In data visualization, it's often necessary to change the range of axes (rescaling) or magnify a specific portion of a curve (zooming) to focus on particular details or make the plot more informative. You can achieve this in R using various packages like `ggplot2` or base R plotting functions. Here's how to change axis ranges and magnify a portion of a curve in R:

Changing the Range of Axes (Rescaling):

To change the range of axes in R, you typically manipulate the limits of the x and y-axes in a plot. This can be done using the `xlim()` and `ylim()` functions in base R graphics or the `coord_cartesian()` function in `ggplot2`.

Using base R graphics:

```R
# Sample data
x <- 1:100
y <- x^2

# Create a basic plot
plot(x, y, type = "l", xlab = "X-axis", ylab = "Y-axis")

# Change the x-axis and y-axis limits
xlim(c(20, 80))  # Adjust the x-axis limits
ylim(c(0, 5000)) # Adjust the y-axis limits
```

In the above example, we used the `xlim()` and `ylim()` functions to rescale the x and y-axes to focus on a specific range of data.

Using ggplot2:

```R
library(ggplot2)

# Sample data
data <- data.frame(x = 1:100, y = (1:100)^2)

# Create a ggplot
p <- ggplot(data, aes(x, y)) +
  geom_line() +
  labs(x = "X-axis", y = "Y-axis")

# Change the x-axis and y-axis limits
p + coord_cartesian(xlim = c(20, 80), ylim = c(0, 5000))
```

In this ggplot2 example, we used the `coord_cartesian()` function to change the axis limits, which effectively rescales the plot.

Magnifying a Portion of the Curve (Zooming):

To magnify a portion of a curve, you can create multiple plots, each showing a different subset of the data. This is often useful for highlighting specific regions of the curve.

Using base R graphics:

```R
# Sample data
x <- 1:100
y <- x^2

# Create two plots: one showing the entire curve and the other focusing on a portion
par(mfrow = c(2, 1))  # Create a 2x1 layout
plot(x, y, type = "l", xlab = "X-axis", ylab = "Y-axis", main = "Entire Curve")

# Zoom in on a portion of the curve
plot(x, y, type = "l", xlim = c(20, 80), ylim = c(0, 5000),
     xlab = "X-axis", ylab = "Y-axis", main = "Zoomed In")
```

In this base R example, we created two separate plots to show the entire curve and a magnified portion.

Using ggplot2:

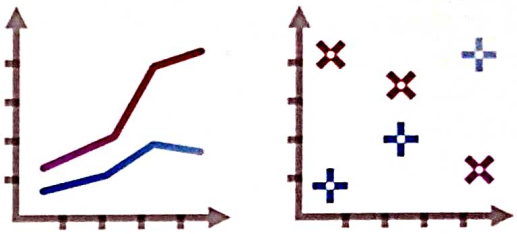
```R
library(ggplot2)

# Sample data
data <- data.frame(x = 1:100, y = (1:100)^2)

# Create a ggplot showing the entire curve
p1 <- ggplot(data, aes(x, y)) +
  geom_line() +
  labs(x = "X-axis", y = "Y-axis")

# Create another ggplot focusing on a portion of the curve
p2 <- ggplot(data, aes(x, y)) +
  geom_line() +
  xlim(20, 80) +
  ylim(0, 5000) +
  labs(x = "X-axis", y = "Y-axis")
```

```
# Arrange and display the plots side by side
library(gridExtra)
grid.arrange(p1, p2, ncol = 2)
```

In the ggplot2 example, we created two separate ggplot objects, one for the entire curve and the other for the magnified portion, and then used `grid.arrange()` from the `gridExtra` package to display them side by side.

These techniques allow you to rescale axes and magnify specific portions of curves to better emphasize the details you want to highlight in your data visualizations.

2 The Workhorse of R Base Graphics: The plot() Function



The plot() function forms the foundation for much of R's base graphing operations, serving as the vehicle for producing many different kinds of graphs. plot() is a generic function, or a placeholder for a family of functions. The function that is actually called depends on the class of the object on which it is called. The basic syntax to create a line chart in R is –

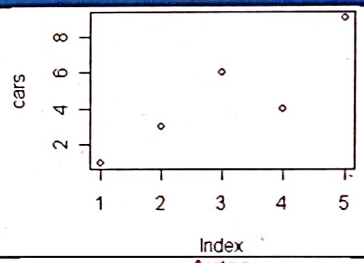
plot(v, type, col, xlab, ylab)

Following is the description of the parameters used –

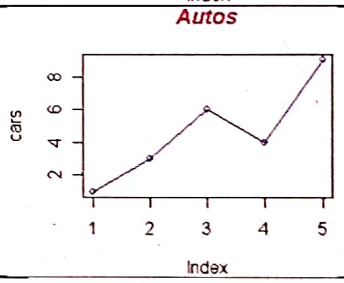
- ✓ **v** is a vector containing the numeric values.
- ✓ **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
- ✓ **xlab** is the label for x axis.
- ✓ **ylab** is the label for y axis.
- ✓ **main** is the Title of the chart.
- ✓ **col** is used to give colors to both the points and lines.

Examples of plot function

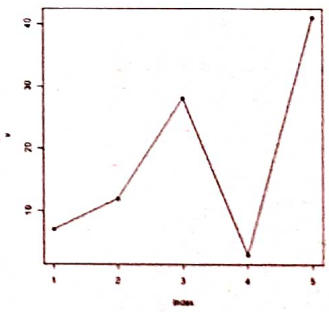
```
# Define the cars vector with 5 values
  cars <- c(1, 3, 6, 4, 9)
# Graph the cars vector with all defaults
   plot(cars)
 The default argument of type is points
```




```
# Define the cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)
# Graph cars using blue points with lines
plot(cars, type="o", col="blue")
# Create a title with a red, bold/italic font
title(main="Autos",col.main="red", font.main=4)
```




```
# Create the data for the chart.
v <- c(7,12,28,3,41)
# Give the chart file a name.
png(file = "line_chart.jpg")
# Plot the bar chart.
plot(v,type = "o")
# Save the file.
dev.off()
```



abline() function in R Language is used to add one or more straight lines to a graph. The abline() function can be used to add vertical, horizontal or regression lines to plot.

Syntax:
abline(a=NULL, b=NULL, h=NULL, v=NULL, …)

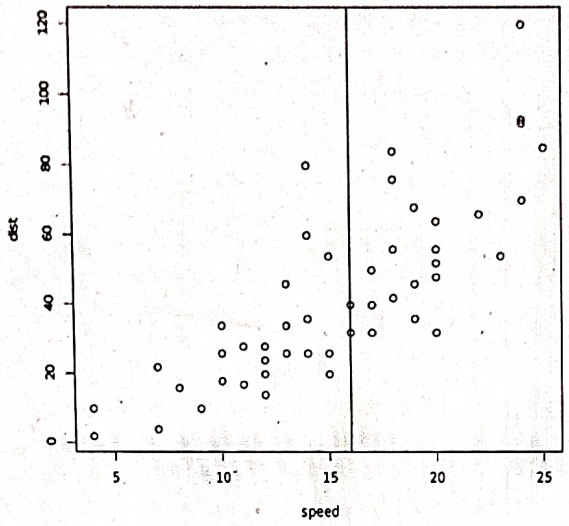
Parameters:
a, b: It specifies the intercept and the slope of the line
h: specifies y-value for horizontal line(s)
v: specifies x-value(s) for vertical line(s)

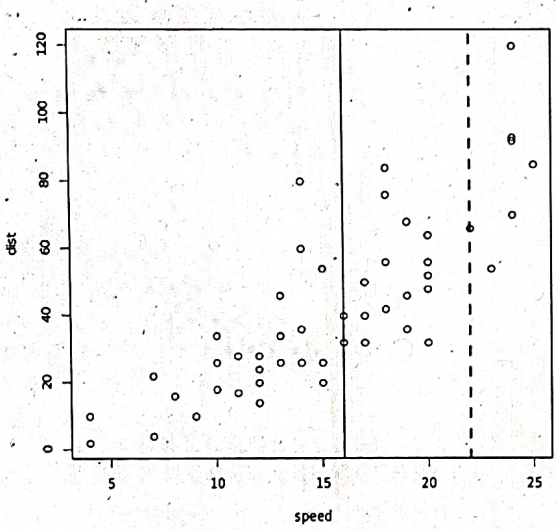
Returns: a straight line in the plot

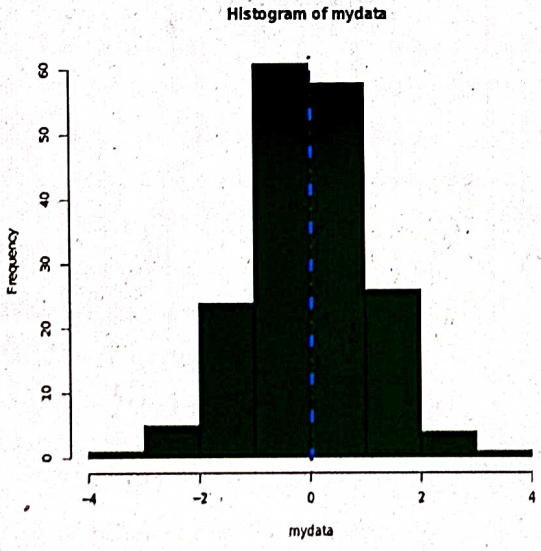
Example 1: To add a vertical line to the plot

```
# add line to square plot
# first example : Add one line
plot(cars)
abline(v = 16, col = "darkgreen")

# second example : add 2 lines
# addline to square plot
# change line colors, sizes and types
plot(cars)
abline(v = c(16, 22), col = c("darkgreen", "blue"),
                      lty = c(1, 2), lwd = c(1, 3))

# third example
set.seed(1200); mydata<-rnorm(180)
hist(mydata, col="darkgreen")

# lwd=line width, lty =linetype
abline(v = mean(mydata), col = "blue", lwd = 4, lty = 4)
```

Output:


dist
speed
0
20
40
60
80
100
120
5
10
15
20
25


dist
speed
0
20
40
60
80
100
120
5
10
15
20
25


Histogram of mydata
Frequency
mydata
0
10
20
30
40
50
60
-4
-2
0
2
4

Displaying two density estimates on the same graph is a common visualization technique used in statistics to compare the distribution of two different datasets or to illustrate the change in a distribution before and after some treatment or transformation. This can be done using various techniques and plotting functions in R. Below, I'll explain how to achieve this with a simple example.

Let's assume you have two datasets, A and B, and you want to create density plots for each of them on the same graph for comparison.

2b

Example in R:

Suppose you have two datasets, A and B, and you want to compare their density estimates on the same graph. You can use the `density()` function to compute density estimates for each dataset and then plot them together using the `plot()` function or the `ggplot2` package. Here's how you can do it in R:

```R
# Sample data for datasets A and B
set.seed(42)
data_A <- rnorm(1000, mean = 0, sd = 1)
data_B <- rnorm(1000, mean = 2, sd = 1)

# Compute density estimates for A and B
density_A <- density(data_A)
density_B <- density(data_B)

# Create a plot with both density estimates
plot(density_A, main = "Density Comparison", xlab = "Value", ylab = "Density", col = "blue", lty = 1)
lines(density_B, col = "red", lty = 2)
legend("topright", legend = c("Dataset A", "Dataset B"), col = c("blue", "red"), lty = c(1, 2))
```

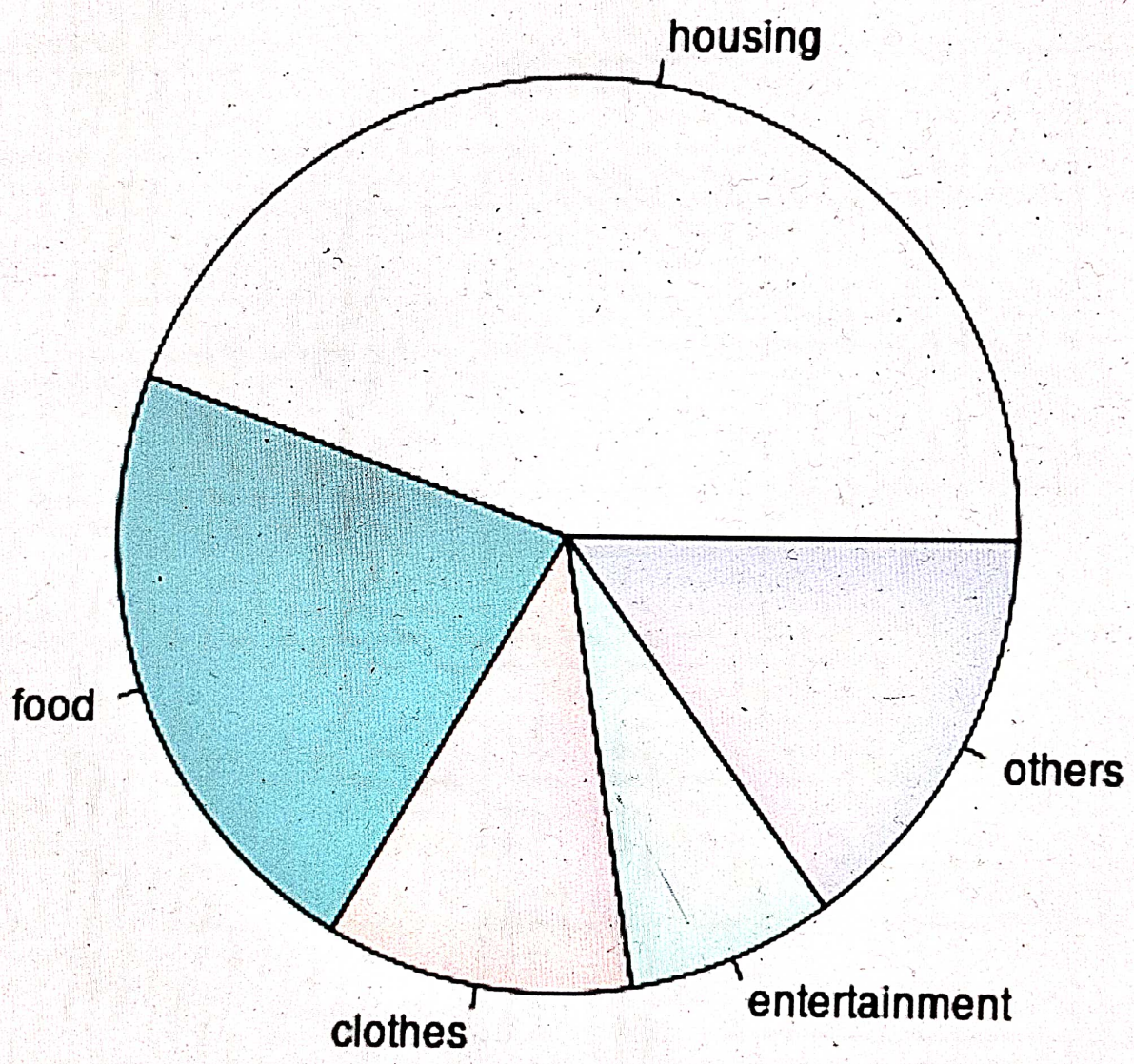
In this example:
- We generate two datasets, A and B, with different means.
- We use the `density()` function to compute density estimates for each dataset.
- We create a plot using `plot()` for the first density estimate (blue) and then add the second density estimate (red) using `lines()`.
- We add a legend to distinguish between the two datasets.

This results in a graph that displays the density estimates of both datasets on the same plot for easy visual comparison.

By comparing the density estimates of two datasets on the same graph, you can quickly identify differences in their distributions, assess the impact of changes or treatments, and make informed statistical inferences.

3a

```
# Create data for the graph.
x <- c(600,300,150,100,200)
labels <- c("housing","food","clothes","entertainment","others")

# Give the chart file a name.
png(file = "city.png")

# Plot the chart.
pie(x,labels)

# Save the file.
dev.off()
```


housing
food
others
clothes
entertainment

You can create a plot of the function \(f(x) = \sin(x)\) in the interval (-3, 3) with a step size of 0.1, and use a triangle as the point character connected by lines. Here's an R program to achieve this:

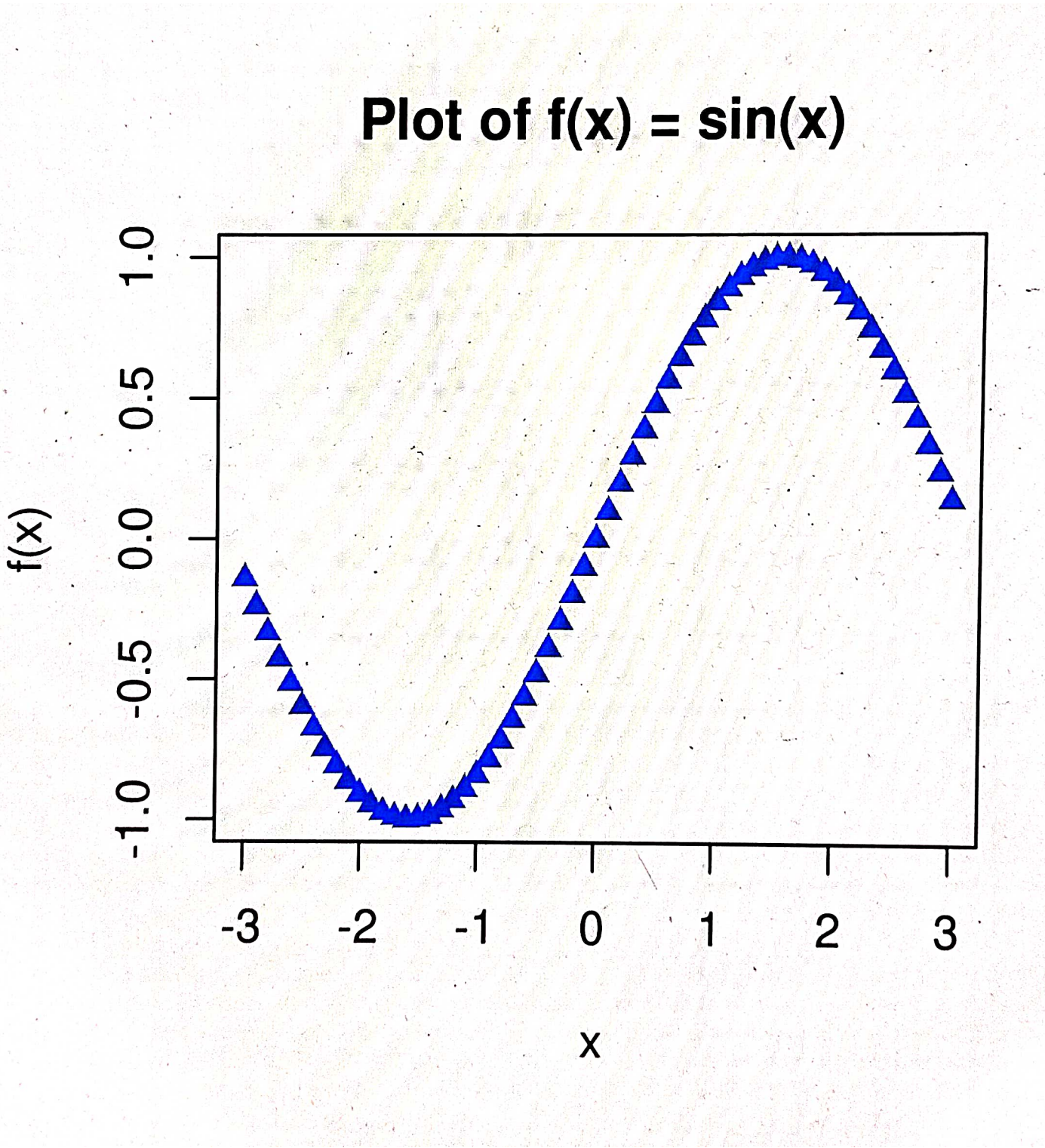
3b

```R
# Define the x values from -3 to 3 with a step size of 0.1
x <- seq(-3, 3, by = 0.1)

# Calculate the corresponding y values using the sine function
y <- sin(x)

# Create a plot with triangles as point characters and lines connecting them
plot(x, y, type = "b", pch = 17, col = "blue", xlab = "x", ylab = "f(x)", main = "Plot of f(x) = sin(x)")
```

In this program:

- We first define the x values using the `seq()` function to create a sequence of values from -3 to 3 with a step size of 0.1.
- We calculate the corresponding y values by applying the sine function `sin(x)` to the x values.
- We create the plot using the `plot()` function with the following options:
  - `type = "b"` specifies that we want both points and lines in the plot.
  - `pch = 17` sets the point character to a triangle.
  - `col = "blue"` sets the color of the points and lines to blue.
  - `xlab` and `ylab` set the labels for the x-axis and y-axis, respectively.
  - `main` sets the title of the plot.

This code will generate a plot of the function \(f(x) = \sin(x)\) in the specified interval with triangles as point characters connected by lines.


Plot of f(x) = sin(x)
f(x)
1.0
0.5
0.0
-0.5
-1.0
-3
-2
-1
0
1
2
3
x

4a

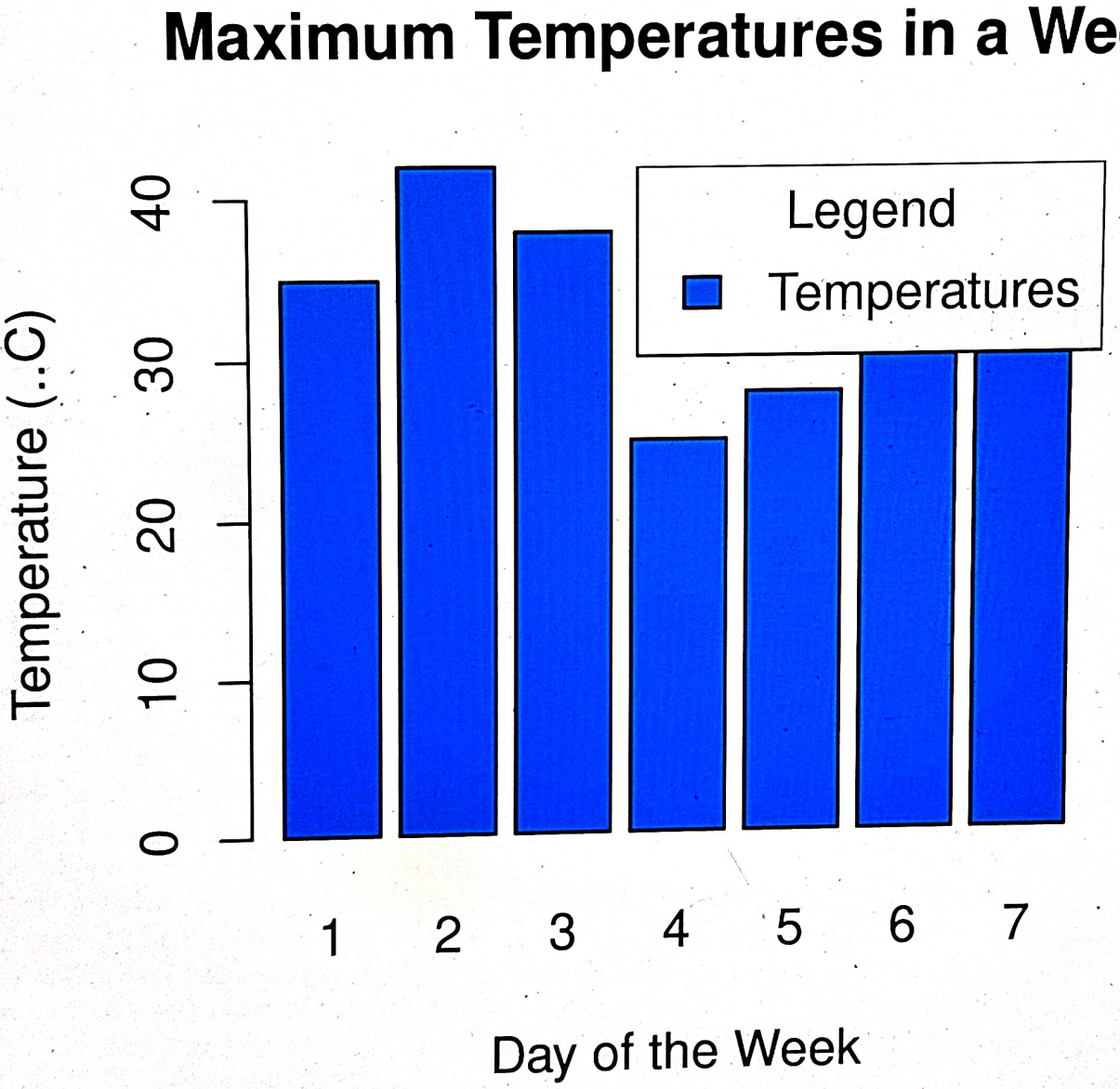
You can create a bar plot for the maximum temperatures in Celsius for a week and use the `legend()` function to add descriptions to the bars. Here's an R program to do that:

```R
# Maximum temperatures in Celsius for a week
temperatures <- c(35, 42, 38, 25, 28, 36, 40)

# Create a bar plot
barplot(temperatures, names.arg = 1:7, col = "blue", main = "Maximum Temperatures in a Week",
        xlab = "Day of the Week", ylab = "Temperature (°C)")

# Add a legend to describe the bars
legend("topright", legend = "Temperatures", fill = "blue", title = "Legend")
```

In this program:

- We define the maximum temperatures for a week in the `temperatures` vector.
- We use the `barplot()` function to create a bar plot of the temperatures. The `names.arg` argument is set to 1:7 to label the bars with the days of the week (assuming the temperatures are ordered by day).
- We set the color of the bars to blue, add a title, and label the axes.
- We use the `legend()` function to add a legend to the plot. The "topright" location is specified for the legend, and it describes the "Temperatures" with a blue fill. The `title` argument is used to provide a title for the legend.

Running this code will produce a bar plot of the maximum temperatures for a week, with a legend describing the meaning of the bars.


Maximum Temperatures in a Week
Legend
Temperatures
Temperature (..C)
0
10
20
30
40
1
2
3
4
5
6
7
Day of the Week

4b

Stacked Bar Plot and **Bar Plot** are both used for visualizing data, but they represent data in different ways and serve different purposes. Here are the key differences between the two:

1. Data Representation:

- **Stacked Bar Plot:** In a stacked bar plot, multiple data series are stacked on top of each other, with each segment representing a category within the data. The height of each segment represents a value, and the total height of each stack represents the total for a given category.

- **Bar Plot:** In a bar plot (or clustered bar plot), multiple bars are placed side by side, where each bar represents a separate category or group. The length or height of each bar represents a value, and there is no stacking of data.

2. Use Cases:

- **Stacked Bar Plot:** Stacked bar plots are used when you want to show the composition or distribution of a whole into various categories. They are useful for visualizing parts-to-whole relationships and are commonly used for displaying proportions.

- **Bar Plot:** Bar plots are typically used to compare values or frequencies of different categories or groups side by side. They are useful for showing the magnitude of different variables within or across categories.

3. Interpretation:

- **Stacked Bar Plot:** In a stacked bar plot, it is easy to see the relationship between each category's contribution to the whole. However, it can be challenging to compare the absolute values of different categories across different stacks.

- **Bar Plot:** Bar plots make it straightforward to compare the values of different categories as they are not stacked on top of each other. You can easily compare the heights or lengths of bars to understand the differences in magnitude.

4. Overlapping Data:

- **Stacked Bar Plot:** Stacked bar plots can handle data where categories overlap (e.g., in a survey where respondents can choose multiple categories), but they may not be the best choice for precise comparisons.

- **Bar Plot:** Bar plots are suitable for displaying non-overlapping data, making them a better choice when you need to compare values directly.

5a

Plotting multiple curves in the same graph in R can be done using the `plot()` function multiple times, or by using functions like `lines()` or `points()` in combination with the `plot()` function. Here's an example of how to plot multiple curves in the same graph:

```R
# Create sample data
x <- seq(0, 2 * pi, length.out = 100)  # x values
y1 <- sin(x)                           # First curve (sine wave)
y2 <- cos(x)                           # Second curve (cosine wave)

# Create the initial plot for the first curve
plot(x, y1, type = "l", col = "blue", lwd = 2, xlab = "x", ylab = "y", main = "Multiple Curves")

# Add the second curve to the same graph using lines()
lines(x, y2, col = "red", lwd = 2)

# Add a legend to describe the curves
legend("topright", legend = c("Sine Wave", "Cosine Wave"), col = c("blue", "red"), lwd = 2)
```

In this example:

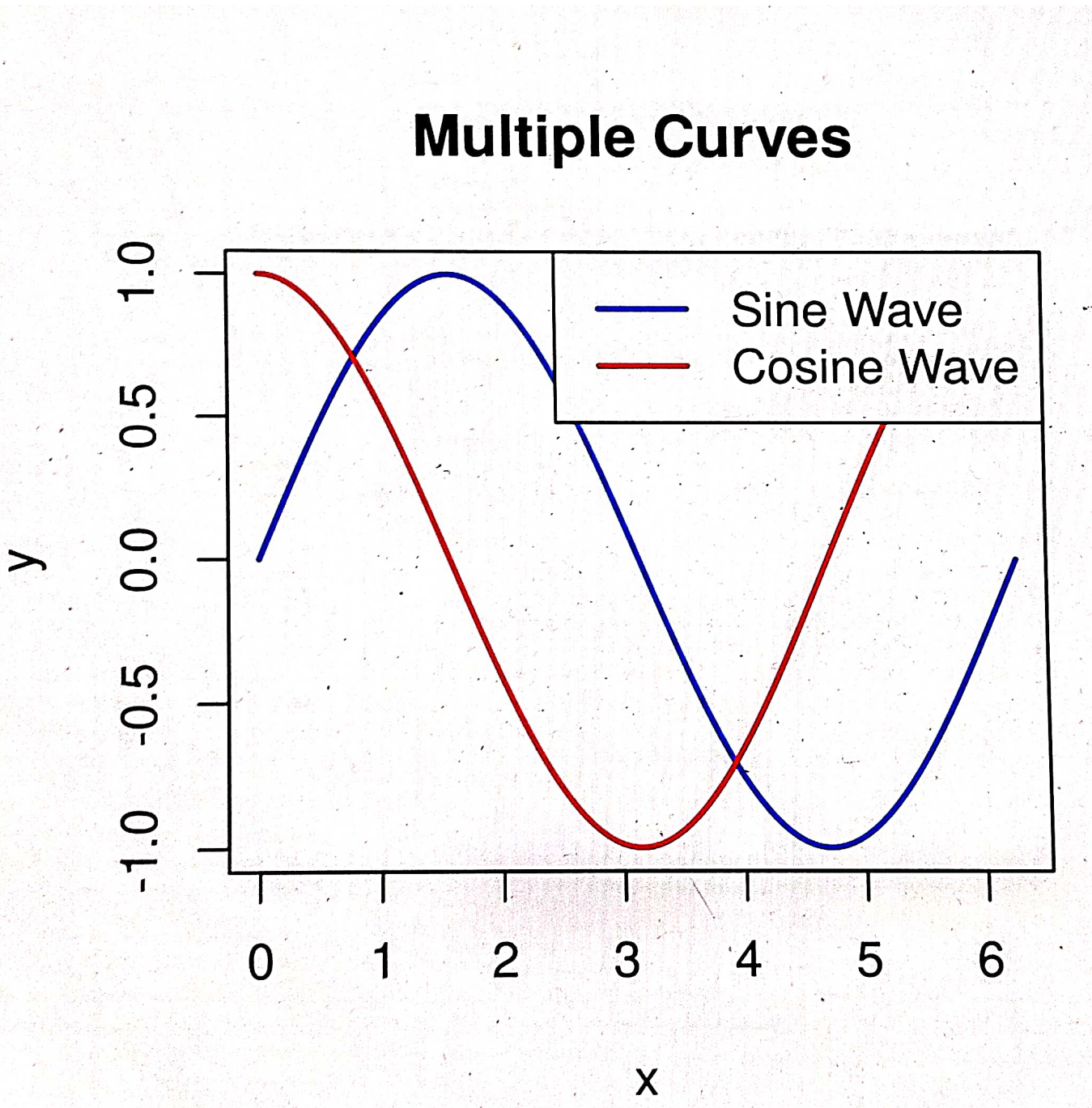
- We create sample data for the x-values, which range from 0 to \(2\pi\), and calculate the corresponding y-values for two curves, a sine wave (\(y_1\)) and a cosine wave (\(y_2\)).
- We create the initial plot using `plot()`, specifying the type as "l" for a line plot. We customize the color, line width, and labels for the x-axis and y-axis, and set a title for the plot.
- We add the second curve to the same graph using the `lines()` function. This adds the cosine wave to the existing plot.
- Finally, we add a legend using the `legend()` function to describe the two curves in the plot.

This code results in a graph that displays both the sine wave and cosine wave in the same plot with different colors and a legend to distinguish between them. You can follow a similar approach to add more curves to the same graph, making it easy to compare multiple data series in a single visualization.

5. Multiple Variables:

- **Stacked Bar Plot:** Stacked bar plots can be used to show multiple variables within a single stack, allowing you to see the composition of different categories based on multiple attributes.
- **Bar Plot:** Bar plots can be used to display multiple variables in separate bars, making it easier to compare values across different groups.

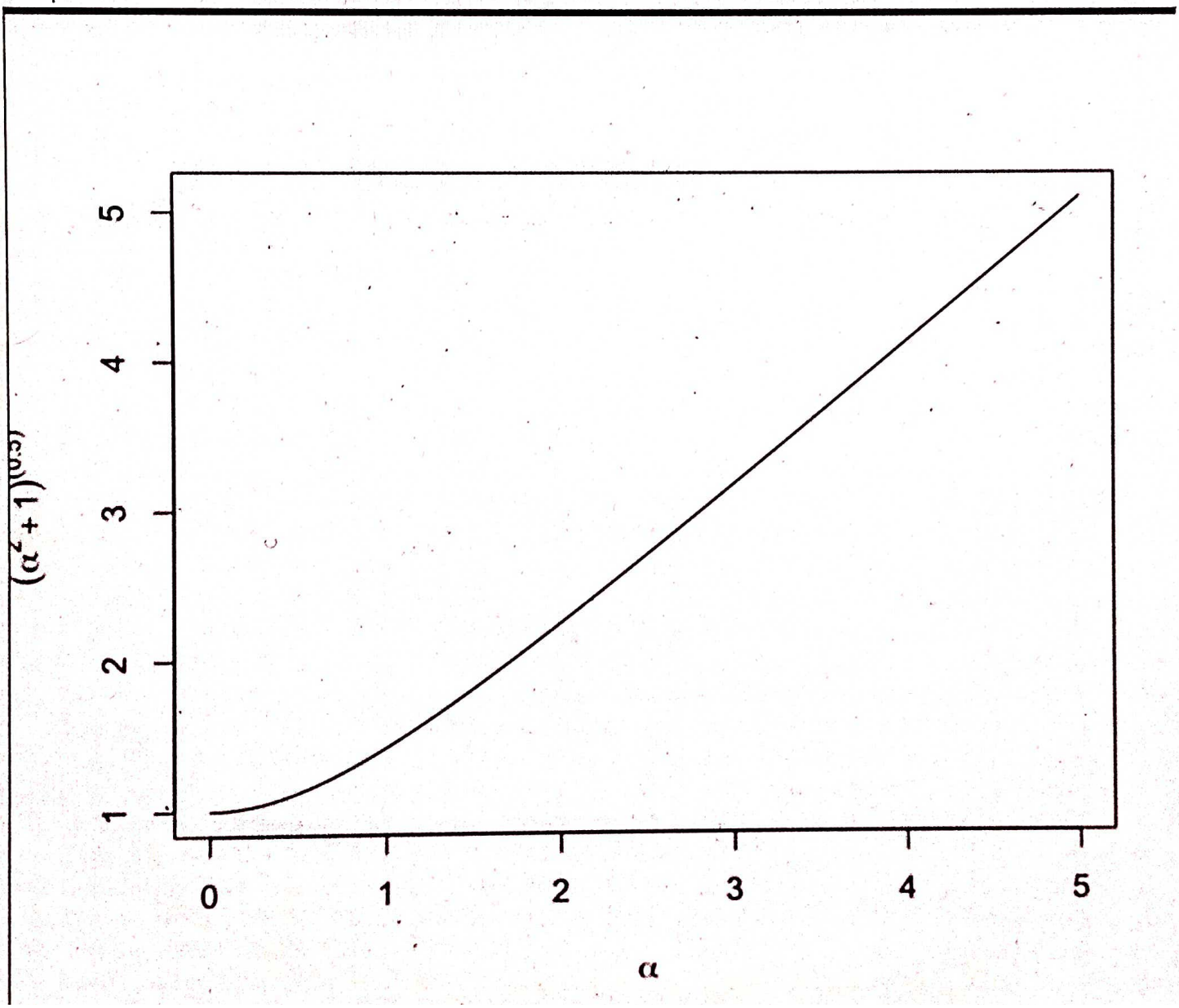
In summary, the choice between a stacked bar plot and a bar plot depends on the type of data you want to visualize and the specific objectives of your analysis. Stacked bar plots are suitable for showing parts-to-whole relationships and proportions, while bar plots are better for comparing values across different categories or groups.


Multiple Curves
Sine Wave
Cosine Wave
y
x
1.0
0.5
0.0
-0.5
-1.0
0
1
2
3
4
5
6

5b

```
curve((x^2 + 1)^(0.5), from=0, to=5,
      xlab = expression(alpha),
      ylab = expression((alpha^2 + 1)^(0.5), # ((alpha
      main = expression(paste("Function : ",
                              f(alpha) == (alpha^2 + 1)^(0.5))))) # ))
```

Output:



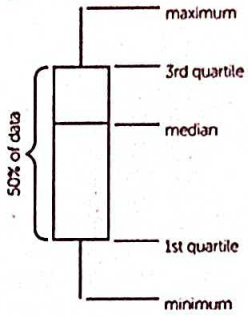
Box plot:- Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them. Boxplots are created in R by using the **boxplot()** function.

6a

Syntax:- **boxplot(x, data, notch, varwidth, names, main)**

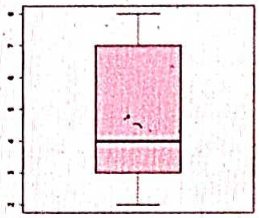
Following is the description of the parameters used –

- **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.

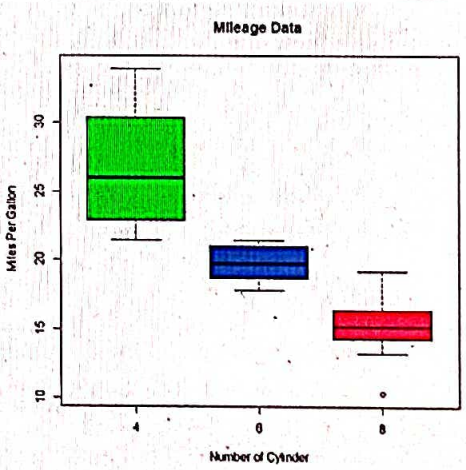


Examples

```
x <- c(7,3,2,4,8)
boxplot(x,col="pink")
```

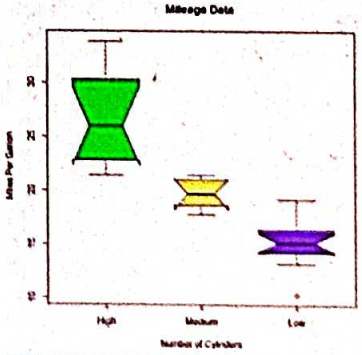



```
> input <- mtcars[,c('mpg','cyl')]
> print(head(input))
                  mpg cyl
Mazda RX4        21.0   6
Mazda RX4 Wag    21.0   6
Datsun 710       22.8   4
Hornet 4 Drive   21.4   6
Hornet Sportabout 18.7  8
Valiant          18.1   6

boxplot(mpg ~ cyl, data = mtcars,
        xlab ="Number of Cylinder",
        ylab = "Miles Per Gallon",
        main = "Mileage Data",
       col= c("green","royalblue","red"))
```



We can draw boxplot with notch to find out how the medians of different data groups match with each other.

```
boxplot(mpg ~ cyl, data = mtcars,
    xlab = "Number of Cylinders",
    ylab = "Miles Per Gallon",
    main = "Mileage Data",
    notch = TRUE,
    varwidth = TRUE,
    col = c("green","yellow","purple"),
    names = c("High","Medium","Low"))
```

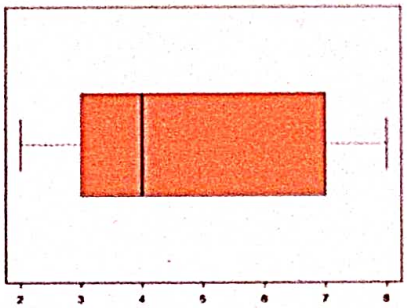


a~z , where the value of z determines the value of a.

a~z , where the value of z determines the value of a.

We can draw boxplot horizontally by making horizontal as TRUE, the default value is FALSE.

```
x <- c(7,3,2,4,8)
boxplot(x,col="orange",horizontal = TRUE,border = "blue")
> b
$stats
     [,1]
[1,]    2
[2,]    3
[3,]    4
[4,]    7
[5,]    8

$n
[1] 5

$conf
         [,1]
[1,] 1.17361
[2,] 6.82639

$out
numeric(0)
$group
numeric(0)

$names
[1] "1"
```



- ***n:*** *It includes number of used to draw box-plot exculding the NA's.*
- ***conf:*** *It represents the lower and upper extremes of notch and the out-value of outliers.*
- ***group:*** *It represents same length vector as out whose elements indicate to which group the outlier belongs and*
- ***names:*** *names vector for a group*

Advantages:

- A box plot is a good way to summarize large amounts of data.
- It displays the range and distribution of data along a number line.

- Box plots provide some indication of the data's symmetry and skew-ness.
- Box plots show outliers.

Disadvantages

- Original data is not clearly shown, in the box plot; also, mean and mode cannot be identified in a box plot.
- Exact values not retained.

6b

A scatter plot is a type of data visualization used to display individual data points or observations as dots on a two-dimensional plane. Scatter plots are useful for understanding relationships between two continuous variables. They allow you to examine patterns, trends, correlations, and outliers in your data. Here's how to create a scatter plot in R with an example:

Example:

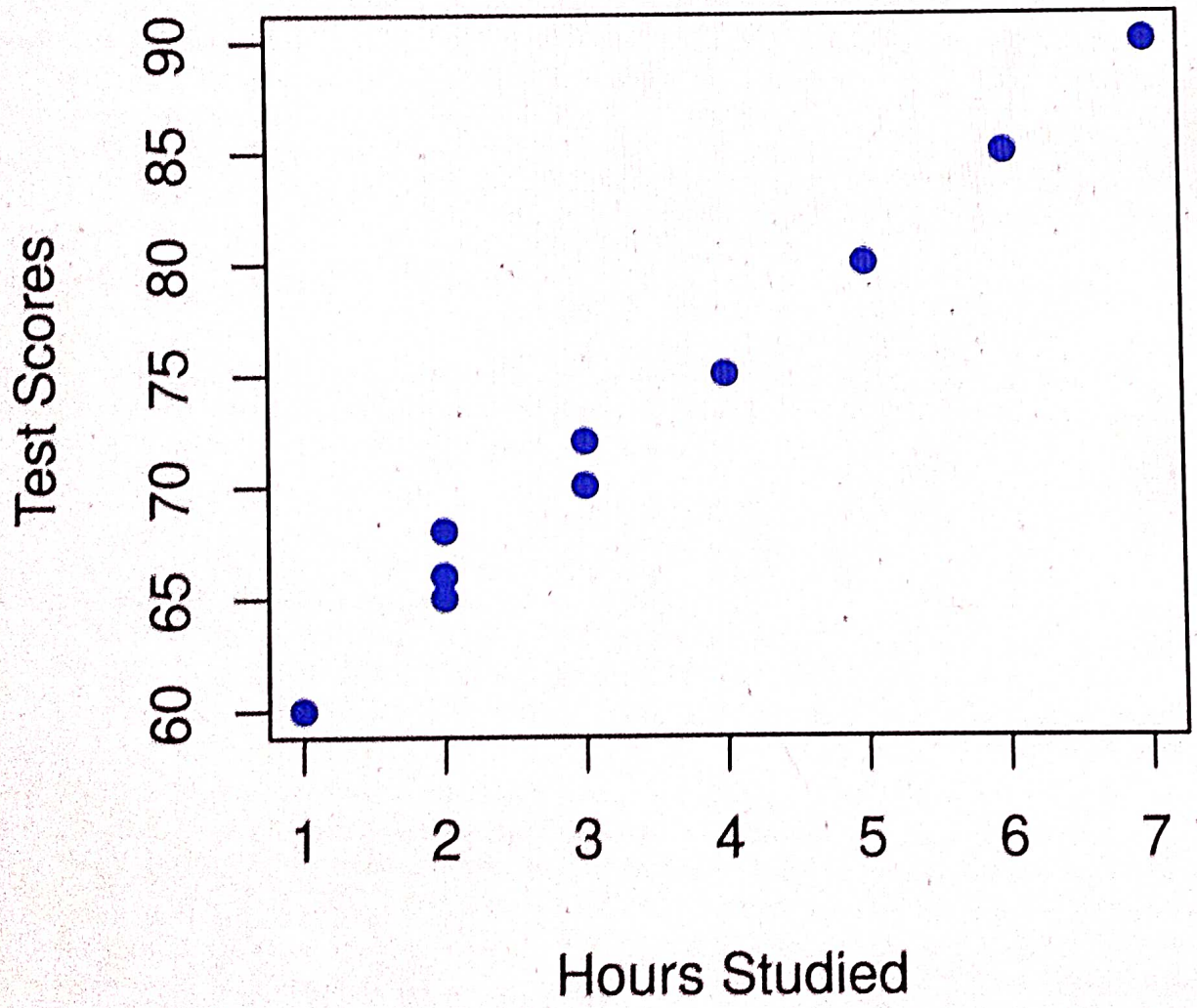
Let's create a scatter plot to visualize the relationship between the number of hours spent studying and the test scores of a group of students.

```R
# Sample data
hours_studied <- c(2, 3, 1, 4, 5, 2, 6, 7, 3, 2)
test_scores <- c(65, 70, 60, 75, 80, 68, 85, 90, 72, 66)

# Create a scatter plot
plot(
  hours_studied,  # x-axis variable
  test_scores,    # y-axis variable
  xlab = "Hours Studied",  # Label for the x-axis
  ylab = "Test Scores",    # Label for the y-axis
  main = "Scatter Plot of Hours Studied vs. Test Scores",  # Title
  col = "blue",   # Color of the points
  pch = 19        # Shape of the points (solid circles)
)
```

In this code:

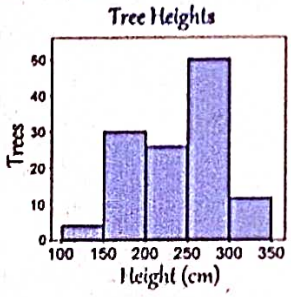
- We create two vectors, `hours_studied` and `test_scores`, to represent the data for the number of hours spent studying and the corresponding test scores.
- We use the `plot()` function to create the scatter plot. The `hours_studied` variable is plotted on the x-axis, and the `test_scores` variable is plotted on the y-axis.
- We provide labels for the x-axis and y-axis using the `xlab` and `ylab` arguments, respectively.
- We set a title for the plot using the `main` argument.
- We customize the appearance of the points by specifying the color as "blue" and the point shape as solid circles (pch = 19).

The resulting scatter plot will show individual data points, with each point representing a student's test score as a function of the number of hours studied. It allows you to visually assess the relationship between these two variables, which can be useful for understanding whether more hours of studying lead to higher test scores.


Scatter Plot of Hours Studied vs. Test Scc
Test Scores
60
65
70
75
80
85
90
1
2
3
4
5
6
7
Hours Studied

Histogram:- A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

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



Syntax:- **hist(v,main,xlab,xlim,ylim,breaks,col,border)**

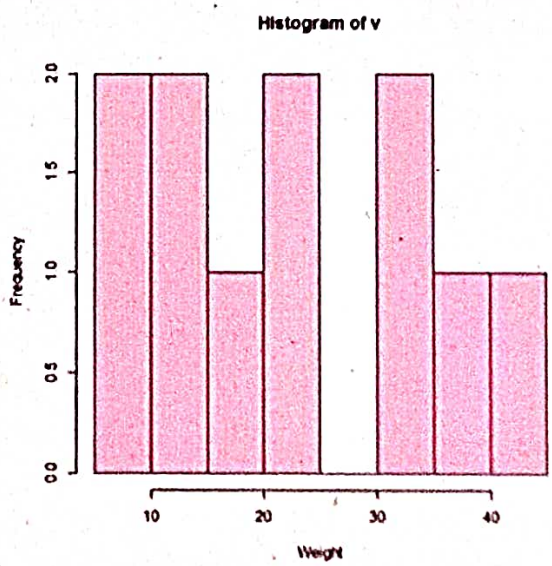
Following is the description of the parameters used –

- v is a vector containing numeric values used in histogram.
- 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.
- counts: The count of values in a particular range.

- mids: center point of multiple cells.
- density: cell density

Examples of histogram

```
#Simple histogram
v <- c(9,13,21,8,36,22,12,41,31,33,19)
h <- hist(v,xlab = "Weight",col = "pink",border = "blue")
> h
$breaks
[1] 5 10 15 20 25 30 35 40 45

$counts
[1] 2 2 1 2 0 2 1 1

$density
[1] 0.03636364 0.03636364 0.01818182 0.03636364
0.00000000 0.03636364 0.01818182
[8] 0.01818182

$mids
[1] 7.5 12.5 17.5 22.5 27.5 32.5 37.5 42.5

$xname
[1] "v"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"
```



To specify the range of values allowed in X axis

Poisson Distribution :- The **Poisson distribution** is the probability distribution of independent event occurrences in an interval. If λ is the mean occurrence per interval, then the probability of having x occurrences within a given interval is:

7a

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

$p(x)$ = Probability of x given λ
λ = Expected (mean) number 'successes'
e = 2.71828 (base of natural logs)
x = Number of 'successes' in per unit

Mean	Standard Deviation
$\mu = E(x) = \lambda$	$\sigma = \sqrt{\lambda}$

Examples:

1. The number of defective electric bulbs manufactured by a reputed company.
2. The number of telephone calls per minute at a switch board
3. The number of cars passing a certain point in one minute.
4. The number of printing mistakes per page in a large text.

R has four in-built functions to generate binomial distribution. They are described below.

- *dpois(x, lambda, log = FALSE)* :- This function gives the probability density distribution at each point.
- *ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)* :- This function gives the cumulative probability of an event. It is a single value representing the probability.
- *qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)*:- This function takes the probability value and gives a number whose cumulative value matches the probability value.
- *rpois(n, lamda)* :- This function generates required number of random values of given probability from a given sample.

  Following is the description of the parameters used –

  - ✓ x is a vector of numbers.
  - ✓ p is a vector of probabilities.
  - ✓ n is number of observations.
  - ✓ size is the number of trials.
  - ✓ prob is the probability of success of each trial.

BINOMIAL DISTRIBUTION:- The binomial distribution is a discrete probability distribution. It describes the outcome of n independent trials in an experiment. Each trial is assumed to have only two outcomes, either success or failure. If the probability of a successful trial is p, then the probability of having x successful outcomes in an experiment of n independent trials is as follows.

9a

$$P(x) = \frac{n!}{(n-x)!x!} p^x q^{n-x}$$

This starts the count of number of ways event can occur.
This is the probability of success for x trials.
This ends the count of number of ways event can occur.
This deletes duplications.
This is the probability of failure for the x trials.

Mean = $\mu = E(x) = np$
Variance = $\sigma^2 = np(1-p)$
Standard Deviation = $\sigma = \sqrt{np(1-p)}$

where
n = number of trials
p = probability of success
$1-p$ = probability of other outcome (failure)

R has four in-built functions to generate binomial distribution. They are described below.

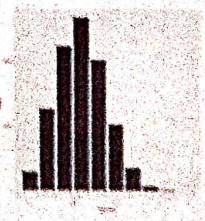
- ***dbinom(x, size, prob)*** :- This function gives the probability density distribution at each point.
- ***pbinom(x, size, prob)*** :- This function gives the cumulative probability of an event. It is a single value representing the probability.
- ***qbinom(p, size, prob)*** :- This function takes the probability value and gives a number whose cumulative value matches the probability value.
- ***rbinom(n, size, prob)*** :- This function generates required number of random values of given probability from a given sample.

  Following is the description of the parameters used –
  - ✓ x is a vector of numbers.
  - ✓ p is a vector of probabilities.
  - ✓ n is number of observations.
  - ✓ size is the number of trials.
  - ✓ prob is the probability of success of each trial.

Examples:

- ➢ *rbinom(n=1,size=10,prob=0.4)* - It generates 1 random number from the binomial distribution basesd on number of successes of 10 independent trails.
- ➢ *rbinom(n=5,size=10,prob=0.4)* - It generates 5 random number from the binomial distribution basesd on number of successes of 10 independent trails with probability 0.4.
- ➢ *rbinom(n=5,size=1,prob=0.4)* – Setting size to 1 turns the numbers into a bernoulli random variable, which can take only value 1 (success) or 0 (failure).
- ➢ To visualize the binomial distribution we randomly generate 10,000 experiments, each with 10 trails and 0.3 probability.

```
b <- data.frame(success=rbinom(n=10000,size=10,prob=0.3))
ggplot(b,aes(x=success))+geom_bar()
```



Descriptive statistics is a branch of statistics that deals with summarizing and describing the main features of a dataset. The primary purpose of descriptive statistics is to provide a concise and meaningful summary of the key characteristics and patterns within the data. It helps to make data more understandable and interpretable by presenting it in a more manageable form. Descriptive statistics include measures of central tendency, measures of dispersion, and data visualization techniques.

Here are some common descriptive statistics and examples:

10a

1. **Measures of Central Tendency:**
   - **Mean:** The mean is the average of a set of values.

     Example: Calculate the mean of the following numbers: 10, 15, 20, 25, 30.

     Mean = (10 + 15 + 20 + 25 + 30) / 5 = 20.

   - **Median:** The median is the middle value in a dataset when it is ordered. If there is an even number of data points, the median is the average of the two middle values.

     Example: Find the median of the following numbers: 12, 15, 18, 21, 24.

     Median = 18.

   - **Mode:** The mode is the value that appears most frequently in a dataset.

     Example: Determine the mode in the following numbers: 5, 8, 8, 12, 12, 15, 15.

     Mode = 8 and 15.
2. **Measures of Dispersion:**
   - **Range:** The range is the difference between the maximum and minimum values in a dataset.

     Example: Calculate the range of the following data: 12, 18, 24, 30, 36.

     Range = 36 - 12 = 24.

   - **Variance:** Variance measures the average squared difference of each data point from the mean.

     Example: Find the variance of the numbers: 5, 8, 12, 15, 18.

     Variance = 20.5.

   - **Standard Deviation:** The standard deviation is the square root of the variance and provides a measure of the dispersion or spread of the data.

     Example: Calculate the standard deviation of the data: 10, 15, 20, 25, 30.

     Standard Deviation ≈ 7.07.

3. **Data Visualization:**
   - Histograms: Histograms show the distribution of data by dividing it into intervals (bins) and displaying the frequency of values in each bin.

   - Box Plots: Box plots display the median, quartiles, and potential outliers in a dataset, making them useful for comparing distributions.

   - Scatter Plots: Scatter plots help visualize the relationship between two variables by displaying individual data points.

   - Bar Charts: Bar charts are used to display the frequency or count of categorical data.

Descriptive statistics are fundamental for understanding data, summarizing its characteristics, and making informed decisions. They provide insights into the central tendencies, variabilities, and overall patterns in the data, making it easier to interpret and draw meaningful conclusions.