SIT718 Real World Analytics

Lecturer: Dr Ye Zhu

School of Information Technology
Deakin University

Week 2: Introduction to Data Analysis

SUMMARISING AND AGGREGATING DATA IN R

Learning aims:

- Data types
- Data summarisation;
- Data aggregation;
- Understand the properties and notation used for aggregation functions
- Develop intuition and understanding in when to use the arithmetic, geometric or harmonic mean to find the 'average'
- Begin to build proficiency in writing functions in R

Read Chapter 1 of the reference book (An Introduction to Data Analysis using Aggregation Functions in R)

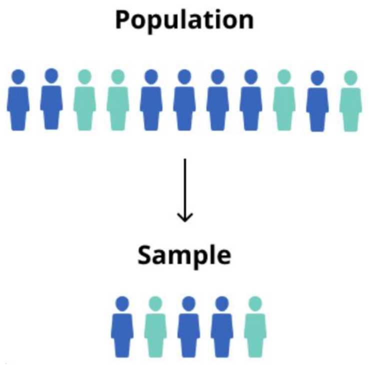
Overview of Using Data

- **Data**: The facts and figures collected, analysed, and summarised for presentation and interpretation.
- **Variable**: A characteristic or a quantity of interest that can take on different values.
- **Observation**: A set of values corresponding to a set of variables.
- **Variation**: The difference in a variable measured over observations.
- **Random variable/uncertain variable**: A quantity whose values are not known with certainty.

Types of Data

Population and Sample Data:

- **Population**: All elements of interest.
- **Sample**: Subset of the population.
    - **Random sampling**: A sampling method to gather a representative sample of the population data.

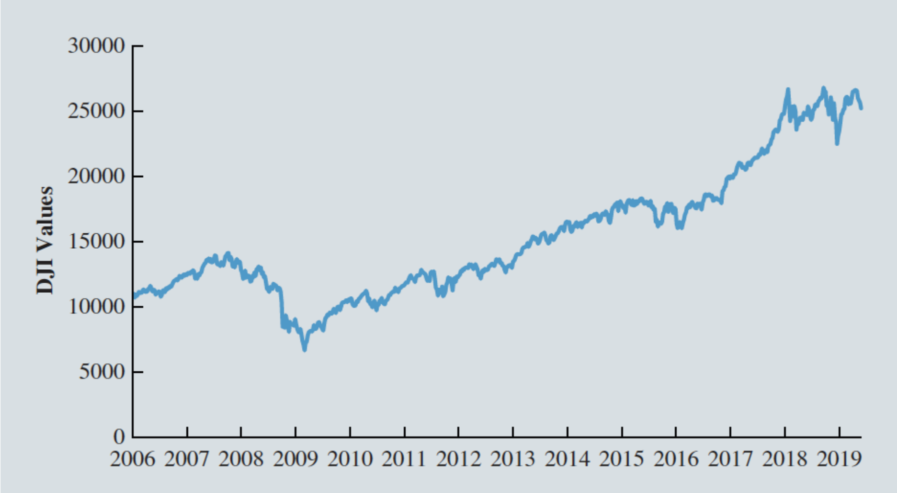


Quantitative and Categorical Data:

- **Quantitative data**: Data on which numeric and arithmetic operations, such as addition, subtraction, multiplication, and division, can be performed (e.g. height, weight, or age).
- **Categorical data**: Data on which arithmetic operations cannot be performed. The data represent groups includes rankings (e.g. finishing places in a race), classifications (e.g. brands of cereal), and binary outcomes (e.g. coin flips).

Cross-Sectional and Time Series Data:

- **Cross-sectional data**: Data collected from several entities at the same, or approximately the same, point in time.
- **Time series data**: Data collected over several time periods.
  - Graphs of time series data are frequently found in business and economic publications.
  - Graphs help analysts understand what happened in the past, identify trends over time, and project future levels for the time series.



Sources of Data:

- Experimental study:
  - A variable of interest is first identified.
  - Then one or more other variables are identified and controlled or manipulated so that data can be obtained about how they influence the variable of interest.
- Nonexperimental study or observational study:
  - Makes no attempt to control the variables of interest.
  - A survey is perhaps the most common type of observational study.

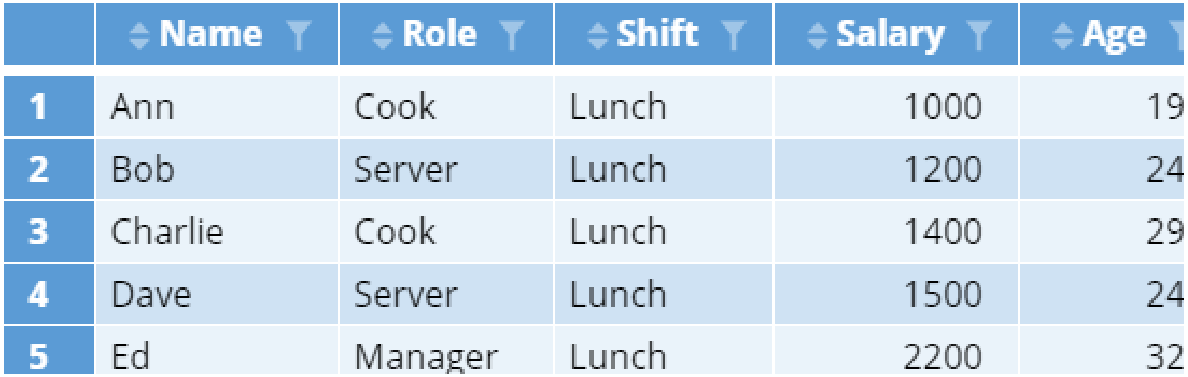
Aggregate function

In database management, an aggregate function or aggregation function is a function where the values of multiple rows are grouped together to form a single summary value. (Wikipedia)

- Mean
- Maximum
- Minimum
- Median
- Mode
- Range
- Sum

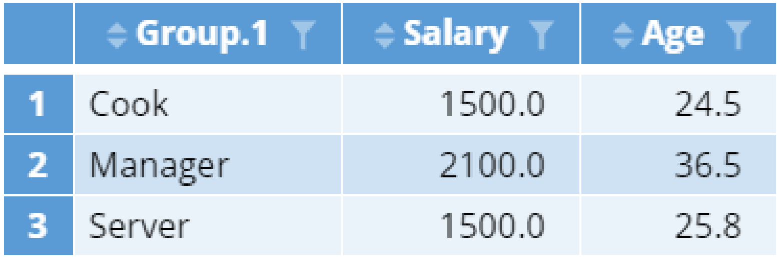
Example

A table consists of different employees at a restaurant:



	Name	Role	Shift	Salary	Age
1	Ann	Cook	Lunch	1000	19
2	Bob	Server	Lunch	1200	24
3	Charlie	Cook	Lunch	1400	29
4	Dave	Server	Lunch	1500	24
5	Ed	Manager	Lunch	2200	32

A table of the average salary and age by role:



	Group.1	Salary	Age
1	Cook	1500.0	24.5
2	Manager	2100.0	36.5
3	Server	1500.0	25.8

Source: https://www.r-bloggers.com/how-to-aggregate-data-in-r/

The Arithmetic Mean

The **arithmetic mean** is at the heart of the majority of aggregation processes. It is also referred to as ‘the average’, ‘the statistical mean’ or just ‘the mean’.

Formula (1)

$$AM(x_1, x_2, \ldots, x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

The Arithmetic Mean

Example

Let $\mathbf{x} = (3, 2, 9, 8)$.

(i) *What is the value of* n*?*

(ii) *Calculate* $AM(\mathbf{x})$.

(iii) *What will be the output if* $\mathbf{x} = (9, 8, 2, 3)$*?*

The Arithmetic Mean

One of our aims in this sequence of units is to be aware of when the arithmetic mean is inappropriate for summarising data, however it is still the main *superstar* when it comes to aggregation.

THE ARITHMETIC MEAN: PROPERTIES

Some properties:

- The order of arguments doesn't affect the output (symmetry)
- If you add a constant to every value and take the average, the output will change by that same constant, e.g. $AM(4, 3, 8) = 5, AM(7, 6, 11) = 8$ (translation invariant),
- If you multiply every input by a constant, the output will change by the same factor, e.g. $AM(3, 7, 8) = 6, AM(15, 35, 40) = 30$ (homogeneous).
- Increasing one of the inputs always increases the output, e.g. $AM(40,\ 57,\ 30) > AM(40,\ 57,\ 29)$, (monotone increasing)
- If all the inputs are the same then the output will be the same too, $AM(5, 5, 5, 5) = 5$ (idempotent).

OTHER MEANS AND AVERAGES

However, despite its usefulness and relative robustness when summarising data, the Arithmetic mean is not always the best choice.

Other means and averages

Formula (2)

$$\text{Median}(\mathbf{x}) = \begin{cases} x_{(k)} \;, & n = 2k - 1 \;, \\ \frac{x_{(k)} + x_{(k+1)}}{2} \;, & n = 2k \end{cases}$$

where $(\cdot)$ *means we first rearranged the data in order (either increasing or decreasing).*

The cases $n = 2k\text{-}1$ and $n = 2k$ refer to whether there is an odd or even number of data to aggregate.

Other means and averages

Example

Let $\mathbf{x} = (4, 6, 9, 1)$ *and* $\mathbf{y} = (2, 6, 9, 4, 106)$. *Calculate* $\text{Median}(\mathbf{x})$ *and* $\text{Median}(\mathbf{y})$.

Other means and averages

In addition, sometimes it just isn't appropriate to use the arithmetic mean to find the 'average'. Consider the following scenarios:

- Your pay goes up by 10% one year and 20% the next. What is the average pay increase?
- Vicky can paint a house in 3 hours, Simon can paint one in 5. How long would it take them to paint two houses if they work together?

Other means and averages

Example

Your pay goes up by 20% one year and 10% the next. What is the average pay increase?

It's not 15%.
e.g. your salary is $10000. First increase $12000. Second increase $13200. If it went up by 15%, then it would be $11500 and then $ 13225.

Other means and averages

Example

Vicky can paint a house in 3 hours, Simon can paint one in 5. How long would it take them to paint two houses if they work together?

Interesting video: (little big league math) https://www.youtube.com/watch?v=pXtFSE7VlL0

It's not 8 hours. It's not even 4 hours.
In 8 hours, Vicky would paint more than 2 houses by herself. In 4 hours, Vicky can paint 1.3 houses, while Simon has painted 0.8.

This would put us over two houses.

OTHER MEANS AND AVERAGES

These are two examples of where the 'average' requires more than just adding the numbers together and dividing by how many observations there are. Respectively, they are scenarios requiring the geometric mean $GM(\mathbf{x})$ and the harmonic mean $HM(\mathbf{x})$.

Formula (3) and (4)

$$GM(\mathbf{x}) = \left(\prod_{i=1}^{n} x_i\right)^{1/n} = (x_1 x_2 \cdots x_n)^{1/n}.$$

$$HM(\mathbf{x}) = n\left(\sum_{i=1}^{n} \frac{1}{x_i}\right)^{-1} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}}.$$

DEAKIN UNIVERSITY AUSTRALIA Worldly

Other means and averages

Example

Your pay goes up by 20% one year and 10% the next. What is the average pay increase?

$$GM(1.2, 1.1) = \sqrt{1.2 \times 1.1}, \\ = 1.1489...$$

So it's about 14.9%.

- Does this change if it went up by 10% the first year and 20% the second year?
- What about the total amount of money earned?

Other means and averages

Example

Vicky can paint a house in 3 hours, Simon can paint one in 5. How long would it take them to paint two houses if they work together? In other words, how long would it take for someone who works at the average of their pace?

this is an average that is appropriate to calculate using the harmonic mean

$$\begin{aligned} HM(3,5) &= 2 \times \left(\frac{1}{3} + \frac{1}{5}\right)^{-1} \\ &= 2\left(\frac{5+3}{15}\right)^{-1} = 2\left(\frac{15}{8}\right) = \frac{30}{8} \end{aligned}$$

Defining aggregation functions

So what do these functions have in common?
Well, there are two properties that we say classes them as **Aggregation functions**

- Monotone increasing
- Boundary conditions $f(a, a, \ldots, a) = a, f(b, b, \ldots, b) = b$ where a and b are the minimum and maximum values possible.

For example, with the boundary condition, if all our values are over the unit interval $[0, 1]$ then we have $f(0, 0, \ldots, 0) = 0$ and $f(1, 1, \ldots, 1)$.

Defining aggregation functions

But further to this, we actually say these functions all belong to the class of *averaging* aggregation functions. An aggregation function A is averaging if,

$$\min(\mathbf{x}) \leq A(\mathbf{x}) \leq \max(\mathbf{x})$$

i.e. for any input vector, the output will always be between the smallest and largest argument.

Averaging aggregation functions

Averaging behaviour is a justifiable condition we may want if we want our functions to give outputs that 'represent' the inputs. The median is also averaging.
Averaging aggregation functions are always idempotent:

$$A(t, t, t, \ldots, t) = t$$

Again, this is an intuitively appealing requirement when we think about the 'averaging'. For example, with the geometric mean example.
We had $GM(1.2, 1.1) = 1.149$ and idempotency means that $GM(1.149, 1.149) = 1.149$ too.

APPLICATIONS

Other aspects of these operators also make them appealing. For example, the geometric mean is used in ecology research to give an overall idea of abundance.

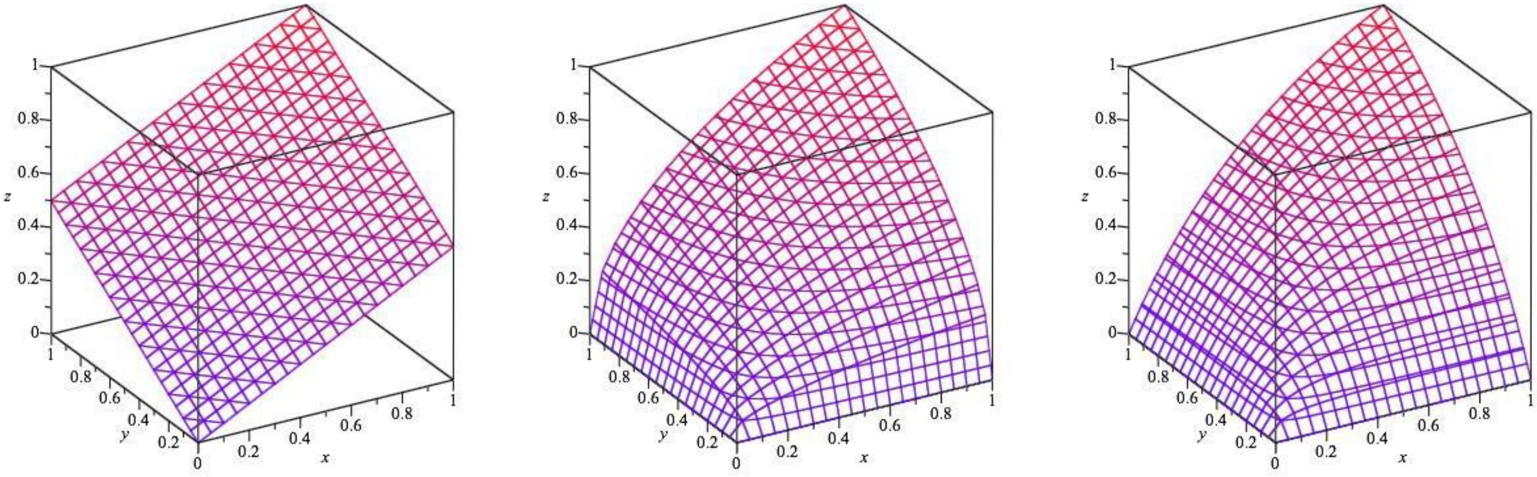
$$GM(\underbrace{s_1, s_2, \ldots, s_n}_{\text{species abundances}}) = \left(\prod_{i=1}^{n} s_i\right)^{\frac{1}{n}}$$

Why?

- The output gets very small if any of the inputs are close to zero. The interpretation here is that we want the function to be more sensitive to increases in rare species.

GRAPHS

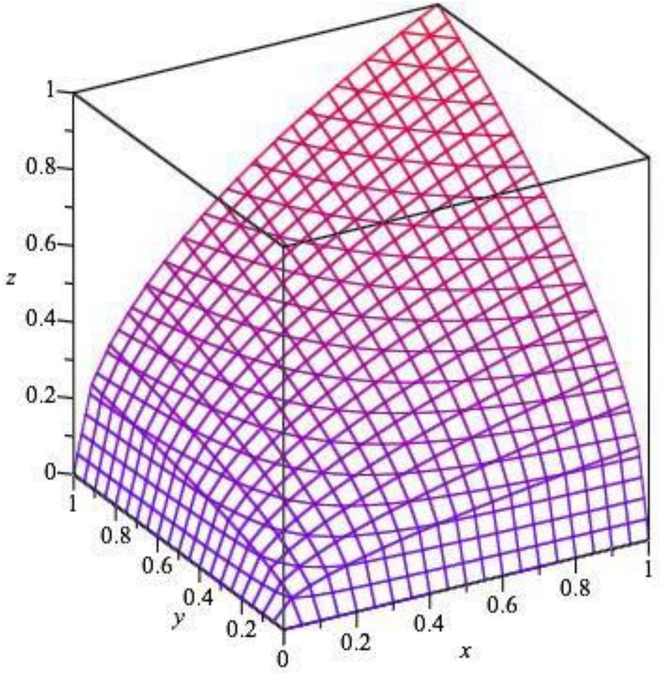
Here are plots of AM, GM, and HM for $\mathbf{x} \in [0,1]^2$.



$$HM(\mathbf{x}) \leq GM(\mathbf{x}) \leq AM(\mathbf{x})$$

LIPSCHITZ CONTINUITY

One thing to be careful of (especially when implementing using software) whenever we use the Geometric mean is that on the boundary the gradient becomes infinitely large.



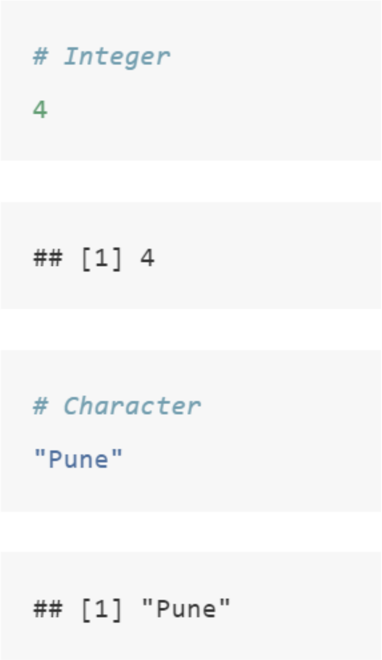
Functions which always have their gradient bounded by some constant are said to be Lipschitz continuous.

Practice Questions

1. Are the harmonic mean and geometric mean homogeneous and translation invariant? How about the median? Hint: you can test them in R
2. If $AM(23, 42, 27, 68) = 40$, without calculating, what will be the value of $AM(29, 48, 33, 74)$? Hint: translation invariant
3. If $AM(x_1, x_2, x_3, x_4) = 15$, what will be the value of $AM(x_1 + 2, x_2 + 2, x_3 + 2, x_4 + 2)$? Hint: translation invariant
4. If $GM(x_1, x_2, x_3, x_4) = 6$, what will be the value of $GM(3x_1, 3x_2, 3x_3, 3x_4)$? Hint: homogeneous
5. Explain how the arithmetic mean, harmonic mean and geometric mean could be affected by outliers.

Vector in R

A vector is a data structure in R. A data structure can be defined as a collection of data values, the relationships among them, and the functions or operations that can be applied to the data. A vector can either be an atomic vector or a list. An atomic vector contains only one data type whereas a list may contain multiple data types. Atomic vector could be Character, Logical, double, integer or complex

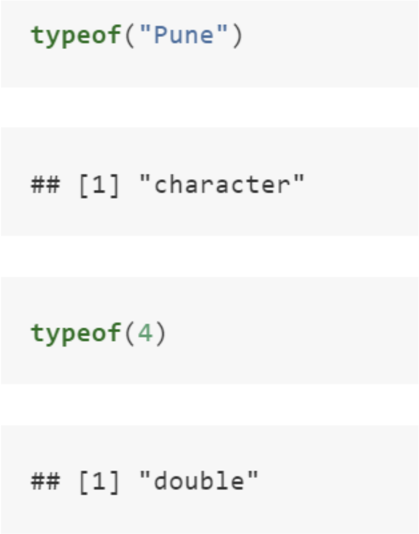


```
# Integer
4
```

```
## [1] 4
```

```
# Character
"Pune"
```

```
## [1] "Pune"
```

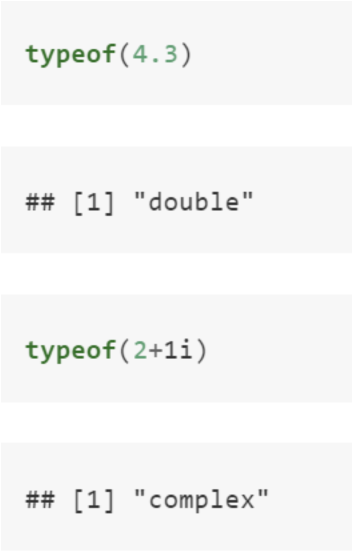


```
typeof("Pune")
```

```
## [1] "character"
```

```
typeof(4)
```

```
## [1] "double"
```



```
typeof(4.3)
```

```
## [1] "double"
```

```
typeof(2+1i)
```

```
## [1] "complex"
```

Since, a vector must have elements of the same type, this function will try and coerce elements to the same type, if they are different.

Coercion is from lower to higher types from logical to integer to double to character.

```
> x <- c(1, 5, 4, 9, 0)
> typeof(x)
[1] "double"
> length(x)
[1] 5
> x <- c(1, 5.4, TRUE, "hello")
> x
[1] "1"       "5.4"    "TRUE"   "hello"
> typeof(x)
[1] "character"
```

If we want to create a vector of consecutive numbers, the `:` operator is very helpful.

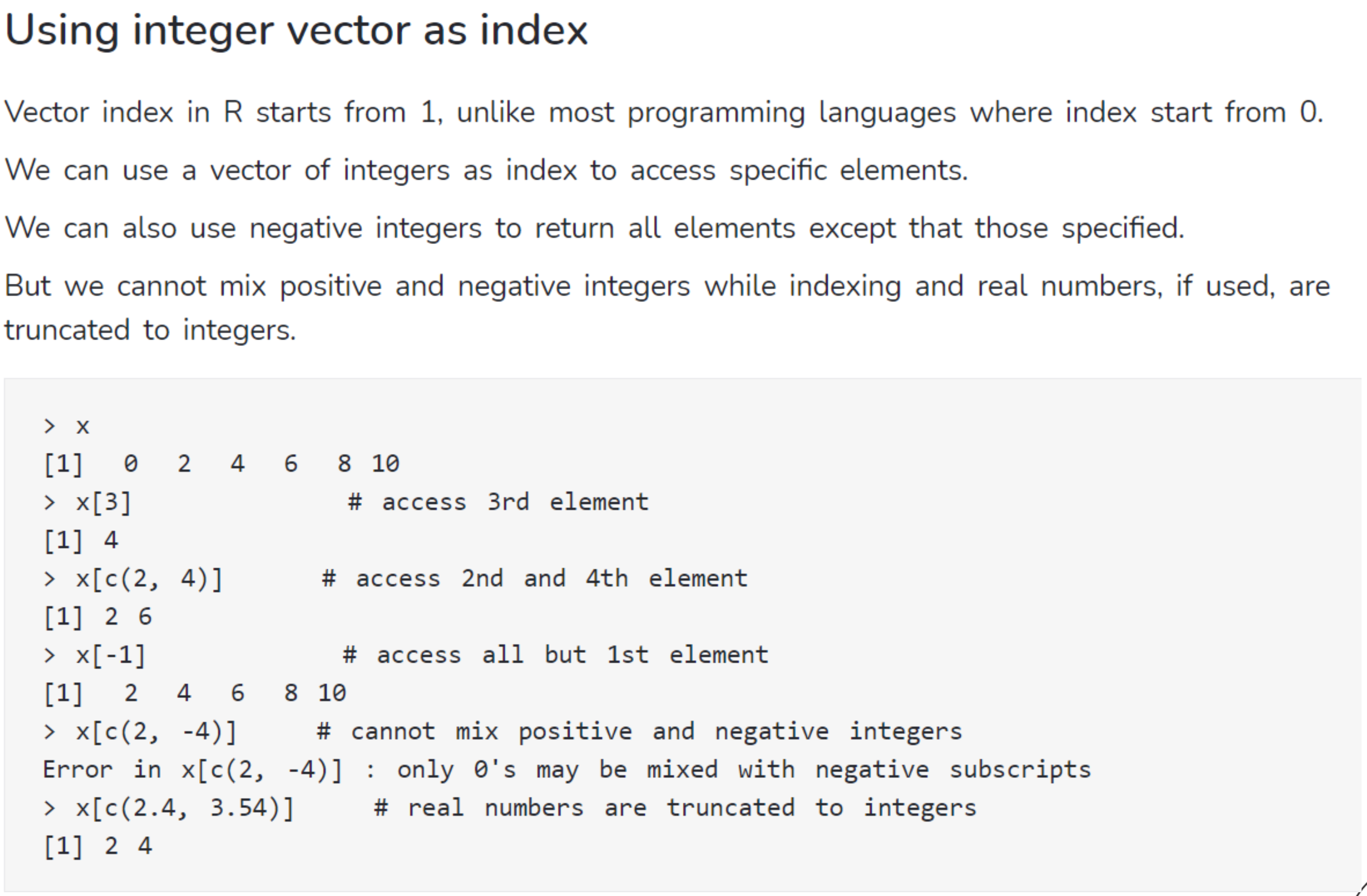
Example 1: Creating a vector using : operator

```
> x <- 1:7; x
[1] 1 2 3 4 5 6 7
> y <- 2:-2; y
[1]  2  1  0 -1 -2
```

More complex sequences can be created using the `seq()` function, like defining number of points in an interval, or the step size.

Example 2: Creating a vector using seq() function

```
> seq(1, 3, by=0.2)            # specify step size
[1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
> seq(1, 5, length.out=4)     # specify length of the vector
[1] 1.000000 2.333333 3.666667 5.000000
```



Using integer vector as index

Vector index in R starts from 1, unlike most programming languages where index start from 0.

We can use a vector of integers as index to access specific elements.

We can also use negative integers to return all elements except that those specified.

But we cannot mix positive and negative integers while indexing and real numbers, if used, are truncated to integers.

```
> x
[1]  0  2  4  6  8 10
> x[3]                # access 3rd element
[1] 4
> x[c(2, 4)]        # access 2nd and 4th element
[1] 2 6
> x[-1]               # access all but 1st element
[1]  2  4  6  8 10
> x[c(2, -4)]       # cannot mix positive and negative integers
Error in x[c(2, -4)] : only 0's may be mixed with negative subscripts
> x[c(2.4, 3.54)]     # real numbers are truncated to integers
[1] 2 4
```

Using logical vector as index

When we use a logical vector for indexing, the position where the logical vector is `TRUE` is returned.

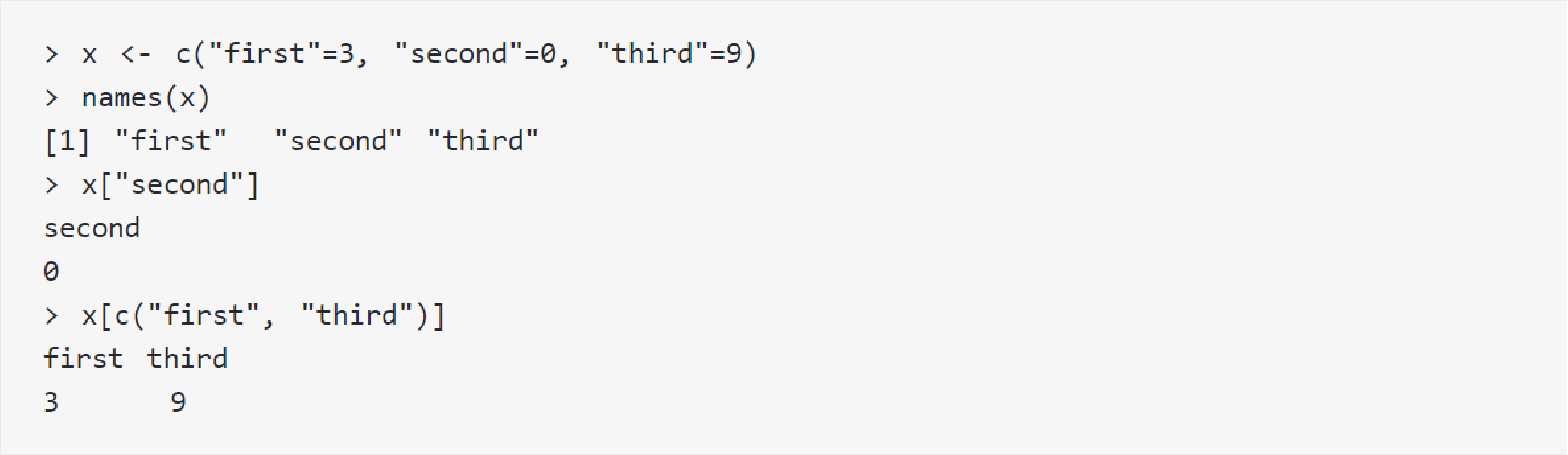
This useful feature helps us in filtering of vector as shown below.

```
> x[c(TRUE, FALSE, FALSE, TRUE)]
[1] -3  3
> x[x < 0]  # filtering vectors based on conditions
[1] -3 -1
> x[x > 0]
[1] 3
```

In the above example, the expression `x>0` will yield a logical vector `(FALSE, FALSE, FALSE, TRUE)` which is then used for indexing.

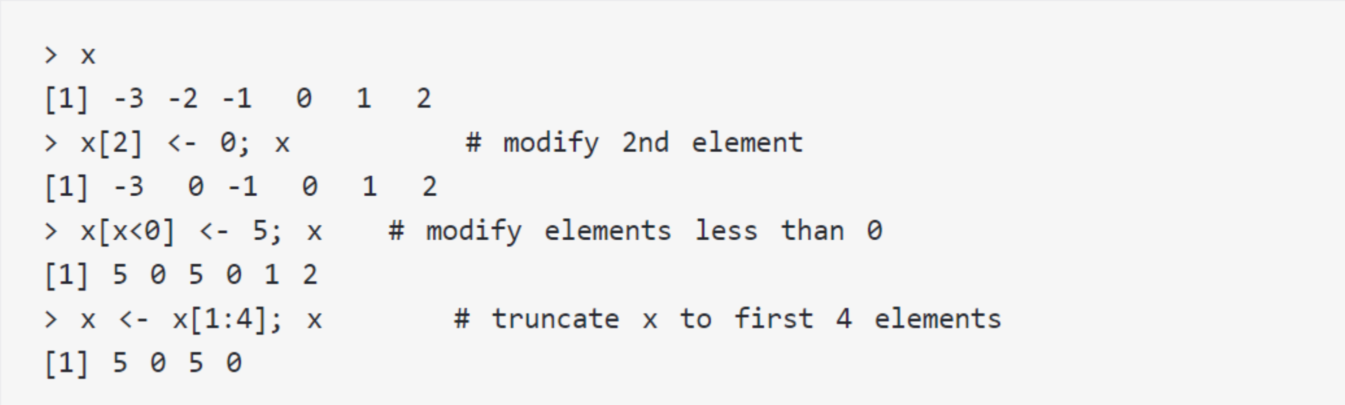
Using character vector as index

This type of indexing is useful when dealing with named vectors. We can name each elements of a vector.



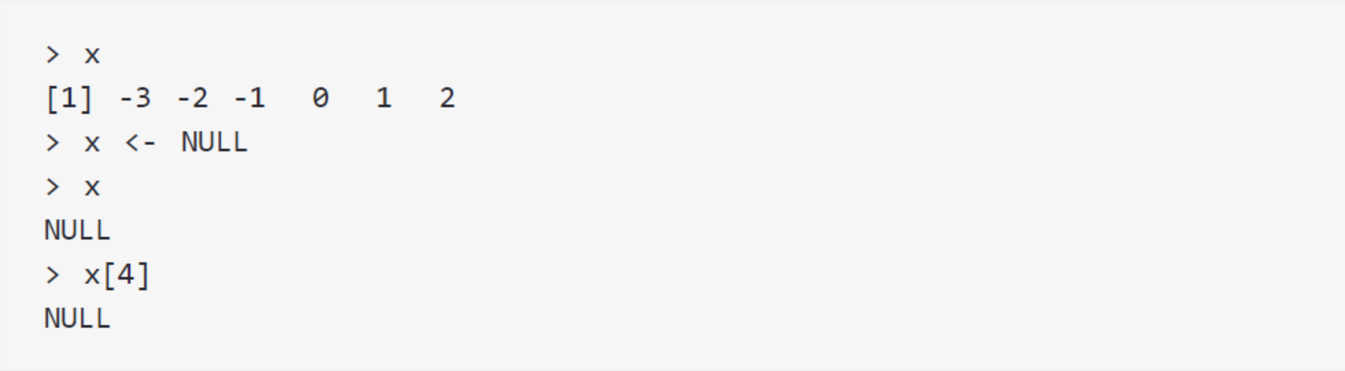
```
> x <- c("first"=3, "second"=0, "third"=9)
> names(x)
[1] "first"  "second" "third"
> x["second"]
second
0
> x[c("first", "third")]
first  third
3       9
```

Modify a vector



```
> x
[1] -3 -2 -1  0  1  2
> x[2] <- 0; x           # modify 2nd element
[1] -3  0 -1  0  1  2
> x[x<0] <- 5; x    # modify elements less than 0
[1] 5 0 5 0 1 2
> x <- x[1:4]; x        # truncate x to first 4 elements
[1] 5 0 5 0
```

Delete a vector



```
> x
[1] -3 -2 -1  0  1  2
> x <- NULL
> x
NULL
> x[4]
NULL
```

Example: Vector Elements Arithmetic

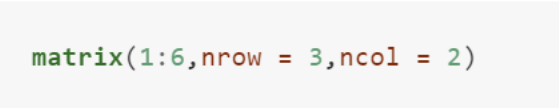
```
> sum(2,7,5)
[1] 14
> x
[1]  2 NA  3  1  4
> sum(x)     # if any element is NA or NaN, result is NA or NaN
[1] NA
> sum(x, na.rm=TRUE)     # this way we can ignore NA and NaN values
[1] 10
> mean(x, na.rm=TRUE)
[1] 2.5
> prod(x, na.rm=TRUE)
[1] 24
```

Whenever a vector contains `NA` (Not Available) or `NaN` (Not a Number), functions such as `sum()`, `mean()`, `prod()` etc. produce `NA` or `NaN` respectively.

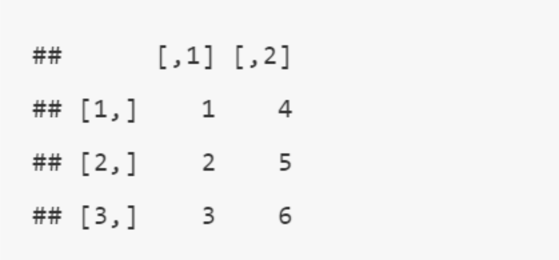
In order to ignore such values, we pass in the argument `na.rm = TRUE`.

Matrix in R

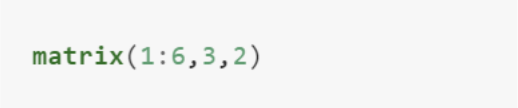
Matrices are two-dimensional data structures in R and are arranged in a rectangular layout. Matrices can contain only one data type. We can create matrices of any of the six data types we discussed before. A matrix can also be thought of as a vector in two dimension.



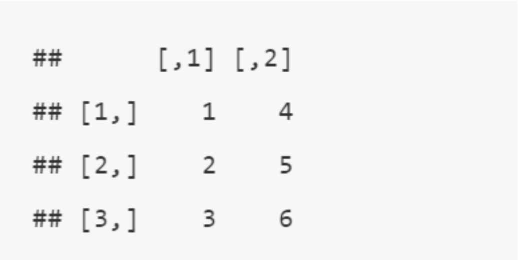
```
matrix(1:6,nrow = 3,ncol = 2)
```



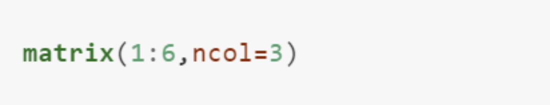
```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```



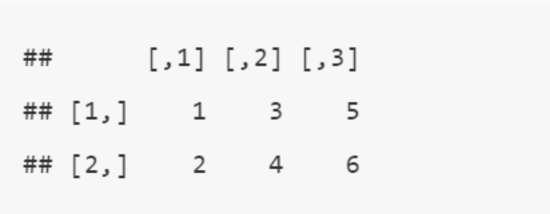
```
matrix(1:6,3,2)
```



```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```



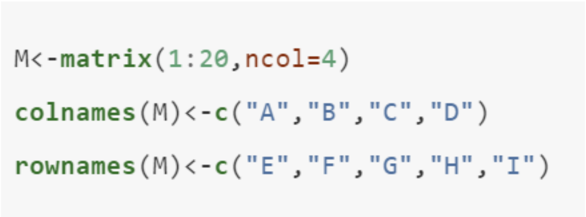
```
matrix(1:6,ncol=3)
```



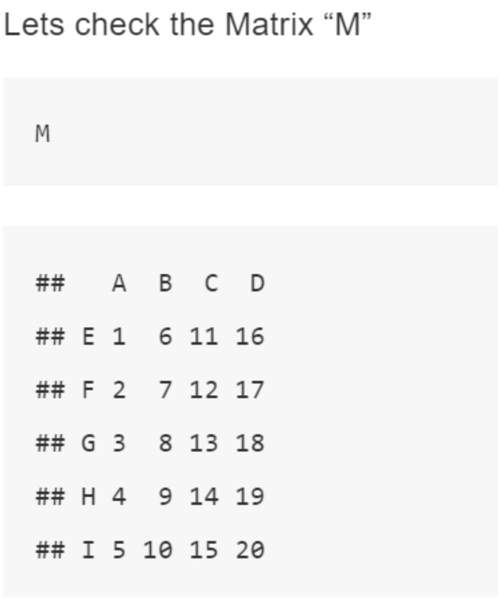
```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Matrix indexing

Create a matrix with 3 rows and two columns



```
M<-matrix(1:20,ncol=4)
colnames(M)<-c("A","B","C","D")
rownames(M)<-c("E","F","G","H","I")
```

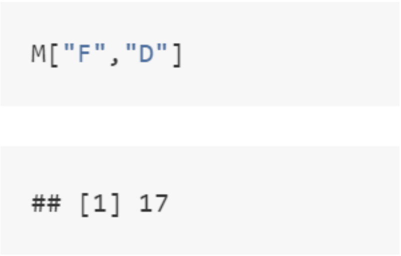


Lets check the Matrix “M”

```
M
```

```
##   A  B  C  D
## E 1  6 11 16
## F 2  7 12 17
## G 3  8 13 18
## H 4  9 14 19
## I 5 10 15 20
```

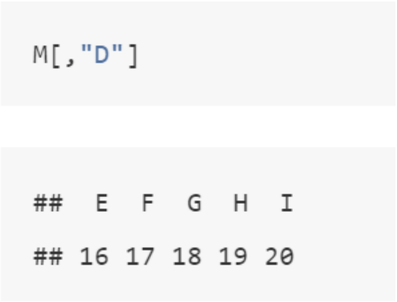
Matrix[rowname,colname] will call the required row and columns by name.



```
M["F","D"]
```

```
## [1] 17
```

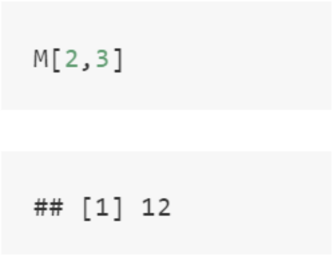
Call complete column



```
M[,"D"]
```

```
##  E  F  G  H  I
## 16 17 18 19 20
```

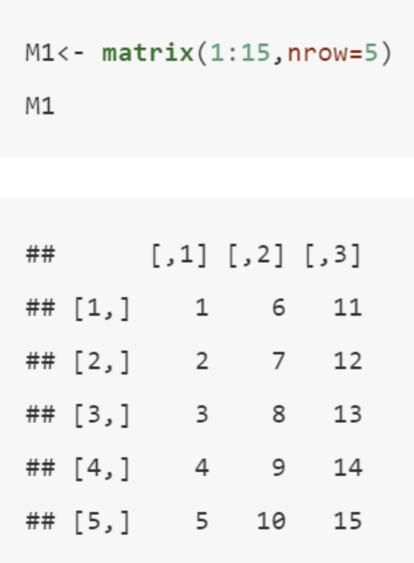
Access specific elements



```
M[2,3]
```

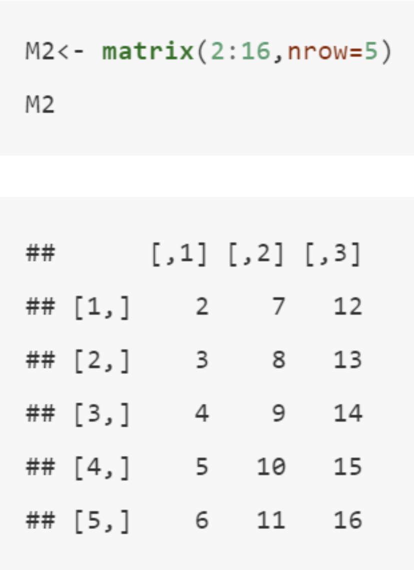
```
## [1] 12
```

Operations in Matrix



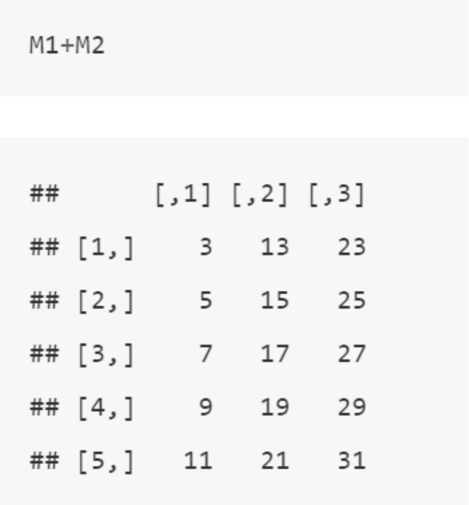
```
M1<- matrix(1:15,nrow=5)
M1
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```



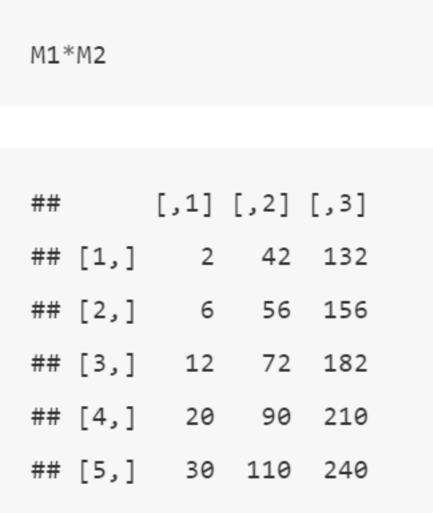
```
M2<- matrix(2:16,nrow=5)
M2
```

```
##      [,1] [,2] [,3]
## [1,]    2    7   12
## [2,]    3    8   13
## [3,]    4    9   14
## [4,]    5   10   15
## [5,]    6   11   16
```



```
M1+M2
```

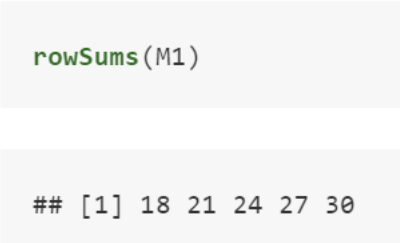
```
##      [,1] [,2] [,3]
## [1,]    3   13   23
## [2,]    5   15   25
## [3,]    7   17   27
## [4,]    9   19   29
## [5,]   11   21   31
```



```
M1*M2
```

```
##      [,1] [,2] [,3]
## [1,]    2   42  132
## [2,]    6   56  156
## [3,]   12   72  182
## [4,]   20   90  210
## [5,]   30  110  240
```

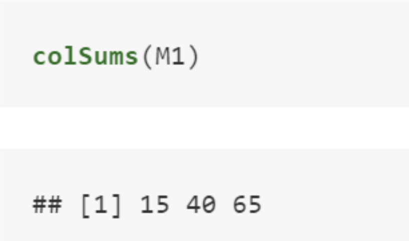
Sum of rows in Matrix



```
rowSums(M1)
```

```
## [1] 18 21 24 27 30
```

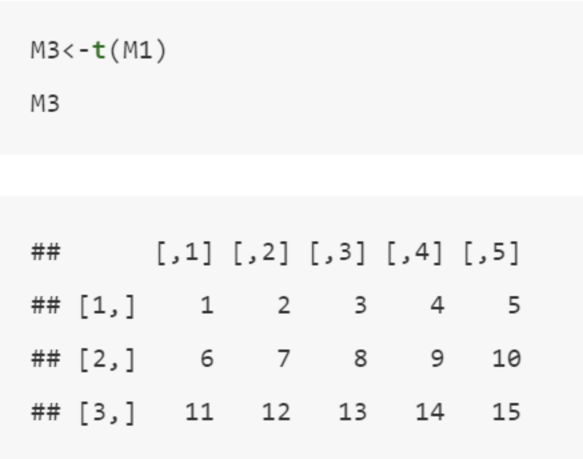
Sum of columns in Matrix



```
colSums(M1)
```

```
## [1] 15 40 65
```

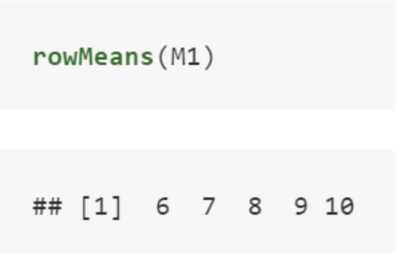
Transposing a Matrix



```
M3<-t(M1)
M3
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
```

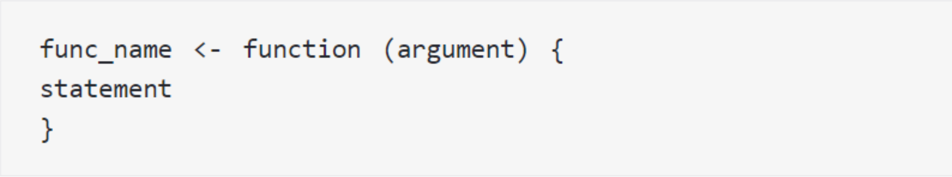
Mean of rows in Matrix



```
rowMeans(M1)
```

```
## [1]  6  7  8  9 10
```

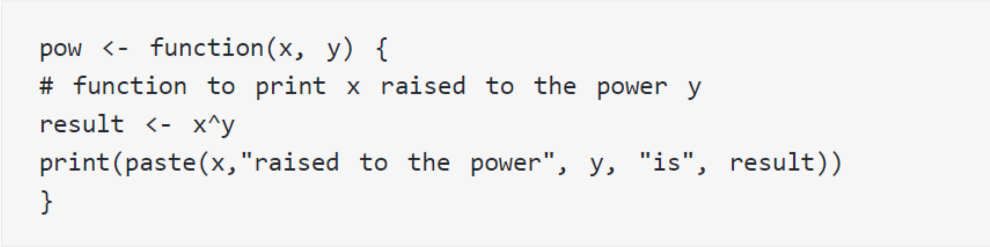
Syntax for Writing Functions in R



```
func_name <- function (argument) {
statement
}
```

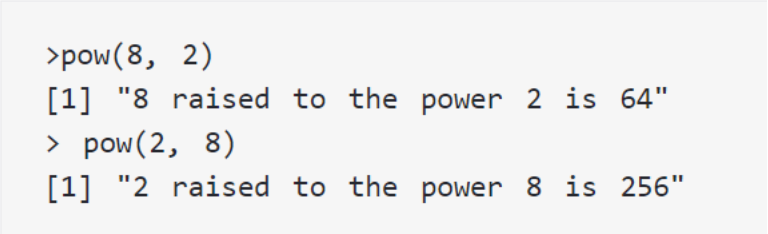
- Here, we can see that the reserved word function is used to declare a function in R.
- The statements within the curly braces form the body of the function. These braces are optional if the body contains only a single expression.
- Finally, this function object is given a name by assigning it to a variable, func_name.

Example of a Function

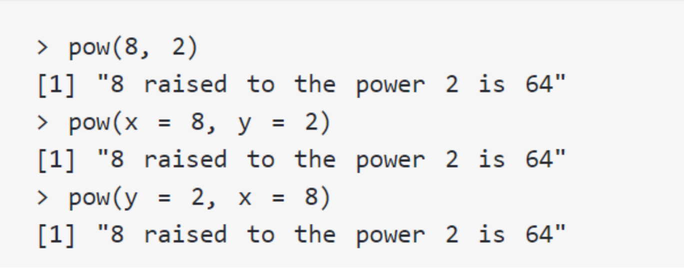


```
pow <- function(x, y) {
# function to print x raised to the power y
result <- x^y
print(paste(x,"raised to the power", y, "is", result))
}
```

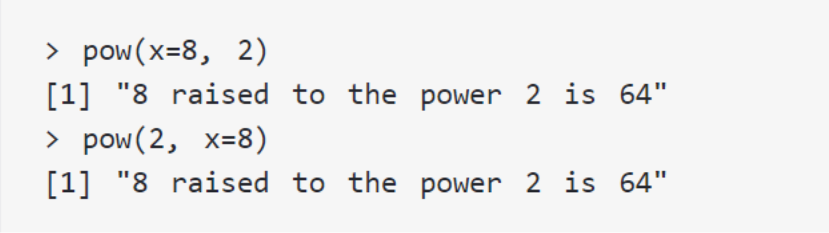
We can call the above function as follows.



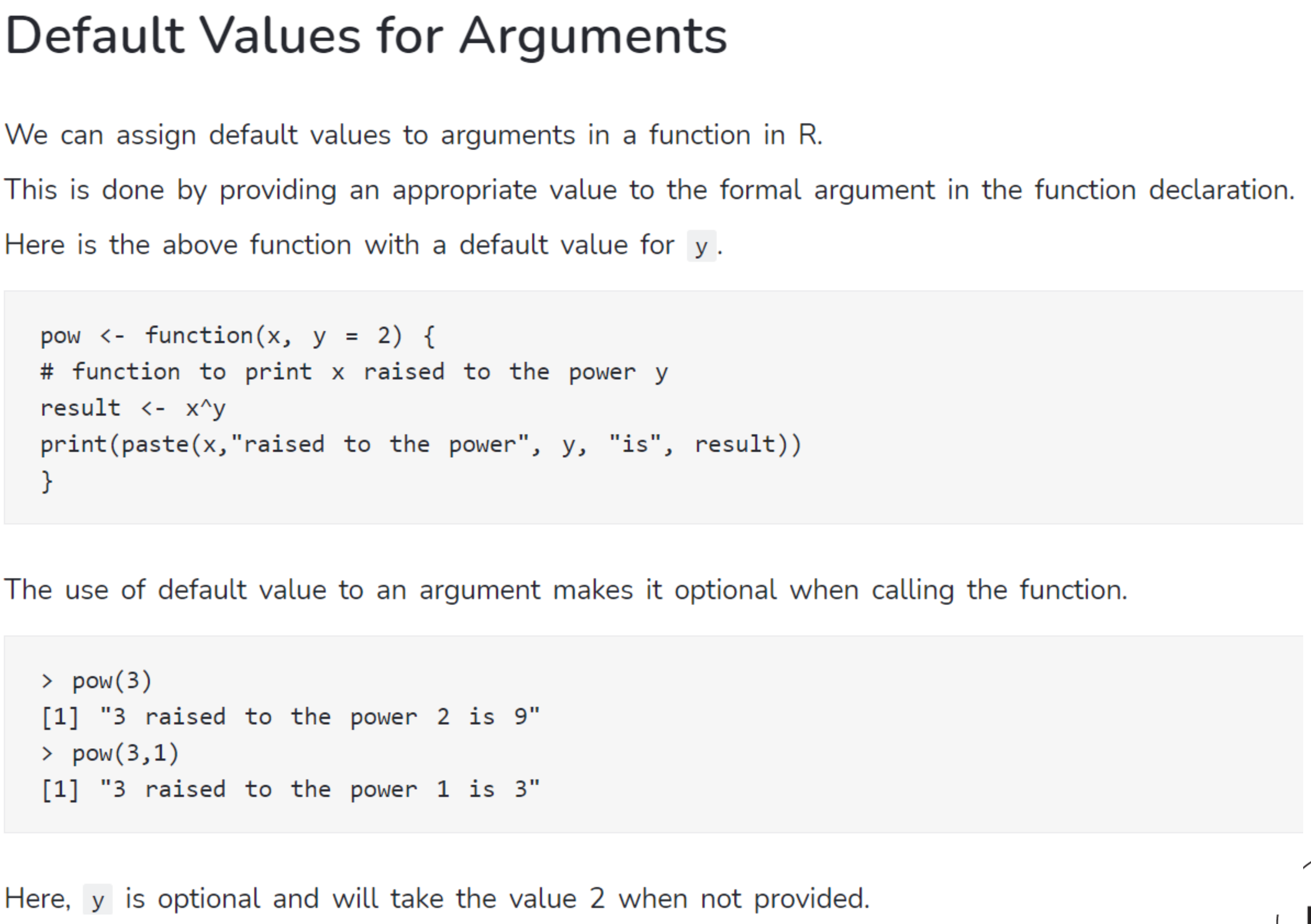
```
>pow(8, 2)
[1] "8 raised to the power 2 is 64"
> pow(2, 8)
[1] "2 raised to the power 8 is 256"
```



```
> pow(8, 2)
[1] "8 raised to the power 2 is 64"
> pow(x = 8, y = 2)
[1] "8 raised to the power 2 is 64"
> pow(y = 2, x = 8)
[1] "8 raised to the power 2 is 64"
```



```
> pow(x=8, 2)
[1] "8 raised to the power 2 is 64"
> pow(2, x=8)
[1] "8 raised to the power 2 is 64"
```



Default Values for Arguments

We can assign default values to arguments in a function in R.

This is done by providing an appropriate value to the formal argument in the function declaration.

Here is the above function with a default value for `y`.

```
pow <- function(x, y = 2) {
# function to print x raised to the power y
result <- x^y
print(paste(x,"raised to the power", y, "is", result))
}
```

The use of default value to an argument makes it optional when calling the function.

```
> pow(3)
[1] "3 raised to the power 2 is 9"
> pow(3,1)
[1] "3 raised to the power 1 is 3"
```

Here, `y` is optional and will take the value 2 when not provided.