.73955
Hornet Sportabout	53.66455
Valiant	35.04909

- Note that the above doesn't make much sense!
- It makes more sense to average each column of the data frame since values within a column are the same units of measurement

```
# find mean of each column, 2 = column
colmeans <- apply(mtcars, 2, mean)
print(colmeans)
```

```
       mpg        cyl       disp         hp       drat         wt       qsec 
 20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750 
        vs         am       gear       carb 
  0.437500   0.406250   3.687500   2.812500 
```

Use `data.frame()` to convert the output vector to a dataframe.

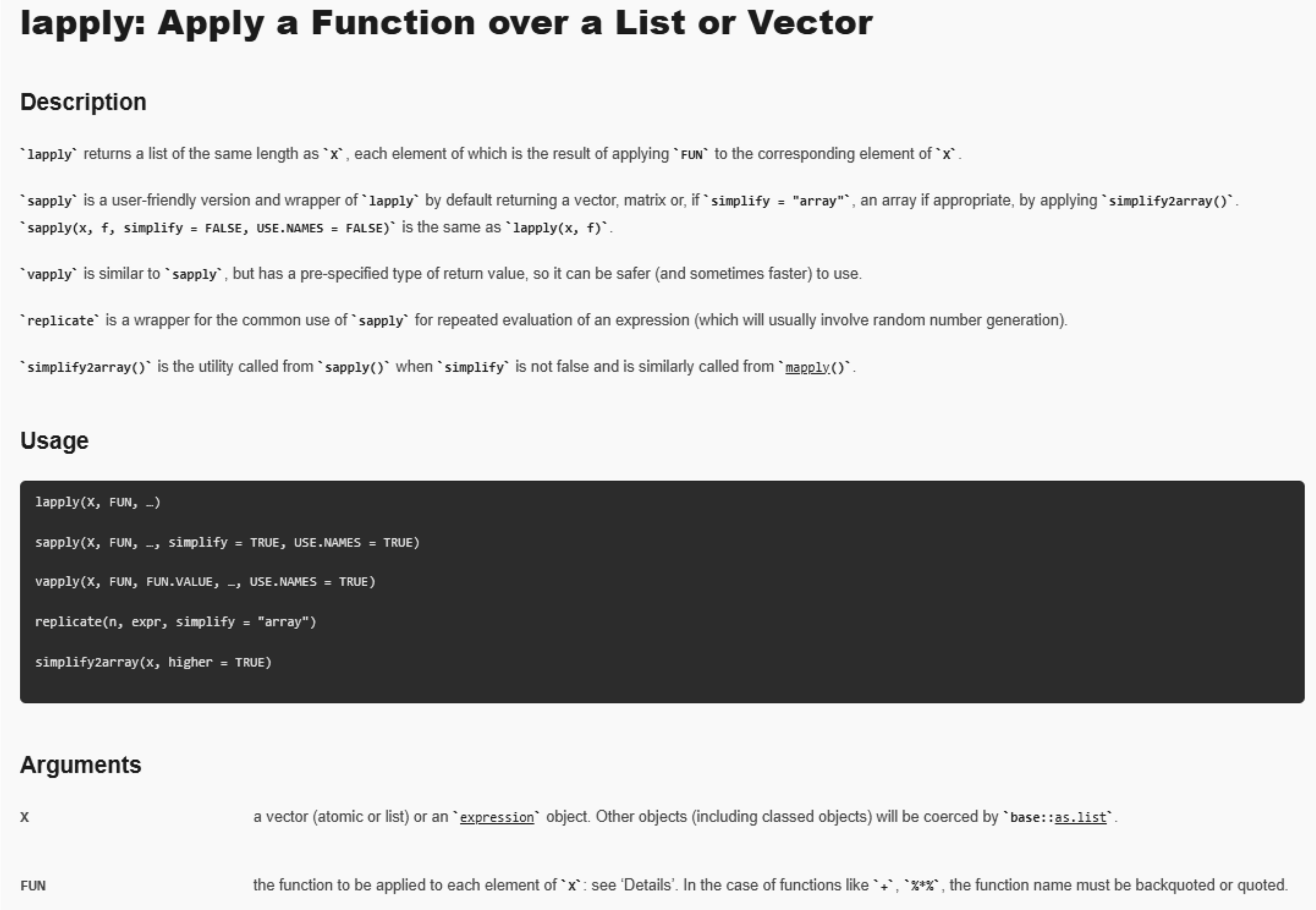
```
# find mean of each column
colmeans <- apply(mtcars, 2, mean)
colmeans <- data.frame(colmeans)

head(colmeans)
```

A data.frame: 6 × 1

	colmeans
	<dbl>
mpg	20.090625
cyl	6.187500
disp	230.721875
hp	146.687500
drat	3.596563
wt	3.217250

lapply()

- The `lapply()` function applies a function to each element of a list and outputs another list



lapply: Apply a Function over a List or Vector

Description

`lapply` returns a list of the same length as `X`, each element of which is the result of applying `FUN` to the corresponding element of `X`.

`sapply` is a user-friendly version and wrapper of `lapply` by default returning a vector, matrix or, if `simplify = "array"`, an array if appropriate, by applying `simplify2array()`. `sapply(x, f, simplify = FALSE, USE.NAMES = FALSE)` is the same as `lapply(x, f)`.

`vapply` is similar to `sapply`, but has a pre-specified type of return value, so it can be safer (and sometimes faster) to use.

`replicate` is a wrapper for the common use of `sapply` for repeated evaluation of an expression (which will usually involve random number generation).

`simplify2array()` is the utility called from `sapply()` when `simplify` is not false and is similarly called from `mapply()`.

Usage

```
lapply(X, FUN, …)

sapply(X, FUN, …, simplify = TRUE, USE.NAMES = TRUE)

vapply(X, FUN, FUN.VALUE, …, USE.NAMES = TRUE)

replicate(n, expr, simplify = "array")

simplify2array(x, higher = TRUE)
```

Arguments

X a vector (atomic or list) or an `expression` object. Other objects (including classed objects) will be coerced by `base::as.list`.

FUN the function to be applied to each element of `X`: see 'Details'. In the case of functions like `+`, `%*%`, the function name must be backquoted or quoted.

- Remember that a dataframe is a structured list!
- We can use `lapply()` find the mean of each variable in the data frame

```
# Find th mean of each column
lapply(mtcars, mean)
```

```
$mpg
    20.090625
```

```
$cyl
        6.1875
$disp
        230.721875
$hp
        146.6875
$drat
        3.5965625
$wt
        3.21725
$qsec
        17.84875
$vs
        0.4375
$am
        0.40625
$gear
        3.6875
$carb
        2.8125
```

```
# Converting unstructured list to a data frame
data.frame(lapply(mtcars, mean))
```

A data.frame: 1 × 11

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	g
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<d
20.09062	6.1875	230.7219	146.6875	3.596563	3.21725	17.84875	0.4375	0.40625	3.6

---

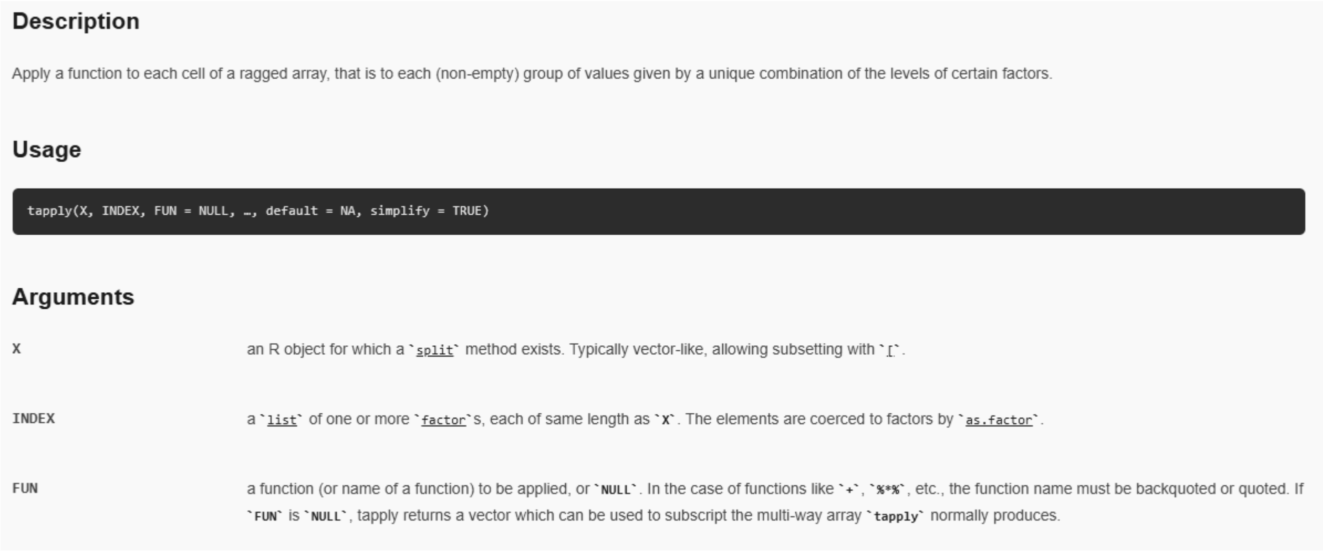
sapply()

[ ] ↳ 7 cellules masquées

tapply()

- The `tapply()` functions applies a function for each factor level in a variable
- This function is very useful!

tapply: Apply a Function Over a Ragged Array



Description

Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

Usage

```
tapply(X, INDEX, FUN = NULL, …, default = NA, simplify = TRUE)
```

Arguments

X	an R object for which a `split` method exists. Typically vector-like, allowing subsetting with `[`.
INDEX	a `list` of one or more `factor`s, each of same length as `X`. The elements are coerced to factors by `as.factor`.
FUN	a function (or name of a function) to be applied, or `NULL`. In the case of functions like `+`, `%*%`, etc., the function name must be backquoted or quoted. If `FUN` is `NULL`, tapply returns a vector which can be used to subscript the multi-way array `tapply` normally produces.

- For example, we can use `tapply()` to calculate the average mpg for each cylinder engine

```
# means for each cylinder
cyl_means <- tapply(mtcars$mpg, mtcars$cyl, mean)
print(cyl_means)
```

```
       4        6        8 
26.66364 19.74286 15.10000 
```

Custom Functions and the `apply()` Family

- Thus far, all we have done is calculate averages
- However, the `apply()` family can use different functions as well...even custom functions!

- The major challenge is understanding
    1. the appropriate inputs to the function
    2. the expected output of the function

Examples

- Finding the maximum of each variable using `apply()`

```
# find maximum of each variable
print(apply(mtcars, 2, max))
```

```
    mpg     cyl    disp      hp    drat      wt    qsec      vs      am    gear 
 33.900   8.000 472.000 335.000   4.930   5.424  22.900   1.000   1.000   5.000 
   carb 
  8.000 
```

- Finding the minimum of each variable using `lapply()`

```
lapply(mtcars, max)
```

```
$mpg
     33.9
$cyl
     8
$disp
     472
$hp
     335
$drat
     4.93
$wt
     5.424
$qsec
     22.9
$vs
     1
$am
     1
$gear
     5
$carb
     8
```

- Finding the median of each variable using `sapply()`

```
print(sapply(mtcars, max))
```

```
    mpg     cyl    disp      hp    drat      wt    qsec      vs      am    gear 
 33.900   8.000 472.000 335.000   4.930   5.424  22.900   1.000   1.000   5.000 
   carb 
  8.000 
```

- Finding the quantiles of the `mpg` variable by the number of cylinders using `tapply()`

```
print(tapply(mtcars$mpg, mtcars$cyl, quantile))
```

```
$`4`
  0%  25%  50%  75% 100% 
21.4 22.8 26.0 30.4 33.9 

$`6`
   0%   25%   50%   75%  100% 
17.80 18.65 19.70 21.00 21.40 

$`8`
   0%   25%   50%   75%  100% 
10.40 14.40 15.20 16.25 19.20 
```

Custom Function Example

- We can build our own function to summarize the data
- For example, `mean()`, `min()`, and `max()` are built-in functions in R, but we can build our own functions to be used by the apply family

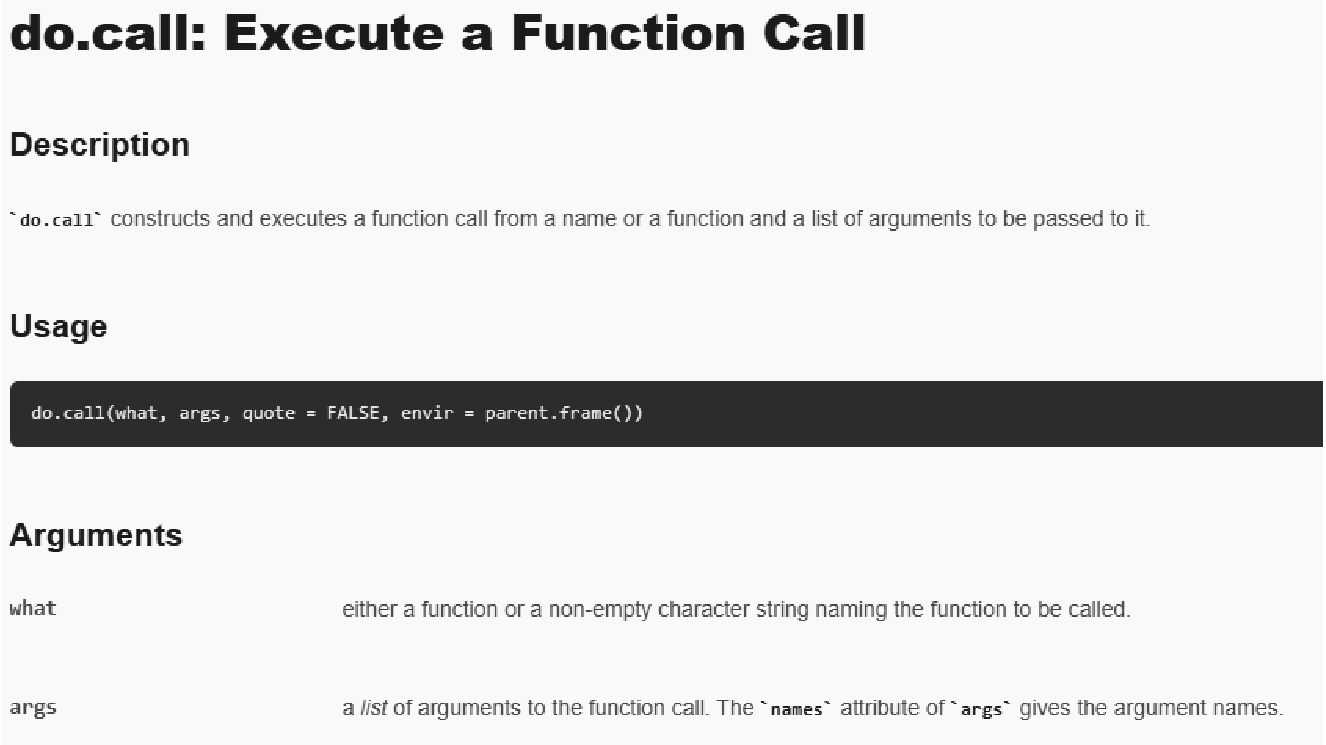
```
# define our own function
vector_summary <- function(x) {

  # calculate descriptive statistics
  mean_out <- mean(x)
  min_out  <- min(x)
  max_out  <- max(x)

  # combine into a vector
  output <- c(mean_out, min_out, max_out)

  # return summary
  # return summary
  return(output)
}


# apply function to mpg for each unique cylinder
mpg_summary <- tapply(mtcars$mpg, mtcars$cyl, vector_summary)
mpg_summary
```

```
$`4`
      26.6636363636364 · 21.4 · 33.9
$`6`
      19.7428571428571 · 17.8 · 21.4
$`8`
      15.1 · 10.4 · 19.2
```

---

do.call()

- The function `do.call()` can be used to combine elements of a list using a specified function
- For example, if we want to concatenate the elements of a list



do.call: Execute a Function Call

Description

`do.call` constructs and executes a function call from a name or a function and a list of arguments to be passed to it.

Usage

```
do.call(what, args, quote = FALSE, envir = parent.frame())
```

Arguments

`what`	either a function or a non-empty character string naming the function to be called.
`args`	a *list* of arguments to the function call. The `names` attribute of `args` gives the argument names.

- For example, we can use `do.call()` to combine the following output into a data summary table

```
# summary table
mpg_summary <- tapply(mtcars$mpg, mtcars$cyl, vector_summary)
mpg_summary
```

```
$`4`
     26.6636363636364 · 21.4 · 33.9
$`6`
     19.7428571428571 · 17.8 · 21.4
$`8`
     15.1 · 10.4 · 19.2
```

```
# row bind each summary in the unstructured list
mpg_summary <- do.call(rbind, mpg_summary)
mpg_summary
```

```
 A matrix: 3 × 3 of type dbl
4  26.66364  21.4  33.9
6  19.74286  17.8  21.4
8  15.10000  10.4  19.2
```

- We can now continue organizing the output for viewing

```
# convert to a data frame
mpg_summary <- data.frame(mpg_summary)
mpg_summary
```

```
      A data.frame: 3 × 3
          X1     X2     X3
       <dbl>  <dbl>  <dbl>
4   26.66364   21.4   33.9
6   19.74286   17.8   21.4
8   15.10000   10.4   19.2
```

```
# change column names
names(mpg_summary) <- c("mean", "min", "max")
```

```
mpg_summary
```

```
A data.frame: 3 × 3
        mean   min    max
       <dbl>  <dbl>  <dbl>
4   26.66364   21.4   33.9
6   19.74286   17.8   21.4
8   15.10000   10.4   19.2
```

```
# change row names
rownames(mpg_summary) <- paste(c(4, 6, 8), "cyl", sep="")
mpg_summary
```

```
A data.frame: 3 × 3
           mean   min    max
          <dbl>  <dbl>  <dbl>
4cyl   26.66364   21.4   33.9
6cyl   19.74286   17.8   21.4
8cyl   15.10000   10.4   19.2
```