R coding

Mariam Iftikhar

2025-02-15

Accessing & Deleting Objects:

```r
ls()
```

```
## character(0)
```

```r
objects()
```

```
## character(0)
```

```r
rm(a)
```

```
## Warning in rm(a): object 'a' not found
```

If you wish to delete all of the variables in the workspace, the list option in the rm command can be combined with the ls command, as follows:

```r
rm(list=ls())
```

Type to check Command

- Integer: is.integer
- Logical: is.logical
- Character: is.character
- Factor: is.factor
- Double: is.double
- Complex: is.complex
- NA: is.NA
- List: is.list

Type to convert to Command:

- Integer: as.integer
- Logical: as.logical

- Character: as.character
- Factor: as.factor
- Double: as.double
- Complex: as.complex
- NA: as.NA
- List: as.list

```r
thisNumber <- as.integer(8/3)
typeof(thisNumber)
```

```
## [1] "integer"
```

Basic Data Types

Basic data types in R can be divided into the following types:

- numeric - (10.5, 55, 787)

- integer - (1L, 55L, 100L, where the letter "L" declares this as an integer)

- complex - (9 + 3i, where "i" is the imaginary part)

- character (a.k.a. string) - ("k", "R is exciting", "FALSE", "11.5")

- logical (a.k.a. boolean) - (TRUE or FALSE)

We can use the class() function to check the data type of a variable

Operators

1. Assignment Operators:

- Addition x + y

- Subtraction x - y

- Multiplication x * y

- Division x / y

- Exponent x ˆ y

- Modulus (Remainder from division) x %% y

- Integer Division x%/%y

2. R Comparison Operators:

Comparison operators are used to compare two values:

- Equal x == y
- Not equal x != y
- Greater than x > y
- Less than x < y
- Greater than or equal to x >= y
- Less than or equal to x <= y

3. R Logical Operators:

Logical operators are used to combine conditional statements:

- & Element-wise Logical AND operator. Returns TRUE if both elements are TRUE
- && Logical AND operator - Returns TRUE if both statements are TRUE
- Elementwise- Logical OR operator. Returns TRUE if one of the statements is TRUE
- || Logical OR operator. Returns TRUE if one of the statements is TRUE
- ! Logical NOT - Returns FALSE if statement is TRUE

4. R Miscellaneous Operators:

Miscellaneous operators are used to manipulate data:

- Creates a series of numbers in a sequence x <- 1:10
- %in% Find out if an element belongs to a vector x %in% y
- *%% Matrix Multiplication x <- Matrix1 %%* Matrix2

String functions

```r
str <- "Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."

cat(str)
```

```
## Lorem ipsum dolor sit amet,
## consectetur adipiscing elit,
## sed do eiusmod tempor incididunt
## ut labore et dolore magna aliqua.
```

```r
str <- "Hello World!"

nchar(str)
```

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

```r
str <- "Hello World!"

grepl("H", str)
```

```
## [1] TRUE
```

```r
grepl("Hello", str)
```

```
## [1] TRUE
```

```r
grepl("X", str)
```

```
## [1] FALSE
```

```r
str1 <- "Hello"
str2 <- "World"

paste(str1, str2)
```

```
## [1] "Hello World"
```

```r
one <- "A"
two <- "B"
numbers <- paste(one,two,sep="/")
numbers
```

```
## [1] "A/B"
```

Factor Function

```r
# Create a factor
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

# Print the factor
music_genre
```

```
## [1] Jazz    Rock    Classic Classic Pop     Jazz    Rock    Jazz
## Levels: Classic Jazz Pop Rock
```

```r
levels(music_genre)
```

```
## [1] "Classic" "Jazz"    "Pop"     "Rock"
```

Matrices Functions

```r
# Create a matrix
thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)

# Print the matrix
thismatrix
```

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

```r
thismatrix <- matrix(c("apple", "banana", "cherry", "orange","grape", "pineapple", "pear", "melon", "fi

thismatrix[c(1,2),]
```

```
##      [,1]     [,2]     [,3]
## [1,] "apple"  "orange" "pear"
## [2,] "banana" "grape"  "melon"
```

```r
thismatrix <- matrix(c("apple", "banana", "cherry", "orange","grape", "pineapple", "pear", "melon", "fi

newmatrix <- cbind(thismatrix, c("strawberry", "blueberry", "raspberry"))
newmatrix
```

```
##      [,1]     [,2]        [,3]    [,4]
## [1,] "apple"  "orange"    "pear"  "strawberry"
## [2,] "banana" "grape"     "melon" "blueberry"
## [3,] "cherry" "pineapple" "fig"   "raspberry"
```

```r
thismatrix <- matrix(c("apple", "banana", "cherry", "orange", "mango", "pineapple"), nrow = 3, ncol =2)

#Remove the first row and the first column
thismatrix <- thismatrix[-c(1), -c(1)]
thismatrix
```

```
## [1] "mango"     "pineapple"
```

Vectors Functions

R Vectors are the same as the arrays in R language which are used to hold multiple data values of the same type. One major key point is that in R Programming Language the indexing of the vector will start from '1' and not from '0'. We can create numeric vectors and character vectors as well.

```r
# Create a numeric vector
x <- c(1, 2, 3, 4, 5)

# Find the length of the vector
length(x)
```

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

```r
# Create a character vector
y <- c("apple", "banana", "cherry")

# Find the length of the vector
length(y)
```

```
## [1] 3
```

```r
# Create a logical vector
z <- c(TRUE, FALSE, TRUE, TRUE)

# Find the length of the vector
length(z)
```

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

```r
# R program to sort elements of a Vector

# Creation of Vector
X<- c(8, 2, 7, 1, 11, 2)

# Sort in ascending order
A<- sort(X)
cat('ascending order', A, '\n')
```

```
## ascending order 1 2 2 7 8 11
```

```r
# sort in descending order
# by setting decreasing as TRUE
B<- sort(X, decreasing = TRUE)
cat('descending order', B)
```

```
## descending order 11 8 7 2 2 1
```

```r
# R program to delete a Vector

# Creating a Vector
M<- c(8, 10, 2, 5)

# set NULL to the vector
M<- NULL
cat('Output vector', M)
```

```
## Output vector
```

Arrays

Arrays are the R data objects which can store data in more than two dimensions. An array is created using the array() function. It takes vectors as input and uses the values in the dim parameter to create an array.

```r
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2))
print(result)
```

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

```r
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,column.names,
   matrix.names))
print(result)
```

```
## , , Matrix1
##
##      COL1 COL2 COL3
## ROW1    5   10   13
## ROW2    9   11   14
## ROW3    3   12   15
##
## , , Matrix2
##
##      COL1 COL2 COL3
## ROW1    5   10   13
## ROW2    9   11   14
## ROW3    3   12   15
```

Lists

```
thislist <- list("apple", "banana", "cherry")
thislist
```

```
## [[1]]
## [1] "apple"
##
## [[2]]
## [1] "banana"
##
## [[3]]
## [1] "cherry"
```

```
thislist <- list("apple", "banana", "cherry")
newlist <- thislist[-1]
newlist
```

```
## [[1]]
## [1] "banana"
##
## [[2]]
## [1] "cherry"
```

```
thislist <- list("apple", "banana", "cherry")
append(thislist, "orange", after = 2)
```

```
## [[1]]
## [1] "apple"
##
## [[2]]
## [1] "banana"
##
## [[3]]
## [1] "orange"
##
## [[4]]
## [1] "cherry"
```

```
list1 <- list("a", "b", "c")
list2 <- list(1,2,3)
list3 <- c(list1,list2)
list3
```

```
## [[1]]
## [1] "a"
##
## [[2]]
## [1] "b"
##
```

```
## [[3]]
## [1] "c"
##
## [[4]]
## [1] 1
##
## [[5]]
## [1] 2
##
## [[6]]
## [1] 3
```

Functions

```r
converttemp <- function(value,from,to)
{
  if (from == "celsius" && to == "fahrenheit") {
    return(value * 9/5 + 32)
  } else if (from == "celsius" && to == "kelvin") {
    return(value + 273.15)
  } else {
  print("Invalid conversion units.")}}


converttemp(100,"celsius","fahrenheit")
```

```
## [1] 212
```

```r
converttemp(100,"kelvin","fahrenheit")
```

```
## [1] "Invalid conversion units."
```

Loops

1. For Loop

```r
week <- c('Sunday',
          'Monday',
          'Tuesday',
          'Wednesday',
          'Thursday',
          'Friday',
          'Saturday')

# using for loop to iterate
# over each string in the vector
for (day in week)
```

```r
{

  # displaying each string in the vector
  print(day)
}
```

```
## [1] "Sunday"
## [1] "Monday"
## [1] "Tuesday"
## [1] "Wednesday"
## [1] "Thursday"
## [1] "Friday"
## [1] "Saturday"
```

2. While Loop

```r
val = 1

# using while loop
while (val <= 5)
{
  # statements
  print(val)
  val = val + 1
}
```

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

```r
# assigning value to the variable whose factorial will be calculated
n <- 5

factorial <- 1
i <- 1

# using while loop
while (i <= n)
{

  # multiplying the factorial variable
  # with the iteration variable
  factorial = factorial * i

  # incrementing the iteration variable
  i = i + 1
}

# displaying the factorial
print(factorial)
```

```
## [1] 120
```

3. Repeat Loop

```
val = 1
# using repeat loop
repeat
{
  # statements
  print(val)
  val = val + 1

  # checking stop condition
  if(val > 5)
  {
    # using break statement
    # to terminate the loop
    break
  }
}
```

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

If-else statements

```
# creating values
var1 <- 6
var2 <- 5
var3 <- -4

# checking if-else if ladder
if(var1 > 10 || var2 < 5){
  print("condition1")
}else{
  if(var1 <4 ){
    print("condition2")
  }
    else{
      print("condition4")
    }
  }
```

```
## [1] "condition4"
```

```
# creating a dataframe
data_frame <- data.frame(col1 = c(1:9),
                         col2 = LETTERS[1:3])

print("Original DataFrame")
```

```
## [1] "Original DataFrame"
```

```
print(data_frame)
```

```
##   col1 col2
## 1    1    A
## 2    2    B
## 3    3    C
## 4    4    A
## 5    5    B
## 6    6    C
## 7    7    A
## 8    8    B
## 9    9    C
```

```
data_frame$col3 = with(data_frame,
                       ifelse(col1>4,"cond1 satisfied",
                              ifelse(col2 %in% c("A","C"),
                                     "cond2 satisfied",
                                     "both failed")))

print("Modified DataFrame")
```

```
## [1] "Modified DataFrame"
```

```
print(data_frame)
```

```
##   col1 col2            col3
## 1    1    A cond2 satisfied
## 2    2    B     both failed
## 3    3    C cond2 satisfied
## 4    4    A cond2 satisfied
## 5    5    B cond1 satisfied
## 6    6    C cond1 satisfied
## 7    7    A cond1 satisfied
## 8    8    B cond1 satisfied
## 9    9    C cond1 satisfied
```

```
if(0) {
   cat("Yes, that is FALSE.\n")
 } else if (1) {
 cat("Yes that is TRUE\n")
 } else {
 cat("Whatever")
 }
```

```
## Yes that is TRUE
```

Input and output Functions

```
n <- readline(prompt="Please, enter your ANSWER: ")
```

```
## Please, enter your ANSWER:
```

Error handling

```
data <- data.frame(
  ID = c(1, 2, 3, 4),
  Name = c("Alice", "Bob", "Charlie", "Diana"),
  Age = c(25, 30, 22, 28),
  Score = c(90, 85, 88, 92)
)

# Print the data frame
print("Original Data Frame:")
```

```
## [1] "Original Data Frame:"
```

```
print(data)
```

```
##   ID    Name Age Score
## 1  1   Alice  25    90
## 2  2     Bob  30    85
## 3  3 Charlie  22    88
## 4  4   Diana  28    92
```

```
tryCatch({
  # Attempt a division by zero
  print("Attempting division by zero:")
  result <- data$Score / 0
  print(result)
}, warning = function(w) {
  print("Warning occurred:")
  print(w)
}, error = function(e) {
  print("Error: Division by zero is not allowed.")
})
```

```
## [1] "Attempting division by zero:"
## [1] Inf Inf Inf Inf
```

Dataframe

```r
data <- data.frame(
  ID = c(1, 2, 3, 4),
  Name = c("Alice", "Bob", "Charlie", "Diana"),
  Age = c(25, 30, 22, 28),
  Score = c(90, 85, 88, 92)
)

# Print the data frame
print("Original Data Frame:")
```

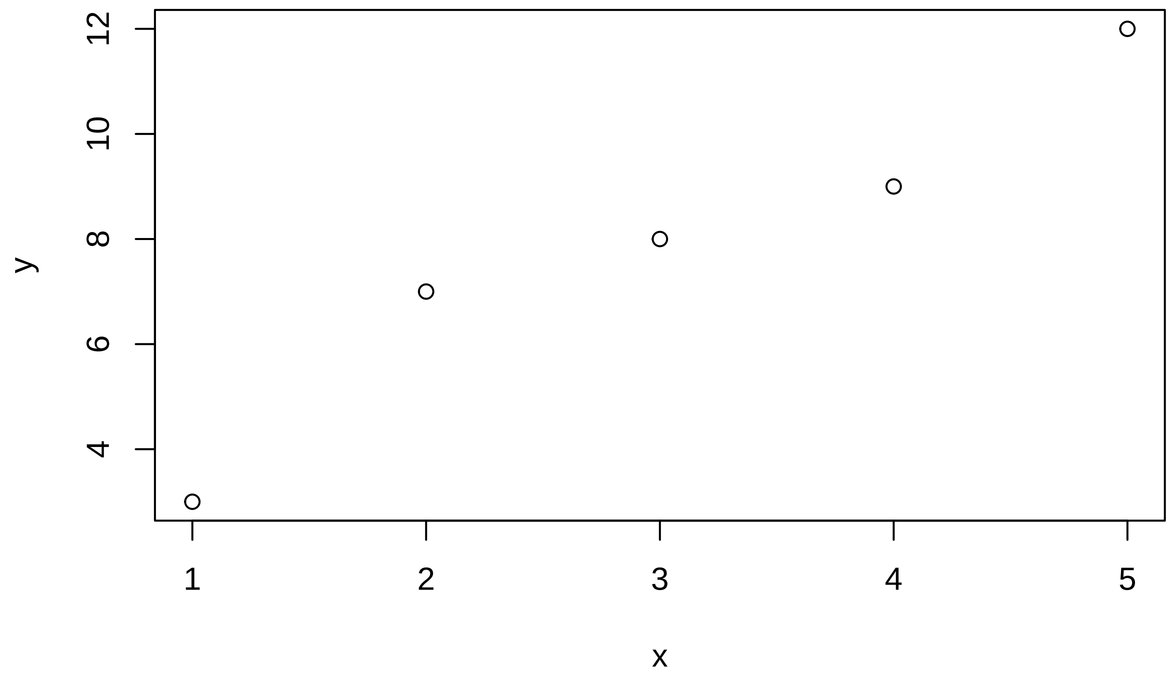
```
## [1] "Original Data Frame:"
```

```r
print(data)
```

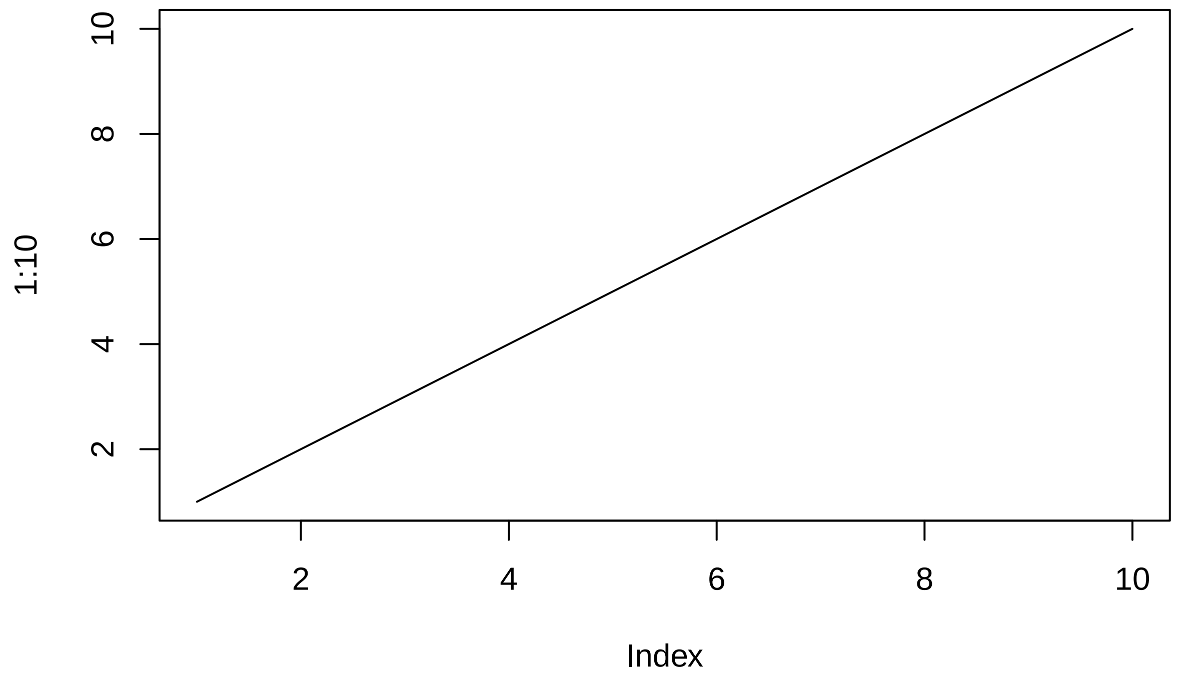
```
##   ID    Name Age Score
## 1  1   Alice  25    90
## 2  2     Bob  30    85
## 3  3 Charlie  22    88
## 4  4   Diana  28    92
```

Plot Functions

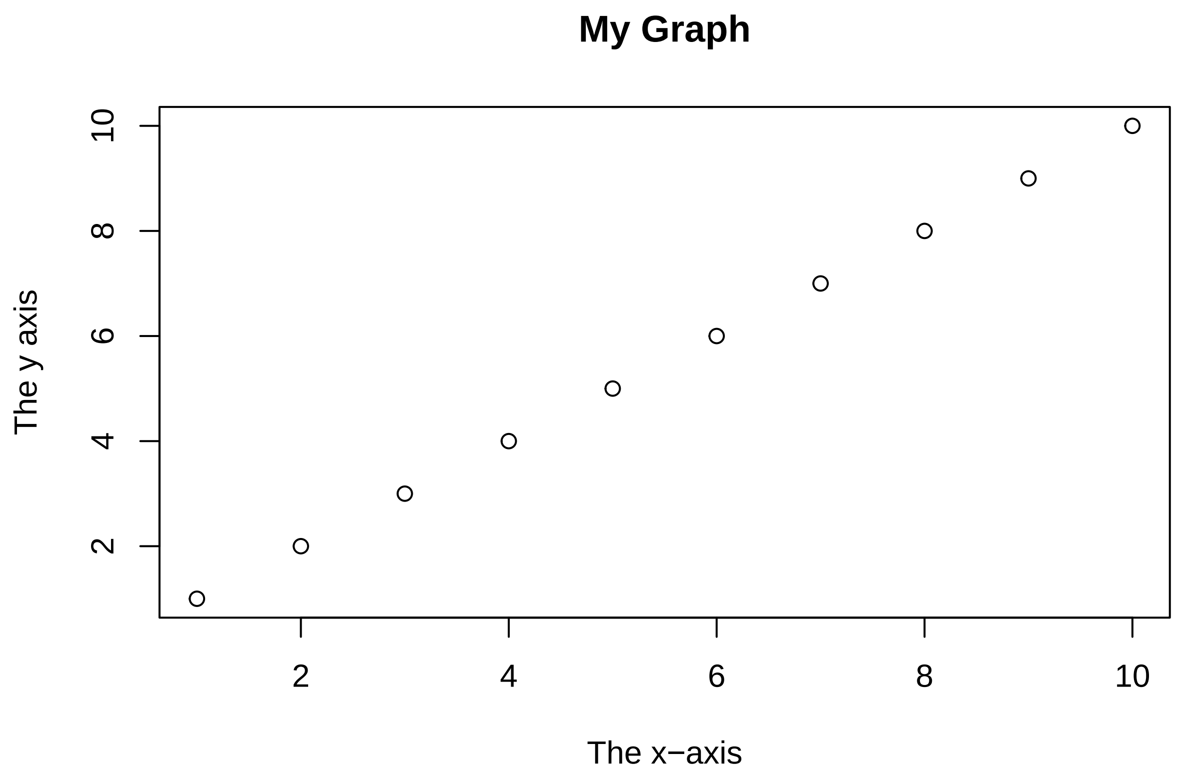
```r
x <- c(1, 2, 3, 4, 5)
y <- c(3, 7, 8, 9, 12)

plot(x, y)
```



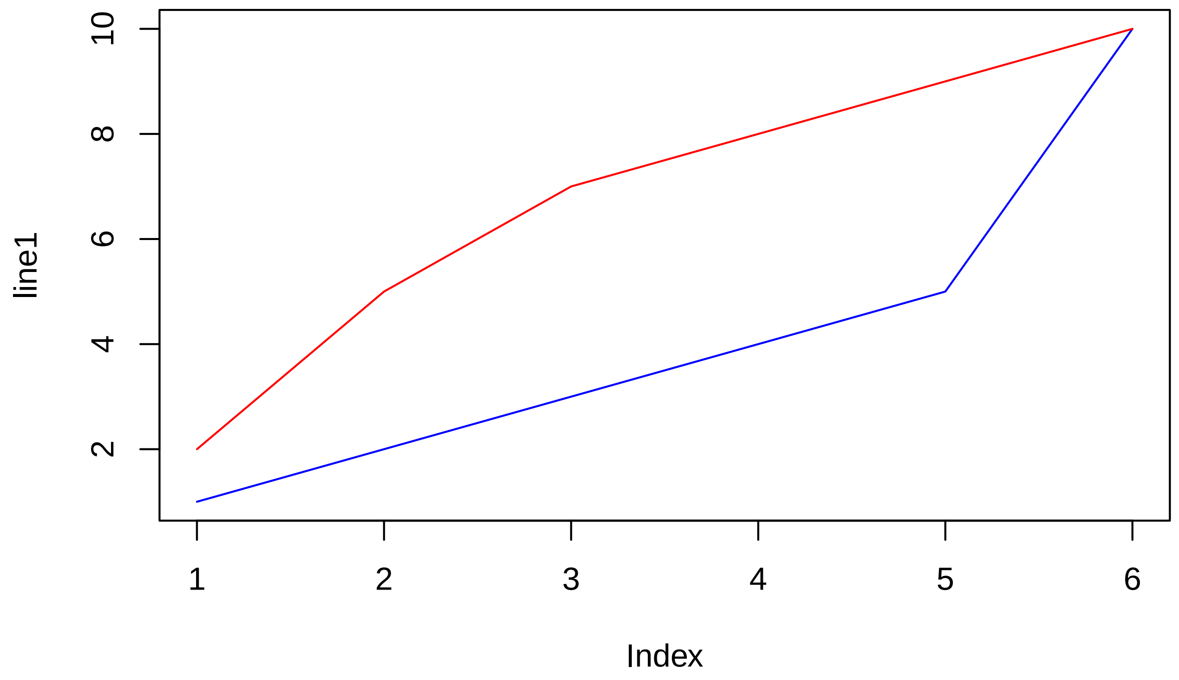
```
plot(1:10, type="l")
```



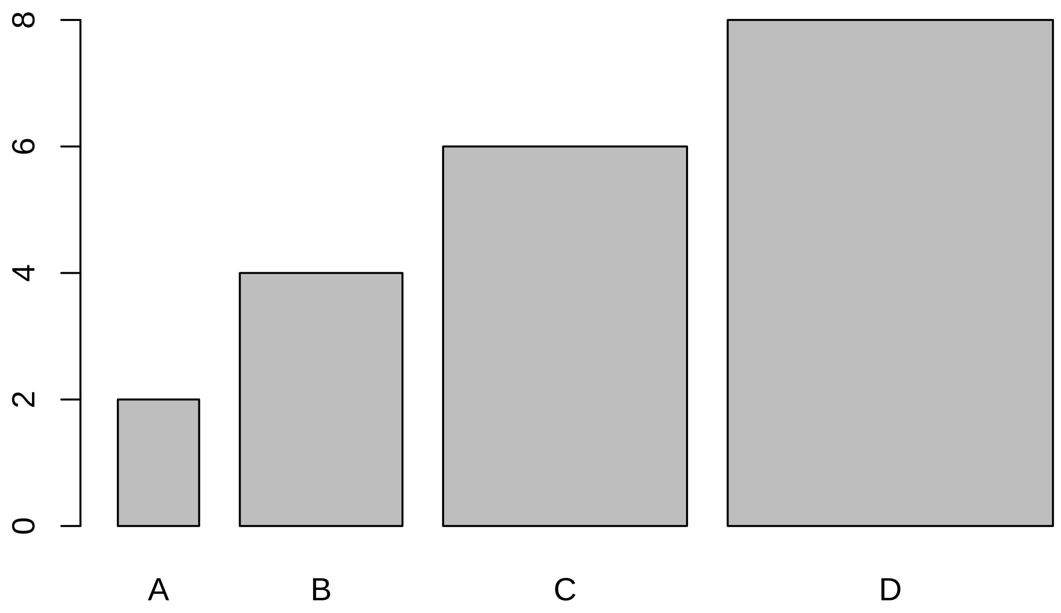
```r
plot(1:10, main="My Graph", xlab="The x-axis", ylab="The y axis")
```



```
line1 <- c(1,2,3,4,5,10)
line2 <- c(2,5,7,8,9,10)

plot(line1, type = "l", col = "blue")
lines(line2, type="l", col = "red")
```



```
x <- c("A", "B", "C", "D")
y <- c(2, 4, 6, 8)

barplot(y, names.arg = x, width = c(1,2,3,4))
```



data.table Functions

```
head(ceosal1,4)
```

```
##   salary pcsalary   sales  roe pcroe ros indus finance consprod utility
## 1   1095       20 27595.0 14.1 106.4 191     1       0        0       0
## 2   1001       32  9958.0 10.9 -30.6  13     1       0        0       0
## 3   1122        9  6125.9 23.5 -16.3  14     1       0        0       0
## 4    578       -9 16246.0  5.9 -25.7 -21     1       0        0       0
##    lsalary    lsales
## 1 6.998509 10.225389
## 2 6.908755  9.206132
## 3 7.022868  8.720281
## 4 6.359574  9.695602
```

```
tail(ceosal1,3)
```

```
##     salary pcsalary  sales  roe pcroe ros indus finance consprod utility
## 207    658       32 4542.6 12.1  -7.8  68     0       0        0       1
## 208    555        6 2023.0 13.7 -14.6  60     0       0        0       1
## 209    626        0 1442.5 14.4 -10.2  62     0       0        0       1
##      lsalary   lsales
## 207 6.489205 8.421255
## 208 6.318968 7.612337
## 209 6.439351 7.274133
```

```
names(ceosal1)
```

```
##  [1] "salary"   "pcsalary" "sales"    "roe"      "pcroe"    "ros"
##  [7] "indus"    "finance"  "consprod" "utility"  "lsalary"  "lsales"
```

```
colnames(ceosal1)
```

```
##  [1] "salary"   "pcsalary" "sales"    "roe"      "pcroe"    "ros"
##  [7] "indus"    "finance"  "consprod" "utility"  "lsalary"  "lsales"
```

```
class(ceosal1)
```

```
## [1] "data.frame"
```

```
x <- as.data.table(ceosal1)
class(x)
```

```
## [1] "data.table" "data.frame"
```

```
x2 <- x[1:4,list(salary,sales)] #dt way
x2
```

```
##    salary   sales
##     <int>   <num>
## 1:   1095 27595.0
## 2:   1001  9958.0
## 3:   1122  6125.9
## 4:    578 16246.0
```

```
x3 <- x[1:4,1:4]
x3
```

```
##    salary pcsalary   sales   roe
##     <int>    <int>   <num> <num>
## 1:   1095       20 27595.0  14.1
## 2:   1001       32  9958.0  10.9
## 3:   1122        9  6125.9  23.5
## 4:    578       -9 16246.0   5.9
```

```
class(x3)
```

```
## [1] "data.table" "data.frame"
```

```
y2 <- x3[salary==1001,sales] #dt way
y2
```

```
## [1] 9958
```

```
newcol <- x3[,addsales:= salary + pcsalary] #dt way
x3
```

```
copydt <- copy(x3) #dt way
```

```
y4 <- x[1:4,list(pcroe)] #dt way
y4
```

```
##    pcroe
##    <num>
## 1: 106.4
## 2: -30.6
## 3: -16.3
## 4: -25.7
```

```
x4 <- cbind(x3,y4) #dt way
x4
```

```
##    salary pcsalary   sales   roe addsales pcroe
##     <int>    <int>   <num> <num>    <int> <num>
## 1:   1095       20 27595.0  14.1     1115 106.4
## 2:   1001       32  9958.0  10.9     1033 -30.6
## 3:   1122        9  6125.9  23.5     1131 -16.3
## 4:    578       -9 16246.0   5.9      569 -25.7
```

```
x4$addsales=NULL # remove col in dt
x4
```

```
##    salary pcsalary   sales   roe pcroe
##     <int>    <int>   <num> <num> <num>
## 1:   1095       20 27595.0  14.1 106.4
## 2:   1001       32  9958.0  10.9 -30.6
## 3:   1122        9  6125.9  23.5 -16.3
## 4:    578       -9 16246.0   5.9 -25.7
```

Aggregations

df[i,j,by=] i=row,j=agg funct on col,by= which col

```
y5<- x4[,.(meansal=mean(salary),maxsales=max(sales))]
y5
```

```
##    meansal maxsales
##      <num>    <num>
## 1:     949    27595
```

```
y6asc <- x4[order(salary)]
y6asc
```

```
##    salary pcsalary   sales   roe pcroe
##     <int>    <int>   <num> <num> <num>
## 1:    578       -9 16246.0   5.9 -25.7
## 2:   1001       32  9958.0  10.9 -30.6
## 3:   1095       20 27595.0  14.1 106.4
## 4:   1122        9  6125.9  23.5 -16.3
```

```
y7desc <- x4[order(-salary)]
```

Delete row

```
x21 <- x2[salary != 1095]
x21
```

```
##    salary   sales
##     <int>   <num>
## 1:   1001  9958.0
## 2:   1122  6125.9
## 3:    578 16246.0
```

Set a key on the ID column

```
dt <- data.table(
  ID = c(1, 2, 3, 4, 5),
  Name = c("John", "Jane", "Paul", "Anna", "Tom"),
  Age = c(28, 34, 29, 22, 40)
)

setkey(dt, ID)
```

Xts Functions

```
xts4 <- xts(x=1:10, order.by=Sys.Date()+seq(1,20,2))
xts4
```

```
##            [,1]
## 2025-02-16    1
## 2025-02-18    2
## 2025-02-20    3
## 2025-02-22    4
## 2025-02-24    5
## 2025-02-26    6
## 2025-02-28    7
## 2025-03-02    8
## 2025-03-04    9
## 2025-03-06   10
```

```
str(xts4)
```

```
## An xts object on 2025-02-16 / 2025-03-06 containing:
##   Data:    integer [10, 1]
##   Index:   Date [10] (TZ: "UTC")
```

```
class(xts4)
```

```
## [1] "xts" "zoo"
```

What is the index?

tclass or indexClass: attribute for extraction

tzone or indexTZ: attribute for time zones

```
index(xts4)
```

```
##  [1] "2025-02-16" "2025-02-18" "2025-02-20" "2025-02-22" "2025-02-24"
##  [6] "2025-02-26" "2025-02-28" "2025-03-02" "2025-03-04" "2025-03-06"
```

```
coredata(xts4)
```

```
##       [,1]
##  [1,]    1
##  [2,]    2
##  [3,]    3
##  [4,]    4
##  [5,]    5
##  [6,]    6
##  [7,]    7
##  [8,]    8
##  [9,]    9
## [10,]   10
```

```
tzone(xts4)
```

```
## [1] "UTC"
```

```
tclass(xts4)
```

```
## [1] "Date"
```

Finding Start And End Points

```r
start(xts4)
```

```
## [1] "2025-02-16"
```

```r
end(xts4)
```

```
## [1] "2025-03-06"
```

```r
frequency(xts4)
```

```
## [1] 0.5
```

Counting No. of Days/ etc

```r
ndays(xts4)
```

```
## [1] 10
```

```r
nmonths(xts4)
```

```
## [1] 2
```

```r
nquarters(xts4)
```

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

```r
nyears(xts4)
```

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

Changing Format of Index

```r
indexFormat(xts4) <- "%b-%d-%Y"
```

```
## Warning in ‘indexFormat<-‘(‘*tmp*‘, value = "%b-%d-%Y"): ’indexFormat<-’ is deprecated.
## Use ’tformat<-’ instead.
## See help("Deprecated") and help("xts-deprecated").
```

```r
index(xts4)[1:3]
```

```
## [1] "2025-02-16" "2025-02-18" "2025-02-20"
```

First & Last Functions

```r
# Create lastweek using the last 1 week of temps
lastweek <- last(xts4, "1 week")

# Print the last 2 observations in lastweek
last(lastweek, 2)
```

```
##             [,1]
## Mar-04-2025    9
## Mar-06-2025   10
```

```r
# Extract all but the first two days of lastweek
first(lastweek, "-2 days")
```

```
##      [,1]
```

Combining first and last

```r
# Last 3 days of first week
last(first(xts4, '1 week'), '3 days')
```

```
## Warning in periodicity(x): can not calculate periodicity of 1 observation
```

```
## Warning in last.xts(first(xts4, "1 week"), "3 days"): requested length is
## greater than original
```

```
##             [,1]
## Feb-16-2025    1
```

```r
# Extract the first three days of the second week of temps
first(last(first(xts4, "2 weeks"), "1 week"), "3 days")
```

```
##             [,1]
## Feb-18-2025    2
## Feb-20-2025    3
## Feb-22-2025    4
```

Arithematic Operations

```r
xts4 * as.numeric(xts4)
```

```
##             [,1]
## Feb-16-2025    1
## Feb-18-2025    4
## Feb-20-2025    9
## Feb-22-2025   16
## Feb-24-2025   25
## Feb-26-2025   36
## Feb-28-2025   49
## Mar-02-2025   64
## Mar-04-2025   81
## Mar-06-2025  100
```

```r
coredata(xts4) - xts4
```

```
##               [,1]
## Feb-16-2025    0
## Feb-18-2025    0
## Feb-20-2025    0
## Feb-22-2025    0
## Feb-24-2025    0
## Feb-26-2025    0
## Feb-28-2025    0
## Mar-02-2025    0
## Mar-04-2025    0
## Mar-06-2025    0
```

```r
coredata(xts4) / xts4
```

```
##               [,1]
## Feb-16-2025    1
## Feb-18-2025    1
## Feb-20-2025    1
## Feb-22-2025    1
## Feb-24-2025    1
## Feb-26-2025    1
## Feb-28-2025    1
## Mar-02-2025    1
## Mar-04-2025    1
## Mar-06-2025    1
```

Periodicity to to.period Function

periodicity Function The periodicity function determines the frequency of an xts object (e.g., daily, weekly, monthly, etc.). It is used to understand the temporal granularity of your time series. Periodicity: daily Frequency: 1

Aggregates data into specified periods. Converting data for analysis or plotting. to.period Function The to.period function aggregates xts data into specified time periods such as daily, weekly, monthly, or yearly. It's often used for financial data, such as converting minute-level stock prices to daily prices.

to.period(x, period = "seconds", k = 1, indexAt = "endof", OHLC = FALSE, …) Parameters: x: An xts object. period: The target period for aggregation (e.g., "seconds", "minutes", "hours", "days", "weeks", "months", "years"). k: Specifies the frequency multiplier (e.g., k = 2 for every 2 days). indexAt: Indicates where to place the period's index ("startof" or "endof"). OHLC: Whether to calculate Open, High, Low, Close prices (for financial data).

Dealing with Missing Values

- Omit NA values

- na.locf is particularly useful for time-series data where missing values must be filled without interpolation. The fromLast argument allows flexibility in filling direction: FALSE (default): Last Observation Carried Forward. TRUE: Next Observation Carried Backward.
- Interpolate NAs using linear approximation

Types of Apply Functions

- apply(): Used for applying functions to rows or columns of matrices or 2D arrays.
- lapply(): Applies a function to each element of a list and returns a list (even if each element is a single value).
- sapply(): Similar to lapply(), but tries to simplify the result into an array (e.g., vector or matrix).
- vapply(): Similar to sapply(), but you specify the expected output type.
- mapply(): Applies a function to multiple arguments (e.g., vectors or lists) simultaneously.
- tapply(): Applies a function to subsets of a vector, grouped by a factor.

apply(): Purpose: Apply a function to the rows or columns of a matrix or a 2-dimensional array. Use Case: When working with matrices or 2D arrays and you want to perform operations on either rows or columns.

```
# Matrix Example
matrix_data <- matrix(1:6, nrow = 2)

apply(matrix_data, 1, sum)  # Apply sum to rows (1 indicates rows)
```

```
## [1]  9 12
```

```
apply(matrix_data, 2, sum)  # Apply sum to columns (2 indicates columns)
```

```
## [1]  3  7 11
```

lapply(): Purpose: Apply a function to each element of a list and return a list (it preserves the structure). Use Case: When working with lists and you want to apply a function to each element but return a list (even if the result is a single value for each element).

```
list_data <- list(a = 1:3, b = 4:6, c = 7:9)
lapply(list_data, sum)  # Apply sum to each list element
```

```
## $a
## [1] 6
##
## $b
## [1] 15
##
## $c
## [1] 24
```

sapply(): Purpose: Apply a function to each element of a list and attempt to simplify the result (e.g., returning a vector or matrix instead of a list). Use Case: When working with lists and you want the result simplified (e.g., a vector if possible).

```r
list_data <- list(a = 1:3, b = 4:6, c = 7:9)
sapply(list_data, sum)  # Apply sum and simplify the result
```

```
##  a  b  c
##  6 15 24
```

vapply(): Purpose: Similar to sapply(), but you explicitly specify the output type and structure. Use Case: When you want to ensure the result is of a specific type or structure, avoiding automatic simplification that might lead to unexpected results.

```r
list_data <- list(a = 1:3, b = 4:6, c = 7:9)
vapply(list_data, sum, numeric(1))  # Specify output as numeric(1)
```

```
##  a  b  c
##  6 15 24
```

mapply(): Purpose: Apply a function to multiple arguments (i.e., vectors or lists) simultaneously, often used for functions that take multiple arguments. Use Case: When you have several inputs and you want to apply a function to corresponding elements across these inputs.

```r
x <- 1:3
y <- 4:6
mapply(sum, x, y)  # Apply sum to corresponding elements of x and y
```

```
## [1] 5 7 9
```

tapply(): Purpose: Apply a function to subsets of a vector based on a factor or grouping variable (often used in data analysis). Use Case: When you need to apply a function to a vector and group the data by some factor or grouping variable.

```r
group <- factor(c("A", "B", "A", "B", "A"))
data <- c(1, 2, 3, 4, 5)
tapply(data, group, sum)  # Apply sum to data, grouped by 'group'
```

```
## A B
## 9 6
```

Lead and Lag Functions

Key Differences: Function Purpose Direction of Shift Example Usage lag Backward shift Past (default k > 0) Comparing current values to prior periods. lead Forward shift Future (via k < 0) Analyzing future relationships or trends.

Rolling functions

Discrete -> lapply(), split()

Continuous -> rollapply()

Discrete rolling windows split() to break up by period lapply() cumulative functions cumsum(), cumprod(), cummin(), cummax()

Continuous rolling windows rollapply(data, width, FUN, . . . , by = 1, by.column = TRUE, fill = if (na.pad) NA, na.pad = TRUE, partial = TRUE, align = c("right", "center", "left"))

data is your xts object width is the window size FUN is your function to apply can add additional arguments to your function

Importing File

You can use read.zoo() for direct import from CSV files. Alternatively, use read.csv() and convert to xts.

```
data <- data.frame(
  Date = seq(as.Date("2023-01-01"), as.Date("2023-01-07"), by = "days"),
  Value = c(10, 15, 20, 18, 25, 30, 28)
)

# Convert to xts object
xts_data <- xts(data$Value, order.by = data$Date)
print("Imported xts Data:")
```

```
## [1] "Imported xts Data:"
```

```
print(xts_data)
```

```
##            [,1]
## 2023-01-01   10
## 2023-01-02   15
## 2023-01-03   20
## 2023-01-04   18
## 2023-01-05   25
## 2023-01-06   30
## 2023-01-07   28
```

Exporting File

Use write.zoo() for exporting to text or CSV formats.

```
data <- data.frame(
  Date = seq(as.Date("2023-01-01"), as.Date("2023-01-10"), by = "days"),
  Sales = c(100, 120, 130, 140, 150, 160, 170, 180, 190, 200)
)
xts_sales <- xts(data$Sales, order.by = data$Date)

first_week <- xts_sales["2023-01-01/2023-01-07"]
write.zoo(first_week, file = "first_week_sales.csv", sep = ",", index.name = "Date")
```

Subsetting

Use date strings in the form “YYYY-MM-DD”. Ranges like “start_date/end_date” or “start_date/” make date selection flexible.

```r
dates <- seq(as.Date("2023-01-01"), as.Date("2023-01-06"), by = "days")
values <- c(10, 20, 30, 40, 50, 60)
xts_data <- xts(values, order.by = dates)

subset1 <- xts_data["2023-01-03"]
subset1
```

```
##            [,1]
## 2023-01-03   30
```

```r
# Select data for a range of dates
subset2 <- xts_data["2023-01-03/2023-01-05"]
subset2
```

```
##            [,1]
## 2023-01-03   30
## 2023-01-04   40
## 2023-01-05   50
```

```r
# Select data before a specific date
subset3 <- xts_data["/2023-01-04"]
subset3
```

```
##            [,1]
## 2023-01-01   10
## 2023-01-02   20
## 2023-01-03   30
## 2023-01-04   40
```

Merging

```r
# Add a to b, and fill all missing rows of b with 0
dates_a <- seq(as.Date("2023-01-01"), as.Date("2023-01-05"), by = "days")
a <- xts(c(10, 20, 30, 40, 50), order.by = dates_a)

# Create a time-series object 'b'
dates_b <- seq(as.Date("2023-01-03"), as.Date("2023-01-06"), by = "days")
b <- xts(c(5, 15, 25, 35), order.by = dates_b)

a + merge(b, index(a), fill = 0)
```

```
##            e1
## 2023-01-01 10
## 2023-01-02 20
## 2023-01-03 35
## 2023-01-04 55
## 2023-01-05 75
```

```r
# Perform a left-join of a and b, fill missing values with 0
merge(a, b, join = "left", fill = 0)
```

```
##             a  b
## 2023-01-01 10  0
## 2023-01-02 20  0
## 2023-01-03 30  5
## 2023-01-04 40 15
## 2023-01-05 50 25
```