Unit: V

Simple linear regression, multiple linear regression, linear model selection and diagnostics. Advanced graphics: plot customization, plotting regions and margins, point and click coordinate interaction, customizing traditional R plots, specialized text and label notation. Defining colors and plotting in higher dimensions, representing and using color, 3D scatter plots.

What is Regression?

- ✓ Regression is a statistical method used to analyze the relationship between one or more independent variables (predictors) and a dependent variable (outcome).

- **Linear Regression Modeling:**
  - ✓ Models the relationship with a straight line.
- **Types of Linear Regression Modeling**
  - ✓ Simple Linear Regression Model: Can be simple (one predictor)
  - ✓ Multiple Linear Regression Model: Can be multiple (multiple predictors).

Simple Linear Regression Modeling:

- ✓ A simple regression model (one independent variable) fits a regression line in 2-dimensional space

OR

- ✓ Simple linear regression is a statistical method used to model the relationship between a single independent variable (predictor) and a dependent variable (response). It assumes a linear relationship between the two variables.

Key Elements of a Simple Regression Model

Model Equation: The equation for a simple linear regression model can be expressed as:

$$y = \beta_0 + \beta_1 x + \epsilon$$

- ✓ y is the dependent variable.
- ✓ x is the independent variable.

- ✓ β0 is the y-intercept (the expected value of y when x=0).
- ✓ β1 is the slope of the line (the expected change in y for a one-unit change in x).
- ✓ ϵ represents the error term (the difference between the observed and predicted values).

Fitting the Model:

- ✓ The model is typically fitted using the method of Ordinary Least Squares (OLS), which finds the line that minimizes the sum of the squared differences (residuals) between the observed values and the predicted values.

Evaluation:

- ✓ **R-squared:** Indicates the proportion of the variance in the dependent variable that is predictable from the independent variable.
- ✓ **Coefficients:** Assess the relationship between the variables.
- ✓ **P-values:** Help determine the statistical significance of the coefficients.

Calculation of the Standard Error of the Regression

Sum of Squared Residuals (SSR): The sum of the squared residuals is calculated as:

$$SSR = \sum(e_i^2) = \sum(y_i - \hat{y}_i)^2$$

Standard Error Formula: The standard error of the regression is then calculated using the formula:

$$s_{Y|X} = \sqrt{\frac{SSR}{n-2}} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{n-2}}$$

Multiple regression model

- ✓ A multiple regression model is a statistical technique used to understand the relationship between one dependent variable and two or more independent variables.

Key Elements of a Multiple Regression Model

- ✓ **Model Equation**: The general form of a multiple regression equation is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$$

- ✓ y is the dependent variable.
- ✓ x1, x2…, xn are the independent variables.
- ✓ β0 is the intercept (the expected value of y when all x variables are zero).
- ✓ β1, β2…, βn are the coefficients representing the change in y for a one-unit change in the corresponding x.
- ✓ ϵ is the error term, capturing the variability not explained by the model.

Fitting the Model: The model is typically fitted using Ordinary Least Squares (OLS), which minimizes the sum of the squared residuals.

Calculation of the Standard Error of the Regression

Sum of Squared Residuals (SSR): The sum of the squared residuals is calculated as:

$$SSR = \sum (e_i^2) = \sum (y_i - \hat{y}_i)^2$$

Standard Error Formula: The standard error of the regression is then calculated using the formula:

$$s_{Y|X} = \sqrt{\frac{SSR}{n-2}} = \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{n-2}}$$

Assumption of linear regression Model

- ✓ When performing regression analysis (whether linear or nonlinear), it's important to ensure that certain **assumptions** are met for the results to be valid.

- ✓ If these assumptions are violated, the estimated coefficients may be biased or inefficient, and the results of statistical tests (such as hypothesis tests) may be misleading.

Here are the key assumptions for linear regression, including both simple and multiple regression:

1. **Linearity of the data**. The relationship between the predictor (x) and the outcome (y) is assumed to be linear.
2. **Normality of residuals**. The residual errors are assumed to be normally distributed
3. **Homoscedasticity** The variance of the residuals is constant across all levels of the independent variables.
4. **No Multicollinearity (for Multiple Regression)** In multiple regression, independent variables should not be too highly correlated
5. **No Specification Error** The model is correctly specified, meaning that all relevant variables are included, and irrelevant variables are excluded.
6. **Independence:** the residuals (errors) of the model should be independent.

You should check whether or not these assumptions hold true

Statistical Assumptions

- ✓ **Linearity:** Relationship between independent variables and dependent variable is linear.
- ✓ **Independence:** Observations are independent of each other.
- ✓ **Homoscedasticity:** Constant variance of residuals across all levels of independent variables.
- ✓ **Normality:** Residuals follow a normal distribution.
- ✓ **No multicollinearity:** Independent variables are not highly correlated.

Model Assumptions

- ✓ **Correct functional form:** Model specifies correct relationship between variables.
- ✓ **No omitted variable:** Relevant variables are included.
- ✓ **No measurement error:** Variables measured accurately.

Regression diagnostics:

- ✓ Diagnosing a linear regression model is crucial to ensuring that the model assumptions hold true and that the results are valid.
- ✓ By conducting diagnostic checks, you can identify potential problems in your model and take corrective actions.

Here are some key aspects of regression diagnostics:

Residual Analysis

- ✓ **Residuals**: The differences between observed values and predicted values. Analyzing residuals helps check the assumptions of linearity, homoscedasticity, and normality.
- ✓ **Residual Plots**: Residual plots help assess the assumptions of linearity, homoscedasticity, and independence.

Normality of Residuals

- ✓ **Q-Q Plots**: **A histogram or quantile-quantile (Q-Q) plot** of the residuals helps assess the **normality** of the residuals.

Homoscedasticity

- **Spread-Location Plot**: A plot of standardized residuals versus predicted values can show if the variance of residuals is constant across levels of the independent variables.

Outliers and Influential Points

- ✓ **Outlier Detection**: Outliers are observations that differ significantly from other data points. They can be identified using residuals or through standardized residuals
- ✓ **Leverage:** Measures how far an independent variable is from the mean of the independent variables..

Cook's Distance:

- ✓ This measure combines leverage and residuals to identify influential observations. Points with a Cook's Distance greater than 1 may be influential.

Multicollinearity

- ✓ Multicollinearity occurs when two or more predictors are highly correlated, making it difficult to estimate their individual effects on the dependent variable.

- ✓ **Variance Inflation Factor (VIF)**: VIF measures how much the variance of an estimated regression coefficient increases if your predictors are correlated. A VIF value above 10 indicates multicollinearity issues.

Correlation Matrix: Examining pairwise correlations among predictors can also help identify multicollinearity.

Model Specification

Link Test: A link test checks whether the functional form of the model is correctly specified. If the squared predicted values significantly affect the outcome, it suggests a specification error.

linear model selection

- ✓ In linear regression, **model selection** refers to the process of choosing the most appropriate set of predictors (independent variables) to include in the model.
- ✓ This is crucial because including too many predictors can lead to **overfitting**, while including too few may result in **underfitting** or omitted variable bias.

Stepwise Selection Methods

- ✓ Stepwise selection involves automatically adding or removing predictors based on statistical criteria. There are three primary variants:

Forward Selection

- ✓ Starts with an empty model (no predictors) and adds predictors one by one
- ✓ The process continues until no more predictors improve the model significantly

Backward Elimination

- ✓ Starts with a full model (including all potential predictors) and removes predictors one by one.

Stepwise Selection: A combination of forward selection and backward elimination. Predictors can be added or removed at each step based on significance criteria.

Best Subset Selection

- ✓ Best subset selection examines all possible combinations of predictors and identifies the subset of predictors.
- ✓ Fit all possible models, using different combinations of predictors.

Regularization Methods

- ✓ Regularization methods are used to prevent overfitting by adding a penalty term to the regression cost function, effectively shrinking the coefficients of less important predictors toward zero

Cross-Validation

- ✓ Cross-validation is a technique where the dataset is split into multiple subsets (folds), and the model is trained and validated on different combinations of these subsets

Advanced graphics:

1. Advanced plot customization in statistics involves fine-tuning visual elements of graphs and charts to convey data effectively.
2. It includes adjusting colors, adding annotations, changing axis scales, incorporating multiple data series, and utilizing various plot types to enhance the representation of statistical information.
3. Advanced graph customization allows you to tailor every aspect of your visualizations.

Characteristics of advanced graphs

1. **Color and Style:**
   - ✓ **Color Palettes:** Choose meaningful color schemes for different data elements.
   - ✓ **Line Styles and Markers:** Customize the appearance of lines, such as dashed or dotted lines, and markers for data points.
2. **Annotations:**
   - ✓ **Text Annotations:** Add labels, titles, or captions to highlight important features or provide additional context.
   - ✓ **Arrows and Lines:** Use arrows or lines with annotations to draw attention to specific data points or trends.
3. **Axis Customization:**

- ✓ **Tick Marks and Labels:** Adjust the appearance of tick marks and labels on both the x and y axes.
- ✓ **Logarithmic Scales:** Utilize logarithmic scales for axes if data spans multiple orders of magnitude

4. **Legend:**
   - ✓ **Positioning:** Move the legend to a suitable position (top, bottom, left, right) for better readability.
   - ✓ **Custom Labels:** Provide clear and concise labels for each series in the legend.
5. **Faceting and Multiple Plots:**
   - ✓ **Facet Grids:** Use facets to create multiple plots based on different categories, making it easier to compare subsets of the data.
   - ✓ **Arrangement:** Adjust the arrangement of subplots to create a coherent and informative layout.
6. **Statistical Summaries:**
   - ✓ **Error Bars:** Include error bars to show variability or uncertainty in data
   - ✓ **Regression Lines:** Add regression lines or other statistical summaries to illustrate trends.
7. **Background and Grid Lines:**
   - ✓ **Background Color:** Customize the background color to improve contrast and aesthetics.
   - ✓ **Grid Lines:** Adjust the appearance of grid lines to guide the viewer's eye.
8. **Export and Save Options:**
   - ✓ **High-Resolution Output:** Ensure that exported graphs are of high resolution for publications or presentations.
   - ✓ **Save in Multiple Formats:** Save plots in different formats (PDF, PNG, SVG) for versatility.

Handling the Graphics Device

In advanced graphics using or Manually Opening a New Device to using three built in function

1. dev. new () --------To open the new device plot

2. dev. off () ---------To close the device plot after completion of the task.

3. dev. set () --------- Switching Between Devices to change something in Device 2 without closing Device 3, use dev. Set

Plot customization:

- ✓ Advanced plot customization in advanced graphs involves fine-tuning visual elements to convey complex data effectively.

✓ Customizing plots in R can be done using various functions and parameters to modify the appearance of different plot elements

Here's a basic overview of how you can customize plots in R.

1. Titles and Labels
2. Colors and Symbols
3. Adding a Legend
4. Axis Customization
5. Lines
6. Annotations
7. Plot Range
8. Defining a Particular Layout

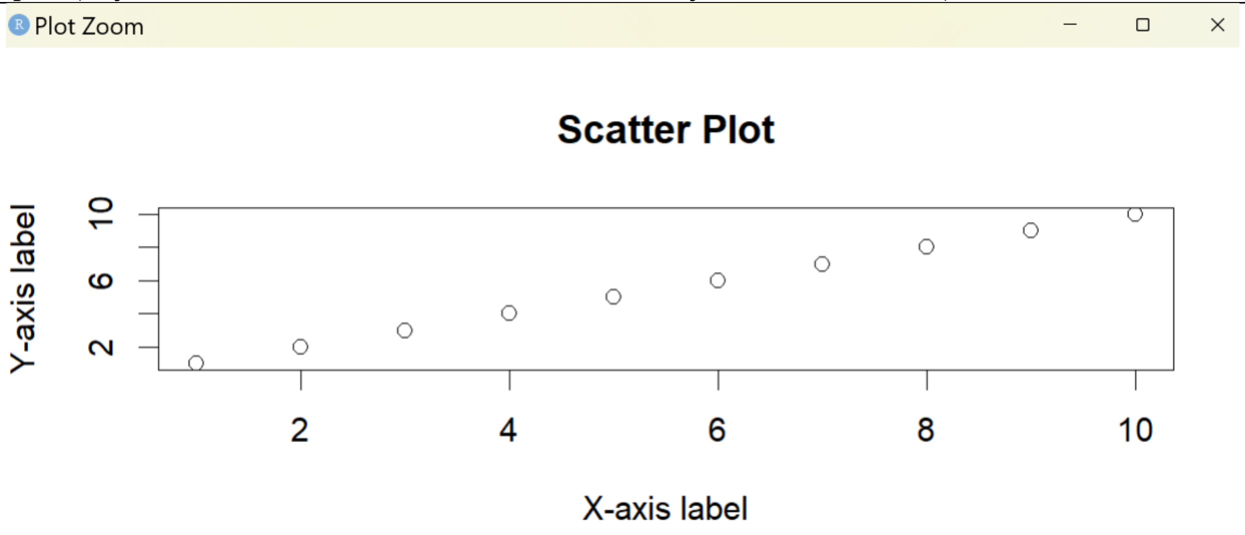
Titles and Labels:

✓ **main:** Main title of the plot.

✓ **xlab and ylab:** Labels for the x and y-axes.

Example:

```
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1,2,3,4,5,6,7,8,9,10)
plot (x, y, main="Scatter Plot", xlab="X-axis label", ylab="Y-axis label")
```

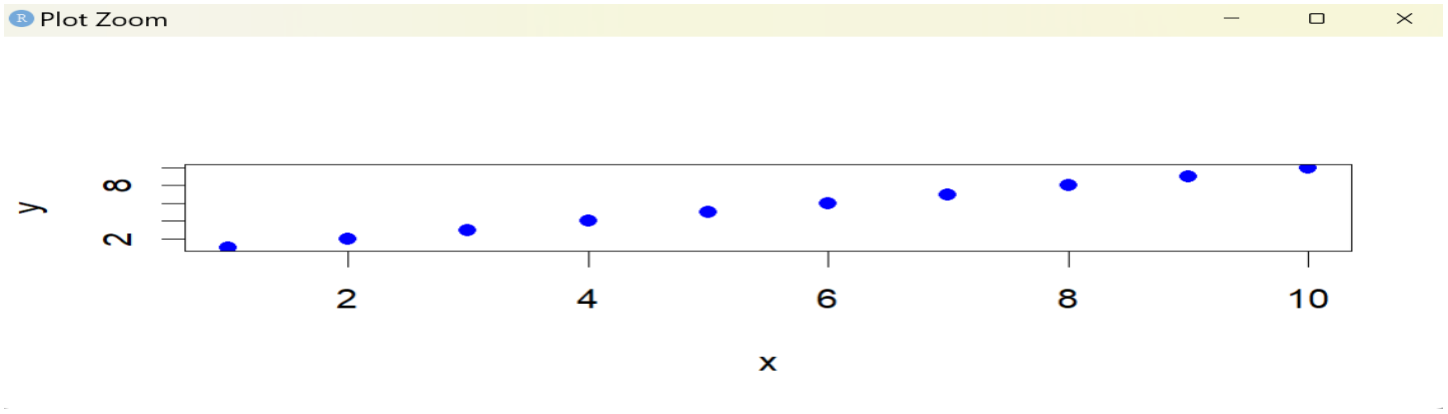


Colors and Symbols:

- ✓ **col:** You can specify colors for points, lines, and backgrounds using the col argument.
- ✓ **Point Characters (pch):**
  - The pch argument specifies the type of point to use in the plot.
  - **Here are some common `pch` values:**

- ✓ `1`: Open circle
- ✓ `2`: Open triangle
- ✓ `3`: Plus, sign
- ✓ `16`: Filled circle
- ✓ `17`: Filled triangle

```
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1,2,3,4,5,6,7,8,9,10)
plot(x, y, col="blue", pch=16)
```

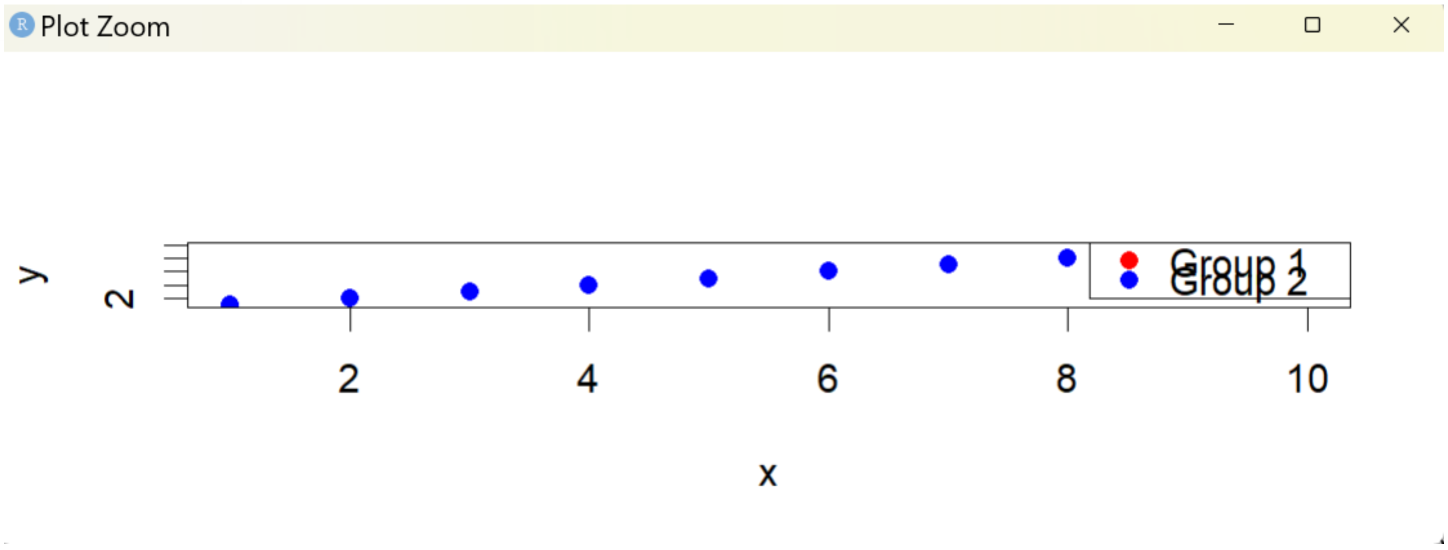


Adding a Legend:

- ✓ **legend:** Adding a legend to the plot.

Example: legend ("topright", legend=c("Group 1", "Group 2"), col=c("red", "blue"), pch=16)

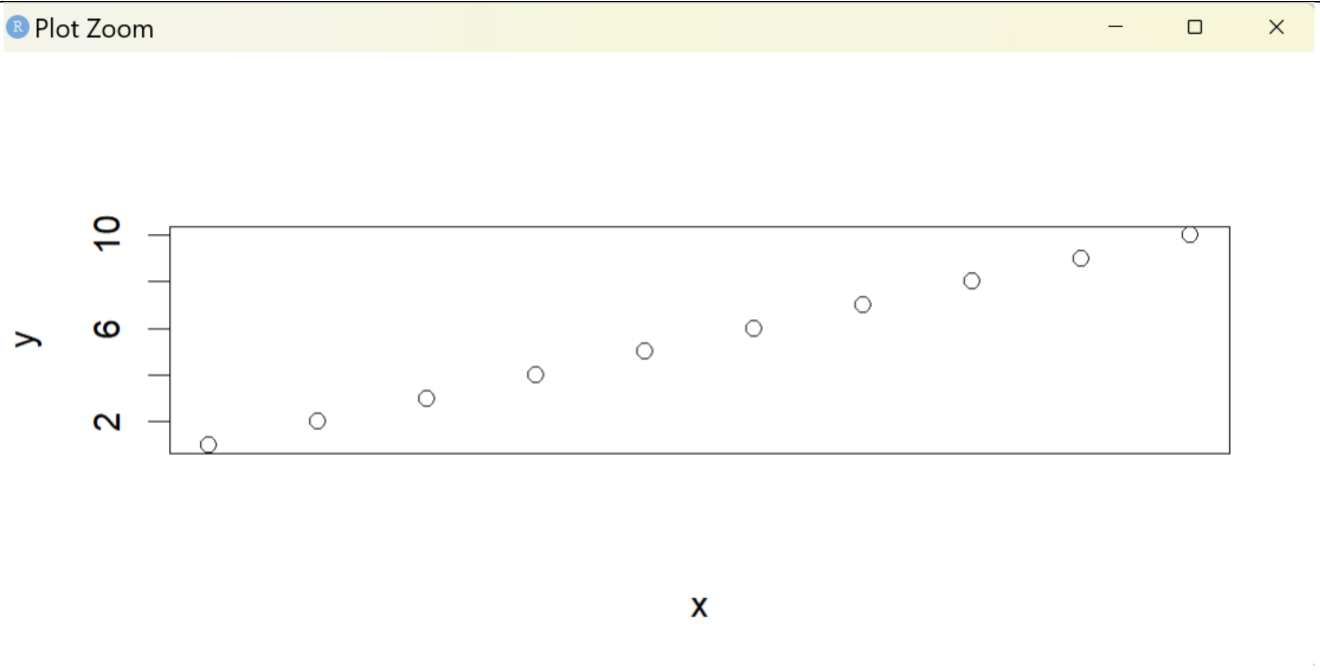
```
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1,2,3,4,5,6,7,8,9,10)
plot (x, y, col="blue", pch=16)
legend ("topright", legend=c("Group 1", "Group 2"), col=c("red", "blue"), pch=16)
```



Axis Customization:

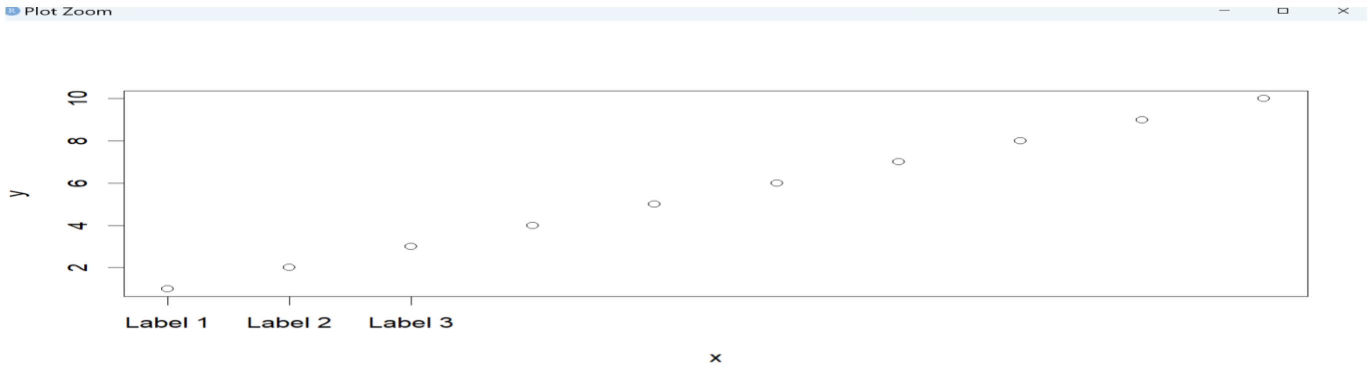
- ✓ **axis:** Customize the axis ticks and labels.
- ✓ The xaxt = "n" parameter in R's base plotting functions is used to suppress the drawing of the x-axis in a plot.

```
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1,2,3,4,5,6,7,8,9,10)
plot(x, y, xaxt="n") # Turn off x-axis
```



- ✓ You can customize the appearance of the tick marks and labels when you add the x-axis manually.

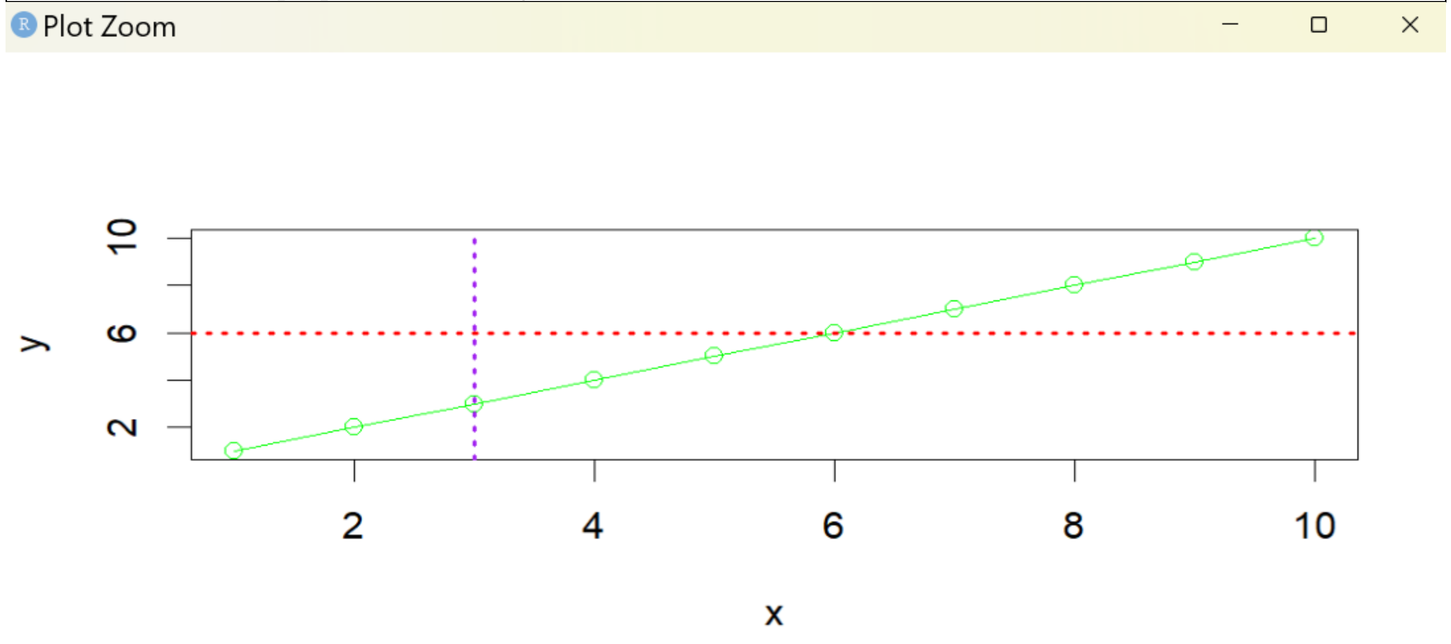
```
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1,2,3,4,5,6,7,8,9,10)
plot(x, y, xaxt="n") # Turn off x-axis
axis(1, at=c(1, 2, 3), labels=c("Label 1", "Label 2", "Label 3"))
```



Lines:

✓ You can customize the appearance of lines by adjusting parameters like col (color), lty (line type), and lwd (line width).

```
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1,2,3,4,5,6,7,8,9,10)
lines (x, y, col="green")
abline(h = 6, col = "red", lwd = 2, lty = 3)  # Horizontal dashed line
# Add a vertical line at x = 3
abline(v = 3, col = "purple", lwd = 2, lty = 3)  # Vertical dashed line
```



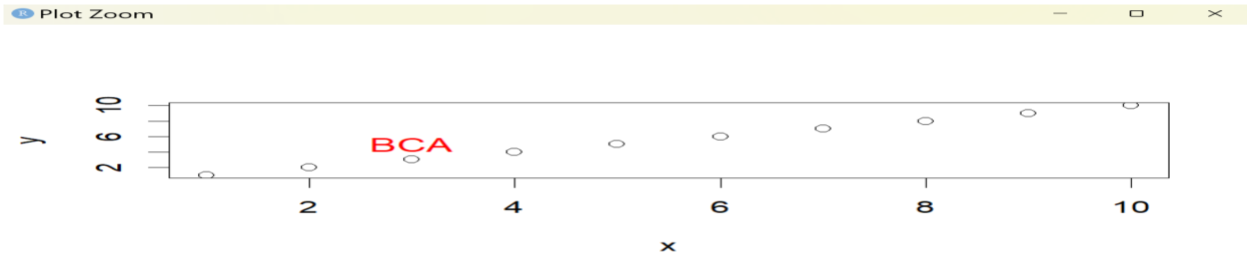
Annotations:

- ✓ In base R, you can use the `text ()`, `points ()`, `arrows ()`, and `segments ()` functions to add annotations.

a. Adding Text Annotations

- ✓ You can use the `text ()` function to add labels at specified coordinates.

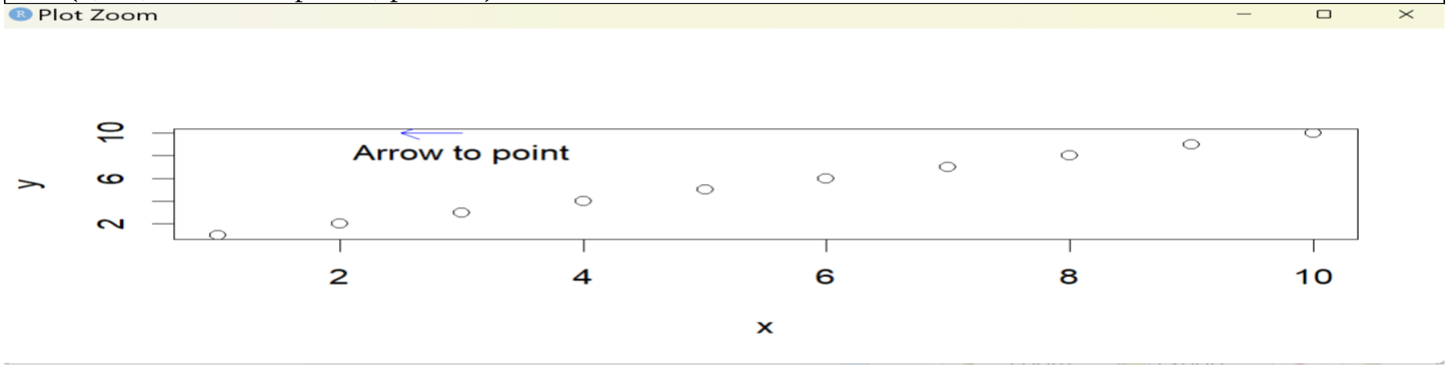
```
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1,2,3,4,5,6,7,8,9,10)
plot(x, y)
text(3, 5, "BCA", col = "red", cex = 1.2)  # cex controls text size
```



Adding Arrows

- ✓ The `arrows ()` function can be used to draw arrows to indicate direction or highlight points.

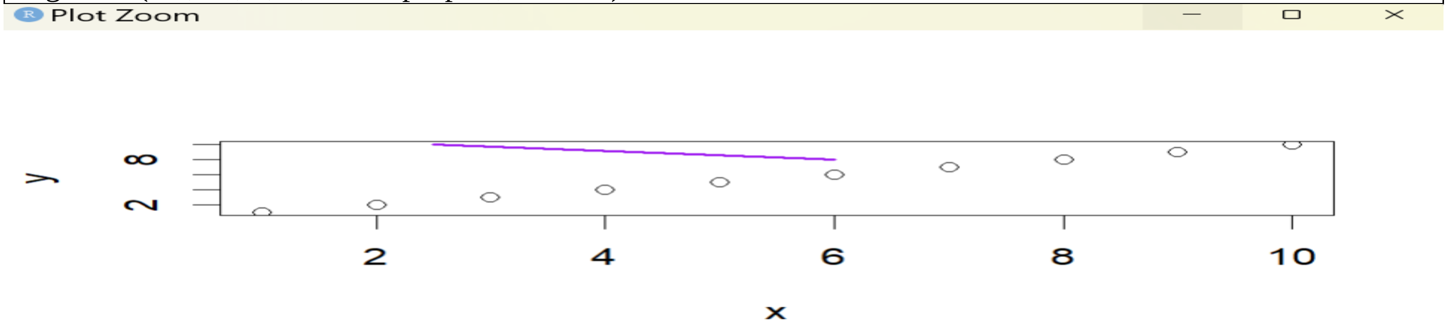
```
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1,2,3,4,5,6,7,8,9,10)
plot(x, y)
arrows(3, 10, 2.5, 10, col = "blue", length = 0.1)
text(3, 10, "Arrow to point", pos = 1)
```



Adding Segments

- ✓ Use `segments ()` to draw lines between points, which can highlight relationships.

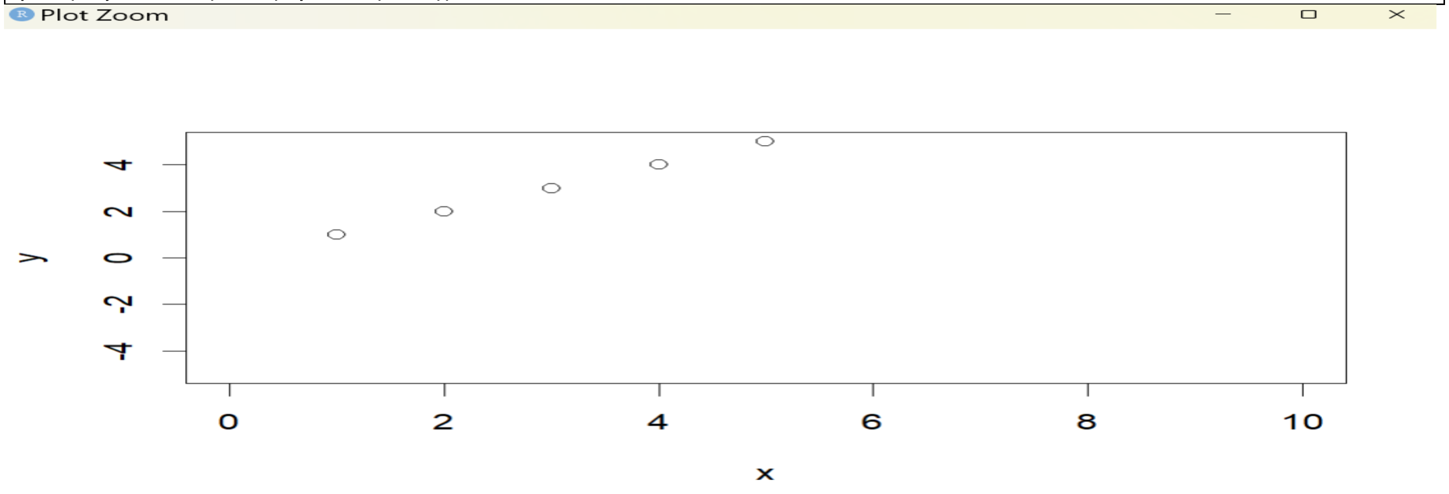
```
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1,2,3,4,5,6,7,8,9,10)
plot(x, y)
segments(2.5, 10, 6, 8, col = "purple", lwd = 2)
```



Plot Range:

- ✓ **xlim and ylim:** Set the range of the x and y-axes.
- ✓ Example: plot (x, y, xlim=c(0, 10), ylim=c(-5, 5))

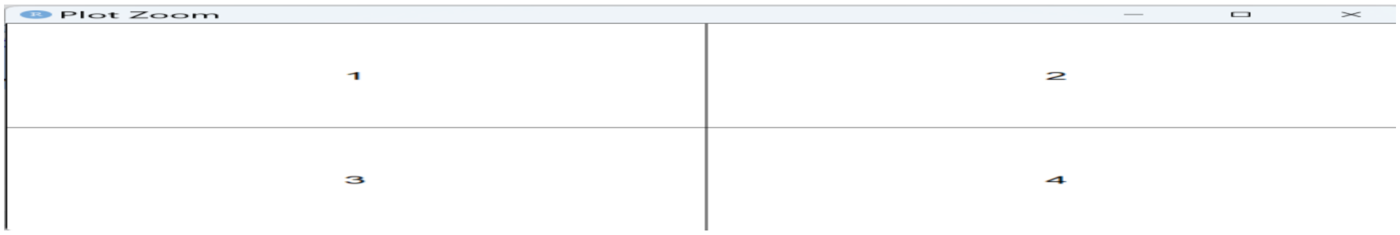
```
x<-c(1,2,3,4,5,6,7,8,9,10)
y<-c(1,2,3,4,5,6,7,8,9,10)
plot(x, y, xlim=c(0, 10), ylim=c(-5, 5))
```



Defining a Particular Layout:

- ✓ The arrangements of plots in a single device using the layout function, which offers more ways to individualize the panels into which the plots will be drawn.

```
lay.Mat <- matrix(c(1,3,2,4),2,2)
lay.Mat
layout(mat=lay.Mat)
layout.show(n=max(lay.Mat))
```



Plotting Regions and Margins:

Plotting Regions

- ✓ In R plotting, **regions** refer to the distinct areas within a plot where graphical elements can be rendered.

For any single plot created using base R graphics, there are three regions that make up the image.

- ✓ **plotting region** is the area where the actual data is plotted.
- ✓ **Figure region:** The figure region is the area that contains the space for your axes, their labels, and any titles. These spaces are also referred to as the figure margins.
- ✓ **Outer region:** The outer region, also referred to as the outer margins, is additional space around the figure region that is not included by default but can be specified if it's needed

Example for region

```
plot(1:10, type="n") # Create an empty
rect(1, 2, 4, 6, col="lightblue", border="blue")
```

Margins:

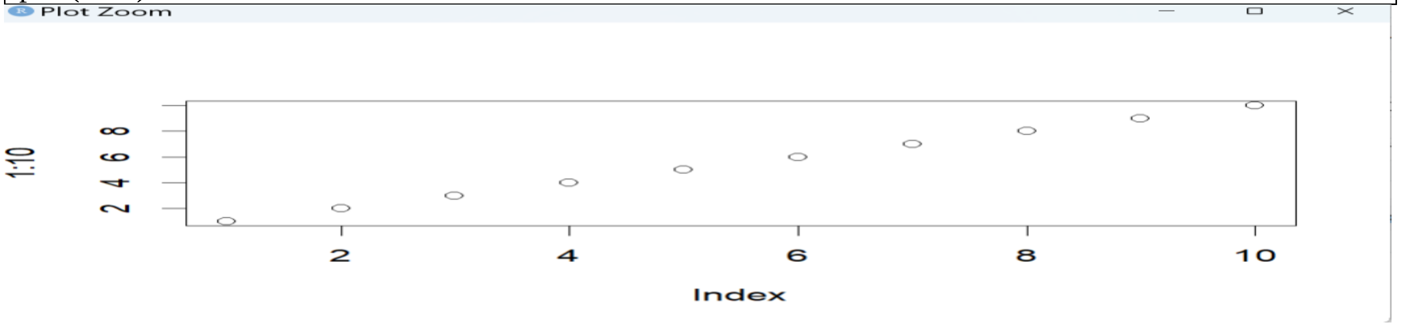
- ✓ In R, **margins** refer to the space around the plotting area in a graph
- ✓ You can control the size of the margins using **the par()** function.

Adjusting Margins with par() :Inner Margin

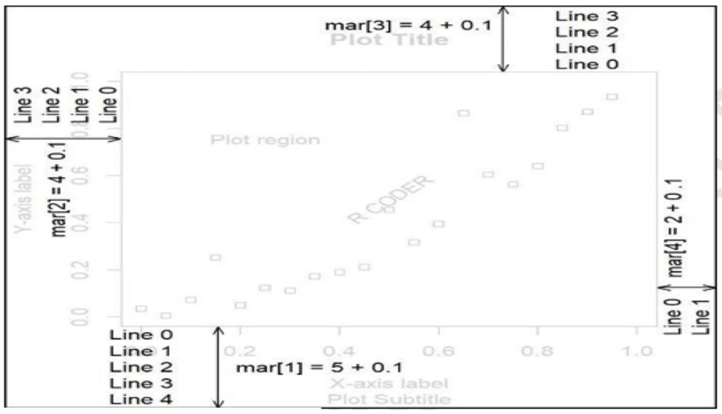
- ✓ The par() function allows you to set the graphical parameters, including margins, using the mar argument. The mar argument takes a numeric vector of four values, which specify the size of the margins in lines (bottom, left, top, right).

- ✓ par(mar = c(bottom, left, top, right))

```
par(mar=c(5, 4, 4, 2)) # Set margins (bottom, left, top, right)
plot(1:10)
```

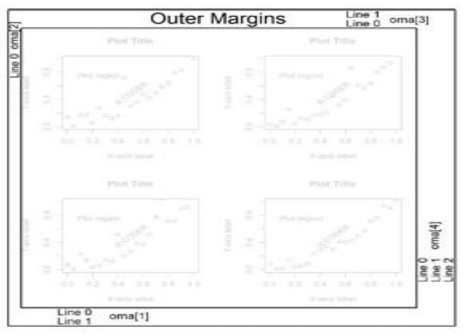


- ✓ By default par(mar = c(5, 4, 4, 2) + 0.1).



Outer margin:

- ✓ In R, **outer margins** refer to the space surrounding the entire plotting area, beyond the inner margins defined by the mar parameter.
- ✓ These outer margins are controlled using the oma parameter in the par() function.
- ✓ par(oma = c(bottom, left, top, right))
  - ✓ **`bottom`**: Size of the outer margin at the bottom of the plot (in lines).
  - ✓ **`left`**: Size of the outer margin on the left side of the plot (in lines).
  - ✓ **`top`**: Size of the outer margin at the top of the plot (in lines).
  - ✓ **`right`**: Size of the outer margin on the right side of the plot (in lines).



```
# Set up the plotting area with default margins and outer margins
par(mar = c(2, 1, 2, 1), oma = c(1, 2, 2, 2))
plot(1:10)
```

- ✓ par(mar = c(4, 4, 2, 1)) sets the inner margins for each plot.
- ✓ par(oma = c(2, 2, 2, 2)) sets the outer margins.

Default Margins

1. **Inner Margins (`mar`)**:

   - ✓ The default inner margins are set using the `mar` parameter in the `par()` function. The typical default value is: par(mar = c(5, 4, 4, 2) + 0.1)

Outer Margins (oma)**:**

- ✓ The default outer margins are specified **by the oma** parameter. By default, this is typically set to:

  par(oma = c(0, 0, 0, 0))

Point-and-Click Coordinate Interaction:

- ✓ Point-and-click coordinate interaction in a graph refers to the capability for users to interact with plotted data points by clicking or hovering over them to trigger specific actions or to display additional information related to those points.
- ✓ To use the **locator() function** for point-and-click coordinate interaction. This function allows you to click on a plot, and it returns the coordinates of the point where you clicked

Features of Point-and-Click Coordinate Interaction:

- ✓ Retrieving Coordinates

- ✓ Visualizing Selected Coordinates
- ✓ Ad Hoc Annotation

Retrieving Coordinates

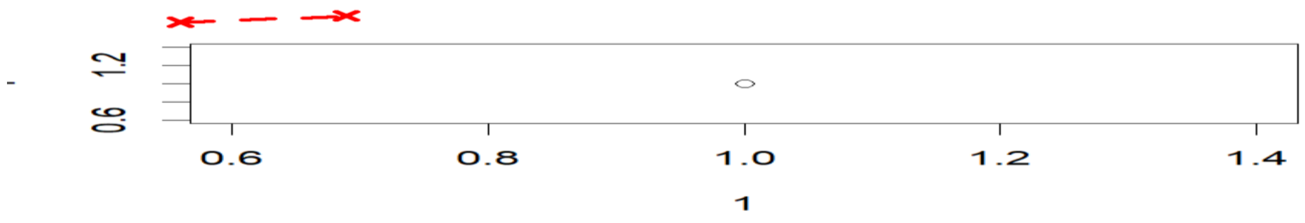
- ✓ Retrieving coordinates using point-and-click interaction in R can be done effectively with the locator() function.
- ✓ This function allows users to click on a plot and capture the x and y coordinates of the clicked points.

```
x <- c(1,2,3)
y <- c(1,2,3)
plot(x, y, main="Click to Select Points", xlab="X-Axis", ylab="Y-Axis")
selected_points <- locator(n = 2)  # Adjust 'n' for more points
print(selected_points)
```

Visualizing Selected Coordinates

- ✓ To use locator to plot the points you select as either individual points or as lines

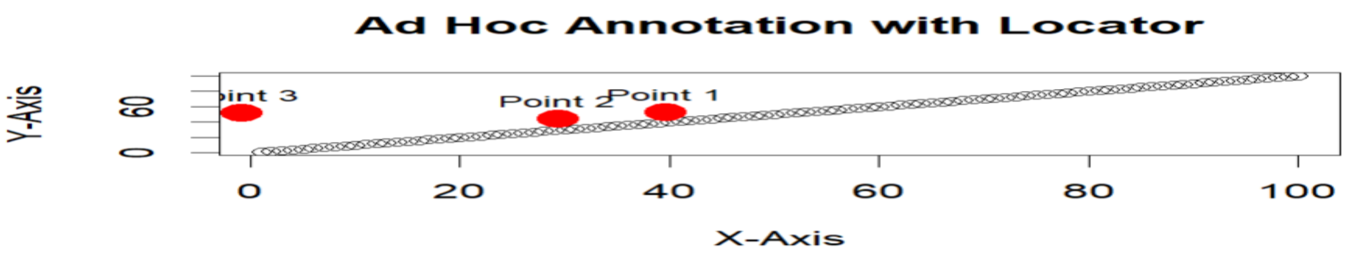
```
plot (1,1)
locator (type="o", pch=4,lty=2,lwd=3,col="red",xpd=TRUE) R>
```



Ad Hoc Annotation

- ✓ Ad hoc annotation using the locator() function in R allows you to interactively select points on a plot and then add text, lines, or other annotations based on those selected points.

```
x <-(1:100)
y <- (1:100)
plot(x, y, main="Ad Hoc Annotation with Locator",
     xlab="X-Axis", ylab="Y-Axis")
selected_points <- locator(n = 3)  # Allow selection of 3 points
points(selected_points$x, selected_points$y, col="red", pch=19, cex=2)
for (i in 1:length(selected_points$x)) {
  text(selected_points$x[i], selected_points$y[i],
       labels=paste("Point", i), pos=3, col="black", cex=0.8)
}
```



Customizing traditional r plots:

- ✓ Customizing traditional R plots allows you to enhance the visual appeal and clarity of your data visualizations. You can modify various elements, including titles, labels, colors, point shapes, line types, and much more...
- ✓ Customizing traditional R plots involves modifying various aspects of the plot to make it visually appealing and informative. To customize different elements of a plot in R.

Types of Customizing Traditional R Plots

- ✓ **Graphical Parameters for Style and Suppression.**
- ✓ **Customizing Boxes**
- ✓ **Customizing Axes**

1) Graphical Parameters for Style and Suppression.

- ✓ Graphical parameters in R, managed by the par() function, allow you to control various aspects of plot appearance and layout.
- ✓ They enable customization of plots by modifying parameters such as margins, axis appearance, text size, colors, and more.

Key Features of the par() Function

1. **Margins**:
   - Control the size of plot margins using the **mar parameter**. This parameter takes a numeric vector of the form c(bottom, left, top, right).
2. **Outer Margins**:
   - Adjust outer margins with the **oma parameter**, which specifies the size of the outer margins in lines.
3. **Plot Layout**:

- Use the **mfrow and mfcol parameters** to arrange multiple plots in a single figure. mfrow fills rows first, while mfcol fills columns first.

4. **Text Size and Font**:
   - Control text size with **cex** and the font with font.
5. **Line Type and Width**:
   - Customize line types with lty and line widths with lwd.
6. **Background Color**:
   - Set the background color of plots using the bg parameter.

```
# Set up the layout for multiple plots
par(mfrow = c(2, 2),   # 2 rows and 2 columns
   mar = c(4, 4, 2, 1),  # Margins: bottom, left, top, right
   col.axis = "blue",    # Axis text color
   col.lab = "red",      # Axis label color
   col.main = "darkgreen")  # Main title color
x <- c (1:100)
y <- x(1:100)
plot(x,y)
```

Customizing Boxes:

✓ **The `box()`** function adds a box around the plot area, and you can customize its properties using parameters.

Here's a breakdown of how to customize the box:

✓ **Line Type (`lty`)**: Specify the type of line (solid, dashed, etc.).

✓ **Line Width (`lwd`)**: Adjust the width of the box line.

✓ **Color (`col`)**: Change the color of the box

```
set.seed(42)
x <- rnorm(100)
y <- rnorm(100)
# Create a scatter plot
plot(x, y, main = "Custom Box around Plot", xlab = "X-Axis", ylab = "Y-Axis")
# Customize the box
box( lwd = 3, col = "darkblue", lty = 2)  # Custom line width, color, and type
```

Customizing Axes

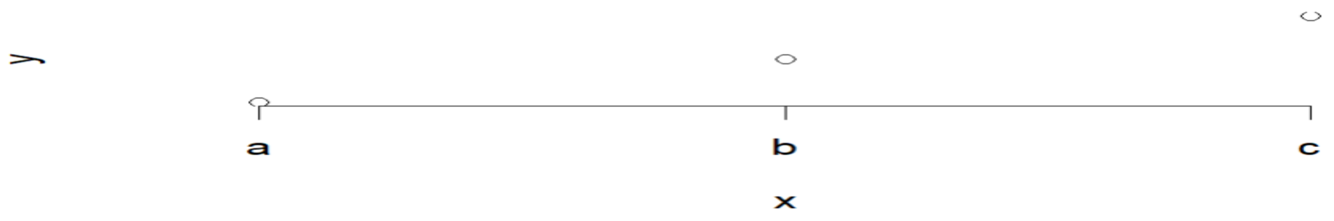
✓ Customizing axes in R plots allows you to improve the readability and presentation of your visualizations. You can adjust various aspects of the axes, including their labels, limits, tick marks, and appearance using **axis() function.**

axis() function

The axis function allows you to control the addition and appearance of an axis on any of the four sides of the plot region.

- ✓ **Syntax: axis (side, at=NULL, labels=TRUE)**
- ✓ Parameters: side: It defines the side of the plot the axis is to be drawn on possible values such as below, left, above, and right.
- ✓ at: Point to draw tick marks
- ✓ label: for axis() labels

```
x <- 1:3
y <-1:3
plot(x, y, axes = FALSE)
# Calling the axis() function
axis(side = 1, at = 1:3,labels = c('a','b','c'))
```



Specialized Text and Label Notation:

- ✓ In R, you can use specialized text and label notation to enhance the annotations and labels on your plots.
- ✓ This includes mathematical notation, formatting options, and positioning text for better clarity and visual appear.

Key Techniques for Specialized Text and Labels

- ✓ **Math Annotation**:
  - o Use expression () to include mathematical notation in titles and labels.
- ✓ **Subscripts and Superscripts**:
  - o Subscripts can be created with [], and superscripts with ^ within expression ().
- ✓ **Positioning Text**:
  - o Use the text () function to place text at specific coordinates on your plot.
- ✓ **Customizing Text Appearance**:
  - o Adjust font size, style, and color using parameters like cex, font, and col.

text ()function

- ✓ The text() function in R is used to add text annotations to plots. This function allows you to specify the coordinates where the text should appear, customize the content, and adjust various text properties like size, color, and font.

  Syntax: text (x, y, labels, pos = NULL, cex = 1, col = "black", font = 1, ...)

  - **x**: The x-coordinate(s) where the text will be placed.
  - **y**: The y-coordinate(s) where the text will be placed.
  - **labels**: The text labels to be displayed.
  - **pos**: Positioning of the text relative to (x, y). Options include 1 (below), 2 (left), 3 (above), and 4 (right).
  - **cex**: Size of the text relative to the default.
  - **col**: Color of the text.
  - **font**: Font type (1 = plain, 2 = bold, 3 = italic, 4 = bold italic).

```
x <- 1:10
y <- 1:10
plot(x, y)
text(5, 4, "Midpoint", col = "red", cex = 1.5)
```

Customizing Text Appearance

- ✓ You can customize font size, style, and color:

```
x <- 1:10
y <- 1:10
plot(x, y)
text(5, 5, "Custom Text", col = "purple", cex = 1.5, font = 2)
```

Use expression ()

- ✓ In R, you can include mathematical expressions in your plots using the expression () function. This allows you to create annotations, axis labels, and titles that feature mathematical symbols, subscripts, superscripts.

Basic Formatting:

- ✓ Use `expression()` to create mathematical text directly in plot titles and labels.

Example

```
plot(1:10, (1:10)^2, main = expression(y == x^2))
```

Subscripts and Superscripts:

- ✓ Use [] for subscripts and ^ for superscripts.

Example

```
plot(1:10, (1:10)^3, main = expression(y[i] == x^3))
```

Special Symbols:

- ✓ R supports a variety of mathematical symbols like Greek letters (alpha, beta, etc.)

```
plot(1:10, (1:10)^2, main = expression(alpha + beta == gamma))
```

Combined Expressions:

- ✓ Combine text and mathematical symbols using paste() inside expression().

```
X<-1:10
Y<-1:10
plot(X,Y, main = expression(paste("Quadratic Function: ", y == x^2 + epsilon)))
```

Defining colors and plotting in higher dimensions:

- ✓ In R, visualizing data in higher dimensions can be accomplished using various techniques, such as color, size, and shape to represent additional variables.

Here's how to define colors and create effective visualizations for higher-dimensional data.

Defining Colors in R

- ✓ You can define colors in R using:

Named Colors:

- ✓ R provides a range of named colors you can use directly.
- ✓ Some common ones include: Red: "red" • Green: "green"

Hexadecimal Color Codes:

- ✓ They consist of a # symbol followed by a combination of six characters (0-9 and A-F) representing the intensity of red, green, and blue (RGB) channels ,For example: 1. Red: "#FF0000" Green: "#00FF00"

RGB Values:

- ✓ You can define colors using RGB values, specifying the intensity of red, green, and blue channels individually within the range of 0 to 1, a) Red: rgb(1, 0, 0) b). Green: rgb(0, 1, 0) c). Blue: rgb(0, 0, 1)

HSL (Hue, Saturation, Lightness)

- ✓ The hue value represents the color (0-360), saturation represents intensity (0-1), and lightness represents the brightness (0-1). For example: 1. Red: hsl(0, 1, 0.5) 2. Green: hsl(120, 1, 0.5)

```
my_color <- "red"
plot(1:10, col=my_color, pch=16)
my_rgb_color <- rgb(0.2, 0.5, 0.8)
plot(1:10, col=my_rgb_color, pch=16)
plot(1:10, col = "#FFA500", main = "Hexadecimal Color")
my_hex_color <- "#FFA500" # Orange
plot(1:10, col=my_hex_color, pch=16)
plot(1:10, col = hsl(140, 1, 0.5), main = "HSL Color")
```

Plotting in higher dimensions:

- ✓ Plotting in higher dimensions often involves techniques like 3D plots, color mapping, facet grids, or even interactive visualizations to represent more than three dimensions effectively

Characteristics of Plotting in higher dimensions:

2D Plots:

- ✓ '2D' stands for 2-dimensional and a 2D line is a line that is moved in 2 dimensions. A line in 2D means that we could move in forward and backward direction but also in any direction like left, right, up, down.

```
X<-1:10
Y<-1:10
plot(x, y, col="blue", pch=16)
```

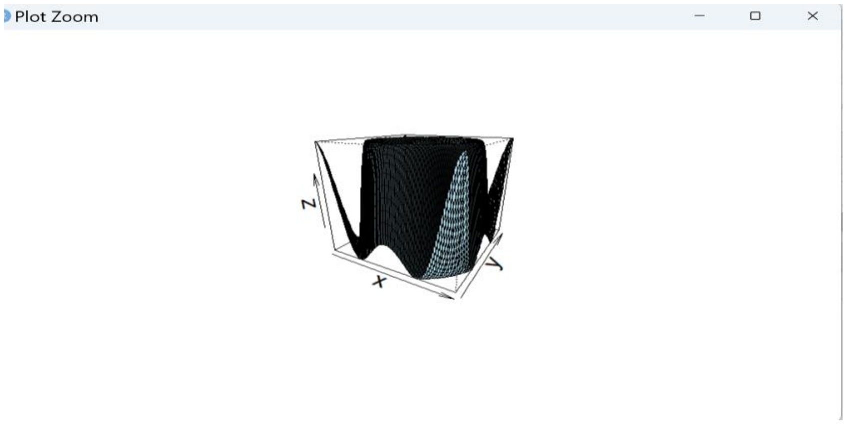
Faceting

- ✓ Faceting allows you to create multiple plots based on a categorical variable, effectively visualizing additional dimensions.

surface plot: is a three-dimensional graphical representation of data that shows the relationship between three continuous variables. It can be visualized as a 3D surface over a grid, where two variables are represented on the x and y axes, and the third variable is represented by the height of the surface above the x-y plane.

Example:

```
x <- seq(-10, 10, length=100)
y <- seq(-10, 10, length=100)
z <- outer(x, y, function(x, y) sin(sqrt(x^2 + y^2)))
persp(x, y, z, theta=30, phi=20, col="lightblue", shade=0.5)
```



REPRESENTING AND USING COLOR:

- ✓ One of the most common methods of color specification is to specify different saturations or intensities of three primaries—red, green, and blue (RGB)—which are then mixed to form the resulting target color.

Palette ():

- ✓ The palette() function in R is used to get or set the colors of the current plotting palette.
- ✓ This function is particularly useful for customizing the color scheme of your plots.
- ✓ By default, R has a predefined palette, but you can modify it to suit your needs.

Using palette()

- ✓ **Get Current Palette**: If you call `palette ()` without any arguments, it will return the current color palette.
- ✓ **Set a New Palette**: You can provide a vector of colors to `palette ()` to change the colors used for plotting.

```
current_palette <- palette()
print(current_palette)
new_colors <- c("red", "blue", "green", "orange", "purple")
palette(new_colors)
print(palette())
```

Built-in Palettes

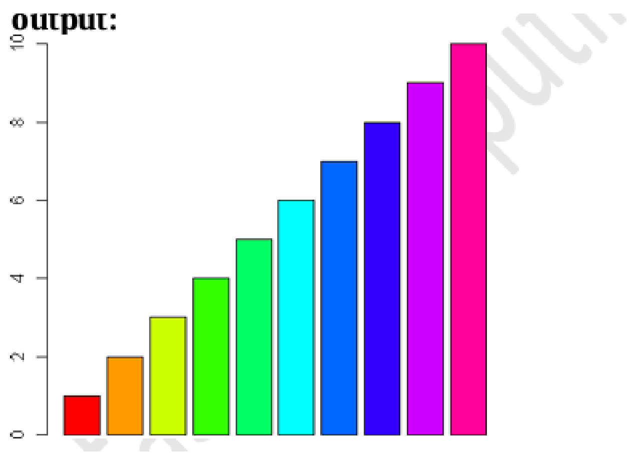
- ✓ Being able to implement your own RGB colors is most useful when you need many colors, the collection of which is referred to as a palette.
- ✓ There are a number of color palettes built into the base R installation. These are defined by the functions rainbow, heat.colors, terrain. colors, topo.colors, cm.colors, gray.colors, and gray.

```
N <- 600
rbow <- rainbow(N)
heat <- heat.colors(N)
terr <- terrain.colors(N)
topo <- topo.colors(N)
cm <- cm.colors(N)
gry1 <- gray.colors(N)
gry2 <- gray(level=seq(0,1,length=N))
```

Color Palettes: R provides color palettes that allow you to use a sequence of colors.

Example:

```
colors <- rainbow(10) # Generates 10 colors of the
barplot(1:10, col=colors)
```



Color Functions:

R has functions for creating color gradients or interpolating colors.

Example:

```
library(RColorBrewer)

# Generate a gradient of colors from the "Blues" palette

colors <- colorRampPalette(brewer.pal(9, "Blues"))(10)
```

```
pie(rep(1, 10), col = colors, main = "Pie Chart with Gradient Colors")
```

Color Scales:

A **color scale** in data visualization refers to the mapping of data values (typically continuous or categorical) to colors. It helps convey information about the distribution, patterns, or relationships within the data through visual means.

in R, color scales can be created and customized using various packages such as RColorBrewer, viridis, and built-in functions like rainbow() or heat.colors(). Here are a few ways to implement color scales:

```
library(viridis)

plot(1:10, col=viridis(10), pch=16)
```

```
# Create a color vector with 10 colors

cols <- heat.colors(10)

# Example plot

plot(1:10, col = cols, pch = 16, cex = 2)
```

3D SCATTER PLOTS

- ✓ Creating 3D scatter plots in R is a great way to visualize three-dimensional data. There are several packages available for this purpose, with popular choices including plotly, rgl, and scatterplot3d
- ✓ 3D scatterplots allow you to plot raw observations based on three continuous variables at once.
- ✓ The syntax of the scatterplot3d function is similar to the default plot function.
- ✓ In the latter, you supply a vector of x- and y-axis coordinates; in the former, you merely supply an additional third vector of values providing the z-axis coordinates.
- ✓ **Syntax: scatterplot3d(x, y=NULL, z=NULL)**
- ✓ x, y, z are the coordinates of points to be plotted.

Example:

```
library(scatterplot3d)
set.seed(42)
n <- 100
x <- rnorm(n)
y <- rnorm(n)
z <- rnorm(n)
```

```
# Create a 3D scatter plot
scatterplot3d(x, y, z, color = "blue", pch = 19, main = "3D Scatter Plot with scatterplot3d",xlab = "X-Axis",
ylab = "Y-Axis", zlab = "Z-Axis")
```

Packages used to create 3D scatterplot

Creating 3D scatter plots in R can be achieved using different packages. Like

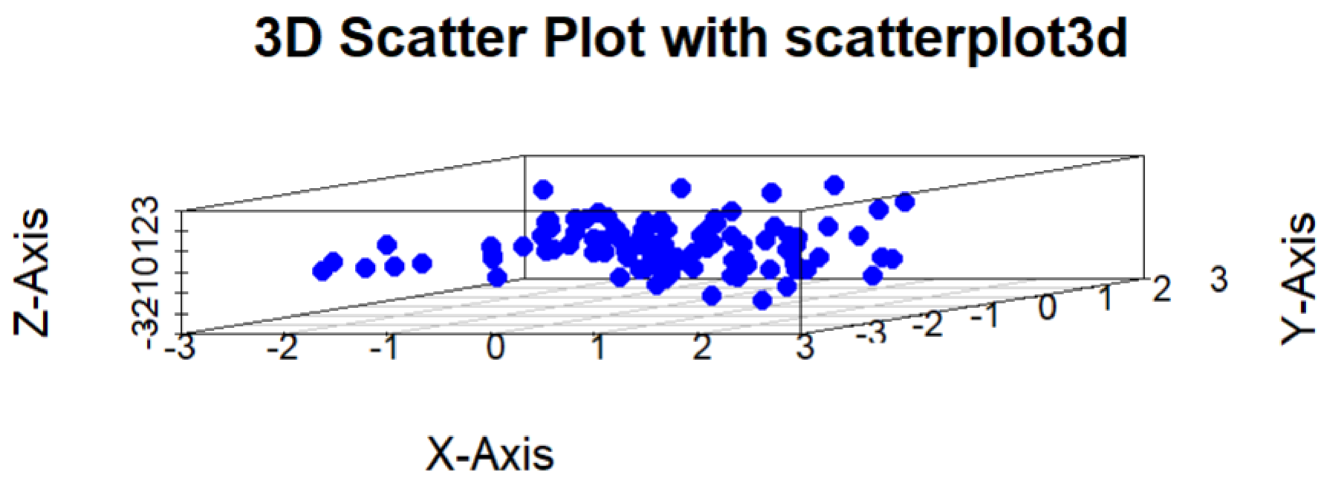
- ✓ 3D Scatter Plot with plotly
- ✓ 3D Scatter Plot with rgl
- ✓ 3D Scatter Plot with scatterplot3d

One popular package for this purpose is scatterplot3d.

Here's a basic example of how to create a 3D scatter plot using this package:

Example:

```
library(scatterplot3d)
set.seed(42)
n <- 100
x <- rnorm(n)
y <- rnorm(n)
z <- rnorm(n)
# Create a 3D scatter plot
scatterplot3d(x, y, z, color = "blue", pch = 19, main = "3D Scatter Plot with scatterplot3d",xlab = "X-Axis",
ylab = "Y-Axis", zlab = "Z-Axis")
```


3D Scatter Plot with scatterplot3d
Z-Axis
Y-Axis
X-Axis