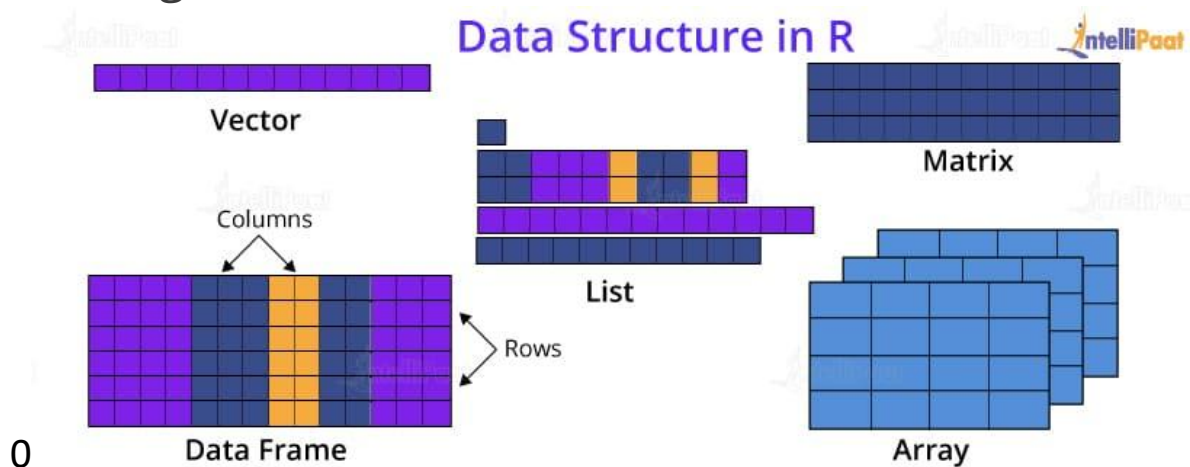


Data structures in R

Vector, matrix, array which are of a homogenous type and the other two are list and data frame which are heterogeneous.



Egs

```
fruits <- c("banana", "apple", "orange") //Vector of strings
```

```
thislist <- list("apple", "banana", 50, 100)
```

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

```
# An array with one dimension with values ranging from 1 to 24
```

```
thisarray <- c(1:24)
```

```
thisarray
```

```
# An array with more than one dimension
```

```
multiarray <- array(thisarray, dim = c(4, 3, 2))
```

```
multiarray
```

```
Data_Frame <- data.frame (
```

```
  Training = c("Strength", "Stamina", "Other"),
```

```
  Pulse = c(100, 150, 120),
```

```
Duration = c(60, 30, 45)  
)
```

Module 2.

Data Visualization using R: Reading and getting data into R (External Data): Using CSV files, XML files, Web Data, JSON files, Databases, Excel files.

Working with R Charts and Graphs: Histograms, Boxplots, Bar Charts, Line Graphs, Scatterplots, Pie Chart

Plot

Plot

The `plot()` function is used to draw points (markers) in a diagram.

Example

Draw one point in the diagram, at position (1) and position (3):

```
plot(1, 3)
```

Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

```
Eg plot(c(1, 8), c(3, 10))
```

```
Eg plot(c(1, 2, 3, 4, 5), c(3, 7, 8, 9, 12))
```

```
Eg x <- c(1, 2, 3, 4, 5)
```

```
y <- c(3, 7, 8, 9, 12)
```

```
plot(x, y)
```



























```
plot(1:10)
```

The `plot()` function also takes a `type` parameter with the value `l` to draw a line to connect all the points in the diagram:

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

```
plot(1:10, col="red")
```

```
plot(1:10, pch=25, cex=2)
```

0	1	2	3	4	
					
5	6	7	8	9	
					
10	11	12	13	14	
					
15	16	17	18	19	
					
20	21	22	23	24	25
					

Line plotting

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

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

```
plot(1:10, type="l", lwd=2)
```

To change the width of the line, use the **lwd** parameter (**1** is default, while **0.5** means 50% smaller, and **2** means 100% larger)

```
plot(1:10, type="l", lwd=5, lty=3)
```

The line is solid by default. Use the **lty** parameter with a value from **0 to 6** to specify the line format.

Multiple Lines

To display more than one line in a graph, use the **plot()** function together with the **lines()** function:

```
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")
```

Scatter Plots

A "scatter plot" is a type of plot used to display the relationship between two numerical variables, and plots one dot for each observation.

It needs two vectors of same length, one for the x-axis (horizontal) and one for the y-axis (vertical):

```
x <- c(5,7,8,7,2,2,9,4,11,12,9,6)
y <- c(99,86,87,88,111,103,87,94,78,77,85,86)
plot(x, y, main="Observation of Cars", xlab="Car
age", ylab="Car speed")
```

-
- Q1: Create a bar plot of the number of cylinders (cyl) in the mtcars dataset. Use different colors to represent the transmission type (am). Add appropriate title, labels, and legend to the plot.
 - Q2: Create a histogram of the miles per gallon (mpg) in the mtcars dataset. Use different shades of blue to represent the frequency of each bin. Add appropriate title and labels to the plot. Calculate and display the mean and standard deviation of mpg on the plot.
 - Q3: Create a box plot of the horsepower (hp) in the mtcars dataset. Use different shapes to represent the number of gears (gear). Add appropriate title, labels, and legend to the plot. Identify and label any outliers on the plot.
 - Q4: Create a scatter plot of the displacement (disp) versus the weight (wt) in the mtcars dataset. Use

different colors and sizes to represent the number of carburetors (carb). Add appropriate title, labels, and legend to the plot. Add a smooth line to show the trend of the relationship.

- Q5: Create a map of India using the maps package. Use different colors to represent the literacy rate of each state. Add appropriate title, labels, and legend to the map. Add a text annotation to show the name and literacy rate of Kerala.
-

- Q1: How can you create a scatterplot of the Sepal.Length and Petal.Length variables in the iris dataset using the plot function? Add appropriate labels and title to the plot.
- Q2: How can you create a scatterplot of the mpg and disp variables in the mtcars dataset using the ggplot2 package? Use different colors to represent the cyl variable and add a smooth line to show the trend. Add appropriate labels, title, and legend to the plot.

-

Pie Charts

A pie chart is a circular graphical view of data.

Use the `pie()` function to draw pie charts:

```
x <- c(10,20,30,40)
```

eg 1

```
# Display the pie chart
```

```
pie(x)
```

eg 2

```
# Create a vector of pies
```

```
x <- c(10,20,30,40)
```

```
# Display the pie chart and start the first pie at  
90 degrees
```

```
pie(x, init.angle = 90)
```

Eg 3

```
# Create a vector of pies
```

```
x <- c(10,20,30,40)
```

```
# Create a vector of labels
```

```
mylabel <-
```

```
c("Apples", "Bananas", "Cherries", "Dates")
```

```
# Display the pie chart with labels
```

```
pie(x, label = mylabel, main = "Fruits")
```

eg 4

```
# Create a vector of colors
```

```
colors <- c("blue", "yellow", "green", "black")
```

```
# Display the pie chart with colors
```

```
pie(x, label = mylabel, main = "Fruits", col =  
colors)
```

Eg 5

```
# Create a vector of colors
colors <- c("blue", "yellow", "green", "black")

# Display the pie chart with colors
pie(x, label = mylabel, main = "Fruits", col =
colors)

legend("bottomright", mylabel, fill = colors)
```

Bar Charts

A bar chart uses rectangular bars to visualize data. Bar charts can be displayed horizontally or vertically. The height or length of the bars are proportional to the values they represent.

Use the `barplot()` function to draw a vertical bar chart:

```
# x-axis values
x <- c("A", "B", "C", "D")
# y-axis values
y <- c(2, 4, 6, 8)
barplot(y, names.arg = x)
```

Eg 2

Use the `col` parameter to change the color of the bars:

```
x <- c("A", "B", "C", "D")
y <- c(2, 4, 6, 8)
barplot(y, names.arg = x, col = "red")
```

eg 3

Horizontal Bars

If you want the bars to be displayed horizontally instead of vertically, use `horiz=TRUE`:

Example

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

```
barplot(y, names.arg = x, horiz = TRUE)
```

Histogram

R creates histogram using **hist()** function. This function takes a vector as an input and uses some more parameters to plot histograms

The basic syntax for creating a histogram using R is –

```
hist(v,main,xlab,xlim,ylim,breaks,col,border)
```

- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.

- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

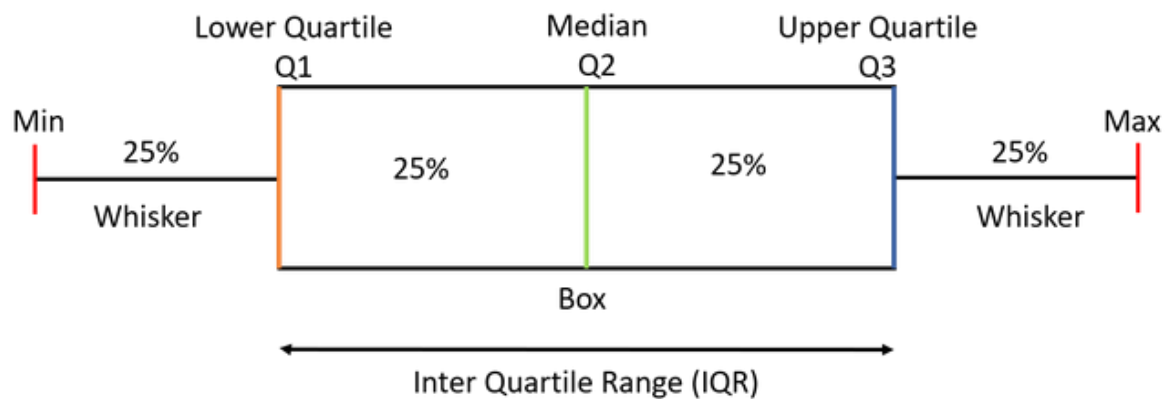
```
# Create data for the graph.  
v <- c(9,13,21,8,36,22,12,41,31,33,19)  
  
# Create the histogram.  
hist(v,xlab = "Weight",col =  
"yellow",border = "blue")
```

```
# Create data for the graph.  
v <- c(9,13,21,8,36,22,12,41,31,33,19)  
  
# Create the histogram.  
hist(v,xlab = "Weight",col =  
"green",border = "red", xlim = c(0,40),  
ylim = c(0,5),  
breaks = 5)
```

Boxplot

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

Boxplots are created in R by using the **boxplot()** function.



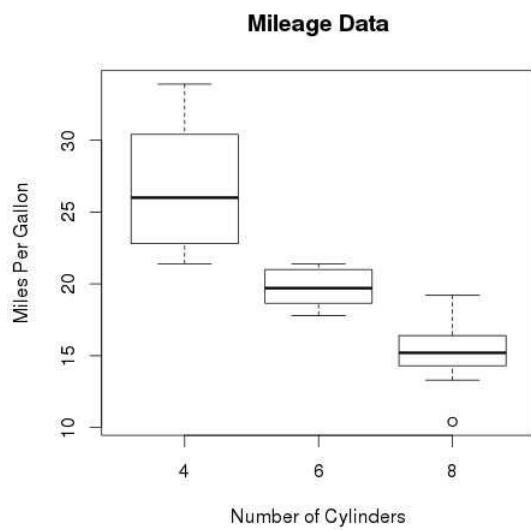
Syntax

```
boxplot(x, data, notch, varwidth, names, main)
```

Following is the description of the parameters used –

- **x** is a vector or a formula.
- **data** is the data frame.
- **notch** is a logical value. Set as TRUE to draw a notch.
- **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
- **names** are the group labels which will be printed under each boxplot.
- **main** is used to give a title to the graph.

```
input <- mtcars[,c('mpg','cyl')]
boxplot(mpg ~ cyl, data = mtcars, xlab =
"Number of Cylinders",
      ylab = "Miles Per Gallon", main =
"Mileage Data")
```



Exclude observations with missing data

Many analyses use what is known as a **complete case analysis** in which you filter the dataset to only include observations with no missing values on any variable in your analysis. In base R, use `na.omit()` to remove all observations with missing data on ANY variable in the dataset, or use `subset()` to filter out cases that are missing on a subset of variables. An alternative to `na.omit()` is `na.exclude()`

R – Handling Missing Values

Missing values are practical in life. For example, some cells in spreadsheets are empty. If an insensible or impossible arithmetic operation is tried then NAs occur.

Dealing Missing Values in R

Missing Values in R, are handled with the use of some pre-defined functions:

is.na() Function for Finding Missing values:

A logical vector is returned by this function that indicates all the NA values present. It returns a Boolean value. If NA is present in a vector it returns TRUE else FALSE.

```
x<- c(NA, 3, 4, NA, NA, NA)
is.na(x)
```

Output:

```
[1] TRUE FALSE FALSE TRUE TRUE TRUE
```

How to Impute Missing Values in R?

Replacing these missing values with another value is known as Data Imputation. There are several ways of imputation. Common ones include replacing with average, minimum, or maximum value in that column/feature. Different datasets and features will require one type of imputation method. For example, considering a dataset of sales performance of a company, if the feature loss has missing values then it would be more logical to replace a minimum value.

```
> data
  marks1 marks2 marks3
1      NA     81 78.500
2      22     14 19.325
3      NA     NA      NA
4      49     61 28.000
5      75     12 48.002
```

```
> summary(data)
      marks1      marks2      marks3
Min.   :22.00  Min.   :12.0  Min.   :19.32
1st Qu.:35.50  1st Qu.:13.5  1st Qu.:25.83
Median :49.00  Median :37.5  Median :38.00
Mean   :48.67  Mean   :42.0  Mean   :43.46
3rd Qu.:62.00  3rd Qu.:66.0  3rd Qu.:55.63
Max.   :75.00  Max.   :81.0  Max.   :78.50
NA's   :2      NA's   :1    NA's   :1
```

Impute One Column

Method 1: Imputing manually with Mean value

Let's impute the missing values of one column of data, i.e marks1 with the **mean** value of this entire column.

Syntax :

`mean(x, trim = 0, na.rm = FALSE, ...)`

Parameter:

- *x* – any object
- *trim* – observations to be trimmed from each end of *x* before the mean is computed
- *na.rm* – FALSE to remove NA values

create a dataframe

```
data <- data.frame(marks1 = c(NA, 22, NA, 49, 75),  
                  marks2 = c(81, 14, NA, 61, 12),  
                  marks3 = c(78.5, 19.325, NA, 28, 48.002))
```

impute manually

```
data$marks1[is.na(data$marks1)] <- mean(data$marks1, na.rm = T)
```

data

```
> data  
  marks1 marks2 marks3  
1 48.66667    81 78.500  
2 22.00000    14 19.325  
3 48.66667    NA    NA  
4 49.00000    61 28.000  
5 75.00000    12 48.002
```

Method 2: Using Hmisc Library and imputing with Median value

Using the function `impute()` inside Hmisc library let's impute the column marks2 of data with the **median** value of this entire column.

Example: Impute missing values

```
# install and load the required packages
```

```
install.packages("Hmisc")
```

```
library(Hmisc)
```

```
# create a dataframe
```

```
data <- data.frame(marks1 = c(NA, 22, NA, 49, 75),  
                  marks2 = c(81, 14, NA, 61, 12),  
                  marks3 = c(78.5, 19.325, NA, 28,  
                             48.002))
```

```
# fill missing values of marks2 with median
```

```
impute(data$marks2, median)
```

Understanding Missing Values in Time Series Data

In general Time Series data is a type of data where observations are collected over some time at successive intervals. Time series are used in various fields such as finance, engineering, and biological sciences, etc,

- **Missing values** will disrupt the order of the data which indirectly results in the inaccurate representation of trends and patterns over some time
- By Imputing missing values we can ensure the statistical analysis done on the Time Serial data is reliable based on the patterns we observed.

- Similar to other models handling missing values in the time series data improves the model performance.

In R Programming there are various ways to handle missing values of Time Series Data using functions that are present under the ZOO package.

Step 1: Load Necessary Libraries and Dataset

```
# Load necessary libraries
library(zoo)
library(ggplot2)

# Generate sample time series data with missing values
set.seed(789)
dates <- seq(as.Date("2022-01-01"), as.Date("2022-01-31"), by = "days")
time_series_data <- zoo(sample(c(50:100, NA), length(dates), replace = TRUE),
                        order.by = dates)
head(time_series_data)
```

Effects of an outlier on the model:

- The format of the data appears to be skewed.

- Modifies the mean, variance, and other statistical characteristics of the data's overall distribution.
- Leads to the model's accuracy level being biased.

Steps involving Outlier detection:

Step 1: In this step, we will be, by default creating the data containing the outlier inside it using the `rnorm()` function and generating 500 different data points. Further, we will be adding 10 random outliers to this data.

```
data <- rnorm(500)
data[1:10] <- c(46,9,15,-90,
               42,50,-82,74,61,-32)
```

Step 2: In this step, we will be analyzing the outlier in the provided data using the `boxplot`, which will be plotting a barplot, and we will be able to analyze the outlier in the data. As said when reviewing a box plot, an outlier is defined as a data point that is located outside the whiskers of the box plot.

boxplot() function:

```
data <- rnorm(500)
data[1:10] <- c(46,9,15,-90,
               42,50,-82,74,61,-32)
boxplot(data)
```