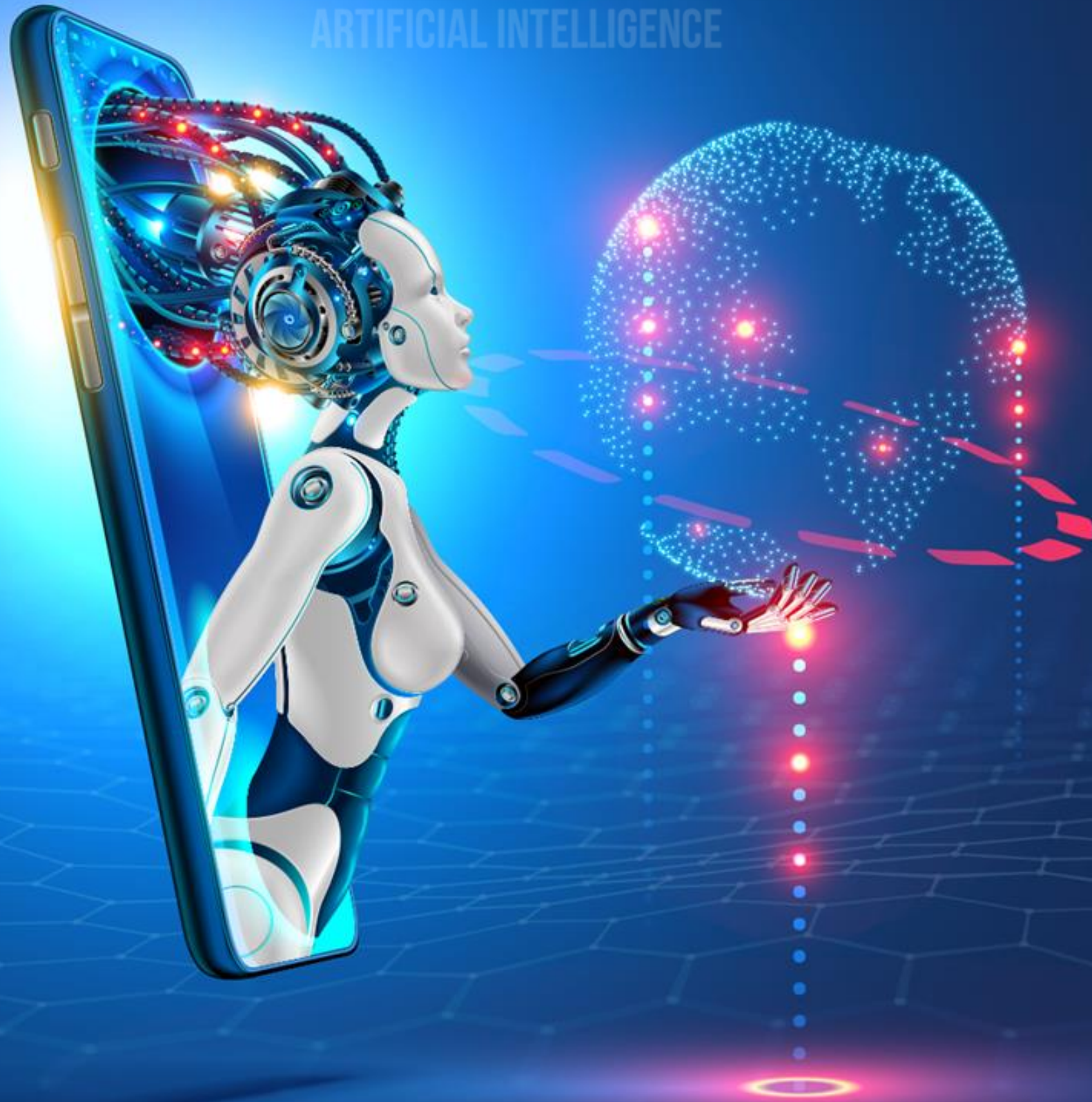


DATA AND ARTIFICIAL INTELLIGENCE



Data Analytics with R



Introduction to R Programming

Learning Objectives

By the end of this lesson, you will be able to:

- 🕒 Define R programming language and its use in data analytics
- 🕒 Utilize various operators, objects, and data structures in R
- 🕒 Subset data from R objects
- 🕒 Perform various operations on R objects



Business Scenario

- Anna has joined as a research associate to a data analyst at a university. She is required to create, store, and manage the research data collected. Since this data has to be statistically analyzed, the preferred tool to do so is R.

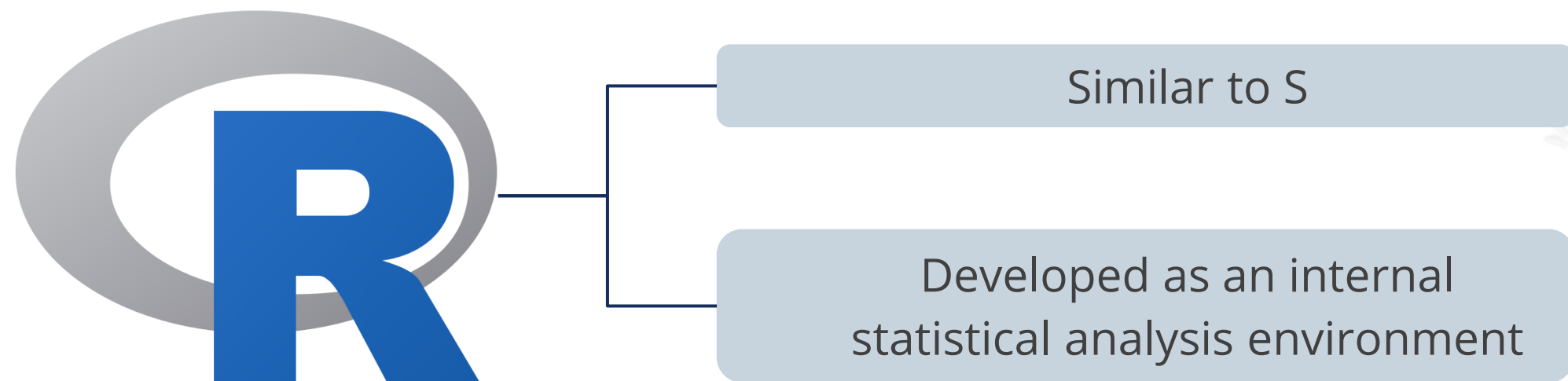
Approach: To create and manage data in R, Anna needs to understand the various data types, data structures, and operations in R and why is R the preferred tool for statistical computing.



Overview and History

Overview and History

R is a programming language and an environment for statistical computing, graphics, and reporting of data.

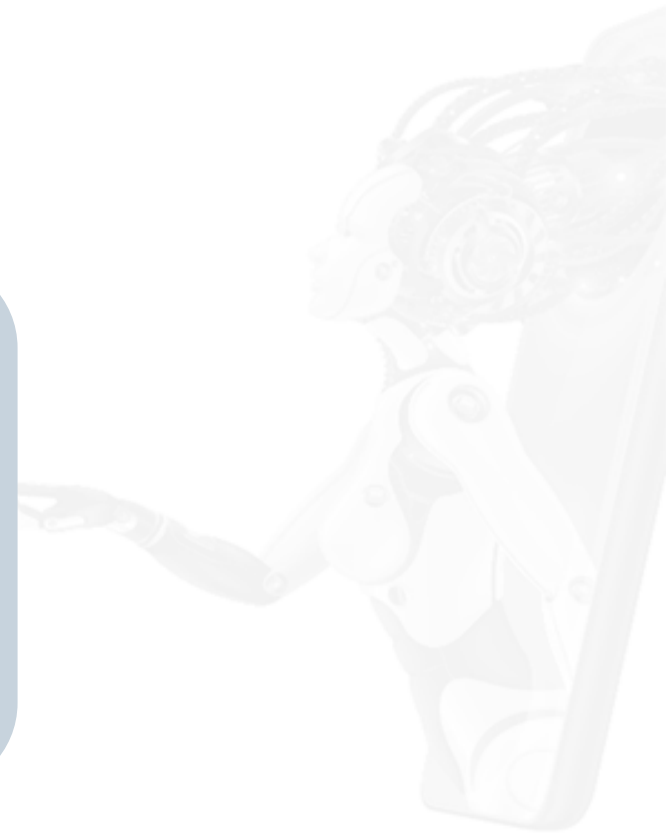


Overview and History

R was initially developed by Ross Ihaka and Robert Gentleman at the University of Auckland.



Since 1997, the R Core Team has been responsible for the development of R.



Overview and History

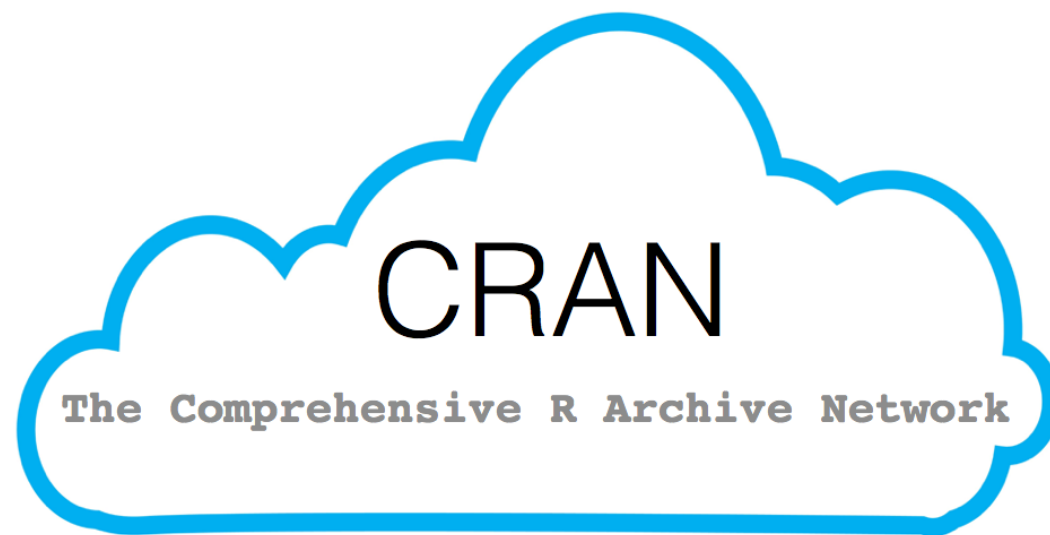
R is an open-source tool.



All users have access to source code which enables users to study, modify, and redistribute the source code.

Overview and History

R has a worldwide repository system – CRAN.



The Comprehensive R Archive Network (CRAN) is a network of sites that acts as the primary web service distributing R sources.

IDEs of R

Integrated Development Environment

An IDE or integrated development environment is a software application that combines all the tools needed for a software development project.



Integrated Development Environment

Popular R IDEs are:



R Studio

R Studio is the most popular IDE for R. There is an open-source and a commercial edition.



Note

The free desktop version of R Studio can be downloaded from:

<https://www.rstudio.com/products/rstudio/download/>

R Vs. Python

R vs. Python

R	Python
It is designed for data analysis, data visualization, statistical computation, and machine learning applications.	It is a multi-purpose language with applications ranging from data analysis and statistical modeling to web development.
It has the support of numerous packages to perform statistical tasks.	It has a few key packages to perform the tasks.
It has its limitations in terms of memory management, which is not scalable.	It can be used for any amount of data and for any applications.

Why R for Analytics?

Why R for Data Analytics?

Open source

R is a free software.

Statistical algorithm

Visualizations

Machine learning algorithms

Recognized by other software

Customized algorithms



Why R for Data Analytics?

Open source

Statistical algorithm

Visualizations

Machine learning algorithms

Recognized by other software

Customized algorithms

R is a statistical software where complex statistical models, such as linear regression, logistic regression, hypothesis testing, analysis of variance (ANOVA), and generalized linear model (GLM), can be run.



Why R for Data Analytics?

Open source

Statistical algorithm

Visualizations

Machine learning algorithms

Recognized by other software

Customized algorithms

R has some great tools to aid data visualization to create graphs, bar charts, multi-panel lattice charts, scatter plots, and new custom-designed graphics.



Why R for Data Analytics?

Open source

Statistical algorithm

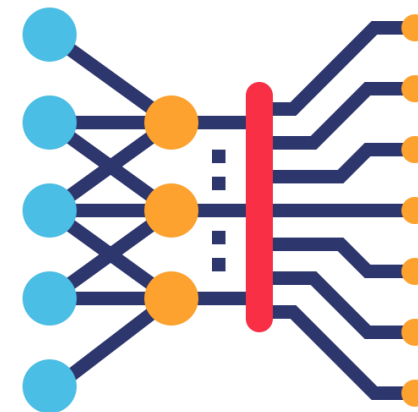
Visualizations

Machine learning algorithms

Recognized by other software

Customized algorithms

ML algorithms like SVM, Naive Bayes theorem, XGBoost, decision tree, and random forest are available in R.



These algorithms have proven to be better over time and provide accurate results.

Why R for Data Analytics?

Open source

Statistical algorithm

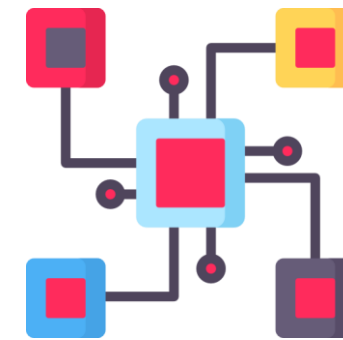
Visualizations

Machine learning algorithms

Recognized by other software

Customized algorithms

R codes can now be written in SAS as R codes are widely used and programmers are getting familiar with these.



R can handle semi-structured data and has built-in algorithms.

Why R for Data Analytics?

Open source

Statistical algorithm

Visualizations

Machine learning algorithms

Recognized by other software

Customized algorithms

Programmers can define their customized algorithms in R and develop their own algorithms and packages.



Industrial Implementation of R

Industrial Implementation of R

R is the tool of choice for a vast majority of Data Analyst for:



Industrial Implementation of R

Many companies use R for statistical and analytical purposes.

- To update status
- To perform behavioral analysis for improved advertising suggestions

Facebook

- To calculate ROI
- To predict economic activity
- To improve the efficiency of online advertising

Google

Industrial Implementation of R

Many companies use R for statistical and analytical purposes.

- To provide Xbox matchmaking service
- To act as a statistical engine within the Azure ML framework

Microsoft

- To perform prediction to automate price settings of last-minute offers

Thomas Cook

R Variables

R Variables

Following rules are considered when naming an identifier:



Identifiers in R may be a combination of letters, digits, periods, and underscores.



Identifiers start with a letter or a period.



Reserved words in R cannot be used as identifiers.



R Variables

R is a case-sensitive language.

Valid Variable Names

var_name2, var.name, var123, .name

Invalid Variable Names

var@name2, .12name, _var123

R Operators

Types of Operators in R

R provides four types of operators:

Assignment Operators

Arithmetic Operators

Relational Operators

Logical Operators



R Operators

Assignment Operators

Operator	Description	Illustration in R
= or <-	Left assignment	<pre>> var1 <- 45 > var2 = 25 > print(var1) [1] 45 > print(var2) [1] 25</pre>
->	Right assignment	<pre>> 56 -> var > print(var) [1] 56</pre>

R Operators

Arithmetic Operators

Operator	Description	Illustration in R
+	Adds two numeric objects	<pre>> val1 <- 45 > val2 <- 56 > print(val1 + val2) [1] 101</pre>
-	Subtracts value of second numerical object from the first	<pre>> val1 <- 45 > val2 <- 56 > print(val1 - val2) [1] -11</pre>
*	Gives product of two numerical objects	<pre>> val1 <- 45 > val2 <- 5 > print(val1 * val2) [1] 255</pre>
/	Divides the value of the first object by the value of the second object	<pre>> val1 <- 87 > val2 <- 5 > print(val1 / val2) [1] 17.4</pre>

R Operators

Arithmetic Operators

Operator	Description	Illustration in R
%%	Gives the remainder of the division	<pre>> val1 <- 87 > val2 <- 5 > print(val1 %% val2) [1] 2</pre>
/%%	Gives the integer part (quotient) of the division	<pre>> val1 <- 87 > val2 <- 5 > print(val1 %/% val2) [1] 17</pre>
^	Gives result as the value of the first object raised to exponent of the second object	<pre>> val1 <- 3 > val2 <- 5 > print(val1 ^ val2) [1] 243</pre>

R Operators

Relational Operators

Operator	Description	Illustration in R
>	Returns True if the first object is greater than the second	<pre>> vec1 <- c(2, 9.2, 4.5, 3) > vec2 <- c(7, 9, 1.2, 3) > vec1 > vec2 [1] FALSE TRUE TRUE FALSE</pre>
<	Returns True if the first object is less than the second	<pre>> vec1 <- c(2, 9.2, 4.5, 3) > vec2 <- c(7, 9, 1.2, 3) > vec1 < vec2 [1] TRUE FALSE FALSE FALSE</pre>
==	Returns True if the first object is equal to the second	<pre>> vec1 <- c(2, 9.2, 4.5, 3) > vec2 <- c(7, 9, 1.2, 3) > vec1 == vec2 [1] FALSE FALSE FALSE TRUE</pre>

R Operators

Relational Operators

Operator	Description	Illustration in R
>=	Returns True if first object is greater than or equal to the second	<pre>> vec1 <- c(2, 9.2, 4.5, 3) > vec2 <- c(7, 9, 1.2, 3) > vec1 >= vec2 [1] FALSE TRUE TRUE TRUE</pre>
<=	Returns True if first object is less than or equal to the second	<pre>> vec1 <- c(2, 9.2, 4.5, 3) > vec2 <- c(7, 9, 1.2, 3) > vec1 <= vec2 [1] TRUE FALSE FALSE TRUE</pre>
!=	Returns True if first object is not equal to the second	<pre>> vec1 <- c(2, 9.2, 4.5, 3) > vec2 <- c(7, 9, 1.2, 3) > vec1 != vec2 [1] TRUE TRUE TRUE FALSE</pre>

R Operators

Logical Operators

Operator	Description	Illustration in R
&&	Combines first element of the two vectors and gives True if both values are True	<pre>> v1 <- TRUE > v2 <- FALSE > print(v1 && v2) [1] FALSE</pre>
&	Combines corresponding elements of the two vectors and gives True if both values are True	<pre>> v1 <- c(TRUE, FALSE, TRUE, TRUE) > v2 <- c(FALSE, FALSE, TRUE, FALSE) > print(v1 & v2) [1] FALSE FALSE TRUE FALSE</pre>
	Combines first element of the two vectors and gives True if either of the values are True	<pre>> v1 <- TRUE > v2 <- FALSE > print(v1 v2) [1] TRUE</pre>

R Operators

Logical Operators

Operator	Description	Illustration in R
	Combines each corresponding element of the two vectors and gives True if either of the values are True	<pre>> v1 <- c(TRUE, FALSE, TRUE, TRUE) > v2 <- c(FALSE, FALSE, TRUE, FALSE) > print(v1 v2) [1] TRUE FALSE TRUE TRUE</pre>
!	Takes each element of the vector and gives the opposite of the logical value	<pre>> v1 <- c(TRUE, FALSE, TRUE, TRUE) > print(!v1) [1] FALSE TRUE FALSE FALSE</pre>

R Objects

Data Types in R

R has five basic classes of objects:

Class of data	Description	Illustration in R
Character	Strings of letters, numbers, and other symbols or any value enclosed within single or double quotes	<pre>> val <- "Hello!!" > print(class(val)) [1] "character"</pre>
Numeric	Real numbers	<pre>> val <- 12.7 > print(class(val)) [1] "numeric"</pre>
Integer	Integers	<pre>> val <- 23L > print(class(val)) [1] "integer"</pre>

Data Types in R

R has five basic classes of objects:

Class of data	Description	Illustration in R
Complex	Represents complex numbers	<pre>> val <- 23 +6i > print(class(val)) [1] "complex"</pre>
Logical	Logical values, usually results of comparisons: TRUE, FALSE, and NA	<pre>> val <- FALSE > print(class(val)) [1] "logical"</pre>

Data Structure in R

Data Structures in R

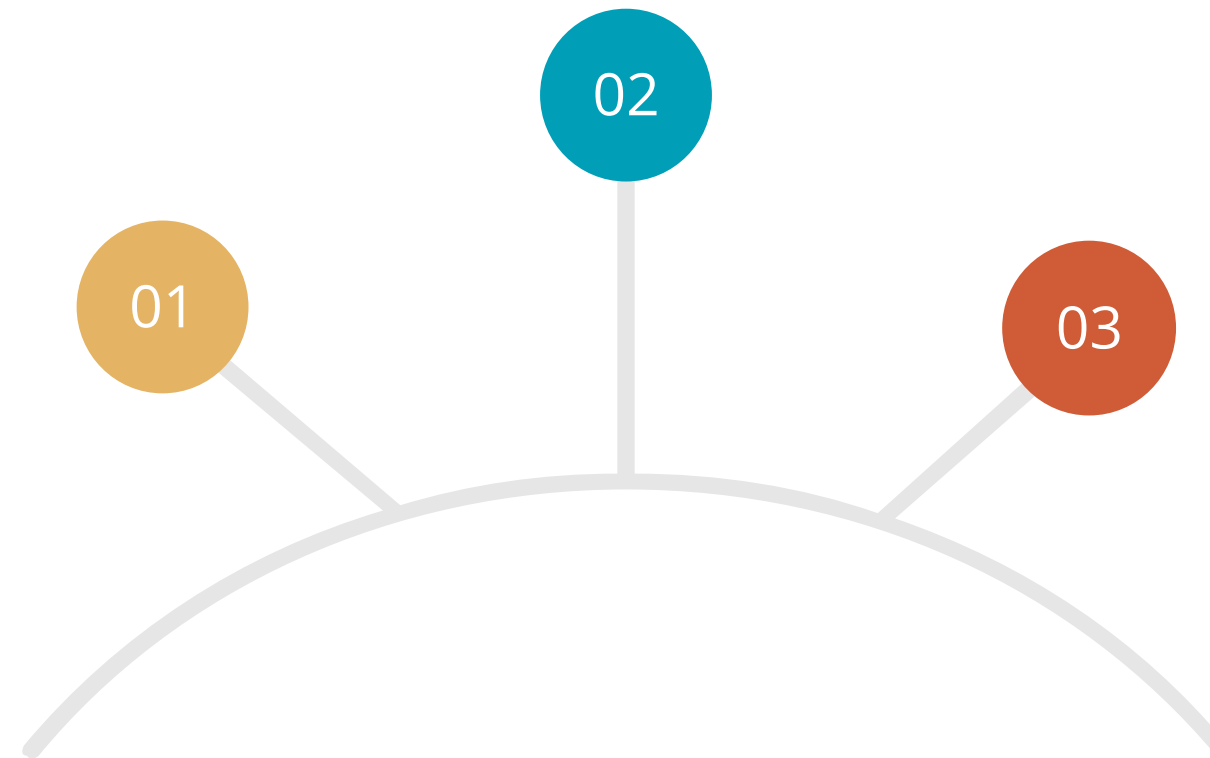
Data structures are ways of organizing and storing data.



Vectors

The `c()` function is used to create vectors of objects.

Vectors are the most basic objects in R.



A vector can only store objects of the same class and are named after them.

Vectors

Different types of vectors:

A character vector is a vector with only character values.

```
> vec <- c("A", "c", 'John')  
> print(class(vec))  
[1] "character"
```

A numeric vector is a vector with only numeric values.

```
> vec <- c(0.5, 0.8, 1.9, 4)  
> print(class(vec))  
[1] "numeric"
```

An integer vector is a vector with only integer values.

```
> vec <- 4:9  
> print(class(vec))  
[1] "integer"
```

A complex vector is a vector with only complex values.

```
> vec <- c(1 +0i, 4.5 +6i)  
> print(class(vec))  
[1] "complex"
```

A logical vector is a vector with only logical values.

```
> vec <- c(TRUE, FALSE, T, F)  
> print(class(vec))  
[1] "logical"
```

Tip: A sequential integer vector can be created using ":" operator.

Mixing Vectors

What is coercion?

When a vector is created with different types of objects (mixing objects), the data type of the objects is implicitly converted or coerced such that each element in the vector is now of the same class.

Mixing Vectors: Examples

```
> # a vector with character and
numeric value coerced to
character type
> vec <- c("a", 1.7)
> print(vec)
[1] "a" "1.7"
> print(class(vec))
[1] "character"
```

A vector with character and
numeric value coerced to
character type

```
> # a vector with logical and
numeric value coerced to
numeric type
> vec <- c(23 , TRUE)
> print(vec)
[1] 23 1
> print(class(vec))
[1] "numeric"
```

A vector with logical and
numeric value coerced to
numeric type

```
> # a vector with character and
logical value coerced to
character type
> vec <- c(TRUE, "a")
> print(vec)
[1] "TRUE" "a"
> print(class(vec))
[1] "character"
```

A vector with character and
logical value coerced to
character type

Typecasting in R

Typecasting is changing the data type of a variable.

Objects can be converted explicitly from one class to another using the “as. *” function.

```
> # code to convert an object to integer/character or
logical.
> vec <- c(0, -1, 2.4, 8, 10)
> print(class(vec))
[1] "numeric"
> as.integer(vec)
[1] 0 -1 2 8 10
> as.character(vec)
[1] "0" "-1" "2.4" "8" "10"
> as.logical(vec)
[1] FALSE TRUE TRUE TRUE TRUE
```



Typecasting in R

If casting is not possible, it results in NA.

```
> vec <- c("a", "TRUE", "12.4", "23")
> print(class(vec))
[1] "character"
> as.integer(vec)
[1] NA NA 12 23
Warning message:
NAs introduced by coercion
> as.numeric(vec)
[1] NA NA 12.4 23.0
Warning message:
NAs introduced by coercion
> as.logical(vec)
[1] NA TRUE NA NA
```



Vector Attributes

The three properties of a vector are class, length, and names.

```
> vec <- c(a = 25, b = 40, c = 85, d = 90, e = 10)
> print(vec)
  a b c d e
25 40 85 90 10
> print(class(vec)) ## the class of the vec
[1] "numeric"
> print(length(vec)) ## the no. of elements
[1] 5
> print(names(vec)) ## the names
[1] "a" "b" "c" "d" "e"
```

Example: Code displaying the class, length, and names of a vector

01

Class attribute gives the data class of the vector.

02

Length gives the count of elements in a vector.

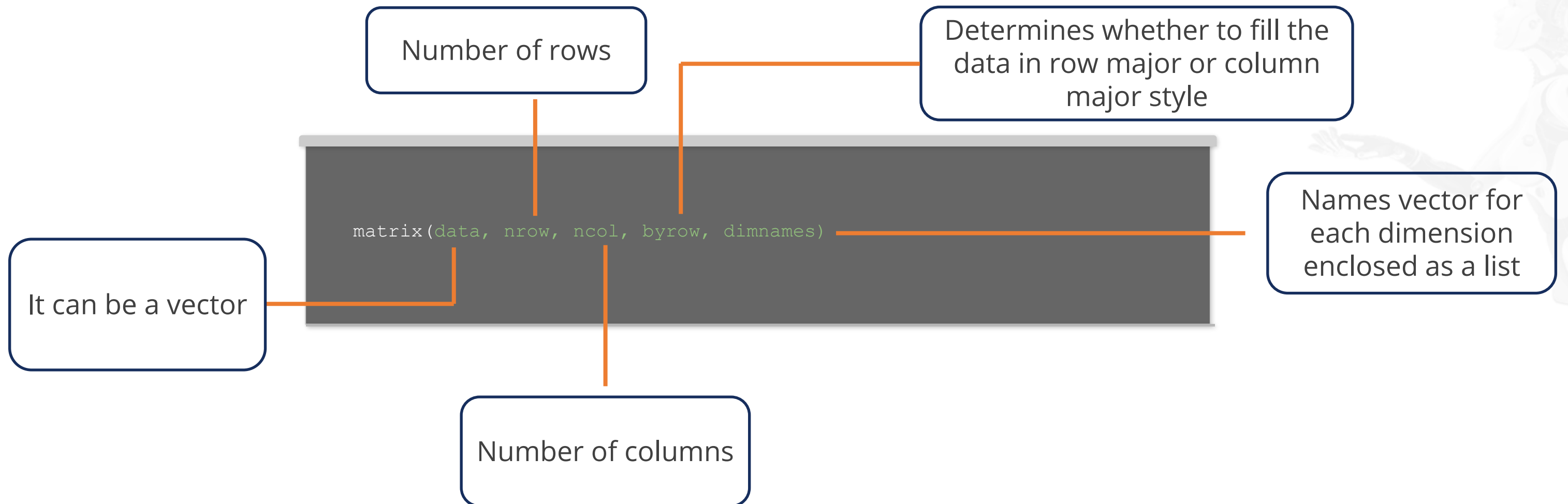
03

Names is an optional attribute and only contains the values for a named vector.

Matrices

Matrices are vectors with two dimensions.

Matrices in R can be created using the `matrix()` function.



Matrices

Creation of an empty matrix:
In this example, an empty matrix with 2 rows and 3 columns is created.

```
> m <- matrix(nrow = 2, ncol = 3)
> m
  [,1] [,2] [,3]
[1,] NA NA NA
[2,] NA NA NA
```

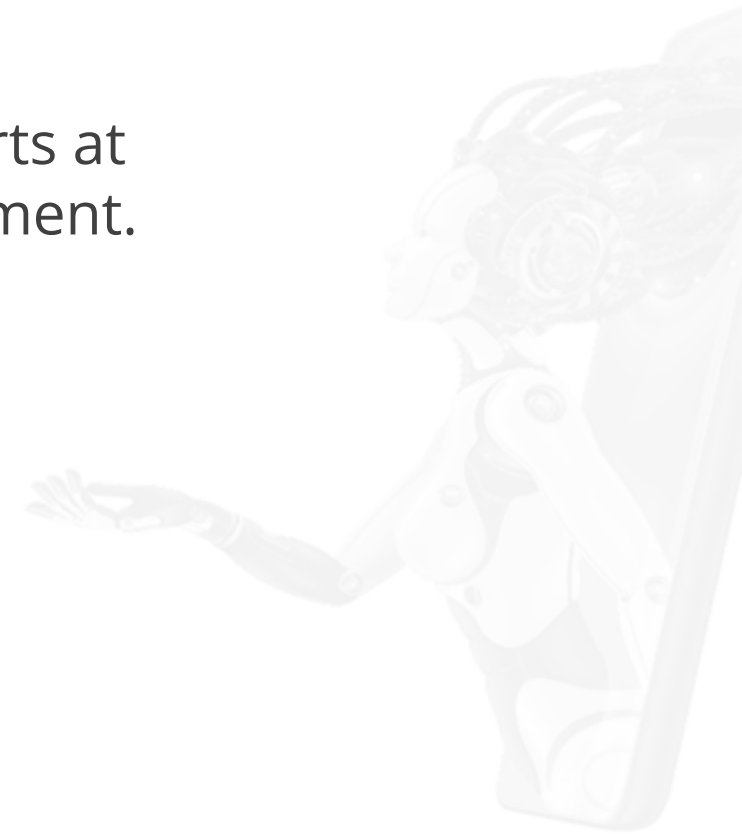


Matrices

Matrices are constructed column-wise by default.

Example: A matrix created from a vector 1:6 with 2 rows and 3 columns starts at the top left and goes down to fill a column first with the default byrow argument.

```
> m <- matrix(1:6,nrow = 2, ncol = 3)
> m
  [,1] [,2] [,3]
[1,]  1  3  5
[2,]  2  4  6
```



Matrices

Matrices can also be constructed row-wise by using the “byrow” argument.

Example: A matrix created from a vector 1:6 with 2 rows and 3 columns starts at top left and goes down to fill a row first when byrow = TRUE.

```
> m <- matrix(1:6,nrow = 2, ncol = 3, byrow = TRUE)
> m
  [,1] [,2] [,3]
[1,]  1  2  3
[2,]  4  5  6
```



Matrices

A matrix can be created directly from a vector by adding dimension attributes.

The dimension attribute for a matrix is a vector with values for rows and columns.

```
> vec <- 1:6
> print(vec)
[1] 1 2 3 4 5 6

> dim (vec) <- c(2,3)
> print(vec)
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 3 4 6
```

Adding dimension
attribute to vector

`cbind()` and `rbind()`

Matrices can be created by column-binding and row-binding vectors.

`rbind`

Binds vectors to create rows of the matrix

`cbind`

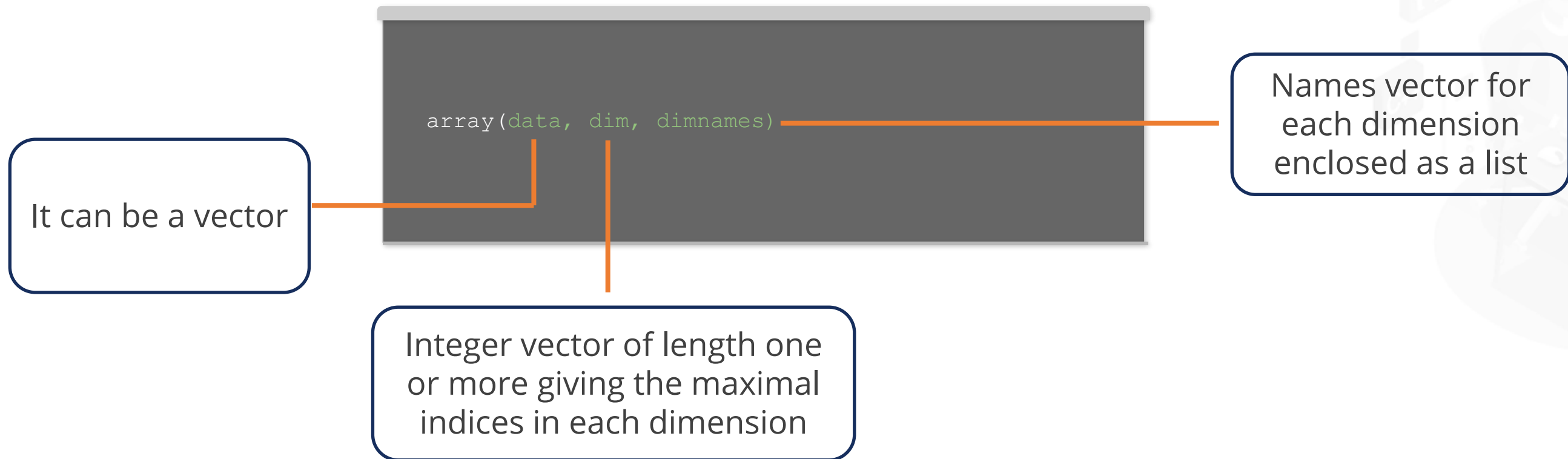
Binds vectors to create columns of the matrix

```
> a <- c(1,3,5,7)
> b <- c(2,4,6,8)
> rbind(a,b)
  [,1] [,2] [,3] [,4]
a  1  3  5  7
b  2  4  6  8
> cbind(a,b)
  a b
[1,] 1 2
[2,] 3 4
[3,] 5 6
[4,] 7 8
```

Arrays

Arrays are vectors that may have more than two dimensions.

Arrays are created using the `array()` function.



Arrays

For example, `dim = c(2,4,3)` creates an array with three matrices, each with two rows and three columns.

```
> arr <- array(1:24, dim = c(2,4,3) )  
> print(arr)  
, , 1  
  
  [,1] [,2] [,3] [,4]  
[1,]  1  3  5  7  
[2,]  2  4  6  8  
  
, , 2  
  
  [,1] [,2] [,3] [,4]  
[1,]  9 11 13 15  
[2,] 10 12 14 16  
  
, , 3  
  
  [,1] [,2] [,3] [,4]  
[1,] 17 19 21 23  
[2,] 18 20 22 24
```

Dimension vector

Factors

Factors are vector-like objects that are used to represent categorical data.

Factors are treated differently by statistical modeling functions.

Factors take only a predefined, finite number of categorical values.



Factors

Levels are the unique categorical values of the factor data.

Levels can be explicitly defined and ordered using the levels argument.

```
> # creating a factor data with responses as yes or no and creating levels automatically
> response <- factor(c("yes", "yes", "no", "no", "yes", "yes", "yes"))
> response
[1] yes yes no no yes yes yes
Levels: no yes
>
> response <- factor(c("yes", "yes", "no", "no", "yes", "yes", "yes"),
+ levels = c("yes", "no", "maybe"))
> response
[1] yes yes no no yes yes yes
Levels: yes no maybe
```

Dataframes

Dataframes are the most commonly used data objects in R for data analysis. They are list-like data objects in which the length of every element needs to be the same.

Example: Creating a dataframe with two vectors as columns.

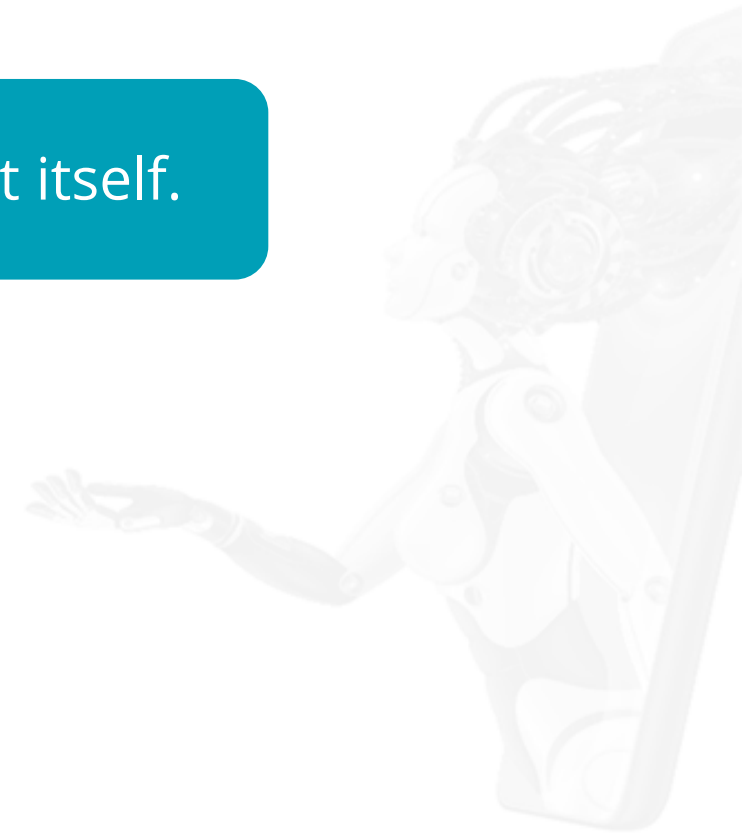
```
> id <- c('A11', 'B12', 'C13', 'D14')
> age <- c(21, 23, 20, 19)
> df <- data.frame(id, age)
> df
  id age
1 A11 21
2 B12 23
3 C13 20
4 D14 19
```



Lists

List is a vector which may contain different types of R objects.

A list may contain numeric, integer, character, matrix, dataframe, and even list itself.



Lists

A list can be created using the `list()` function.

Example:

```
> l <- list("A", 23.5, TRUE, 1+8i, c('John', 'Jake'),  
+ matrix(1:4, nrow = 2, ncol = 2))  
> print(l)  
[[1]]  
[1] "A"  
  
[[2]]  
[1] 23.5  
  
[[3]]  
[1] TRUE  
  
[[4]]  
[1] 1+8i  
  
[[5]]  
[1] "John" "Jake"  
  
[[6]]  
  [,1] [,2]  
[1,]  1  3  
[2,]  2  4
```



Names Attribute

Objects in R can have names for individual elements, rows, and columns.

		Rows			
Columns					



Named Vector

Named vector can also be created by adding value to the names attribute of a vector.

```
> # creating a unnamed vector
> vec <- c(12, 14.5, 67)
> names(vec)
NULL
> # assigning values to names attributes
to convert to named vector.
> names(vec) <- c("a", "b", "c")
> print(vec)
  a b c
12.0 14.5 67.0
> names(vec)
[1] "a" "b" "c"
```

Vector elements can also be assigned names.

```
> # creating a named vector within c() function
itself.
> vec <- c(a = 12, b = 14.5, c = 67)
> print(vec)
  a b c
12.0 14.5 67.0
> names(vec)
[1] "a" "b" "c"
```

Named List

List elements can also be custom-named.

```
# creating a list with named elements
> info <- list(name = "John", age = 45, hobbies = c('golf', 'swimming'),
+ member = "TRUE")
> print(info)
$name
[1] "John"

$age
[1] 45

$hobbies
[1] "golf" "swimming"

$member
[1] "TRUE"

> names(info)
[1] "name" "age" "hobbies" "member"
```



Matrices with Dimension Names

Example: Creating a matrix with dimnames assigned after creation.

```
> # creating a matrix with dimension names specified separately.
> m<- matrix(26:40, nrow = 5, ncol = 3, byrow = TRUE)
> row.names(m) <- c("a", "b", "c", "d", "e")
> colnames(m) <- c("A1", "B1", "C1")
> print(m)
  A1 B1 C1
a 26 27 28
b 29 30 31
c 32 33 34
d 35 36 37
e 38 39 40

> row.names(m)
[1] "a" "b" "c" "d" "e"
> colnames(m)
[1] "A1" "B1" "C1"
```



Matrices with Dimension Names

Example: Creating a matrix with dimnames defined as a part of the matrix function.

```
> # Provide dimension names as argument to the matrix function
> m<- matrix(26:40, nrow = 5, ncol = 3, byrow = TRUE,
+ dimnames = list(c("a", "b", "c", "d", "e"),
+ c("A1", "B1", "C1")))
>
> print(m)
  A1 B1 C1
a 26 27 28
b 29 30 31
c 32 33 34
d 35 36 37
e 38 39 40
> row.names(m)
[1] "a" "b" "c" "d" "e"
> colnames(m)
[1] "A1" "B1" "C1"
```



Arrays with Dimension Names

dimnames in an array is a list of vectors containing the dimension names for each dimension. The dimension names in array() functions are supposed to be in the same order as the dimension shape.

```
> arr <- array(1:24, dim = c(2,4,3),  
+ dimnames = list( c("Row1", "Row2"),  
+ c("Col1", "Col2", "Col3", "Col4"),  
+ c("Matrix1", "Matrix2", "Matrix3"))) )  
> print(arr)  
, , Matrix1  
  
  Col1 Col2 Col3 Col4  
Row1  1  3  5  7  
Row2  2  4  6  8  
  
, , Matrix2  
  
  Col1 Col2 Col3 Col4  
Row1  9 11 13 15  
Row2 10 12 14 16  
  
, , Matrix3  
  
  Col1 Col2 Col3 Col4  
Row1 17 19 21 23  
Row2 18 20 22 24
```



Subsetting R Objects

Subsetting Data in R

Subsetting is retrieving parts of a data for specific purposes.

There are a number of operators that can be used to extract subsets of R objects.

[]: square brackets

- It returns an object of the same class as the original.
- It can be used to select more than one element.

[[]]: double square brackets

- It extracts elements of a list or a dataframe.
- It can only be used to extract a single element.
- Class of the returned object will not necessarily be a list or dataframe.

\$: dollar operator

- It extracts elements of a list or dataframe by name.
- Its semantics are similar to that of [[]].

Subsetting in Vectors

Example: Subsetting single-element vectors using index number.

```
> vec <- c("A", "c", "E", "g")
> print(vec)
[1] "A" "c" "E" "g"
> # gives first element
> print(vec[1])
[1] "A"
> # for second element
> print(vec[2])
[1] "c"
```

Gives first
element

Subsetting in Vectors

Multiple elements can also be extracted at the same time for a specific group of indices or in sequential manner.

```
> vec <- 10:100
> # for specific elements
> vec[c(4, 8, 11, 20, 51)]
[1] 13 17 20 29 60
> # for elements through 26 to 40 :
> print(vec[26:35])
[1] 35 36 37 38 39 40 41 42 43 44
```

For specific
group elements

For a sequence
of elements

Subsetting in Vectors

Subset of elements can also be created based on condition.

```
> vec <- c(-34, 26, 78, 10, -12)
> # for elements > 0
> vec[vec > 0]
[1] 26 78 10
```

Results are only values > 0

Subsetting in Vectors

The names assigned to elements can also be used to extract specific elements.

```
> vec <- c(a = 12, b = 15, c = 20)
> print(vec[1])
a
12
> print(vec["a"])
a
12
> print(vec[c("a", "b")])
a b
12 15
```

Returns the values
corresponding to the names

Subsetting in Matrices

A subset of a matrix can be derived by providing row number(s) and column number(s).

`m[row, column]`

```
> m <- matrix(1:20, nrow = 4, ncol = 5, byrow = TRUE)
> print(m)
  [,1] [,2] [,3] [,4] [,5]
[1,]  1  2  3  4  5
[2,]  6  7  8  9 10
[3,] 11 12 13 14 15
[4,] 16 17 18 19 20
> # row no. 2 and column no. 4
> m[2, 4]
[1] 9
> # row no. 1 to 3 columns 3 to 5
> m[1:3, 3:5]
  [,1] [,2] [,3]
[1,]  3  4  5
[2,]  8  9 10
[3,] 13 14 15
```

Specific value

Multiple values

Subset Using Dimension Names in Matrix

Data can be extracted by using the corresponding dimension name for the values as row name and column name.

```
m<- matrix(26:40, nrow = 5, ncol = 3, byrow = TRUE,  
+ dimnames = list(c("Row1", "Row2", "Row3", "Row4", "Row5"),  
+ c("Col1", "Col2", "Col3"))  
>  
> print(m)  
  Col1 Col2 Col3  
Row1 26 27 28  
Row2 29 30 31  
Row3 32 33 34  
Row4 35 36 37  
Row5 38 39 40  
  
> # row 3 col 2  
> m['Row3', 'Col2']  
[1] 33
```

Extracting data from
Row 3 and Col 2

Subset in Lists

```
> info <- list(name = "John", age = 45, hobbies = c('golf',  
'swimming'),  
+ member = "TRUE")  
>  
> print(info[1])  
$name  
[1] "John"  
  
> class(info[1])  
[1] "list"  
> print(info[c(1,4)])  
$name  
[1] "John"  
  
$member  
[1] "TRUE"  
  
>  
> print(info[[1]])  
[1] "John"  
> class(info[[1]])  
[1] "character"
```

Element extracted using [] returns a list and [] can be used to extract multiple elements.

Element extracted using [[]] returns the same class as the element itself and can extract only a single element at a time.

Subsetting Lists: Using Names Attribute

```
> info <- list(name = "John", age = 45,  
+ hobbies = c('golf', 'swimming'),  
+ member = "TRUE")  
> print(info['hobbies'])  
$hobbies  
[1] "golf" "swimming"  
  
> class(info['hobbies'])  
[1] "list"  
>  
> print(info[['hobbies']])  
[1] "golf" "swimming"  
> class(info[['hobbies']])  
[1] "character"  
>  
> print(info$hobbies)  
[1] "golf" "swimming"  
> class(info$hobbies)  
[1] "character"
```

“\$” operator is used to extract individual elements of a list

Missing Values

Missing Values

Missing values in R are represented by NA.

Undefined mathematical operations, such as division by zero, are represented by NaN (Not a Number).

```
> vec <- c(23, NA, 89, 45, 12)
> is.na(vec)
[1] FALSE TRUE FALSE FALSE FALSE
> is.nan(vec)
[1] FALSE FALSE FALSE FALSE FALSE
```

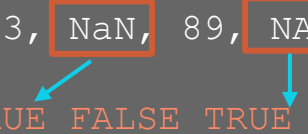
Note

is.na() and is.nan() functions are used to test for NA and NaN respectively.

Missing Values

An NaN value is also an NA, but an NA value is not a NaN value.

```
> vec <- c(23, NaN, 89, NA, 45, 12)
> is.na(vec)
[1] FALSE TRUE FALSE TRUE FALSE FALSE
```



is.na() returns TRUE for both NA and NaN

```
> vec <- c(23, NaN, 89, NA, 45, 12)
> is.nan(vec)
[1] FALSE TRUE FALSE FALSE FALSE FALSE
```



is.nan() returns TRUE only for NaN

Missing Values

Using a not operator (!), which converses the logical value, makes the statement return the nonmissing values.

```
> vec <- c(23, NaN, 89, NA, 45, 12)
> missing <- is.na(vec)
> print(!missing)
[1] TRUE FALSE TRUE FALSE TRUE TRUE
> vec[!missing]
[1] 23 89 45 12
```

Returns a vector with the opposite values of missing vector

Vectorized Operations

Vectorized Operations

In vectorized operations, operations are applied to each element of the object simultaneously.

```
> vec1 <- c(-12, 30, 15, -19, 22)
> vec1 * 2
[1] -24 60 30 -38 44
>
> vec1 <- 1 : 10
> vec2 <- 11 : 20
> vec1 + vec2
[1] 12 14 16 18 20 22 24 26 28 30
>
>
> vec1 <- c(45, 36, 20, 15, 49, 30)
> vec1 < 30
[1] FALSE FALSE TRUE TRUE FALSE FALSE
```

Each element being multiplied by 2

Corresponding elements of the vectors being added

Each element being evaluated for the expression

Vectorized Operations in Matrices

`%*%` is an operator used to compute inner product of matrices.

```
> m1 <- matrix(1:4, nrow = 2, ncol = 2)
> m2 <- matrix(rep(10,times = 4), nrow = 2, ncol
= 2)
>
> m1 * m2
  [,1] [,2]
[1,] 10 30
[2,] 20 40
>
> m1%*% m2
  [,1] [,2]
[1,] 40 40
[2,] 60 60
```

Element-wise multiplication

True matrix multiplication

Converting a Vector



Duration: 2 minutes

Problem Statement: Consider the following input and write a code to convert the temperatures stored in a vector as degree Celsius to degree Fahrenheit.

$$F = 9C/5 + 32$$

Temp_vector = c(34, 20, -40, 0, 45, 55)

Note: Please download the solution document from the Course Resources section and follow the steps given in the document.

ASSISTED PRACTICE

Key Takeaways

- R is an open-source programming language developed for statistical computing and applied data science.
- R has several IDEs for implementation; R Studio being the most popular of them all.
- R has five basic classes of object namely character, numeric, integer, complex, and logical.
- R provides for different data structures for storage, such as vectors, matrices, data-frames and lists.





Knowledge Check

**Knowledge
Check**

1

What will be the output for the below code?

```
print(17 %/% 3)
```

- A. [1] 5.67
- B. [1] 5
- C. [1] 2
- D. [1] 0.67



Knowledge
Check

1

What will be the output for the below code?

```
print(17 %/% 3)
```

- A. [1] 5.67
- B. [1] 5
- C. [1] 2
- D. [1] 0.67



The correct answer is **B**

The `%/%` is the integer divide operator which returns the quotient of the division.

Knowledge Check

2

Consider the code: `vec <- c('Allan', 'Dave', 'John', 'Kim', 'Jack', 'Kate')`
What will be the output of `vec[2:5]`?

- A. [1] "John" "Kim" "Jack" "Kate"
- B. [1] "Dave" "John" "Kim" "Jack"
- C. [1] "Dave" "John" "Kim"
- D. [1] "Allan" "Dave" "John"



**Knowledge
Check**

2

Consider the code: `vec <- c('Allan', 'Dave', 'John', 'Kim', 'Jack', 'Kate')`
What will be the output of `vec[2:5]`?

- A. [1] "John" "Kim" "Jack" "Kate"
- B. [1] "Dave" "John" "Kim" "Jack"
- C. [1] "Dave" "John" "Kim"
- D. [1] "Allan" "Dave" "John"



The correct answer is **B**

The code will create a subset of vector, from the index location 2 to index 5 where both are included.

**Knowledge
Check**
3

**is.na() functions returns TRUE when the value is:
(select all that apply)**

- A. NA
- B. NaN
- C. -1
- D. 0



Knowledge
Check

3

`is.na()` functions returns TRUE when the value is:
(select all that apply)

- A. NA
- B. NaN
- C. -1
- D. 0



The correct answers are **A and B**

`is.na()` recognizes both NAs and NaNs. Therefore, it returns TRUE for both NA and NaN values.