Factors

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling.

Factors are created using the **factor ()** function by taking a vector as input.

Factors are used to categorize data.

Examples of factors are:

Demography: Male/Female

Music: Rock, Pop, Classic, Jazz

Training: Strength, Stamina

To create a factor, use the **factor()** function and add a vector as argument:

Example

```
# Create a factor

music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

# Print the factor

music_genre
```

Result:

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

Levels: Classic Jazz Pop Rock
```

You can see from the example above that that the factor has **four levels** (categories): Classic, Jazz, Pop and Rock.

To only print the **levels**, use the **levels()** function:

Example

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

levels(music_genre)
```

Result:

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

You can also set the levels, by adding the levels argument inside the **factor() function**

Example

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"), levels = c("Classic", "Jazz", "Pop", "Rock", "Other"))

levels(music_genre)
```

Result:

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

Factor Length

Use the **length()** function to find out how many items there are in the factor:

Example

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

length(music_genre)
```

Result:

```
[1] 8
```

Access Factors

To access the items in a factor, refer to the index number, using [] brackets:

Example

Access the third item

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

music_genre[3]
```

Result:

```
[1] Classic

Levels: Classic Jazz Pop Rock
```

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the value of the third item:

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

music_genre[3] <- "Pop"

music_genre[3]
```

Result:

```
[1] Pop

Levels: Classic Jazz Pop Rock
```

Note that you cannot change the value of a specific item if it is not already specified in the factor. The following example will produce an error:

Example

Trying to change the value of the third item ("Classic") to an item that does not exist/not predefined ("Opera"):

music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

music_genre[3] <- "Opera"

music_genre[3]

Result:

Warning message:

In `[<-.factor`(`*tmp*`, 3, value = "Opera") :

invalid factor level, NA generated

However, if you have already specified it inside the levels argument, it will work:

Example

Change the value of the third item:

music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"), levels = c("Classic", "Jazz", "Pop", "Rock", "Opera"))

music_genre[3] <- "Opera"

music_genre[3]

Result:

[1] Opera

Levels: Classic Jazz Pop Rock Opera

Changing the Order of Levels

The order of the levels in a factor can be changed by applying the factor function again with new order of the levels.

```
data <- c("East","West","East","North","North","East","West",
  "West","West","East","North")
# Create the factors
factor_data <- factor(data)
```

```
print(factor_data)

# Apply the factor function with required order of the level.
new_order_data <- factor(factor_data,levels = c("East","West","North"))
print(new_order_data)
```

When we execute the above code, it produces the following result −

```
 [1] East  West  East  North North East  West  West  West  East  North
Levels: East North West
 [1] East  West  East  North North East  West  West  West  East  North
Levels: East West North
```

Generating Factor Levels

We can generate factor levels by using the **gl()** function. It takes two integers as input which indicates how many levels and how many times each level.

Syntax

gl(n, k, labels)

Following is the description of the parameters used −

- **n** is a integer giving the number of levels.
- **k** is a integer giving the number of replications.
- **labels** is a vector of labels for the resulting factor levels.

Example

```
v <- gl(3, 4, labels = c("Tampa", "Seattle","Boston"))
print(v)
```

When we execute the above code, it produces the following result −

```
Tampa   Tampa   Tampa   Tampa   Seattle Seattle Seattle Seattle Boston
[10] Boston  Boston  Boston
Levels: Tampa Seattle Boston
```

Data Frames

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Data Frames are data displayed in a format as a table.

Data Frames can have different types of data inside it. While the first column can be character, the second and third can be numeric or logical. However, each column should have the same type of data.

Use the data.frame() function to create a data frame:

Example

```
# Create a data frame
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45))
# Print the data frame
Data_Frame
```

```
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
```

Summarize the Data

Use the summary() function to summarize the data from a Data Frame:

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame

summary(Data_Frame)
```

```
Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
     Training     Pulse          Duration
 Other   :1   Min.   :100.0   Min.   :30.0
 Stamina :1   1st Qu.:110.0   1st Qu.:37.5
 Strength:1   Median :120.0   Median :45.0
              Mean   :123.3   Mean   :45.0
              3rd Qu.:135.0   3rd Qu.:52.5
              Max.   :150.0   Max.   :60.0
```

Subsetting of data frames

Access Items

We can use single brackets [ ], double brackets [[ ]] or $ to access columns from a data frame:

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame[1]

Data_Frame[["Training"]]
```

Data_Frame$Training

```
Training
1 Strength
2  Stamina
3    Other
[1] Strength Stamina  Other
Levels: Other Stamina Strength
[1] Strength Stamina  Other
Levels: Other Stamina Strength
```

Expand Data Frame

A data frame can be expanded by adding columns and rows.

Add Rows

Use the rbind() function to add new rows in a Data Frame:

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Add a new row
New_row_DF <- rbind(Data_Frame, c("Strength", 110, 110))

# Print the new row
New_row_DF
```

```
Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
4 Strength   110      110
```

Add Columns

Use the cbind() function to add new columns in a Data Frame:

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Add a new column
New_col_DF <- cbind(Data_Frame, Steps = c(1000, 6000, 2000))

# Print the new column
New_col_DF
```

```
  Training Pulse Duration Steps
1 Strength   100       60  1000
2  Stamina   150       30  6000
3    Other   120       45  2000
```

Remove Rows and Columns

Use the c() function to remove rows and columns in a Data Frame:

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

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

# Print the new data frame
Data_Frame_New
```

```
  Pulse Duration
2   150       30
```

```
3    120        45
```

Amount of Rows and Columns

Use the dim() function to find the amount of rows and columns in a Data Frame:

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

dim(Data_Frame)
```

```
 [1] 3 3
```

You can also use the ncol() function to find the number of columns and nrow() to find the number of rows:

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

ncol(Data_Frame)
nrow(Data_Frame)
```

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

Data Frame Length

Use the length() function to find the number of columns in a Data Frame (similar to ncol()):

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

length(Data_Frame)
```

```
[1] 3
```

Combining Data Frames

Use the rbind() function to combine two or more data frames in R vertically:

Example

```
Data_Frame1 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame2 <- data.frame (
  Training = c("Stamina", "Stamina", "Strength"),
  Pulse = c(140, 150, 160),
  Duration = c(30, 30, 20)
)

New_Data_Frame <- rbind(Data_Frame1, Data_Frame2)
New_Data_Frame
```

```
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
4  Stamina   140       30
5  Stamina   150       30
6 Strength   160       20
```

And use the cbind() function to combine two or more data frames in R horizontally:

Example

```
Data_Frame3 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame4 <- data.frame (
  Steps = c(3000, 6000, 2000),
  Calories = c(300, 400, 300)
)

New_Data_Frame1 <- cbind(Data_Frame3, Data_Frame4)
New_Data_Frame1
```

```
  Training Pulse Duration Steps Calories
1 Strength   100       60  3000      300
2  Stamina   150       30  6000      400
3    Other   120       45  2000      300
```

R Sort a Data Frame using Order()

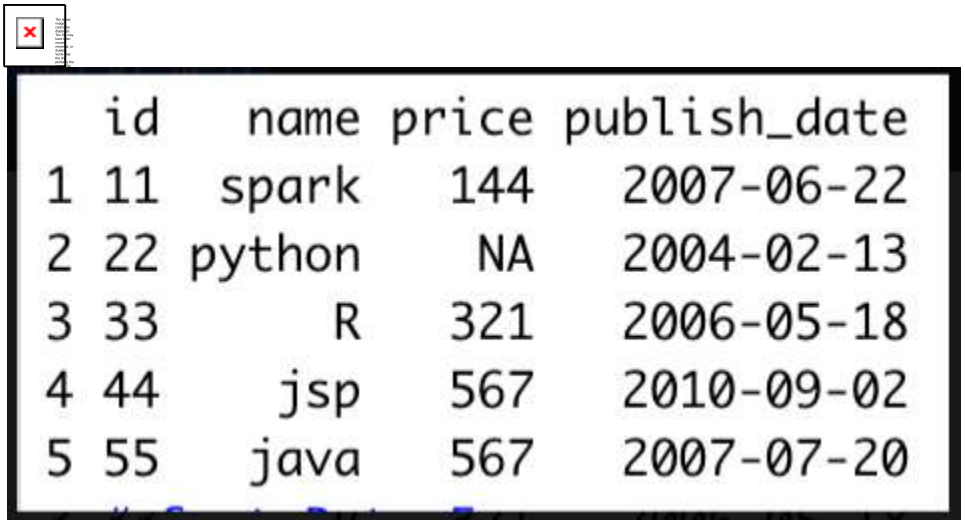
In data analysis you can **sort** your data according to a certain variable in the dataset. In R, we can use the help of the function order(). In R, we can easily sort a vector of continuous variable or factor variable. Arranging the data can be of **ascending** or **descending** order.

Syntax:sort(x, decreasing = FALSE, na.last = TRUE):

Argument:

- **x**: A vector containing continuous or factor variable
- **decreasing**: Control for the order of the sort method. By default, decreasing is set to `FALSE`.
- **last**: Indicates whether the `NA` ‘s value should be put last or not

```
# Create Data Frame
df=data.frame(id=c(11,22,33,44,55),
           name=c("spark","python","R","jsp","java"),
           price=c(144,NA,321,567,567),
           publish_date= as.Date(
             c("2007-06-22", "2004-02-13", "2006-05-18",
               "2010-09-02","2007-07-20"))
           )
```



	id	name	price	publish_date
1	11	spark	144	2007-06-22
2	22	python	NA	2004-02-13
3	33	R	321	2006-05-18
4	44	jsp	567	2010-09-02
5	55	java	567	2007-07-20

EXAMPLE

Let's use the above-created data.frame and order() function to sort the R dataframe by column value in ascending order. The following example sorts the data by column price. Since the order() function takes the vector as an argument, use df$price as an argument. Note that in R, every column in DataFrame is a vector.

```
# Sort DataFrame
df2 <- df[order(df$price),]
df2
```

	id	name	price	publish_date
1	11	spark	144	2007-06-22
3	33	R	321	2006-05-18
4	44	jsp	567	2010-09-02
5	55	java	567	2007-07-20
2	22	python	NA	2004-02-13

Lists

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

A list in R can contain many different data types inside it. A list is a collection of data which is ordered and changeable.

To create a list, use the list() function:

Example

```
# List of strings
thislist <- list("apple", "banana", "cherry")

# Print the list
thislist
```

Output

```
[[1]]
[1] "apple"
```

```
[[2]]
[1] "banana"

[[3]]
[1] "cherry"
```

Access Lists

You can access the list items by referring to its index number, inside brackets. The first item has index 1, the second item has index 2, and so on:

Example

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

thislist[1]
```

Output

```
[[1]]
[1] "apple"
```

Change Item Value

To change the value of a specific item, refer to the index number:

Example

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

# Print the updated list
thislist
```

Output

```
[[1]]
[1] "blackcurrant"

[[2]]
[1] "banana"

[[3]]
[1] "cherry"
```

List Length

To find out how many items a list has, use the length() function:

Example

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

length(thislist)
```

Output

```
[1] 3
```

Check if Item Exists

To find out if a specified item is present in a list, use the %in% operator:

Example

Check if "apple" is present in the list:

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

"apple" %in% thislist
```

Output

```
[1] TRUE
```

Add List Items

To add an item to the end of the list, use the append() function:

Example

Add "orange" to the list:

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

append(thislist, "orange")
```

Output

```
[[1]]
[1] "apple"

[[2]]
[1] "banana"

[[3]]
[1] "cherry"

[[4]]
[1] "orange"
```

To add an item to the right of a specified index, add "after=*index number*" in the append() function:

Example

Add "orange" to the list after "banana" (index 2):

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

append(thislist, "orange", after = 2)
```

Output

```
[[1]]
[1] "apple"

[[2]]
[1] "banana"

[[3]]
[1] "orange"

[[4]]
[1] "cherry"
```

Remove List Items

You can also remove list items. The following example creates a new, updated list without an "apple" item:

Example

Remove "apple" from the list:

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

newlist <- thislist[-1]

# Print the new list
newlist
```

Output

```
[[1]]
[1] "banana"

[[2]]
[1] "cherry"
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range, by using the : operator:

Example

Return the second, third, fourth and fifth item:

```
thislist <-
list("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")

(thislist)[2:5]
```

Output

```
[[1]]
[1] "banana"

[[2]]
[1] "cherry"

[[3]]
[1] "orange"

[[4]]
[1] "kiwi"
```

Note: The search will start at index 2 (included) and end at index 5 (included).

Remember that the first item has index 1.

Join Two Lists

There are several ways to join, or concatenate, two or more lists in R.

The most common way is to use the c() function, which combines two elements together:

Example

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

list3
```

Output

```
[[1]]
[1] "a"

[[2]]
[1] "b"

[[3]]
[1] "c"

[[4]]
[1] 1

[[5]]
[1] 2

[[6]]
[1] 3
```

Converting a List to Vector in R Language –

unlist() Function

unlist() function in R Language is used to convert a list to vector. It simplifies to produce a vector by preserving all components.

Syntax: unlist(list)
Parameters:
list: It is a list or Vector
use.name: Boolean value to prserve or not the position names

Example 1: Converting list numeric vector into a single vector

```
# R  program to illustrate

# converting list to vector

   # Creating a list.

my_list <- list(l1 = c(1, 3, 5, 7),

                l2 = c(1, 2, 3),

                l3 = c(1, 1, 10, 5, 8, 65, 90))

# Apply unlist R function
print(unlist(my_list))
```

Output:

```
l11 l12 l13 l14 l21 l22 l23 l31 l32 l33 l34 l35 l36 l37
  1   3   5   7   1   2   3   1   1  10   5   8  65  90
```

Here in the above code we have unlisted my_list using unlist() and convert it to a single vector.
As illustrated above, the list will dissolve and every element will be in the same line as shown above.